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

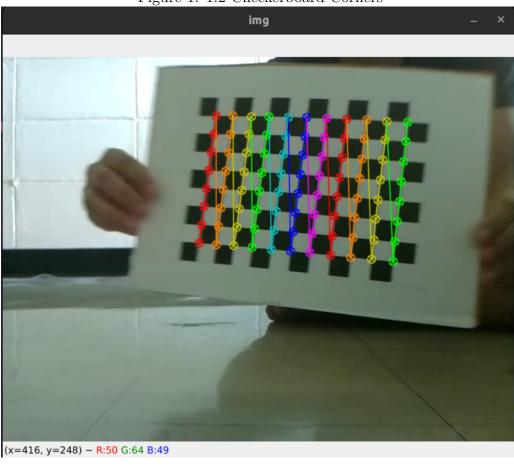


Figure 1: 1.2 Checkerboard Corners

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

```
🗬 main
 Camera matrix:
 [[544.06254343
                          321.767787 ]
                          271.29350075]
              548.01458
                Θ.
                            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

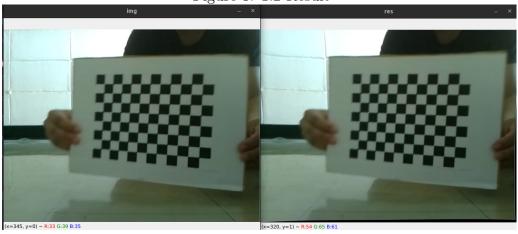


Figure 4: 1.2 Re-projection Error

```
Total error: 0.10270851690199258

Process finished with exit code 0
```

```
6 # Define the dimensions of checkerboard
7 \text{ CHECKERBOARD} = (8, 11)
8 \text{ criteria} = (\text{cv.TERM\_CRITERIA\_EPS} + \text{cv.}
     TERM_CRITERIA_MAX_ITER, 30, 0.001)
10 # Create vector to store vectors of 3D points for each
      checkerboard image
11 \text{ obj-points} = []
12 # Create vector to store vectors of 2D points for each
      checkerboard image
13 \text{ img-points} = []
15 # Define the world coordinates for 3D points
16 obj_point = np.zeros((1, CHECKERBOARD[0] * CHECKERBOARD
      [1], 3), dtype=np.float32)
17 \text{ obj-point}[0, :, :2] = \text{np.mgrid}[0:CHECKERBOARD}[0], 0:
     CHECKERBOARD[1]]. T. reshape (-1, 2)
18 \text{ prev_img\_shape} = \text{None}
19
20 # Extracting path of individual image stored in a given
       directory
21 images = glob.glob("../../Data/Lab03/images/*.jpg")
22 \text{ img} = \text{None}
23 \text{ gray} = \text{None}
24 for f_name in images:
25
26
       img = cv.imread(f_name)
       gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
27
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 If desired number of corner are detected, 34 refine the pixel coordinates and display them on the images of checkerboard 35 36 if ret: 37 38 obj_points.append(obj_point) 39 # Refining pixel coordinates for given 2D 40 points. 41 corners2 = cv.cornerSubPix(gray, corners, (11, (-1, -1), criteria) 42 43 img_points.append(corners2) 44 45# Draw and display the corners img = cv.drawChessboardCorners(img, 46 CHECKERBOARD, corners2, ret) cv.imshow('img', img) 47 cv.waitKey(0) 48 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 corresponding pixel coordinates of the detected 55 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:
       img = cv.imread(f_name)
70
       res = cv.undistort(img, mtx, dist)
71
       cv.imshow('img', img)
72
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(
      mtx, dist, (w, h), 1, (w, h))
85
       res = cv.undistort(img, mtx, dist)
86
87
       # crop the image
       x, y, w, h = roi
88
89
       res = res[y:y+h, x:x+w]
90
91
       cv.imshow('img', img)
92
       cv.imshow('res', res)
93
       cv.waitKey(0)
94
95
96 # Find a mapping function from the distorted image to
      undistorted image.
97 \# Then use the remap function.
98 print ("Find mapping from distorted to undistorted image
      .\nThen use the remap function.")
99 for f_name in images:
100
       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)
       res = cv.remap(img, map_x, map_y, cv.INTER_LINEAR)
107
108
109
       # Crop the image
110
       x, y, w, h = roi
111
       res = res[y:y+h, x:x+w]
112
       cv.imshow('img', img)
113
       cv.imshow('res', res)
114
115
       cv.waitKey(0)
116
117 \text{ mean\_error} = 0
118 for i in range(len(obj_points)):
       img_points2, _ = cv.projectPoints(obj_points[i],
119
      r_vectors[i], t_vectors[i], mtx, dist)
        error = cv.norm(img_points[i], img_points2, cv.
120
      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

```
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
 1 # calibrate_camera.py
 2 import glob
 3 import cv2 as cv
 4 import numpy as np
 5
```

```
6
7 class CameraCalibration:
      camera_mtx = None
9
       distortion_coefficient = None
       rotation_vectors = None
10
       translation_vectors = None
11
12
      # Define the dimensions of checkerboard
13
14
      \_CHECKERBOARD = (8, 11)
       _{-criteria} = (cv.TERM\_CRITERIA\_EPS + cv.
15
     TERM_CRITERIA_MAX_ITER, 30, 0.001)
16
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
       \_\_obj\_point = np.zeros((1, \_CHECKERBOARD[0] *
24
     _CHECKERBOARD[1], 3), dtype=np.float32)
25
       -10 \text{ bj-point} [0, :, :2] = \text{np.mgrid} [0:\_CHECKERBOARD]
      [0], 0:_CHECKERBOARD[1]]. T. reshape(-1, 2)
26
       _prev_img_shape = None
27
28
       _{-img} = None
       -gray = None
29
30
31
       def __init__(self, file_name=None):
           if file_name is not None:
32
33
               self.read_calibration_file(file_name)
34
35
       def calibrate (self, file_path):
           images = glob.glob(file_path + "*.jpg")
36
37
38
           for f_name in images:
               self.__img = cv.imread(f_name)
39
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
     {
m cv} . CALIB_CB_FAST_CHECK + {
m cv} . CALIB_CB_NORMALIZE_IMAGE
45
46
               # If the desired number of corner are
     detected
47
               if ret:
                    self.__obj_points.append(self.
48
      __obj_point)
49
                   # Refining pixel coordinates for given
50
     2D points.
51
                    corners2 = cv.cornerSubPix(self._gray,
      corners, (11, 11), (-1, -1), \text{ self.}_{-criteria}
52
53
                    self.__img_points.append(corners2)
54
                   # Draw and display the corners
55
56
                    self.__img = cv.drawChessboardCorners(
     self._img, self._CHECKERBOARD, corners2, True)
57
               cv.imshow('img', self.__img)
58
               cv.waitKey(0)
59
           cv.destroyAllWindows()
60
61
62
           # Calibrate the camera
           ret, self.camera_mtx, self.
63
      distortion_coefficient, self.rotation_vectors, self.
      translation\_vectors = \setminus
               cv.calibrateCamera(self._obj_points, self.
64
      _{-img\_points}, self._{-gray.shape}[::-1], None, None)
65
66
       def write_calibration_file(self, file_name):
           file_storage = cv.FileStorage(file_name, cv.
67
     FILE_STORAGE_WRITE)
```

68

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

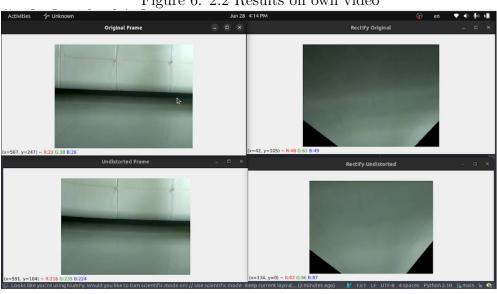


Figure 6: 2.2 Results on own video

Calibrate camera, re-calculate homography from 2.2own video

```
1 # capture_video.py
2 import cv2 as cv
3
5 def capture_video(file_name: str):
       video_capture = cv. VideoCapture (0)
7
8
       fourcc = cv.VideoWriter_fourcc(*"mp4v")
9
       out = cv. VideoWriter(file_name, fourcc, 20.0, (640,
       480))
10
11
       while True:
12
            ret , frame = video_capture.read()
13
14
            out.write(frame)
15
            cv.imshow("Web Camera", frame)
16
17
18
            if \operatorname{cv.waitKey}(30) \& 0xFF = \operatorname{ord}('q'):
19
                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):
       video_capture = cv. VideoCapture(path + file_name)
27
28
       count = 0
29
30
       while True:
           ret , frame = video_capture.read()
31
32
           if frame is None:
33
               break
           cv.imwrite(path + "images/frame%d.jpg" % count,
34
      frame)
35
36
           count += 1
37
           cv.waitKey(30)
38
39
40
1 # calibrate_camera.py
2 import glob
3 import cv2 as cv
4 import numpy as np
5
6
7 class CameraCalibration:
       camera_mtx = None
9
       distortion_coefficient = None
10
       rotation_vectors = None
       translation_vectors = None
11
12
      # Define the dimensions of checkerboard
13
14
      \_CHECKERBOARD = (8, 11)
       _{-}criteria = (cv.TERM_CRITERIA_EPS + cv.
15
     TERM_CRITERIA_MAX_ITER, 30, 0.001)
16
      # Create vector to store vectors of 3D points for
17
```

```
each checkerboard image
       \_obi_points = []
18
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
       \_obj\_point = np.zeros((1, \_CHECKERBOARD[0] *
24
     _CHECKERBOARD[1], 3), dtype=np.float32)
       - obj-point[0, :, :2] = np.mgrid[0:\_CHECKERBOARD]
25
      [0], 0:_CHECKERBOARD[1]]. T. reshape(-1, 2)
       __prev_img_shape = None
26
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
       def calibrate (self, file_path):
35
           images = glob.glob(file_path + "*.jpg")
36
37
38
           for f_name in images:
39
               self.__img = cv.imread(f_name)
               self.__gray = cv.cvtColor(self.__img, cv.
40
     COLOR_BGR2GRAY)
41
42
               # Find the checkerboard corners
43
               ret, corners = cv.findChessboardCorners(
     self._gray, self._CHECKERBOARD, cv.
     CALIB_CB_ADAPTIVE_THRESH
44
     {\rm cv}. CALIB_CB_FAST_CHECK +~{\rm cv}. CALIB_CB_NORMALIZE_IMAGE
45
46
               # If the desired number of corner are
     detected
               if ret:
47
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
                   self.__img_points.append(corners2)
53
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)
               cv.waitKey(30)
59
           cv.destroyAllWindows()
60
61
62
          # Calibrate the camera
63
           ret, self.camera_mtx, self.
     distortion_coefficient, self.rotation_vectors, self.
     translation\_vectors = \setminus
               cv.calibrateCamera(self._obj_points, self.
64
     _{-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
           if not file_storage.isOpened():
69
               return False
70
71
72
           file_storage.write("MTX", self.camera_mtx)
73
           file_storage.write("DIST", self.
     distortion_coefficient)
74
           for i in range (10):
               file_storage.write("R" + str(i), self.
75
     rotation_vectors[i])
76
               file_storage.write("T" + str(i), self.
     translation_vectors[i])
77
           file_storage.release()
78
```

```
79
           return True
80
       def read_calibration_file(self, file_name):
81
82
           file_storage = cv. FileStorage (file_name, cv.
     FILE_STORAGE_READ)
           if not file_storage.isOpened():
83
84
                return False
85
           self.camera_mtx = file_storage.getNode("MTX").
86
     mat()
87
           self.distortion_coefficient = file_storage.
     getNode ("DIST") . mat ()
           r_{\text{vectors}} = \text{tuple}()
88
89
           t_vectors = tuple()
           for i in range (10):
90
                r_vectors += (file_storage.getNode("R" +
91
     str(i)).mat(),)
                t_vectors += (file_storage.getNode("T" +
92
     str(i)).mat(),)
           self.rotation_vectors = r_vectors
93
           self.translation_vectors = t_vectors
94
95
           file_storage.release()
96
97
           return True
98
1 # homography.py
2 import cv2 as cv
3 import numpy as np
5
6 class Homography:
       mat_h = np.zeros((3, 3))
       width_out: int
8
9
       height_out: int
       c_points: int
10
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
           self.c_points = 0
30
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
           self.mat_h = file_storage.getNode("matH").mat()
36
           self.width_out = int(file_storage.getNode("
37
     widthOut").real())
38
           self.height_out = int(file_storage.getNode("
     heightOut").real())
39
           file_storage.release()
           return True
40
41
42
       def write (self, homography_file):
           file_storage = cv. FileStorage (homography_file,
43
     cv . FILE_STORAGE_WRITE)
           if not file_storage.isOpened():
44
45
               return False
46
47
           for i in range (4):
48
               file_storage.write("aPoints" + str(i), self
      .a_points[i])
49
           file_storage.write("matH", self.mat_h)
50
```

```
file_storage.write("widthOut", self.width_out)
51
           file_storage.write("heightOut", self.height_out
52
     )
53
           file_storage.release()
54
           return True
55
56
57
       def __draw_circle_and_line(self, x, y):
           self.__mat_result = self.__mat_final.copy()
58
59
           self._point = (x, y)
60
61
           if self._-var >= 1:
               cv.line(self.__mat_result, self.__pts[self.
62
     [-var - 1], self. [-point, (0, 255, 0, 255), 2)
           cv.circle(self._mat_result, self._point, 2,
63
     (0, 255, 0), -1, 8, 0)
           cv.imshow("Source", self.__mat_result)
64
65
66
      def _mouse_handler(self, event, x, y, flags, param
     ):
           if self._-var >= 4:
67
68
               return
69
           if event == cv.EVENTLBUTTONDOWN:
70
71
               self._-drag = 1
72
               self.__draw_circle_and_line(x, y)
73
74
           if event = cv.EVENTLBUTTONUP and self.__drag:
               self._-drag = 0
75
76
               self.__pts.append(self.__point)
77
               self._-var += 1
78
               self.__mat_final = self.__mat_result.copy()
79
               if self._var >= 4:
80
81
                   cv.line(self.__mat_final, self.__pts
     [0], self.__pts[3], (0, 255, 0, 255), (0, 255)
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
           if self.__drag:
85
```

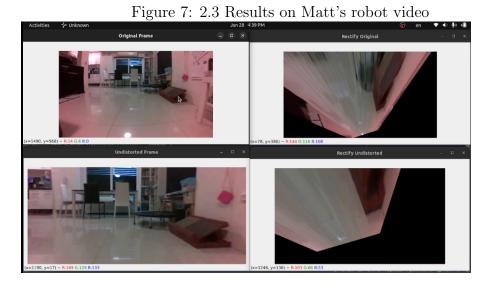
```
86
                self.__draw_circle_and_line(x, y)
87
        def calculate (self, file_name):
88
89
            mat_pause_screen = np.array([])
90
            mat\_frame\_capture = np.array([])
91
            \text{key} = -1
92
93
                               ----- [STEP 1: Make video
      capture from file | ----
           # Open video file
94
            video_capture = cv. VideoCapture(file_name)
95
            if not video_capture.isOpened():
96
97
                print ("ERROR! Unable to open input video
      file ", file_name)
                return False
98
99
100
            width = video_capture.get(cv.
      CAP_PROP_FRAME_WIDTH)
101
            height = video_capture.get(cv.
      CAP_PROP_FRAME_HEIGHT)
            ratio = 640.0 / width
102
            dim = (int(width * ratio), int(height * ratio))
103
104
105
            while key < 0:
106
                # Get the next frame
                _, mat_frame_capture = video_capture.read()
107
                if mat_frame_capture is None:
108
109
                     break
110
111
                mat\_frame\_display = cv.resize(
      mat_frame_capture, dim)
112
                cv.imshow("Original", mat_frame_display)
113
                key = cv.waitKey(30)
114
115
                                 ---- [STEP 2: pause the
116
      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 ] -
            if mat_frame_capture is not None:
124
125
                self._-var = 0
                self._pts.clear()
126
127
                cv.namedWindow("Source", cv.
      WINDOW_GULNORMAL)
128
                cv.setMouseCallback("Source", self.
      _mouse_handler)
                cv.imshow("Source", mat_pause_screen)
129
130
                cv.waitKey(0)
                cv.destroyWindow("Source")
131
132
                if len(self._pts) == 4:
133
134
                    src = np.array(self.__pts).astype(np.
      float32)
135
136
                    reals = np.array([
                        (200, 200),
137
                        (429, 200),
138
139
                        (429, 429),
140
                        (200, 429)
                    dtype=np.float32)
141
142
                       ----- [STEP 4: Calculate
143
      Homography] -
                    homography_matrix = cv.
144
      getPerspectiveTransform(src, reals)
145
                          ----- [STEP 4: Calculate
146
      Homography | -
                    self.mat_h = homography_matrix
147
                    self.c_points = 0
148
149
                    for i in range (4):
150
                        self.a_points.append(src[i])
                        self.c_points += 1
151
                    self.width_out = int(width)
152
153
                    self.height_out = int(height)
```

```
154
155
                    return True
156
            else:
157
                return False
158
159
 1 # main.py
 2 import sys
 3 import cv2 as cv
 4 # from capture_video import capture_video,
      convert_video_to_images
 5 from calibrate_camera import CameraCalibration
 6 from homography import Homography
 8 \text{ PATH} = ".../.../.../ Data/Lab03/checkerboard/"
 9 CALIBRATION_FILE = "checkerboard.mp4"
10 VIDEO_FILE = "car.mp4"
12 # Capture the checkerboard video
13 # capture_video (PATH + CALIBRATION_FILE)
15 # Convert the checkerboard video to images
16 # convert_video_to_images (CALIBRATION_FILE, PATH)
17
18 # Calibrate the camera
19 camera_calibration = CameraCalibration()
20 camera_calibration.calibrate(PATH + "images/")
21 camera_calibration.write_calibration_file("
      camera_calibration.yml")
22
23 print ("Camera matrix: ", camera_calibration.camera_mtx)
24 print ("Distortion coefficient: ", camera_calibration.
      distortion_coefficient)
25 print ("Rotation vectors: ", camera_calibration.
      rotation_vectors)
26 print ("Translation vectors: ", camera_calibration.
      translation_vectors)
27
28 # Calculate homography
29 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():
       print ("ERROR! Unable to open video file",
38
     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 \text{ ratio} = 640.0 / \text{ width}
43 dim = (int(width * ratio), int(height * ratio))
44
45 \text{ duration} = 0
46 while True:
       _, frame = video_capture.read()
47
      if frame is None:
48
           break
49
50
51
      # Undistorted and Cropped the frame
52
      undistorted_frame = cv.undistort(frame,
     camera_calibration.camera_mtx,
53
                                          camera_calibration
     . distortion_coefficient)
      h, w = frame.shape[:2]
54
55
      new_camera_mtx, roi = cv.getOptimalNewCameraMatrix(
     camera_calibration.camera_mtx,
56
     camera_calibration.distortion_coefficient,
57
     (w, h), 1, (w, h)
      x, y, w, h = roi
58
59
      cropped_undistorted_frame = undistorted_frame[y:y+h
     , x:x+w
60
      # Show the original and undistorted frames
61
      cv.namedWindow("Original Frame", cv.WINDOWNORMAL |
62
```

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



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:
      camera_mtx = None
      distortion_coefficient = None
9
10
      rotation_vectors = None
      translation_vectors = None
11
12
      # Define the dimensions of checkerboard
13
      \_CHECKERBOARD = (6, 9)
14
15
       _{-}criteria = (cv.TERM_CRITERIA_EPS + cv.
     TERM_CRITERIA_MAX_ITER, 30, 0.001)
16
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
       -10 \text{ bj-point} [0, :, :2] = \text{np.mgrid} [0:\_CHECKERBOARD]
      [0], 0:_CHECKERBOARD[1]]. T. reshape(-1, 2)
       __prev_img_shape = None
26
27
28
       _{-img} = None
29
       -gray = None
30
       def __init__(self, file_name=None):
31
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
           for f_name in images:
38
39
               self.__img = cv.imread(f_name)
               self.__gray = cv.cvtColor(self.__img, cv.
40
     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
               # If the desired number of corner are
46
     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, (9, 9), (-1, -1), self._criteria)
52
53
                    self.__img_points.append(corners2)
54
                    # Draw and display the corners
55
                    self.__img = cv.drawChessboardCorners(
56
      self._img, self._CHECKERBOARD, corners2, True)
57
                \operatorname{cv.imshow}(\operatorname{'img'}, \operatorname{self.}\operatorname{--img})
58
59
                cv.waitKey(30)
60
           cv.destroyAllWindows()
61
62
           # Calibrate the camera
           ret, self.camera_mtx, self.
63
      distortion_coefficient, self.rotation_vectors, self.
      translation\_vectors = \setminus
64
                cv.calibrateCamera(self._obj_points, self.
      _{-img-points}, self. _{-gray.shape}[::-1], None, None)
65
       def write_calibration_file(self, file_name):
66
            file_storage = cv.FileStorage(file_name, cv.
67
     FILE_STORAGE_WRITE)
68
69
            if not file_storage.isOpened():
70
                return False
71
72
            file_storage.write("MTX", self.camera_mtx)
            file_storage.write("DIST", self.
73
      distortion_coefficient)
74
            for i in range (10):
                file_storage.write("R" + str(i), self.
75
      rotation_vectors[i])
                file_storage.write("T" + str(i), self.
76
      translation_vectors[i])
77
            file_storage.release()
78
79
           return True
80
       def read_calibration_file(self, file_name):
81
82
            file_storage = cv. FileStorage (file_name, cv.
     FILE_STORAGE_READ)
```

```
83
           if not file_storage.isOpened():
                return False
84
85
86
            self.camera_mtx = file_storage.getNode("MTX").
     mat()
            self.distortion_coefficient = file_storage.
87
      getNode("DIST").mat()
           r_{\text{vectors}} = \text{tuple}()
88
            t_{\text{vectors}} = \text{tuple}()
89
90
           for i in range (10):
91
                r_vectors += (file_storage.getNode("R" +
      str(i)).mat(),)
                t_vectors += (file_storage.getNode("T" +
92
      str(i)).mat(),)
93
            self.rotation_vectors = r_vectors
            self.translation_vectors = t_vectors
94
95
            file_storage.release()
96
97
           return True
98
1 # homography.py
2 import cv2 as cv
3 import numpy as np
5
6 class Homography:
       mat_h = np.zeros((3, 3))
8
       width_out: int
9
       height_out: int
10
       c_points: int
       a_points: list = []
11
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:
               self.read(homography_file)
23
24
25
       def read(self, homography_file: str):
26
           file_storage = cv. FileStorage (homography_file,
     cv . FILE_STORAGE_READ)
           if not file_storage.isOpened():
27
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()
           self.width_out = int(file_storage.getNode("
37
     widthOut").real())
           self.height_out = int(file_storage.getNode("
38
     heightOut").real())
39
           file_storage.release()
40
           return True
41
       def write(self, homography_file):
42
           file_storage = cv. FileStorage (homography_file,
43
     cv . FILE_STORAGE_WRITE)
           if not file_storage.isOpened():
44
               return False
45
46
           for i in range (4):
47
               file_storage.write("aPoints" + str(i), self
48
     .a_points[i])
49
           file_storage.write("matH", self.mat_h)
50
           file_storage.write("widthOut", self.width_out)
51
52
           file_storage.write("heightOut", self.height_out
     )
53
           file_storage.release()
54
```

```
return True
55
56
       def __draw_circle_and_line(self, x, y):
57
58
           self.__mat_result = self.__mat_final.copy()
59
           self._-point = (x, y)
60
           if self._-var >= 1:
61
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)
           cv.imshow("Source", self.__mat_result)
64
65
       def __mouse_handler(self, event, x, y, flags, param
66
     ):
67
           if self._-var >= 4:
68
               return
69
70
           if event = cv.EVENTLBUTTONDOWN:
               self._-drag = 1
71
72
               self.__draw_circle_and_line(x, y)
73
74
           if event = \text{cv.EVENTLBUTTONUP} and \text{self.}_{--}\text{drag}:
               self._-drag = 0
75
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), (0, 255)
82
                    cv.fillConvexPoly(self.__mat_final, np.
     array (self.__pts, dtype=np.int32), (0, 120, 0, 20))
               cv.imshow("Source", self.__mat_final)
83
84
           if self.__drag:
85
               self._draw_circle_and_line(x, y)
86
87
88
       def calculate (self, file_name):
89
           mat_pause_screen = np.array([])
90
           mat\_frame\_capture = np.array([])
```

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

```
if mat_frame_capture is not None:
124
                self._var = 0
125
                self.__pts.clear()
126
127
                cv.namedWindow("Source", cv.
      WINDOW_GULNORMAL)
128
                cv.setMouseCallback("Source", self.
      _mouse_handler)
                cv.imshow("Source", mat_pause_screen)
129
                cv.waitKey(0)
130
                cv.destroyWindow("Source")
131
132
                if len(self._-pts) == 4:
133
                    src = np.array(self._pts).astype(np.
134
      float32)
135
                    reals = np.array([
136
137
                        (800, 800),
                        (1000, 800),
138
                        (1000, 1000),
139
                        (800, 1000)
140
                    ], dtype=np.float32)
141
142
                       ----- [STEP 4: Calculate
143
      Homography | -
                    homography_matrix = cv.
144
      getPerspectiveTransform(src, reals)
145
                         STEP 4: Calculate
146
      Homography] -
                    self.mat_h = homography_matrix
147
148
                    self.c_points = 0
                    for i in range (4):
149
                        self.a_points.append(src[i])
150
                        self.c_points += 1
151
152
                    self.width_out = int(width)
                    self.height_out = int(height)
153
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
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")
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():
       print ("ERROR! Unable to open video file",
30
     VIDEO_FILE)
      sys.exit()
31
32 width = video_capture.get(cv.CAP_PROP_FRAME_WIDTH)
33 height = video_capture.get(cv.CAP_PROP_FRAME_HEIGHT)
34 \text{ ratio} = 600.0 / \text{ width}
35 dim = (int(width * ratio), int(height * ratio))
```

```
36
37 \text{ duration} = 0
38 while True:
39
       _, frame = video_capture.read()
       if frame is None:
40
           break
41
42
43
      # Undistorted and Cropped the frame
       h, w = frame.shape[:2]
44
       new_camera_mtx, roi = cv.getOptimalNewCameraMatrix(
45
      camera_calibration.camera_mtx,
46
      camera_calibration.distortion_coefficient,
47
      (w, h), 1, (w, h)
       undistorted_frame = cv.undistort(frame,
48
      camera_calibration.camera_mtx,
49
                                           camera_calibration
      . distortion_coefficient)
       x, y, w, h = roi
50
51
       cropped_undistorted_frame = undistorted_frame[y:y+h
      , x:x+w
52
53
      # Show the original and undistorted frames
       cv.namedWindow("Original Frame", cv.WINDOW_NORMAL |
54
       cv.WINDOW_KEEPRATIO | cv.WINDOW_GULEXPANDED)
       cv.imshow("Original Frame", frame)
55
       \operatorname{cv.namedWindow}(\operatorname{"Undistorted Frame"}, \operatorname{cv.}
56
     WINDOW.NORMAL | cv.WINDOW.KEEPRATIO | cv.
     WINDOW_GULEXPANDED)
       cv.imshow("Undistorted Frame",
57
      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)
      cv.namedWindow("Rectify Original", cv.WINDOW_NORMAL
66
        cv.WINDOW_KEEPRATIO | cv.WINDOW_GULEXPANDED)
      cv.imshow("Rectify Original", result_original)
67
      cv.namedWindow("Rectify Undistorted", cv.
68
     WINDOWNORMAL | cv.WINDOWKEEPRATIO | cv.
     WINDOW_GULEXPANDED)
      cv.imshow("Rectify Undistorted", result_undistorted
69
70
71
      cv.waitKey(duration)
72
      if duration = 0:
73
           duration = 30
74 cv. destroy All Windows ()
75
```

2.4 Do you get better result?

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