# Lab-02 Report

st122246

June 21, 2022

## 1 Q1: Show Image

```
1 import cv2 as cv
2
3 IMAGE_FILE = "../../../Data/sample.jpg"
4
5 if __name__ == "__main__":
6   img = cv.imread(IMAGE_FILE)
7   if img is None:
8     print("Error: No image to show")
9
10  cv.imshow("Input image", img)
11
12  # Wait up to 5s for a key press
13  cv.waitKey(5000)
14
```

## 2 Q2: Show Video

```
1 import sys
2 import cv2 as cv
3
4 VIDEO_FILE = "../../../Data/robot.mp4"
5 ROTATE = False
6
7 if __name__ == "__main__":
8     key = -1
9
```

Figure 1: Q1: Show Image

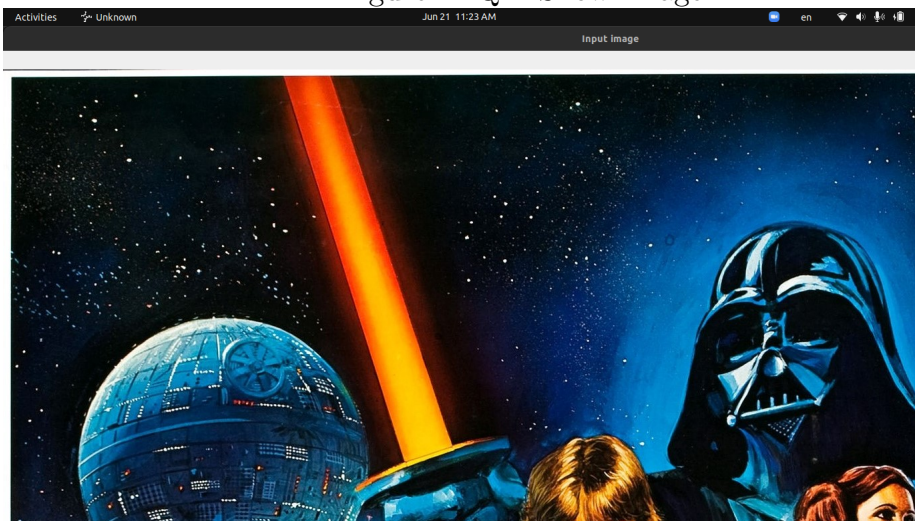
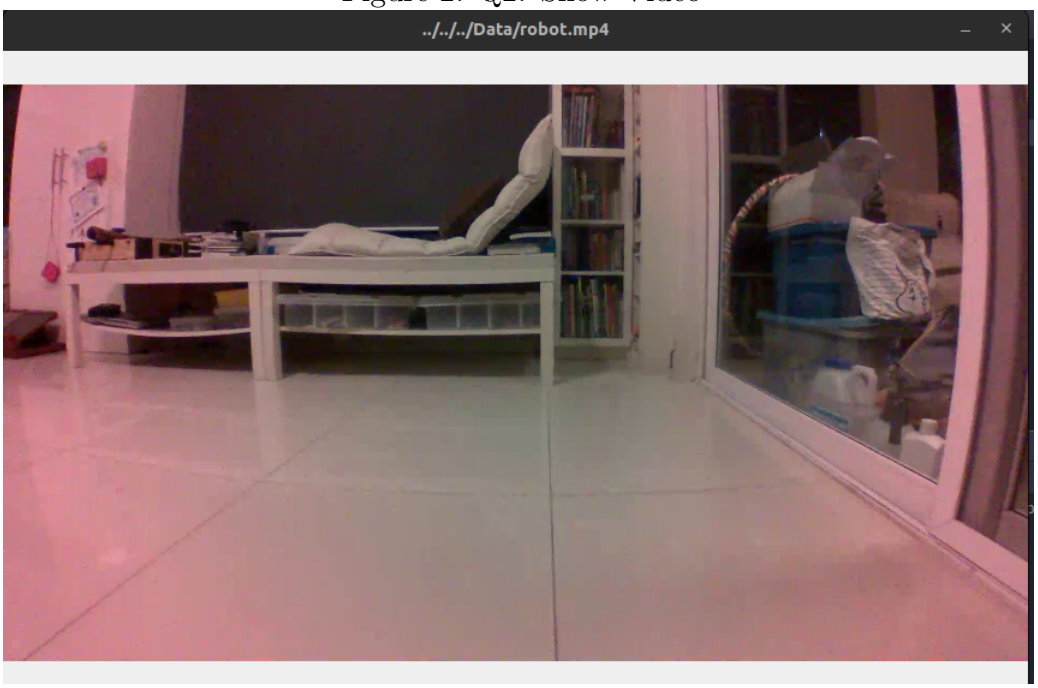
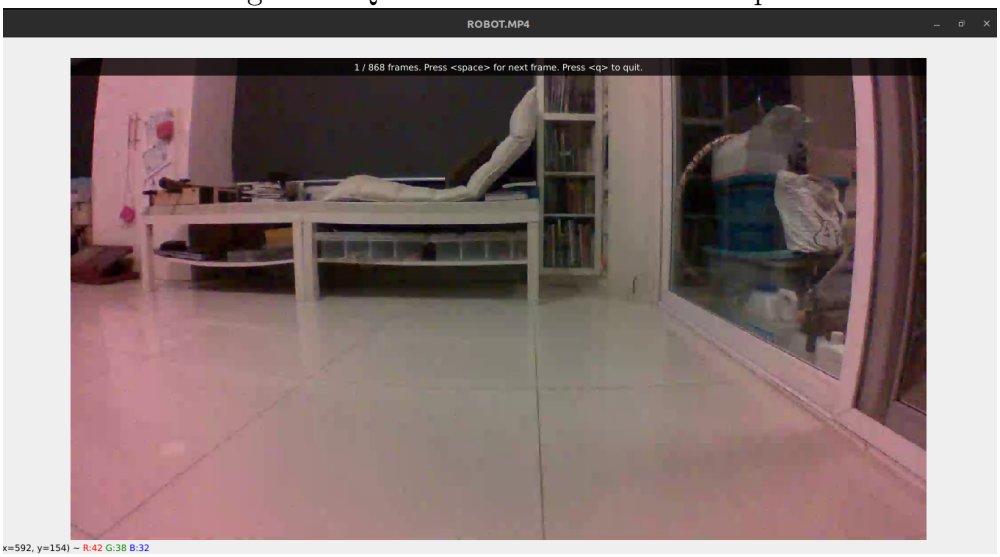



Figure 2: Q2: Show Video



```
10      # Open video file
11      videoCapture = cv.VideoCapture(VIDEO_FILE)
12      height = videoCapture.get(cv.CAP_PROP_FRAME_HEIGHT)
13      width = videoCapture.get(cv.CAP_PROP_FRAME_WIDTH)
14
15      if not videoCapture.isOpened():
16          print("ERROR! Unable to open \
17      video file {}".format(VIDEO_FILE))
18          sys.exit()
19
20      # Capture loop
21      while key != ord(' '):
22          # Get the next frame
23          _, matFrameCapture = videoCapture.read()
24          if matFrameCapture is None:
25              # End of video
26              break
27
28          # Rotate Video
29          if ROTATE:
30              # Rotate 180 degree and put \
31        image to matFrameDisplay
32              _, matFrameDisplay = cv.rotate(
    matFrameCapture,
33        cv.ROTATE_180)
34          else:
35              matFrameDisplay = matFrameCapture
36
37          # Resize the frame
38          ratio = 480.0 / height
39          down_height = int(height * ratio)
40          down_width = int(width * ratio)
41          matFrameDisplay = cv.resize(matFrameDisplay,
42                                      (down_width,
    down_height),
43                                      ratio, ratio, cv.
    INTER_LINEAR)
44
45          # Display the frame
46          cv.imshow(VIDEO_FILE, matFrameDisplay)
47
```

Figure 3: Q3: Show Video With Description




```
48          key = cv.waitKey(30)
```

# 3 Q3: In-lab exercise 1

1. Use waitKey() to wait for the user to press ¡space¿ to advance to the next frame or 'q' to quit. Check the documentation for waitKey(), change the delay parameter to 0 for an infinite wait, and perform the necessary action on a 'spacebar' or 'q' key.

2. Display the full 1080p or 720p frame from the video without making the display window too big for your desktop. Take a look at the documentation for namedWindow() and figure out which flags you should use to set up your display window to be resizable but keep the aspect ratio and display the expanded GUI.

3. Next we probably want to give the user some useful information. Check out the documentation for displayOverlay() and add some explanatory information for the user about frame number, total frames, and user control actions.

4. Last little detail: currently, if the user closes the window, the program doesn't exit.Modify your program to exit when image display window is closed.

```
1 import sys
2 import cv2 as cv
3
4 VIDEO_FILE = "../../../Data/robot.mp4"
5 ROTATE = False
6
7 if __name__ == "__main__":
8   key = ord(" ")
9
10  # Open video file
11  videoCapture = cv.VideoCapture(VIDEO_FILE)
12  height = videoCapture.get(cv.CAP_PROP_FRAME_HEIGHT)
13  width = videoCapture.get(cv.CAP_PROP_FRAME_WIDTH)
14  totalFrameNum = int(videoCapture.get(cv.
    CAP_PROP_FRAME_COUNT))
15  if not videoCapture.isOpened():
16    print("ERROR! Unable to open video file {}".format(
    VIDEO_FILE))
17    sys.exit()
18
19  # Capture loop
20  while True:
21    # Get the next frame when press <space>
22    if key == ord(" "):
23      _, matFrameCapture = videoCapture.read()
24      currentFrameNum = int(videoCapture.get(cv.
    CAP_PROP_POS_FRAMES))
25      if matFrameCapture is None:
26        # End of video
27        break
28
29      # Rotate Video
30      if ROTATE:
31        # Rotate 180 degree and put image to
    matFrameDisplay
32        matFrameDisplay = cv.rotate(matFrameCapture, cv
    .ROTATE_180)
33      else:
34        matFrameDisplay = matFrameCapture
35
36      # Resize the frame
```

```
37        height_ratio = 768.0 / height
38        width_ratio = 1366.0 / width
39        down_height = int(height * height_ratio)
40        down_width = int(width * width_ratio)
41        matFrameDisplay = cv.resize(matFrameDisplay,
42                        (down_width, down_height),
43                        width_ratio, height_ratio,
44                        cv.INTER_LINEAR)
45
46       # 1366p x 768p frame display in resizeable, keep
      aspect ratio, and show expended GUI
47       cv.namedWindow("ROBOT.MP4", flags=cv.
      WINDOW_NORMAL | cv.WINDOW_KEEPRATIO | cv.
      WINDOW_GUI_EXPANDED)
48
49       # Display the frame
50       cv.imshow("ROBOT.MP4", matFrameDisplay)
51
52       # Display overlay explanatory information
53       explanatory_info = f"{currentFrameNum} / {
      totalFrameNum} " \
54                   f"frames. Press <space> for next frame.
       Press <q> to quit."
55       cv.displayOverlay("ROBOT.MP4", explanatory_info)
56
57     # Quit when press <q>
58     if key == ord('q'):
59       break
60
61     # Exit program on window close
62     if cv.getWindowProperty("ROBOT.MP4", cv.
      WND_PROP_VISIBLE) != 1:
63       break
64
65     key = cv.pollKey()
66   cv.destroyAllWindows()
```

# 4 Q4: In-lab exercise 2

Figure 4: Q4: Select 4 Points

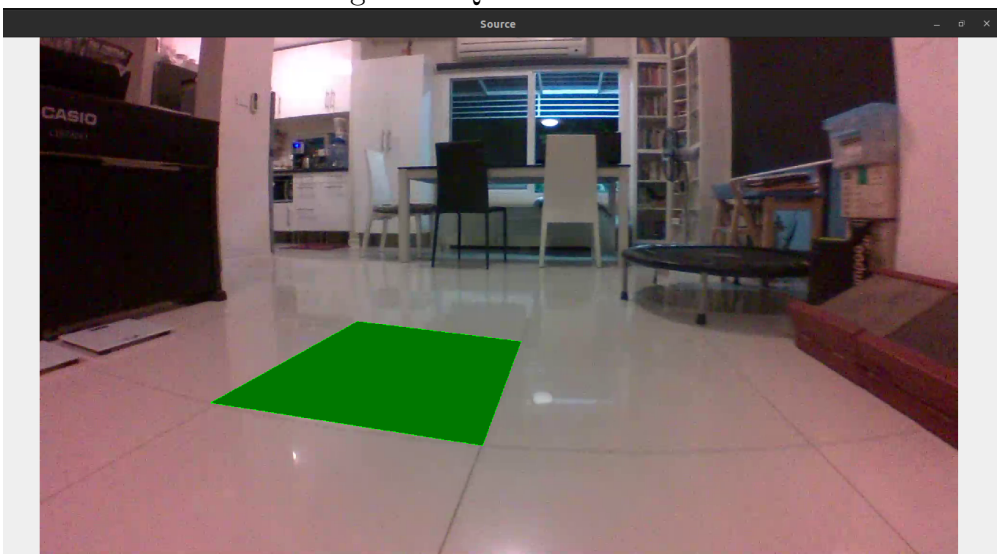



Figure 5: Q4: Estimated Homography

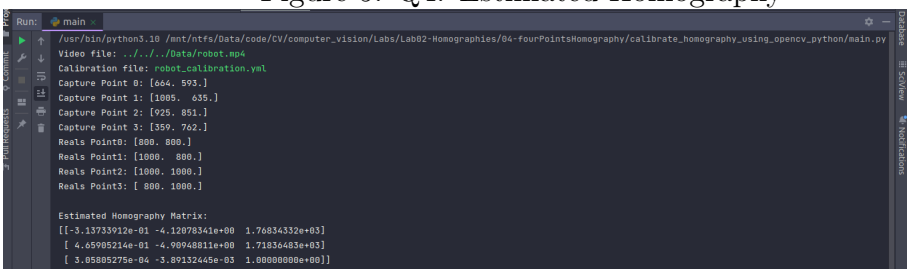



Figure 6: Q4: Warped Image

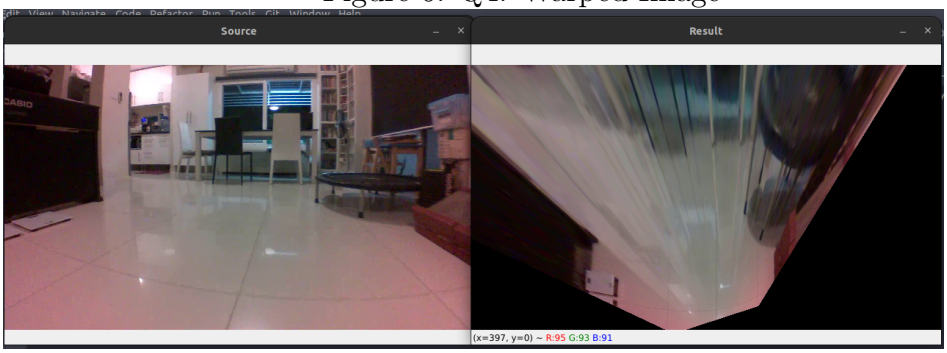
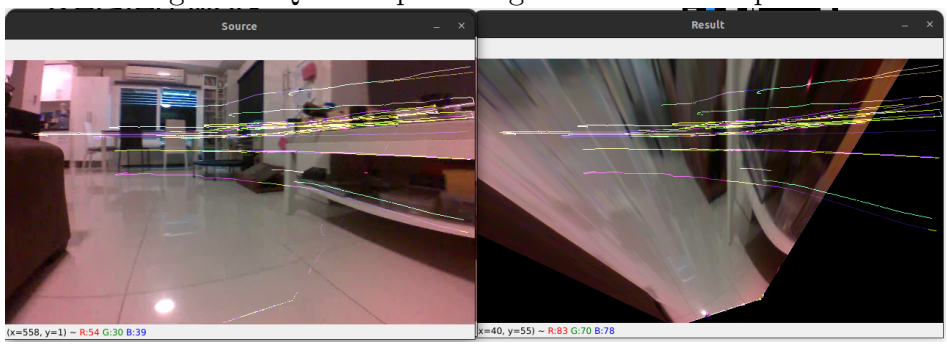


Figure 7: Q4: Warped Image and Perform Optical Flow




```
1   import sys
2   import cv2 as cv
3   import numpy as np
4   from os.path import exists
5
6   point = (-1, -1)
7   pts = []
8   var = 0
9   drag = 0
10  mat_final = np.array([])
11  mat_result = np.array([])
12
13  feature_params = dict(
14    maxCorners=100,
15    qualityLevel=0.3,
16    minDistance=7,
17    blockSize=7
18  )
19  lk_params = dict(
20    winSize=(15, 15),
21    maxLevel=2,
22    criteria=(cv.TERM_CRITERIA_EPS | cv.
     TERM_CRITERIA_COUNT, 10, 0.03)
23  )
24
25  color = np.random.randint(0, 255, (100, 3))
26  VIDEO_FILE = "../../../Data/robot.mp4"
27  ROTATE = False
28  CALIBRATION_FILE = "calibration.yml"
```

```
29  CALIBRATE = False
30
31
32  def draw_circle_and_line(x, y):
33    global point, pts, mat_result
34    mat_result = mat_final.copy()
35    point = (x, y)
36
37    if var >= 1:
38      cv.line(mat_result, pts[var - 1], point, (0, 255,
      0, 255), 2)
39    cv.circle(mat_result, point, 2, (0, 255, 0), -1, 8,
      0)
40    cv.imshow("Source", mat_result)
41
42
43  def open_video(file_name):
44    video_capture = cv.VideoCapture(file_name)
45    if not video_capture.isOpened():
46      print("ERROR! Unable to open input video file ",
     VIDEO_FILE)
47      sys.exit('Unable to open input video file')
48
49    width = video_capture.get(cv.CAP_PROP_FRAME_WIDTH)
50    height = video_capture.get(cv.CAP_PROP_FRAME_HEIGHT
     )
51    ratio = 640.0 / width
52    dim = (int(width * ratio), int(height * ratio))
53
54    return video_capture, width, height, ratio, dim
55
56
57  def mouse_handler(event, x, y, flags, param):
58    global point, pts, var, drag, mat_final, mat_result
59
60    if var >= 4:
61      return
62
63    if event == cv.EVENT_LBUTTONDOWN:
64      drag = 1
65      draw_circle_and_line(x, y)
```

```
66
67      if event == cv.EVENT_LBUTTONUP and drag:
68        drag = 0
69        pts.append(point)
70        var += 1
71        mat_final = mat_result.copy()
72
73        if var >= 4:
74          cv.line(mat_final, pts[0], pts[3], (0, 255, 0,
      255), 2)
75          cv.fillConvexPoly(mat_final, np.array(pts,
      dtype=np.int32), (0, 120, 0, 20))
76        cv.imshow("Source", mat_final)
77
78      if drag:
79        draw_circle_and_line(x, y)
80
81
82    def calibrate():
83      global mat_final, mat_result, var, CALIBRATE
84      CALIBRATE = True
85
86      mat_pause_screen = np.array([])
87      mat_frame_capture = np.array([])
88      key = -1
89
90      # ---------------------- [STEP 1: Make video capture
      from file] ----------------------
91      # Open video file
92      video_capture, width, height, ratio, dim =
      open_video(VIDEO_FILE)
93
94      # Capture loop
95      while key < 0:
96        # Get the next frame
97        _, mat_frame_capture = video_capture.read()
98        if mat_frame_capture is None:
99          break
100
101       # Rotate
102       if ROTATE:
```

```
103         _, mat_frame_display = cv.rotate(
    mat_frame_capture, cv.ROTATE_180)
104       else:
105         mat_frame_display = mat_frame_capture
106
107       mat_frame_display = cv.resize(mat_frame_display,
    dim)
108
109       cv.imshow("ROBOT.MP4", mat_frame_display)
110       key = cv.waitKey(30)
111
112       # ---------------------- [STEP 2: pause the screen
     and show an image] ----------------------
113       if key >= 0:
114         mat_pause_screen = mat_frame_capture
115         mat_final = mat_pause_screen.copy()
116
117     # ---------------------- [STEP 3: use mouse handler
    to select 4 points] ----------------------
118     if mat_frame_capture is not None:
119       var = 0
120       pts.clear()
121       cv.namedWindow("Source", cv.WINDOW_GUI_NORMAL)
122       cv.setMouseCallback("Source", mouse_handler)
123       cv.imshow("Source", mat_pause_screen)
124       cv.waitKey(0)
125       cv.destroyWindow("Source")
126
127       if len(pts) == 4:
128         src = np.array(pts).astype(np.float32)
129         for i, item in enumerate(src):
130           print(f"Capture Point {i}: {item}")
131
132         reals = np.array([
133           (800, 800),
134           (1000, 800),
135           (1000, 1000),
136           (800, 1000)
137         ], dtype=np.float32)
138         for i, item in enumerate(reals):
139           print(f"Reals Point{i}: {item}")
```

```
140
141         # ---------- [STEP 4: Calculate Homography] ----------
142         homography_matrix = cv.getPerspectiveTransform(src, reals)
143         print("\nEstimated Homography Matrix: ")
144         print(homography_matrix)
145
146         # ---------- [STEP 5: Save Homography to file] ----------
147         cv_file = cv.FileStorage(CALIBRATION_FILE, cv.FILE_STORAGE_WRITE)
148         cv_file.write("H", homography_matrix)
149         cv_file.release()
150
151         # ---------- [STEP 6: Warp Inverse Bi-linear Interpolation] ----------
152         h, w, ch = mat_pause_screen.shape
153         mat_result_frame = cv.warpPerspective(mat_pause_screen, homography_matrix,
154                                               (mat_pause_screen[1], mat_pause_screen[0]), cv.INTER_LINEAR)
155
156         mat_source_frame = cv.resize(mat_pause_screen, dim)
157         cv.imshow("Source", mat_source_frame)
158
159         mat_result_frame = cv.resize(mat_result_frame, dim)
160         cv.imshow("Result", mat_result_frame)
161
162         cv.waitKey(0)
163       else:
164         print("Required 4 points to calculate Homography!")
165     else:
166       print("No pause before end of video finish. Exiting.")
167
168
169   def show_source_result_with_optical_flow():
```

```
170     cv_file = cv.FileStorage(CALIBRATION_FILE, cv.
        FILE_STORAGE_READ)
171     homography_matrix = cv_file.getNode("H").mat()
172     cv_file.release()
173
174     # Open video file
175     video_capture, width, height, ratio, dim =
        open_video(VIDEO_FILE)
176     _, frame = video_capture.read()
177     old_frame_source = frame
178     old_frame_result = cv.warpPerspective(frame,
        homography_matrix, (frame.shape[1], frame.shape[0]),
        cv.INTER_LINEAR)
179
180     old_gray_source = cv.cvtColor(old_frame_source, cv.
        COLOR_BGR2GRAY)
181     old_gray_result = cv.cvtColor(old_frame_result, cv.
        COLOR_BGR2GRAY)
182
183     p0_source = cv.goodFeaturesToTrack(old_gray_source,
        mask=None, **feature_params)
184     p0_result = cv.goodFeaturesToTrack(old_gray_result,
        mask=None, **feature_params)
185
186     mask_source = np.zeros_like(old_frame_source)
187     mask_result = np.zeros_like(old_gray_result)
188
189     while True:
190       ret, frame = video_capture.read()
191       if not ret:
192         print("No frames grabbed!")
193         break
194
195       frame_source = frame
196       frame_result = cv.warpPerspective(frame,
        homography_matrix, (frame.shape[1], frame.shape[0]),
        cv.INTER_LINEAR)
197
198       frame_gray_source = cv.cvtColor(frame_source, cv.
        COLOR_BGR2GRAY)
199       frame_gray_result = cv.cvtColor(frame_result, cv.
```

```
    COLOR_BGR2GRAY)
200
201         p1_source, st_source, err_source = cv.
    calcOpticalFlowPyrLK(old_gray_source,
    frame_gray_source, p0_source, None,
202                                             **lk_params)
203         p1_result, st_result, err_result = cv.
    calcOpticalFlowPyrLK(old_gray_result,
    frame_gray_result, p0_result, None,
204                                             **lk_params)
205
206         if p1_source is not None and p1_result is not
    None:
207             good_new_source = p1_source[st_source == 1]
208             good_old_source = p0_source[st_source == 1]
209             good_new_result = p1_result[st_result == 1]
210             good_old_result = p1_result[st_result == 1]
211
212         for i, (new_source, old_source, new_result,
    old_result) in enumerate(zip(good_new_source,
    good_old_source,
213
    good_new_result, good_old_result)):
214             a_source, b_source = new_source.ravel()
215             c_source, d_source = old_source.ravel()
216             a_result, b_result = new_result.ravel()
217             c_result, d_result = old_result.ravel()
218
219             mask_source = cv.line(mask_source, (int(
    a_source), int(b_source)), (int(c_source), int(
    d_source)),
220                                   color[i].tolist(), 2)
221             mask_result = cv.line(mask_source, (int(
    a_result), int(b_result)), (int(c_result), int(
    d_result)),
222                                   color[i].tolist(), 2)
223             frame_source = cv.circle(frame_source, (int(
    a_source), int(b_source)), 5, color[i].tolist(), -1)
224             frame_result = cv.circle(frame_result, (int(
    a_result), int(b_result)), 5, color[i].tolist(), -1)
225
```

```
226         img_source = cv.add(frame_source, mask_source)
227         img_result = cv.add(frame_result, mask_result)
228 
229         img_source_resize = cv.resize(img_source, dim)
230         img_result_resize = cv.resize(img_result, dim)
231 
232         cv.imshow('Source', img_source_resize)
233         cv.imshow('Result', img_result_resize)
234 
235         key = cv.waitKey(30) & 0xff
236         if key == 27:
237           break
238 
239         old_gray_source = frame_gray_source.copy()
240         p0_source = good_new_source.reshape(-1, 1, 2)
241         old_gray_result = frame_gray_result.copy()
242         p0_result = good_new_result.reshape(-1, 1, 2)
243 
244       cv.destroyAllWindows()
245 
246 
247     if __name__ == "__main__":
248       VIDEO_FILE = input("Video file: ")
249       CALIBRATION_FILE = input("Calibration file: ")
250 
251       if exists(CALIBRATION_FILE):
252         show_source_result_with_optical_flow()
253       else:
254         calibrate()
255         if exists(CALIBRATION_FILE):
256           show_source_result_with_optical_flow()
```

# 5 Q5: Exercise

1. Calculate the homography manually using the SVD of the linear system design matrix similar to the null space solution from class.

2. Compute the warped image manually using the inverse of the homography and bilinear interpolation in the input image.

Figure 8: Q5: Homography Verified by Octave

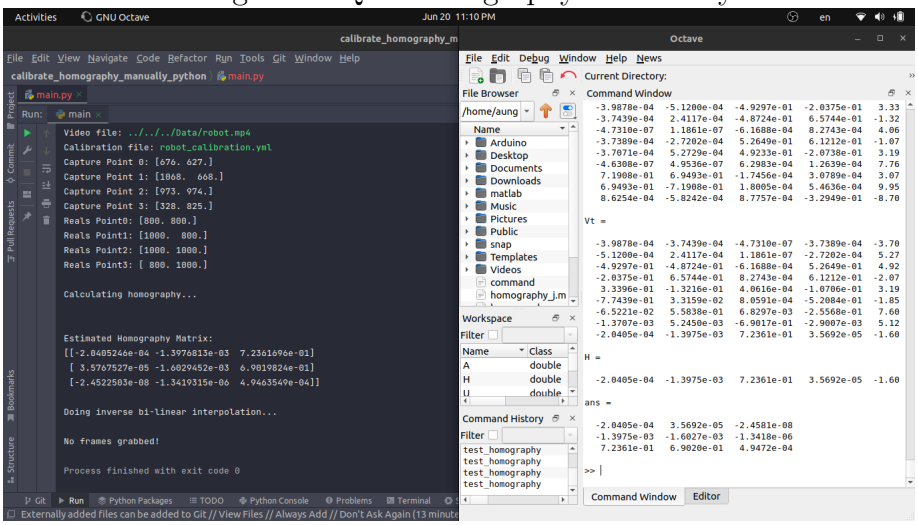
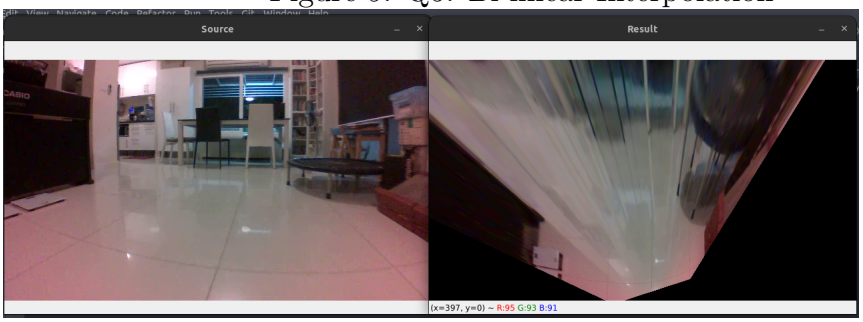



Figure 9: Q5: Bi-linear Interpolation

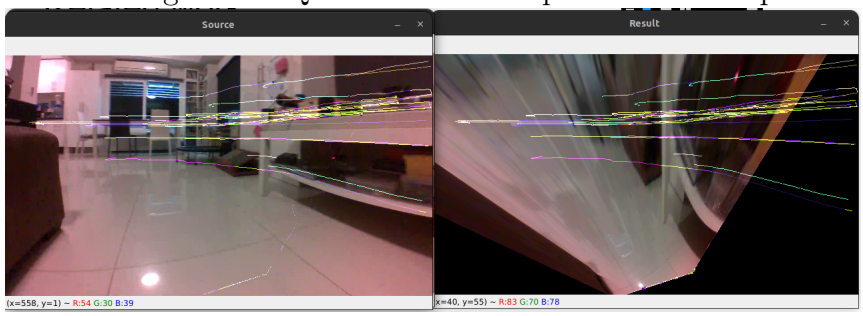



3. Reuse the homography from your last run: If your program finds a file homography.yml in the working directory, it should read the homography from that file and use it to display the transformed image. For this, you will have to learn how OpenCV stores data files in YML format using the FileStorage class. When the user selects four points in a frame, output the resulting homography to the data file and re-read that file when the program starts again. This way, the user only has to do the "calibration" once.

Python Code

```
1 import sys
2 import cv2 as cv
3 import numpy as np
4 from os.path import exists
```

Figure 10: Q5: Bi-linear Interpolation with Optical Flow




```
5
6   point = (-1, -1)
7   pts = []
8   var = 0
9   drag = 0
10  mat_final = np.array([])
11  mat_result = np.array([])
12
13  feature_params = dict(
14    maxCorners=100,
15    qualityLevel=0.3,
16    minDistance=7,
17    blockSize=7
18  )
19  lk_params = dict(
20    winSize=(15, 15),
21    maxLevel=2,
22    criteria=(cv.TERM_CRITERIA_EPS | cv.
     TERM_CRITERIA_COUNT, 10, 0.03)
23  )
24
25  color = np.random.randint(0, 255, (100, 3))
26  VIDEO_FILE = "../../../Data/robot.mp4"
27  ROTATE = False
28  CALIBRATION_FILE = "calibration.yml"
29  CALIBRATE = False
30
31
32  def draw_circle_and_line(x, y):
33    global point, pts, mat_result
```

```
34      mat_result = mat_final.copy()
35      point = (x, y)
36
37      if var >= 1:
38        cv.line(mat_result, pts[var - 1], point, (0, 255,
        0, 255), 2)
39      cv.circle(mat_result, point, 2, (0, 255, 0), -1, 8,
        0)
40      cv.imshow("Source", mat_result)
41
42
43    def open_video(file_name):
44      video_capture = cv.VideoCapture(file_name)
45      if not video_capture.isOpened():
46        print("ERROR! Unable to open input video file ",
       VIDEO_FILE)
47        sys.exit('Unable to open input video file')
48
49      width = video_capture.get(cv.CAP_PROP_FRAME_WIDTH)
50      height = video_capture.get(cv.CAP_PROP_FRAME_HEIGHT
       )
51      ratio = 640.0 / width
52      dim = (int(width * ratio), int(height * ratio))
53
54      return video_capture, width, height, ratio, dim
55
56
57    def mouse_handler(event, x, y, flags, param):
58      global point, pts, var, drag, mat_final, mat_result
59
60      if var >= 4:
61        return
62
63      if event == cv.EVENT_LBUTTONDOWN:
64        drag = 1
65        draw_circle_and_line(x, y)
66
67      if event == cv.EVENT_LBUTTONUP and drag:
68        drag = 0
69        pts.append(point)
70        var += 1
```

```
71        mat_final = mat_result.copy()
72
73        if var >= 4:
74          cv.line(mat_final, pts[0], pts[3], (0, 255, 0,
      255), 2)
75          cv.fillConvexPoly(mat_final, np.array(pts,
      dtype=np.int32), (0, 120, 0, 20))
76        cv.imshow("Source", mat_final)
77
78      if drag:
79        draw_circle_and_line(x, y)
80
81
82    def calc_homography(src, dst):
83
84      if len(src) != 4 or len(dst) != 4:
85        print("Four points are needed for a homography
      ...")
86        return
87
88      print("\nCalculating homography...\n")
89
90      mat_a = np.zeros((9, 9), dtype=np.float32)
91      x_primes = dst[:, 0]
92      y_primes = dst[:, 1]
93
94      for i in range(len(src)):
95        x = src[i][0]
96        y = src[i][1]
97        x_prime = x_primes[i]
98        y_prime = y_primes[i]
99
100       mat_a[i * 2, 0] = -x
101       mat_a[i * 2, 1] = -y
102       mat_a[i * 2, 2] = -1
103       mat_a[i * 2, 3] = 0
104       mat_a[i * 2, 4] = 0
105       mat_a[i * 2, 5] = 0
106       mat_a[i * 2, 6] = x * x_prime
107       mat_a[i * 2, 7] = y * x_prime
108       mat_a[i * 2, 8] = x_prime
```

```
109
110         mat_a[i * 2 + 1, 0] = 0
111         mat_a[i * 2 + 1, 1] = 0
112         mat_a[i * 2 + 1, 2] = 0
113         mat_a[i * 2 + 1, 3] = -x
114         mat_a[i * 2 + 1, 4] = -y
115         mat_a[i * 2 + 1, 5] = -1
116         mat_a[i * 2 + 1, 6] = x * y_prime
117         mat_a[i * 2 + 1, 7] = y * y_prime
118         mat_a[i * 2 + 1, 8] = y_prime
119       mat_a[8, 8] = 1
120
121       mat_w, mat_u, mat_vt = cv.SVDecomp(src=mat_a)
122       mat_h = mat_vt[8, :].reshape(3, 3)
123       return mat_h
124
125
126   def bi_linear_interpolation(img, mat_t, new_width,
      new_height):
127       print("\nDoing inverse bi-linear interpolation... \
      n")
128
129       # Get dimensions of original image
130       old_height, old_width, channels = img.shape
131
132       # Create new image array
133       new_img = np.zeros((new_height, new_width, channels
      ))
134
135       for i in range(new_height):
136           for j in range(new_width):
137               # Map the coordinates back to the original
      image
138               pnt = np.array((j, i, 1)).reshape(3, 1)
139               l_x, l_y, z = np.matmul(np.linalg.pinv(mat_t),
      pnt)
140               y = l_y / z
141               x = l_x / z
142
143               # Check the inverse points lines in original
      image
```

```
144         if 0 <= y < old_height-1 and 0 <= x < old_width
    -1:
145             # Calculate the coordinate values for 4
    surrounding pixels
146             x0 = int(np.floor(x))
147             x1 = x0 + 1
148             y0 = int(np.floor(y))
149             y1 = y0 + 1
150 
151             # Get the neighbouring pixel values
152             i_a = img[y0, x0]
153             i_b = img[y1, x0]
154             i_c = img[y0, x1]
155             i_d = img[y1, x1]
156 
157             # Estimate the pixel value using pixel values
     of neighbors
158             color1 = (x1 - x) * (y1 - y) * i_a
159             color2 = (x1 - x) * (y - y0) * i_b
160             color3 = (x - x0) * (y1 - y) * i_c
161             color4 = (x - x0) * (y - y0) * i_d
162 
163             weighted_avg_color = (color1 + color2 +
    color3 + color4) / 255.0
164             new_img[i, j] = weighted_avg_color
165     return new_img
166 
167 
168 def calibrate():
169     global mat_final, mat_result, var, CALIBRATE
170     CALIBRATE = True
171 
172     mat_pause_screen = np.array([])
173     mat_frame_capture = np.array([])
174     key = -1
175 
176     # ---------------------- [STEP 1: Make video capture
    from file] ----------------------
177     # Open video file
178     video_capture, width, height, ratio, dim =
    open_video(VIDEO_FILE)
```

```
179
180     # Capture loop
181     while key < 0:
182       # Get the next frame
183       _, mat_frame_capture = video_capture.read()
184       if mat_frame_capture is None:
185         break
186
187       # Rotate
188       if ROTATE:
189         _, mat_frame_display = cv.rotate(
    mat_frame_capture, cv.ROTATE_180)
190       else:
191         mat_frame_display = mat_frame_capture
192
193       mat_frame_display = cv.resize(mat_frame_display,
    dim)
194
195       cv.imshow("ROBOT.MP4", mat_frame_display)
196       key = cv.waitKey(30)
197
198       # ---------------------- [STEP 2: pause the screen
     and show an image] ----------------------
199       if key >= 0:
200         mat_pause_screen = mat_frame_capture
201         mat_final = mat_pause_screen.copy()
202
203     # ---------------------- [STEP 3: use mouse handler
    to select 4 points] ----------------------
204     if mat_frame_capture is not None:
205       var = 0
206       pts.clear()
207       cv.namedWindow("Source", cv.WINDOW_GUI_NORMAL)
208       cv.setMouseCallback("Source", mouse_handler)
209       cv.imshow("Source", mat_pause_screen)
210       cv.waitKey(0)
211       cv.destroyWindow("Source")
212
213       if len(pts) == 4:
214         src = np.array(pts).astype(np.float32)
215         for i, item in enumerate(src):
```

```
216             print(f"Capture Point {i}: {item}")
217
218         reals = np.array([
219             (800, 800),
220             (1000, 800),
221             (1000, 1000),
222             (800, 1000)
223         ], dtype=np.float32)
224         for i, item in enumerate(reals):
225             print(f"Reals Point{i}: {item}")
226
227         # ------------------------ [STEP 4: Calculate
    Homography] ------------------------
228         homography_matrix = calc_homography(src, reals)
229         print("\nEstimated Homography Matrix: ")
230         print(homography_matrix)
231
232         # ------------------------ [STEP 5: Save
    Homography to file] ------------------------
233         cv_file = cv.FileStorage(CALIBRATION_FILE, cv.
    FILE_STORAGE_WRITE)
234         cv_file.write("H", homography_matrix)
235         cv_file.release()
236
237         # ------------------------ [STEP 6: Warp Inverse
    Bi-linear Interpolation] ------------------------
238         h, w, ch = mat_pause_screen.shape
239         mat_result_frame = bi_linear_interpolation(
    mat_pause_screen, homography_matrix, w, h)
240
241         mat_source_frame = cv.resize(mat_pause_screen,
    dim)
242         cv.imshow("Source", mat_source_frame)
243
244         mat_result_frame = cv.resize(mat_result_frame,
    dim)
245         cv.imshow("Result", mat_result_frame)
246
247         cv.waitKey(0)
248     else:
249         print("Required 4 points to calculate
```

```
      Homography!")
250     else:
251       print("No pause before end of video finish.
      Exiting.")
252
253
254   def show_source_result_with_optical_flow():
255     cv_file = cv.FileStorage(CALIBRATION_FILE, cv.
      FILE_STORAGE_READ)
256     homography_matrix = cv_file.getNode("H").mat()
257     cv_file.release()
258
259     # Open video file
260     video_capture, width, height, ratio, dim =
      open_video(VIDEO_FILE)
261     _, frame = video_capture.read()
262     old_frame_source = frame
263     old_frame_result = cv.warpPerspective(frame,
      homography_matrix, (frame.shape[1], frame.shape[0]),
       cv.INTER_LINEAR)
264
265     old_gray_source = cv.cvtColor(old_frame_source, cv.
      COLOR_BGR2GRAY)
266     old_gray_result = cv.cvtColor(old_frame_result, cv.
      COLOR_BGR2GRAY)
267
268     p0_source = cv.goodFeaturesToTrack(old_gray_source,
      mask=None, **feature_params)
269     p0_result = cv.goodFeaturesToTrack(old_gray_result,
      mask=None, **feature_params)
270
271     mask_source = np.zeros_like(old_frame_source)
272     mask_result = np.zeros_like(old_gray_result)
273
274     while True:
275       ret, frame = video_capture.read()
276       if not ret:
277         print("No frames grabbed!")
278         break
279
280       frame_source = frame
```

```
281         frame_result = cv.warpPerspective(frame,
        homography_matrix, (frame.shape[1], frame.shape[0]),
         cv.INTER_LINEAR)
282
283         frame_gray_source = cv.cvtColor(frame_source, cv.
        COLOR_BGR2GRAY)
284         frame_gray_result = cv.cvtColor(frame_result, cv.
        COLOR_BGR2GRAY)
285
286         p1_source, st_source, err_source = cv.
        calcOpticalFlowPyrLK(old_gray_source,
        frame_gray_source, p0_source, None,
287                                             **lk_params)
288         p1_result, st_result, err_result = cv.
        calcOpticalFlowPyrLK(old_gray_result,
        frame_gray_result, p0_result, None,
289                                             **lk_params)
290
291         if p1_source is not None and p1_result is not
        None:
292             good_new_source = p1_source[st_source == 1]
293             good_old_source = p0_source[st_source == 1]
294             good_new_result = p1_result[st_result == 1]
295             good_old_result = p1_result[st_result == 1]
296
297         for i, (new_source, old_source, new_result,
        old_result) in enumerate(zip(good_new_source,
        good_old_source,
298
        good_new_result, good_old_result)):
299             a_source, b_source = new_source.ravel()
300             c_source, d_source = old_source.ravel()
301             a_result, b_result = new_result.ravel()
302             c_result, d_result = old_result.ravel()
303
304             mask_source = cv.line(mask_source, (int(
        a_source), int(b_source)), (int(c_source), int(
        d_source)),
305                                   color[i].tolist(), 2)
306             mask_result = cv.line(mask_source, (int(
        a_result), int(b_result)), (int(c_result), int(
```

```
    d_result)),
307                     color[i].tolist(), 2)
308         frame_source = cv.circle(frame_source, (int(
    a_source), int(b_source)), 5, color[i].tolist(), -1)
309         frame_result = cv.circle(frame_result, (int(
    a_result), int(b_result)), 5, color[i].tolist(), -1)
310
311       img_source = cv.add(frame_source, mask_source)
312       img_result = cv.add(frame_result, mask_result)
313
314       img_source_resize = cv.resize(img_source, dim)
315       img_result_resize = cv.resize(img_result, dim)
316
317       cv.imshow('Source', img_source_resize)
318       cv.imshow('Result', img_result_resize)
319
320       key = cv.waitKey(30) & 0xff
321       if key == 27:
322         break
323
324       old_gray_source = frame_gray_source.copy()
325       p0_source = good_new_source.reshape(-1, 1, 2)
326       old_gray_result = frame_gray_result.copy()
327       p0_result = good_new_result.reshape(-1, 1, 2)
328
329     cv.destroyAllWindows()
330
331
332   if __name__ == "__main__":
333     VIDEO_FILE = input("Video file: ")
334     CALIBRATION_FILE = input("Calibration file: ")
335
336     if exists(CALIBRATION_FILE):
337       show_source_result_with_optical_flow()
338     else:
339       calibrate()
340       if exists(CALIBRATION_FILE):
341         show_source_result_with_optical_flow()
342
```

Octave Code

```
1   X = [ 676, 1068, 973, 328 ; 627, 668, 974, 825 ]
2
3   Xprime = [800, 1000, 1000, 800 ; 800, 800, 1000,
      1000]
4
5   A = []
6
7   for iPoint = 1:size(X,2)
8     x = X(:, iPoint)
9     xprime = Xprime(:, iPoint)
10
11    coeffs = ...
12    [-x(1), -x(2), -1, 0, 0, 0, xprime(1)*x(1), xprime
      (1)*x(2), xprime(1)]
13
14    A = [A ; coeffs]
15
16    coeffs = ...
17    [0, 0, 0, -x(1), -x(2), -1, xprime(2)*x(1), xprime
      (2)*x(2), xprime(2)]
18
19    A = [A ; coeffs]
20  end
21
22  h = null(A)
23
24  [U, W, V] = svd(A)
25
26  Vt = V'
27
28  H = Vt(9,:)
29
30  reshape(H, 3, 3)
```