IMPLEMENTATION OF ROBUST ESTIMATION OF HEIGHTS OF MOVING PEOPLE USING A SINGLE CAMERA

A PREPRINT

Saad Manzur
83166813
smanzur@uci.edu

June 14, 2019

1 Overview

Markerless motion capture, to my best knowledge, is an active research interest in current Computer Vision community. Even though multi-view motion capture has a richer dataset, estimating human pose from monocular camera has been receiving attention recently. But estimating motion from a single camera view is a fundamentally ill-posed problem, since multiple pose can generate similar images. The paper [1] has used heightmaps to map the skeleton joint structure onto a human figure. The height estimation algorithm they used is [2].

Since single camera does not provide enough information to project a 2D point, back to it's original 3D coordinates, most of the monocular algorithm rely on geometric structure or information (i.e. vanishing points) present in the image. There are numerous work, which show methods to obtain reference frame from geometric cues of images. Once, the reference frame is known, by using relative ratios of lengths present within the image, we can obtain multiple features (height, gait etc.). However, the object has to be in contact with the reference plane to work. In [2], the authors use a rectangular marker to generate a reference frame in the scene. By back projecting the 2D coordinates and intersecting them with the ground plane and global vertical line, they estimated height.

This project is an implementation of [2] which uses a single camera view to estimate height of moving people. The overview of the project can roughly be seen in Figure 1.

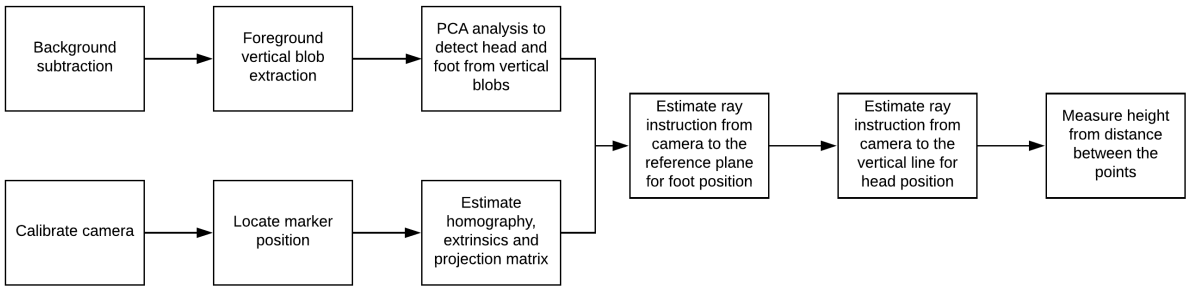


Figure 1: Overview of the process

Experimentation process can be broken down into four steps -

- Dataset collection
- Estimating the projection matrix
- Estimating head and foot position on image

- Measuring height

I used Logitech C920 webcam for data collection and OpenCV for implementation.

2 Technical Approach

2.1 Dataset collection

To collect video dataset, I first tried to get the original video data, used in the paper [2]. Since there was no references to the dataset used, I ended up making the dataset myself. Multiple videos were captured from multiple points around the campus. However, the video in front of the computer sciences building turned out to be best for the experiment. Since, the place has a relatively static background of the engineering building, moderate volume of traffic and tables to mount the camera. I first took a couple of shots to see where the marker can be located (even with a low resolution image). I opted for lower resolution to reduce the processing time.

(a) Setup for collecting the video dataset (b) Setup for capturing data for calibration

Figure 2: Setup for dataset

For capturing the videos, I used a table mount to stabilize the camera and a Logitech C920 webcam. The video was captured at 640×480 resolution. The dataset can be found at [3]. When the marker was in the shadow, it could not detect sometimes. Hence, it was not used in the experiment.

2.2 Estimating the projection matrix

Camera Calibration

For camera calibration, I used a chess board pattern and took multiple images from different angles and used opencv provided camera calibration method to get the intrinsic matrix.

Finding Projection Matrix

For this step, I generated a marker image and placed it on the reference plane. Since the dimension of sides of the markers are known (19.6 inch each), we have the 3d location of the four corners of the marker, assuming the origin is at center. Introducing $z = 0$, reduces the problem to estimating the homography between the reference plane and the image plane [4]. So if we have 2d coordinates of the corners on the image plane (x_n, y_n) and actual 3d point (X_n, Y_n), we can estimate the homography from the following equation,

$$K^{-1} \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = [r_1 \quad r_2 \quad t] \begin{bmatrix} X_n \\ Y_n \\ 1 \end{bmatrix} \tag{1}$$

r_3 is cancelled out since $z = 0$. We can get r_3 by $r_1 \times r_2$. The recovered rotation matrix is then fine tuned by doing an SVD, $R = U\Sigma V^T$ and setting $R = UV^T$. Now we can compute the projection matrix P by $K[R|t]$. To see if the projection matrix is working properly, I projected the origin and reference axis on the image (shown in Figure 3

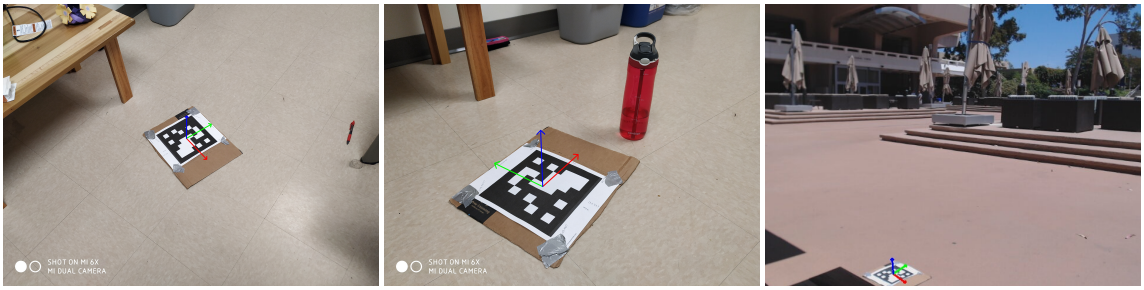
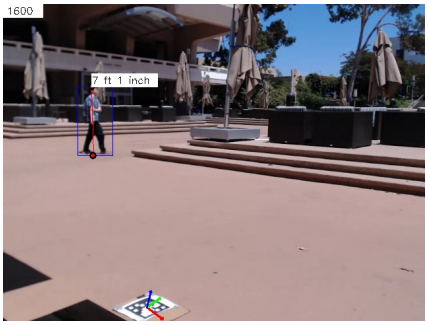


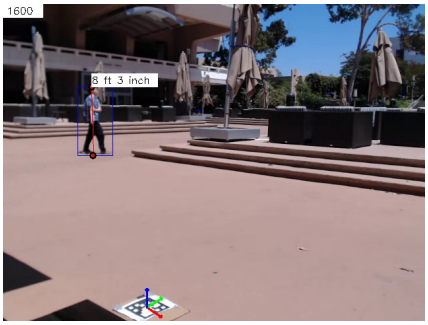
Figure 3: Verification of projection matrix and visualizing reference frame (first two were captured with smart phone camera)

(a) Without correcting the marker corners (frame 1000)

(b) After correcting the marker corners (frame 1000)



(c) Without correcting the marker corners (frame 1600)



(d) After correcting the marker corners (frame 1600)

Figure 4: Effect of stabilizing the frame

First, I computed the projection matrix for every frame, which did not yield a good result, due to wrong estimation of the corner locations, the projection matrix was off and can clearly be seen with the misplacement of the reference frame within the image. My first naive attempt was to take the mean projection matrix, which did stabilize the sudden movement of the reference frame, but in the end, the reference frame was misplaced by the outliers. This is obvious from the fact that the homography matrix has a relatively large condition number (679.6363), it leaves room for error if there is already perturbation in the input (the corners of the marker). Therefore, I iterated through the frames and measured the average location for each of the four corners to get a more stable reference frame (Figure 4).

2.3 Estimating head and foot position on image

In the original paper [2], the authors used Kernel Density Estimator to do background subtraction. However, for my dataset, opencv provided MOG2 and KNN based background subtraction worked fine. After performing the background subtraction and some morphological operations, the foreground blobs were detected using opencv's contour detection. Similarly, bounding rectangle was also extracted for the contours. Using the contours, the foreground points were isolated to perform principal component analysis to get eigen vectors and eigen values. The vertical eigen vector is used to find the intersection with the vertical bounds of the rectangle. These points are used as head and foot points on the image plane.

(a) Random person (b) Random person (c) Me

Figure 5: Head and foot position by intersecting the vertical eigen vector with bounding box

2.4 Measuring height

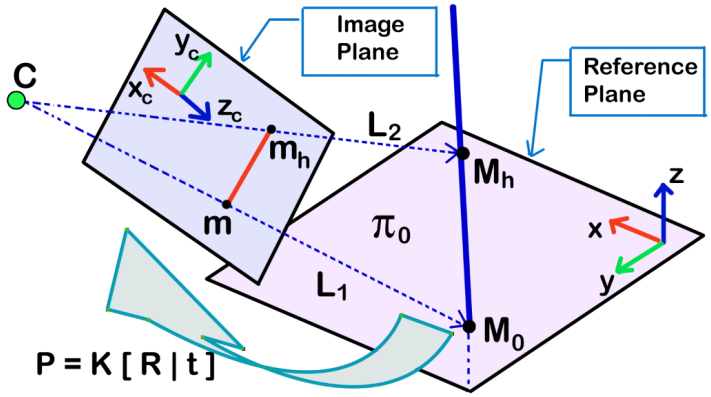


Figure 6: Estimating 3D coordinate by back projection [2]

To measure the height, we have to back project the foot point to intersect with the reference plane and the head point to intersect with the global vertical line as shown in Figure 6. If the projection matrix is given by $\widetilde{P} = [P\widetilde{p}]$, where P is a 3×3 submatrix. The camera center $C = -P^{-1}p$. Now, the 3d position of the foot, M_0, can be computed by -

$$\begin{bmatrix}0\\0\\1\end{bmatrix}^T M_0 = \begin{bmatrix}0\\0\\1\end{bmatrix}^T (C + \lambda_0 P^{-1}\widetilde{m}) \tag{2}$$

Since LHS is 0, λ_0 can be deterministically computed by $\lambda_0 = \dfrac{\begin{bmatrix}0\\0\\1\end{bmatrix}^T C}{\begin{bmatrix}0\\0\\1\end{bmatrix}^T P^{-1}\widetilde{m}}$

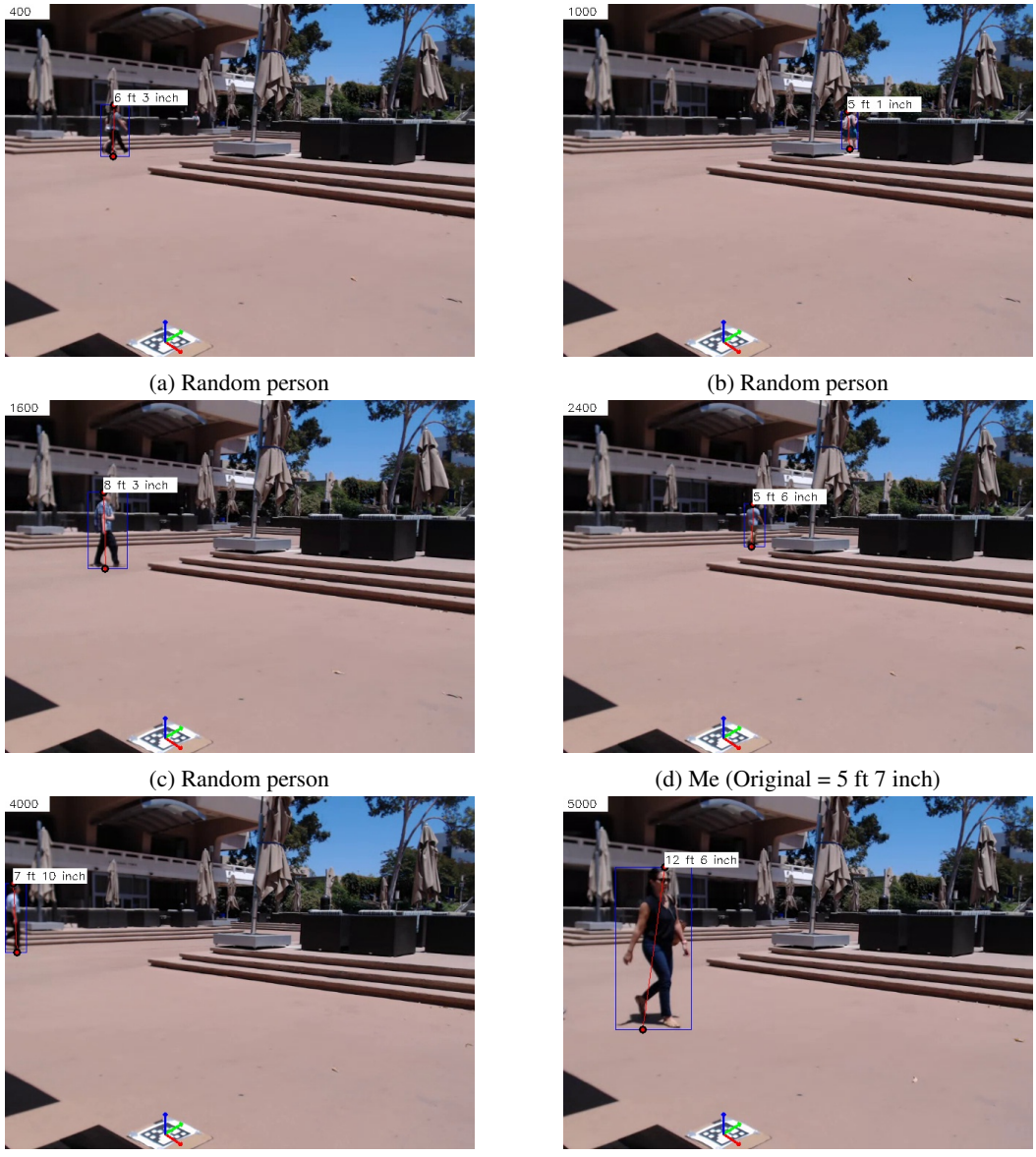
M_h is given by the intersection of the vertical line L_h with line L_2. If $L_h = \widetilde{M_0} + \mu\,[0 \quad 0 \quad 1 \quad 0]$ and $L_2 = C + \lambda P^{-1}\widetilde{m_h}$, then $L_h = L_2 = M_h$ yields,

$$\begin{bmatrix}m_1 - c_1\\ m_2 - c_2\\ m_3 - c_3\end{bmatrix} = \begin{bmatrix}dh_1 & -dv_1\\ dh_2 & -dv_2\\ dh_3 & -dv_3\end{bmatrix} \begin{bmatrix}\lambda\\ \mu\end{bmatrix} \tag{3}$$

where dh_i is the i'th element of the $P^{-1}\widetilde{m_h}$ and dv_i is the i'th element of $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$. Solving the system of linear equation, we can get the 3D coordinate of the foot [2]. The distance between these points is the height.

3 Result

The output of the experiment is a video. A few frames from the video can be seen in Figure 7. The full video is uploaded on https://www.youtube.com/watch?v=9v9Y_5T8WBo



(a) Random person

(b) Random person

(c) Random person

(d) Me (Original = 5 ft 7 inch)

(e) Random person

(f) Random person

Figure 7: Estimated height from the algorithm

Since, I did not manage to get the actual height input from the persons present in the video. However, I walked past the video at certain point, in front of the camera, which estimated my height within (5 ft 4 inch to 6 ft 3 inch) range whereas my actual height is 5 ft 7 inch. I have given an image of a frame that shows me in Figure 7.

4 Discussion

Even without ground truth, it can clearly be seen that the estimated heights are erroneous, especially one where the person is closer to the camera. The authors in [2], explained this. If we back project the corners of the marker, we will get a set of points different from our current estimation. Which form another reference plane, computing a homography that transforms that plane to our reference plane, will correct the error. On another note, the height showed a consistent trend. One observation is whenever the person is closer to our camera, the height jumps up, which in some sense gives an idea that the reference plane is not detected properly (somewhat tilted).

References

[1] Yu Du, Yongkang Wong, Yonghao Liu, Feilin Han, Yilin Gui, Zhen Wang, Mohan Kankanhalli, and Weidong Geng. Marker-less 3d human motion capture with monocular image sequence and height-maps. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 20–36, Cham, 2016. Springer International Publishing.

[2] Sang-Wook Park, Tae-Eun Kim, and Jong-Soo Choi. Robust estimation of heights of moving people using a single camera. In Kuinam J. Kim and Seong Jin Ahn, editors, *Proceedings of the International Conference on IT Convergence and Security 2011*, pages 389–405, Dordrecht, 2012. Springer Netherlands.

[3] Dataset for height estimation. https://www.youtube.com/watch?v=UOipQVaY2mU&list=PLVaC0E1yxiQ7ttgGKO5o4Tw7vf99pi2ot&index=2.

[4] Z Zhang. A flexible new technique for camera calibration, ieee transactions on pattern analysis and machine intelligence, vol. 22, no. 11, pages 1330-1334. 2000.

A Height Estimation Code

```
"""
Tested with Python 3.7 and OpenCV 4.1.0
"""

import cv2 as opencv
import numpy as np

MARKER_SIZE = 19.6

# Logitech C920, computed from calibration code separately
K = np.array([[662.25801378, 0., 294.92509514],
              [0., 663.05010424, 209.48385376],
              [0., 0., 1.]])


def locate_marker(marker_id, input_img, show=False):
    # input_img = opencv.resize(input_img, (int(input_img.shape[1] / 5),
    ↪  int(input_img.shape[0] / 5)))

    marker_dictionary = opencv.aruco.getPredefinedDictionary(opencv.aruco.DICT_7X7_1000)

    corners, ids, rejected_img_points = opencv.aruco.detectMarkers(input_img,
    ↪  marker_dictionary)

    points3d = np.array([[-MARKER_SIZE / 2, MARKER_SIZE / 2],
                         [MARKER_SIZE / 2, MARKER_SIZE / 2],
                         [MARKER_SIZE / 2, -MARKER_SIZE / 2],
                         [-MARKER_SIZE / 2, -MARKER_SIZE / 2]])

    if show:
        output = opencv.aruco.drawDetectedMarkers(input_img, corners, ids)

```

```
        for i in range(ids.shape[0]):
            if ids[i] == marker_id:
                counter = 0
                for corner in corners[i][0]:
                    coord = '(' + str(points3d[counter, 0]) + ', ' +
                    ↪  str(points3d[counter, 1]) + ', 0)'
                    opencv.circle(output, (corner[0], corner[1]), 4, (255, 0, 0), -1)
                    opencv.putText(output, coord,
                                   (int(corner[0] - 20),
                                    int(corner[1] - (-1) ** (counter // 2) * 20)),
                                   opencv.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255))
                    counter = counter + 1

        opencv.imshow('marker', output)
        opencv.waitKey(10000)

    if len(corners) == 0:
        return None, None

    return corners[0][0], points3d


def get_extrinsic(img_points, points_3d):
    img_points_h = np.hstack((img_points, np.ones((4, 1))))
    normalized_img_points = np.linalg.inv(K) @ img_points_h.T

    H, mask = opencv.findHomography(points_3d, normalized_img_points.T[:, :2])
    H /= np.linalg.norm(H[:, 0])

    r1 = H[:, 0]
    r2 = H[:, 1]
    r3 = np.cross(r1, r2)

    t = (2 * H[:, 2]) / (np.linalg.norm(r1) + np.linalg.norm(r2))

    R = np.hstack((r1[:, np.newaxis], r2[:, np.newaxis], r3[:, np.newaxis]))

    U, E, Vt = np.linalg.svd(R)

    R = U @ Vt

    return R, t[:, np.newaxis]


def get_projection_matrix(rotation, translation):
    extrinsic = np.hstack((rotation, translation))
    return K @ extrinsic


def find_marker_in_frame(frame, prev_P):
    img_points, points_3d = locate_marker(20, frame)

    if img_points is None:
        return prev_P

    R, t = get_extrinsic(img_points, points_3d)

    P = get_projection_matrix(R, t)

```

```
    return P


def show_axis(input_img, P, thickness=20):
    axis_3d = np.vstack((np.identity(3) * 15, np.ones((1, 3))))
    axis_2d = P @ axis_3d

    origin_3d = np.vstack((np.zeros((3, 1)), np.ones((1, 1))))
    origin_2d = P @ origin_3d
    origin_2d /= origin_2d[2, 0]

    # opencv.circle(input_img, (int(origin_2d[0, 0]), int(origin_2d[1, 0])), 20, (128,
    ↪  135, 12), -1)

    for i in range(3):
        color = np.zeros((3,))
        color[2 - i] = 255
        axis_2d[:, i] /= axis_2d[2, i]
        opencv.arrowedLine(input_img,
                           (int(origin_2d[0, 0]), int(origin_2d[1, 0])),
                           (int(axis_2d[0, i]), int(axis_2d[1, i])),
                           (color[0], color[1], color[2]),
                           thickness
                           )


def subtract_background_and_get_frames(filename, P = None):
    video = opencv.VideoCapture(filename)

    out_video = opencv.VideoWriter("output.avi",
                                   opencv.VideoWriter_fourcc('X', 'V', 'I', 'D'), 30.0,
                                   (640, 480))

    while not video.isOpened():
        video = opencv.VideoCapture(filename)

        if opencv.waitKey(1000) == 'q':
            break

        print("Attempting again.")

    print('Frame count = ', video.get(opencv.CAP_PROP_FRAME_COUNT))
    print('Frame width = ', video.get(opencv.CAP_PROP_FRAME_WIDTH))
    print('Frame height = ', video.get(opencv.CAP_PROP_FRAME_HEIGHT))

    subtractor = opencv.createBackgroundSubtractorMOG2(detectShadows=False)

    kernel = np.ones((9, 9), np.uint8)

    frames = []
    rects = []

    FRAME_SKIP = 200

    summation_P = np.zeros((3, 4))

    while True:
        ret, frame = video.read()

```

```
147         if frame is not None:
148             #P = find_marker_in_frame(frame, P)
149 
150             #if P is not None:
151             #    summation_P = summation_P + P
152 
153             foreground_mask = subtractor.apply(frame)
154 
155             foreground_mask = opencv.morphologyEx(foreground_mask, opencv.MORPH_OPEN,
                ↪ np.ones((3, 3), np.uint8))
156             foreground_mask = opencv.morphologyEx(foreground_mask, opencv.MORPH_CLOSE,
                ↪ kernel)
157 
158             contours, ret = opencv.findContours(foreground_mask,
                ↪ mode=opencv.CHAIN_APPROX_SIMPLE,
159                                                 method=opencv.RETR_FLOODFILL)
160 
161             if len(frames) % FRAME_SKIP == 0:
162                 opencv.imwrite('resources/frames/fg' + str(len(frames)) + '.jpg',
                    ↪ foreground_mask)
163 
164             frame_contours = []
165             for c in contours:
166                 area = opencv.contourArea(c)
167                 if 600 < area:
168                     x, y, w, h = opencv.boundingRect(c)
169 
170                     if h > 1.5 * w:
171                         frame_contours.append(np.array([x, y, w, h]))
172                         opencv.rectangle(frame, (x, y), (x + w, y + h), color=(255, 0,
                            ↪ 0))
173                         head, foot = draw_head_and_foot(frame, np.array([x, y, w, h]), c)
174 
175                         if len(frames) % FRAME_SKIP == 0:
176                             opencv.imwrite('resources/frames/hf' + str(len(frames)) +
                                ↪ '.jpg', frame)
177 
178                         foot3d = estimate_foot(foot, P, frame)
179                         head3d = estimate_head(head, P, frame, foot3d)
180                         height = np.linalg.norm(head3d - foot3d)
181                         ft_inch = str(int(height) // 12) + ' ft ' + str(int(height) % 12)
                            ↪ + ' inch'
182                         opencv.rectangle(frame,
183                                          (int(head[0]), int(head[1]) - 20),
184                                          (int(head[0]) + 100, int(head[1])),
185                                          (255, 255, 255), -1)
186                         opencv.putText(frame, ft_inch,
187                                        (int(head[0]), int(head[1]) - 5),
188                                        opencv.FONT_HERSHEY_SIMPLEX,
189                                        0.5, (0, 0, 0), 1)
190 
191             if P is not None:
192                 show_axis(frame, P, thickness=2)
193 
194             opencv.rectangle(frame, (0, 0), (60, 20), (255, 255, 255), -1)
195             opencv.putText(frame, str(len(frames)), (5, 15), opencv.FONT_HERSHEY_SIMPLEX,
                ↪ 0.5, color=(0, 0, 0))
196 
197             if len(frames) % FRAME_SKIP == 0:
```

```
                opencv.imwrite('resources/frames/final' + str(len(frames)) + '.jpg',
                ↪  frame)

            out_video.write(frame)

            opencv.imshow('video', frame)
            opencv.waitKey(20)

            frames.append(frame)
            rects.append(frame_contours)

        else:
            break

    print("Background subtraction complete.")

    video.release()
    out_video.release()

    opencv.destroyAllWindows()

    return frames, rects, summation_P / len(frames)


def get_orientation(pts):
    sz = len(pts)
    data_pts = np.empty((sz, 2), dtype=np.float64)
    for i in range(data_pts.shape[0]):
        data_pts[i, 0] = pts[i, 0, 0]
        data_pts[i, 1] = pts[i, 0, 1]

    mean = np.empty((0))
    mean, eigenvectors, eigenvalues = opencv.PCACompute2(data_pts, mean)

    return mean, eigenvectors, eigenvalues


def draw_line_through(img, p_, q_, color, rect):
    p = list(p_)
    q = list(q_)

    m = (q[0] - p[0]) / (q[1] - p[1])

    pt1 = (int(m * (rect[1] - p[1]) + p[0]), rect[1])
    pt2 = (int(m * (rect[1] + rect[3] - p[1]) + p[0]), rect[1] + rect[3])

    opencv.circle(img, pt1, 3, color, -1)
    opencv.circle(img, pt2, 3, color, -1)
    opencv.line(img, pt1, pt2, color, 1)

    return pt1, pt2


def draw_head_and_foot(frame, rect, contour):
    mean, eigen_vectors, eigen_values = get_orientation(contour)

    center = (int(mean[0, 0]), int(mean[0, 1]))

    p1 = (center[0] + 0.02 * eigen_vectors[0, 0] * eigen_values[0, 0],
```

```
            center[1] + 0.02 * eigen_vectors[0, 1] * eigen_values[0, 0])

    head, foot = draw_line_through(frame, center, p1, (0, 0, 255), rect)

    return head, foot


def estimate_foot(foot, projection_matrix, frame):
    P = projection_matrix[:, :3]
    p = projection_matrix[:, 3]

    inv_P = np.linalg.inv(P)

    C = - inv_P @ p

    z = np.zeros((1, 3))
    z[0, 2] = 1

    foot_h = np.array([np.array([foot[0], foot[1], 1])]).T

    l = - (z @ C[:, np.newaxis]) / (z @ inv_P @ foot_h)

    foot_org = C[:, np.newaxis] + l * (inv_P @ foot_h)

    projected_foot = projection_matrix @ np.vstack((foot_org, np.ones((1, 1))))
    projected_foot /= projected_foot[2, 0]

    opencv.circle(frame, (int(projected_foot[0, 0]), int(projected_foot[1, 0])),
                  4, (0, 0, 0), 2)

    return foot_org


def estimate_head(head, projection_matrix, frame, foot3d):
    P = projection_matrix[:, :3]
    p = projection_matrix[:, 3]

    inv_P = np.linalg.inv(P)

    C = - inv_P @ p

    Dv = np.zeros((1, 3))
    Dv[0, 2] = 1

    head_h = np.array([np.array([head[0], head[1], 1])]).T

    Dh = inv_P @ head_h

    B = foot3d - C[:, np.newaxis]
    A = np.hstack((Dh, -Dv.T))

    result = np.linalg.lstsq(A, B, rcond=None)[0]

    head_org = C[:, np.newaxis] + result[0] * Dh

    projected_head = projection_matrix @ np.vstack((head_org, np.ones((1, 1))))
    projected_head /= projected_head[2, 0]

```

```
    opencv.circle(frame, (int(projected_head[0, 0]), int(projected_head[1, 0])),
                  4, (0, 0, 0), 2)

    return head_org


def get_marker_corners_from_video(filename):
    video = opencv.VideoCapture(filename)

    while not video.isOpened():
        video = opencv.VideoCapture(filename)

        if opencv.waitKey(1000) == 'q':
            break

        print("Attempting again.")

    marker_dictionary = opencv.aruco.getPredefinedDictionary(opencv.aruco.DICT_7X7_1000)

    points3d = np.array([[-MARKER_SIZE / 2, MARKER_SIZE / 2],
                         [MARKER_SIZE / 2, MARKER_SIZE / 2],
                         [MARKER_SIZE / 2, -MARKER_SIZE / 2],
                         [-MARKER_SIZE / 2, -MARKER_SIZE / 2]])

    summation_corners = np.zeros((4, 2))
    counter = 0

    print("Estimating marker corners")
    while True:
        ret, frame = video.read()

        if frame is not None:
            corners, ids, rejected_img_points = opencv.aruco.detectMarkers(frame,
            ↪  marker_dictionary)

            if len(corners) is not 0:
                summation_corners += corners[0][0]
                counter += 1
        else:
            break

    video.release()

    opencv.destroyAllWindows()

    corners = summation_corners / counter

    print("Marker corners estimated")

    return corners, points3d


if __name__ == '__main__':
    filename = 'resources/engg_m3.webm'

    img_points, points_3d = get_marker_corners_from_video(filename)

    R, t = get_extrinsic(img_points, points_3d)

```

```
372     P = get_projection_matrix(R, t)
373
374     frames, rects, projection_matrix = subtract_background_and_get_frames(filename, P)
```

B Calibration Code

Followed from sample provided in OpenCV documentation.

```
1  import cv2 as opencv
2  import numpy as np
3  import glob
4
5  images = glob.glob('images/Logitech/*.jpg')
6
7  CHESS_CELL_DIM = 2.5
8
9  patternX = 8
10 patternY = 6
11
12 criteria = (opencv.TERM_CRITERIA_EPS + opencv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
13
14 objp = np.zeros((patternX * patternY, 3), np.float32)
15 objp[:, :2] = np.mgrid[0:8*CHESS_CELL_DIM:CHESS_CELL_DIM,
   ↪  0:6*CHESS_CELL_DIM:CHESS_CELL_DIM].T.reshape(-1, 2)
16
17 points2d = []
18 points3d = []
19
20 for filename in images:
21     image = opencv.imread(filename)
22     gray = opencv.cvtColor(image, opencv.COLOR_BGR2GRAY)
23
24     ret, corners = opencv.findChessboardCorners(gray, (patternX, patternY), None)
25
26     if ret:
27         points3d.append(objp)
28
29         finesedCorners = opencv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
30         points2d.append(finesedCorners)
31
32         opencv.drawChessboardCorners(image, (patternX, patternY), finesedCorners, ret)
33
34         opencv.imshow(filename, image)
35         opencv.waitKey(500)
36
37
38 ret, mtx, dist, rvecs, tvecs = opencv.calibrateCamera(points3d, points2d,
   ↪  gray.shape[::-1], None, None)
39
40 print(mtx)
41
42 mean_error = 0
43 for i in range(len(points3d)):
44     imgpoints2, _ = opencv.projectPoints(points3d[i], rvecs[i], tvecs[i], mtx, dist)
45     error = opencv.norm(points2d[i], imgpoints2, opencv.NORM_L2)/len(imgpoints2)
46     mean_error += error
47 print("total error: {}".format(mean_error/len(objp)))
```