Vision-based toe-walking detection in unconscious condition

Introduction

Toe-walking is normally diagnosed in children who walk on their toes instead of heal-toe gait [1]. The risk of limited joints angle [2], falling [3] and leg pain [4] are predicted for those children without medical rehabilitation. Before rehabilitation, toe-walking detection is the most important part. Researchers have conducted toe-walking detection researches in monitored condition on wearable sensor-based detection [5], electromyographic [6], Inertial Measurement Unit (IMU) [7] and pressure sensors [8]. However, those monitored detection designs fail to ensure accuracy and robustness, since monitored design make the patient walk abnormal from their usual gait especially children, which heavily impacts the detection accuracy. It is normal that when being monitored, clinical outcome is likely to be different [20].

An unconscious design proposes a system that captures toe-walking problem through computer vision based on MediaPipe [9]. Targeted on the 33 pose landmarks (**Figure1**), MediaPipe labels the 3D coordinates of the key points (**Figure2**). The angles between the foot keypoints can determine whether the patient is toe-walking through simple calculation. The video will be recorded on CCTV on the corridor of GP's office to monitor the walking gait of the patient in terms of rehabilitation process and toe-walking recognition.

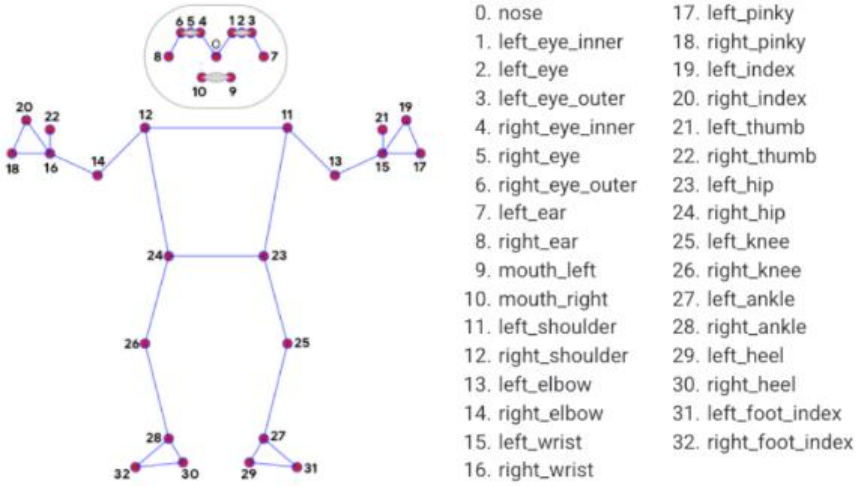

Figure1: General 33 pose landmarks [9]

Figure2: Example of detection [9]

According to [10], since the design is independent computer programming with a clear medical purpose, assisting people to diagnose toe-walking, it is defined as In Vitro Diagnostics (IVD) medical device. According to the International Organization for Standardization (ISO) 20916:2019 [11] for IVD medical device, as the design is contactless, there will be no safety consideration. Following the ISO 20916:2019 requirements [11], reliability and robustness of the design are in priority. Thus, this report aims to (1) demonstrate the detailed design of vision-based toe-walking detection and (2) verify the design following ISO 20916:2019 [11] in terms of reliability and robustness.

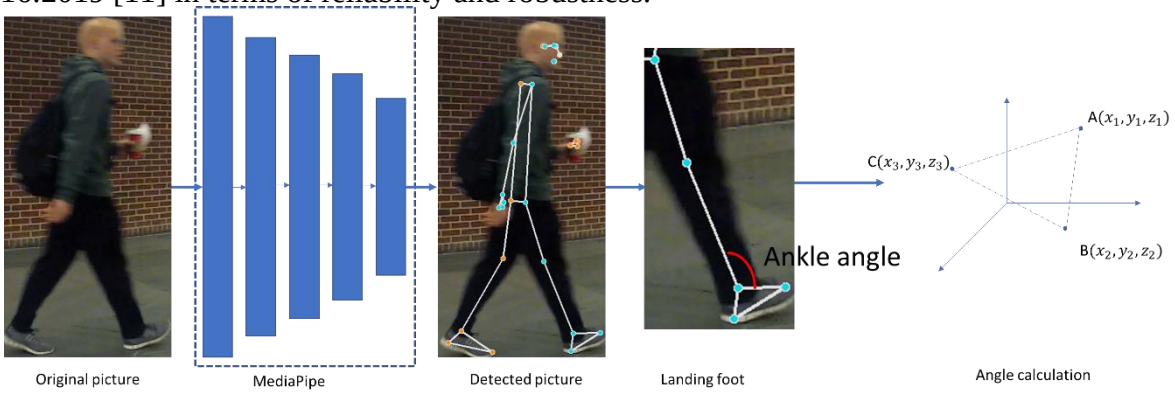

Figure3: Flowchart of vision-based toe-walking detection

Detailed Design

As shown in **Figure3**, when the camera captures a video clip, MediaPipe model processes the clip and outputs the coordinates of keypoints. To ensure robustness, the doctor or GP has to find when the person lands on the ground and which foot is his landing foot. With simple angle calculation, it is simple to get the ankle angle (shown as **Figure3** in the second picture from the right) for GP to evaluate whether the patient is toe-walking or their rehabilitation process through the following calculation:

$$\overrightarrow{AB} = (x_1 - x_2, y_1 - y_2, z_1 - z_2)$$

$$\overrightarrow{BC} = (x_2 - x_3, y_2 - y_3, z_2 - z_3)$$

$$\theta = \cos^{-1} \frac{\overrightarrow{AB} \cdot \overrightarrow{BC}}{\|\overrightarrow{AB}\| \|\overrightarrow{BC}\|}$$

Where $\overrightarrow{AB}$ and $\overrightarrow{BC}$ are vectors in the first picture of **Figure3** from the right and θ is the angle between the two vectors.

Following ISO 20916:2019 [11], as the major concerns for this IVD medical device are accuracy and robustness, this design needs corporation between the doctor or GP and artificial intelligence. Since it is not easy and accurate for the software to recognize the landing foot, the doctor or GP has to get involved in such job which is easier for human and makes it more robust than leaving it alone to artificial intelligence. Secondly, although putting the detected keypoints on the video clip needs additional computation consumption, human is able to get involved to check out whether the detection is reliable from subjective results (the third picture from the left on **Figure3**). Thirdly, among all the vision-based human posture estimation technology such as OpenPose [12] and ViTPose-G [13], MediaPipe has the advantage of training dataset [9], which is composed of single person instead of a crowd of people (the right picture on **Figure4**). Such single-person situation better fits in the toe-walking detection problem, forming samples in similar situations. Thus, this model is likely to be more robust than others. Although there is no model that directly focuses on toe-walking detection or single-person walking detection, MediaPipe are able to handle such problems since its Percentage of Correct Keypoints (PCK) [14] is 84.5 when threshold is 0.2 [9], which shows a reliable detection accuracy for single person.

Figure4: different kinds of dataset

Experiments

Following ISO 20916:2019 [11], this session focus on accuracy and robustness experiments on Zero-Occlusion-Object-Tracking-Data-Set [14]. Zero-Occlusion-Object-Tracking-Data-Set is composed of 1,400 images on single-person walking in the corridor captured by a CCTV on the corner. Samples are passing the corridor unconsciously in the city of Michigan. Since the dataset mainly focuses on gender classification instead of toe-walking, this report collects all the video clips that pedestrians are landing their foot to narrow the samples of the experiments and annotates keypoints of the samples through labelme toolbox [15].

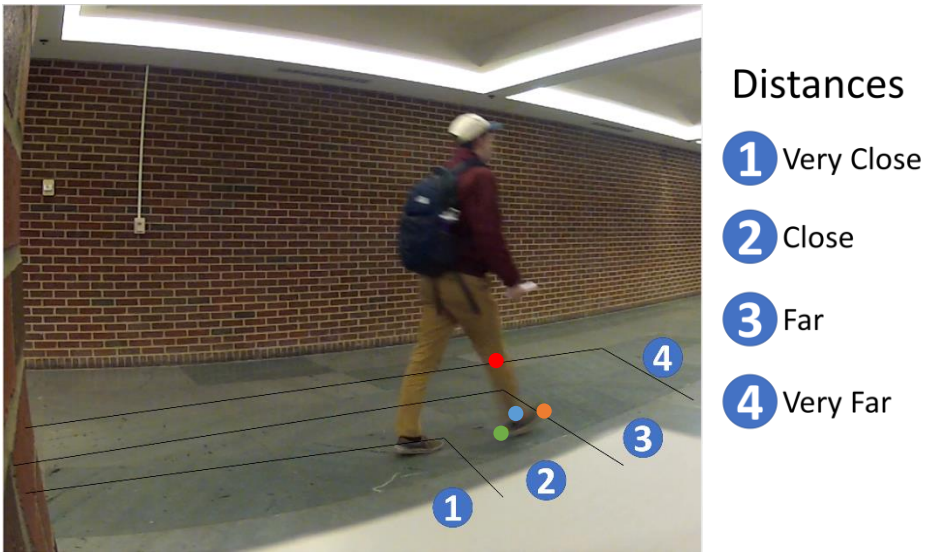

Figure5: sample of the dataset

The final dataset consists 172 samples with 46 different persons in different distance from the camera (**Figure5**). To ensure accuracy, samples with high-heeled shoes are ignored in the dataset. As shown in **Figure5**, the samples are annotated with four keypoints: knee (red point), ankle (blue point), heel (green point) and index (orange point). Those four keypoints are only annotated on the landing foot. Shown as **Figure5**, the distance to the camera is set to four levels (very close, close, far, and very far) based on the location of heel. The shoes are also classified as ankle shoes (shoes that cover the ankle), long boots (boots that cover lower legs) and others (no cover of the ankle and lower legs). The left graph of **Figure6** shows that most persons wear shoes that do not cover ankles and a few of them wear shoes with much coverage of the body. The right graph of **Figure6** shows most samples are far from the camera.

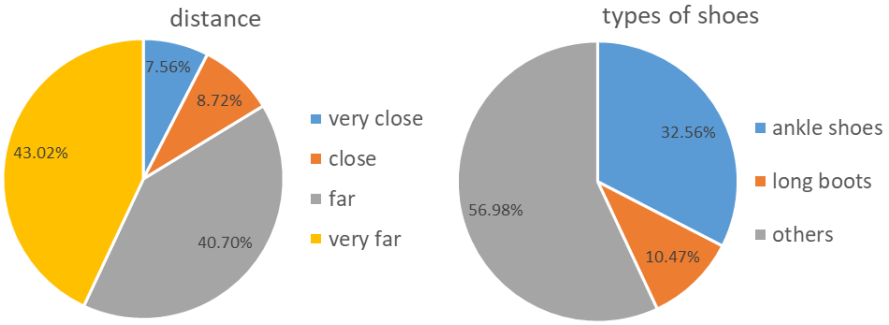

Figure6: composition of the dataset

Since the data annotation part may not be as accurate as the official worldwide datasets like COCO [16], the project analyses two indicators of performance: PCK and ankle angle difference of MediaPipe. The ankle angle difference is the target to recognize toe-walking symptom and PCK is the bias of keypoints coordinates of MediaPipe compared to the annotation (groundtruth). In this report, the model only considers the four keypoints on the landing foot. The formula of PCK [17] is shown as below:

$$PCK_{mean} = \frac{\sum_p \sum_i \delta\left(\frac{d_{pi}}{d_p^{def}} \leq T\right)}{\sum_p \sum_i 1}$$

Where i = the order of keypoint, p = the number of persons, d_{pi} = the Euclidean distance [18] d_p^{def} = the diameter of person's head, and T is threshold, for example, PCK@0.2 means 0.2 threshold and PCK@0.5 means 0.5 threshold. The formula of ankle angle difference is shown as below:

$$\Delta = \frac{|\theta - \hat{\theta}|}{\theta}$$

Where Δ is ankle angle difference, θ is the groundtruth angle and $\hat{\theta}$ is the angle estimated from MediaPipe.

The experiment result is shown in **Table1**. Where μ_{PCK} is the average value of PCK, σ_{PCK} is standard deviation of PCK, μ_{AD} is average value of angle difference and σ_{AD} is standard deviation of angle difference.

Table1: Experiment results

Shoe Type	Distance	Number	μ_{PCK}	σ_{PCK}	μ_{AD}	σ_{AD}
Ankle Shoes	Very Close	0				
	Close	2	0.0	0	45%	0.12
	Far	20	30.0	0.36	21%	0.12
	Very Far	16	18.8	0.19	23%	0.28
Long Boots	Very Close	0				
	Close	2	12.5	0.13	11%	0.05
	Far	7	28.6	0.28	23%	0.17
	Very Far	5	30.0	0.29	22%	0.12
Others	Very Close	8	31.3	0.24	16%	0.11
	Close	4	68.8	0.21	28%	0.19
	Far	29	41.4	0.33	26%	0.28
	Very Far	30	36.7	0.33	28%	0.28

Discussion

Following ISO 20916:2019 [11], the verification and validation for the design should focus on accuracy and robustness. **Table1** shows that the keypoints coordinates estimation performance fails expectation (PCK@0.2 33.3), which is less than half of the evaluation on Yoga dataset (PCK@0.2 84.5) [9] and the angle difference is 24%. Furthermore, only 123 out of 172 samples are detected successfully. Although the results seem to be disappointing for an IVD medical device, error analysis demonstrates the improvement and possible application for the design.

Normally, a larger PCK@0.2 value points to a smaller angle difference. However, **Figure7** shows that when angle difference decreases, PCK@0.2 value decreases as well. Given the human annotation error, there are only 4 keypoints to detect, which means the PCK@0.2 value will only be 0, 0.25, 0.50, 0.75 and 1.0. Compared with 33 keypoints detection in MediaPipe [9], there is likely to be huge deviation especially in a small dataset. Thus, this project focuses more on angle difference, a continuous indicator, instead of PCK@0.2.

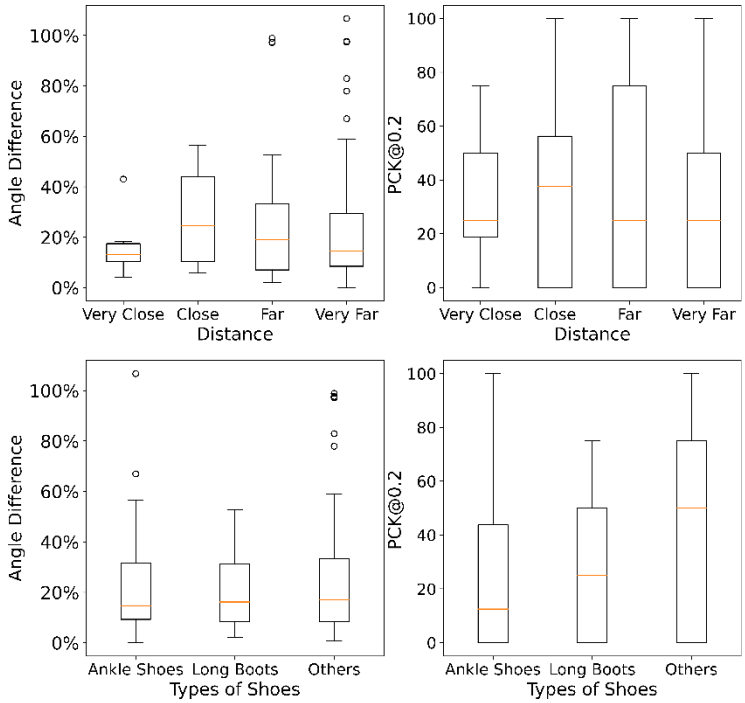

Figure7: Performance of the model

The graph on the bottom left of **Figure7** shows that different types of shoes do not impact the performance. Both standard deviation and expected value (the average) are closer. Since **Figure6** shows relative equal samples of three types of shoes, the coverage problem on the feet is reasonable to be ignored. Ignoring the insufficient sample problem of 'very close' samples, the graph on the top left of **Figure7** explains that when the distance increases, the angle difference decreases gradually. Although it seems that several samples have huge deviation from the normal distribution of 'very far' distance, when the person comes closer to the camera, the results come to be stable and robust. Thus, in practice, the output of design should be combination of the entire process of patient walking closer to the camera instead of a single picture.

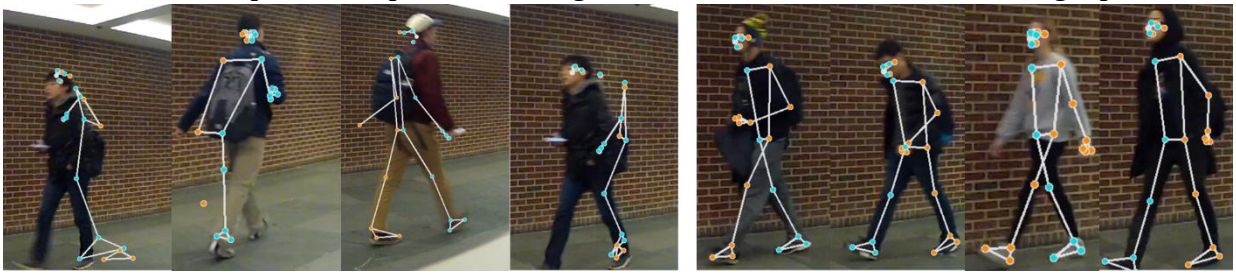

Figure8: Comparison of failures and success due to distance

Secondly, although the dataset is not sufficient in the 'close' and 'very close' samples, it may still be reasonable to suppose that due to the high-speed walking movement on the corridor and coverage problems of their only body (**Figure8**) in close distance, the model cannot correctly capture the keypoints. Most samples in far distance seem to be accurate from subjective analysis on the outputs (**Figure8**). For medical use, asking the patient to slow their walking speed will help the camera to capture their gait. Improving the revolution of camera also helps with the problem since the captured video will be clear for the model or setting multiple cameras will help.

Another interesting finding is that the coverage of the body instead of the feet has a larger impact on performance. As shown in the upper figures of **Figure9**, the person with a long jacket that covers her knee and a schoolbag confuses the model. Although there is almost no coverage on the feet, MediaPipe cannot recognize the person correctly. The bottom figures of **Figure9** show that when the sample is holding something or doing unexpected movement, the model has trouble detecting the body. Whereas, in reality, this problem can be solved just by asking the patients to take off their jacket and holding nothing on their hand.

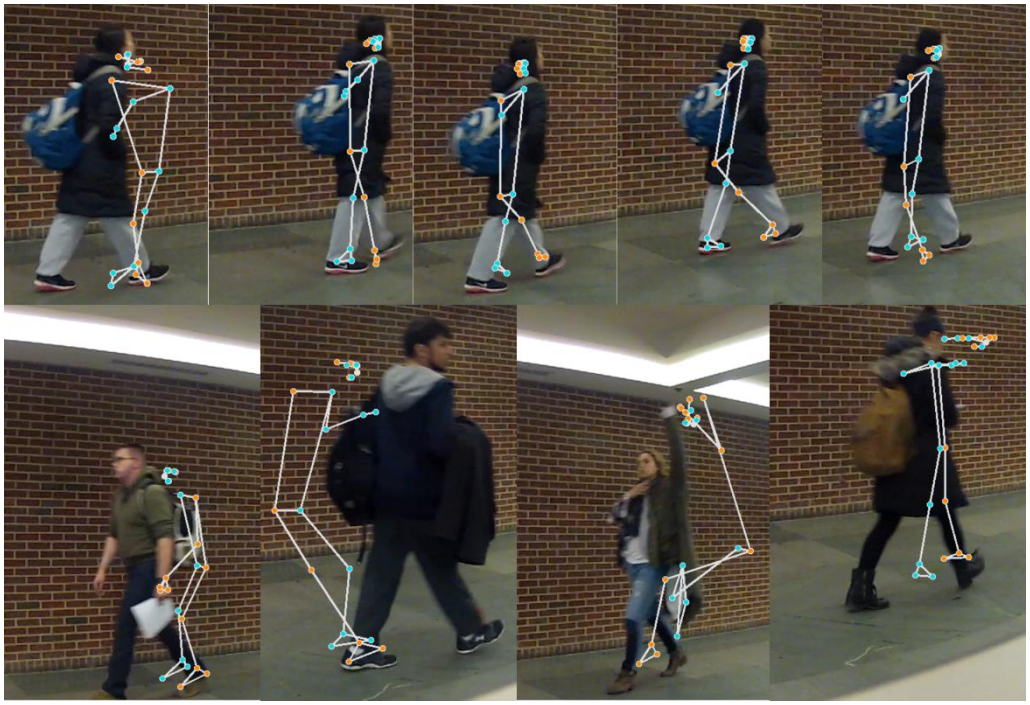

Figure9: Failed samples due to coverage and unexpected movement

FMEA form

To prevent the possible risks of failure detection and reduce its influence, Failure Mode and Effects Analysis (FMEA) is essential for the risk assessment. Under ISO 20916:2019 [11], mitigation for the potential risks shows the accuracy and robustness of the design in practical use. Focusing on reliability of the design, this report puts tasks, risk value, evidence and likelihood of occurrence into consideration, which are mitigated by several solutions.

Table2: FMEA [19]

Task	Potential problem	Evidence	Engineering Requirement	Risk	Likelihood	Score	Mitigation	New score
Model evaluation and accuracy detection	No evaluation indicator for improvement	**Discussion** session and **Figure7**	A reliable and consistent evaluation method	4	5	20	Only focus on angle difference instead of PCK.	5
Robust detection	MediaPipe is not reliable to estimate the coverage part	**Discussion** session and **Figure8**	Find a reliable method not only from the model but from the external solutions	4	4	16	Set multiple cameras and evaluate the both.	4
Robust and accurate detection	High-speed walking is hard to detect	**Discussion** session and **Figure8**	An easy way to solve the problem is enough.	4	5	20	Ask the patient to slow down, set multiple cameras and increase revolution.	5
Robust and accurate detection	MediaPipe may fail in a close distance	**Discussion** session and **Figure8**	Improve accuracy from all parts	5	5	25	Evaluate from a whole video instead of a closed picture.	5
Robust detection	Jacket coverage misleads model	**Discussion** session and **Figure9**	Solve the problem from patient side is better	4	3	12	Ask the patient to remove their jacket when detection.	3
Robust detection	Holding something or unexpected movement mislead model	**Discussion** session and **Figure9**	Solve the problem from a simple method is better	4	3	12	Ask the patient to hold nothing and behave normal when detecting.	3

Thoroughly, although the average ankle angle difference is 24%, that dataset is not aiming for a medical purpose. For the viewing difficulty such as distances and angles, longer video and multiple cameras are thought to help the problem. With some interaction with the patient such as wearing simply for the detection, the deviation will probably be controlled within an acceptable range.

Conclusion

This report creates a walking-toe-detection dataset from Zero-Occlusion-Object-Tracking-Data-Set and verifies accuracy and robustness of MediaPipe on it with a detailed design of vision-based toe-walking detection in unconscious condition. The walking-detection dataset consists feet-hitting-ground clips with annotations of four joints.

This report also verifies the reliability of the design following ISO 20916:2019 [11] and investigates several reasons for the unexpected result including high-speed walking, body and jacket coverage and close distance detection. Although PCK@0.2 is 33.3 and angle difference is 24%, which is not enough for medical use, in practice, camera settings and interaction with patient help to improve the results.

Further work involves verifying the suggested improvements, creating a new dataset with the majority of toe-walking patients, children, and validating the angle difference result with IMU suit. However, since MediaPipe is founded based on probability, there is not definite answer for the detection.

Appendix

Reference

[1] A. J. Caserta, V. Pacey, M. C. Fahey, K. Gray, R. H. H. Engelbert, and C. M. Williams, “Interventions for idiopathic toe walking,” Cochrane Database of Systematic Reviews, 2019.

[2] V. De Oliveira, L. Arrebola, P. De Oliveira, and L. Yi, “Investigation of Muscle Strength, motor coordination and balance in children with idiopathic toe walking: A case-control study,” Developmental Neurorehabilitation, vol. 24, no. 8, pp. 540–546, 2021.

[3] R. Soangra, M. Shiraishi, R. Beuttler, M. Gwerder, L. A. Boyd, V. Muthukumar, M. Trabia, A. Aminian, and M. Grant-Beuttler, “Foot contact dynamics and fall risk among children diagnosed with idiopathic toe walking,” Applied Sciences, vol. 11, no. 6, p. 2862, 2021.

[4] J. J. Ruzbarsky, D. Scher, and E. Dodwell, “Toe walking,” Current Opinion in Pediatrics, vol. 28, no. 1, pp. 40–46, 2016.

[5] Kim S, Soangra R, Grant-Beuttler M, Aminian A. WEARABLE SENSOR-BASED GAIT CLASSIFICATION IN IDIOPATHIC TOE WALKING ADOLESCENTS. Biomed Sci Instrum. 2019 Apr;55(2):178-185. PMID: 32214530; PMCID: PMC7094809.

[6] J. F. Policy, L. Torburn, L. A. Rinsky, and J. Rose, “Electromyographic test to differentiate mild diplegic cerebral palsy and idiopathic toe-walking,” Journal of Pediatric Orthopaedics, vol. 21, no. 6, pp. 784–789, 2001.

[7] G. Ershadi, M. Gwak, J. Liu, G. Lee, A. Aminian, and M. Sarrafzadeh, “Gaitoe: Gait analysis utilizing an IMU for toe walking detection and intervention,” Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 180–195, 2022.

[8] G. Ershadi, M. Gwak, A. Aminian, R. Soangra, M. Grant-Beuttler and M. Sarrafzadeh, "Smart Insole: Remote Gait Detection Algorithm Using Pressure Sensors For Toe Walking Rehabilitation," 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), 2021, pp. 332-337, doi: 10.1109/WF-IoT51360.2021.9595676.

[9] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, “Blazepose: On-device real-time body pose tracking,” Computer Vision and Pattern Recognition, 2020.

[10]

Medicines & Healthcare products Regulatory Agency. (2022, September). “Guidance: Medical device stand-alone software including apps (including IVDMDs)" [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1105233/Medical_device_stand-alone_software_including_apps.pdf

[11] ISO. (2019, May). “In vitro diagnostic medical devices. Clinical performance studies using specimens from human subjects. good study practice” [Online] Available at: https://doi.org/10.3403/30358014.

[12] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, “OpenPose: Realtime multi-person 2D pose estimation using part affinity fields,” Computer Vision and Pattern Recognition, 2019.

[13] Y. Xu, J. Zhang, Q. Zhang, and D. Tao, “Vitpose: Simple vision transformer baselines for human pose estimation,” Computer Vision and Pattern Recognition, 2022

[14] “Human pose estimation: Deep learning approach [2022 guide],” Human Pose Estimation: Deep Learning Approach [2022 Guide]. [Online]. Available: https://www.v7labs.com/blog/human-pose-estimation-guide. [Accessed: 13-Nov-2022].

[15] Wkentaro, “WKENTARO/labelme: Image polygonal annotation with python (polygon, rectangle, circle, line, point and image-level flag annotation).,” *GitHub*. [Online]. Available: https://github.com/wkentaro/labelme. [Accessed: 18-Jan-2023].

[16] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft Coco: Common Objects in Context,” Computer Vision and Pattern Recognition, 2015

[17] “Human pose estimation: Deep learning approach [2022 guide],” Human Pose Estimation: Deep Learning Approach [2022 Guide]. [Online]. Available: https://www.v7labs.com/blog/human-pose-estimation-guide. [Accessed: 13-Nov-2022].

[18] K. J. Smith, Precalculus: A functional approach to graphing and problem solving. Burlington, MA: Jones & Bartlett Learning, 2013.

[19] J. Liu, D. Wang, Q. Lin, and M. Deng, “Risk assessment based on FMEA combining DEA and cloud model: A case application in robot-assisted rehabilitation,” Expert Systems with Applications, vol. 214, p. 119119, 2023.

[20] D. Hal Unwin, C. A. Giller, and T. A. Kopitnik, “Central Nervous System Monitoring: What helps, what does not,” *Surgical Clinics of North America*, vol. 71, no. 4, pp. 733–747, 1991.

Programming

Landing_recognition.py

```
import os
import cv2
import shutil


samples = []
files = 'Zero-Occlusion-Object-Tracking-Data-Set-master'

for i in range(1, 21):
    if i < 10:
        samples.append('000' + str(i) + '.png')
    else:
        samples.append('00' + str(i) + '.png')
count = 0
for file in os.listdir(files):
    for sample in samples:
        img = cv2.imread(os.path.join(files, file, sample))
        cv2.imshow('1', img)
        pressedKey = cv2.waitKey(100000)
        if pressedKey == ord('s'):
            # landing foot
            count += 1
            saving_files = os.path.join(files, file, sample)
            shutil.copy(os.path.join(files, file, sample), 'new_dataset')
            os.rename(os.path.join('new_dataset', sample), os.path.join('new_dataset',
str(count) + '.png'))
            continue

        elif pressedKey == ord('j'):
            continue
```

```
for i in saving_files:
    shutil.copy(i, 'new_dataset')
```

composition_recognition.py

```
import os
import cv2
import json

dataset = 'landing-foot-recognition/images'

distance = {'1': [], '2': [], '3': [], '4': []}
types = {'elevator shoes': [], 'ankle shoes': [], 'long boots': [], 'others': []}

cv2.namedWindow("output", cv2.WINDOW_NORMAL)        # Create window with freedom of
dimensions
cv2.resizeWindow("output", 400, 300)

print('processing distance')
for file in os.listdir(dataset):

    img = os.path.join(dataset, file)
    image = cv2.imread(img)
    cv2.imshow("output", image)
    pressedKey = cv2.waitKey(100000)
    if pressedKey == ord('1'):
        distance['1'].append(img)

    elif pressedKey == ord('2'):
        distance['2'].append(img)
    elif pressedKey == ord('3'):
        distance['3'].append(img)
    elif pressedKey == ord('4'):
        distance['4'].append(img)


with open('distance.txt', 'w+') as f:
    json.dump(distance, f)


print('processing type shoe')
for file in os.listdir(dataset):

    img = os.path.join(dataset, file)
    image = cv2.imread(img)
    cv2.imshow("output", image)
    pressedKey = cv2.waitKey(100000)
    if pressedKey == ord('1'):
        types['elevator shoes'].append(img)

    elif pressedKey == ord('2'):
        types['ankle shoes'].append(img)
    elif pressedKey == ord('3'):
        types['long boots'].append(img)
    elif pressedKey == ord('4'):
        types['others'].append(img)

with open('types.txt', 'w+') as f:
    json.dump(types, f)
```

experiments.py

```
import mediapipe as mp
import cv2
import os
```

```
import json
import numpy as np
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_pose = mp.solutions.pose


def angle_calc(mat):
    x1 = mat[0, 0]
    y1 = mat[0, 1]
    x2 = mat[1, 0]
    y2 = mat[1, 1]
    x4 = mat[3, 0]
    y4 = mat[3, 1]
    angle = np.arccos(((x1 - x2) * (x2 - x4) + (y1 - y2) * (y2 - y4))
                      / np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2) / np.sqrt((x2
- x4) ** 2 + (y2 - y4) ** 2))
    return angle


def dis_calc(a, b):
    return np.sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2)


annotations = 'landing-foot-recognition/annotation'
images = 'landing-foot-recognition/images'
# cv2.namedWindow("output", cv2.WINDOW_NORMAL)        # Create window with freedom of
dimensions
# cv2.resizeWindow("output", 400, 300)
estimated_data = np.zeros((4, 2))
output = {}
diff_angle_list = []
pck_list = []
with mp_pose.Pose(
        min_detection_confidence=0.5,
        min_tracking_confidence=0.5) as pose:
    for file in os.listdir(images):
        data = np.zeros((4, 2))
        img = os.path.join(images, file)
        annotation = os.path.join(annotations, file.replace('png', 'json'))
        with open(annotation, 'r+') as f:
            gt = json.load(f)
        for i in range(4):
            if gt['shapes'][i]['label'] == 'knee':
                data[0, :] = gt['shapes'][i]['points'][0]
            elif gt['shapes'][i]['label'] == 'ankle':
                data[1, :] = gt['shapes'][i]['points'][0]
            elif gt['shapes'][i]['label'] == 'heel':
                data[2, :] = gt['shapes'][i]['points'][0]
            elif gt['shapes'][i]['label'] == 'index':
                data[3, :] = gt['shapes'][i]['points'][0]

        image = cv2.imread(img)
        image.flags.writeable = False
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = pose.process(image)
        if results.pose_landmarks == None:
            output[file] = {'detected': 0}
            continue
        # Draw the pose annotation on the image.
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        mp_drawing.draw_landmarks(
            image,
            results.pose_landmarks,
            mp_pose.POSE_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles.get_default_pose_landmarks_style())
```

```
        # Flip the image horizontally for a selfie-view display.
        '''
        for i in range(data.shape[0]):
            cv2.circle(image, (int(data[i][0]), int(data[i][1])), 5, (21,123,0), 10)
        cv2.imshow('output', image)
        '''
        cv2.imshow('output', image)
        pressedKey = cv2.waitKey(100000)
        if pressedKey == ord('l'):
            key = [25, 27, 29, 31]
            estimated_data[0, :] = [results.pose_landmarks.landmark[key[0]].x *
image.shape[1],
                                    results.pose_landmarks.landmark[key[0]].y *
image.shape[0]]
            estimated_data[1, :] = [results.pose_landmarks.landmark[key[1]].x *
image.shape[1],
                                    results.pose_landmarks.landmark[key[1]].y *
image.shape[0]]
            estimated_data[2, :] = [results.pose_landmarks.landmark[key[2]].x *
image.shape[1],
                                    results.pose_landmarks.landmark[key[2]].y *
image.shape[0]]
            estimated_data[3, :] = [results.pose_landmarks.landmark[key[3]].x *
image.shape[1],
                                    results.pose_landmarks.landmark[key[3]].y *
image.shape[0]]
            '''
            for i in range(data.shape[0]):
                cv2.circle(image, (int(estimated_data[i][0]),
int(estimated_data[i][1])), 5, (121,123,0), 10)
            cv2.imshow('output', image)
            '''
            cv2.imshow('output', image)
            pressedKey2 = cv2.waitKey(100000)
            if pressedKey2 != pressedKey:
                key = [26, 28, 30, 32]
                estimated_data[0, :] = [results.pose_landmarks.landmark[key[0]].x *
image.shape[1],
                                        results.pose_landmarks.landmark[key[0]].y *
image.shape[0]]
                estimated_data[1, :] = [results.pose_landmarks.landmark[key[1]].x *
image.shape[1],
                                        results.pose_landmarks.landmark[key[1]].y *
image.shape[0]]
                estimated_data[2, :] = [results.pose_landmarks.landmark[key[2]].x *
image.shape[1],
                                        results.pose_landmarks.landmark[key[2]].y *
image.shape[0]]
                estimated_data[3, :] = [results.pose_landmarks.landmark[key[3]].x *
image.shape[1],
                                        results.pose_landmarks.landmark[key[3]].y *
image.shape[0]]

        elif pressedKey == ord('r'):
            key = [26, 28, 30, 32]
            estimated_data[0, :] = [results.pose_landmarks.landmark[key[0]].x *
image.shape[1],
                                    results.pose_landmarks.landmark[key[0]].y *
image.shape[0]]
            estimated_data[1, :] = [results.pose_landmarks.landmark[key[1]].x *
image.shape[1],
                                    results.pose_landmarks.landmark[key[1]].y *
image.shape[0]]
            estimated_data[2, :] = [results.pose_landmarks.landmark[key[2]].x *
image.shape[1],
                                    results.pose_landmarks.landmark[key[2]].y *
image.shape[0]]
```

```
            estimated_data[3, :] = [results.pose_landmarks.landmark[key[3]].x *
image.shape[1],
                                    results.pose_landmarks.landmark[key[3]].y *
image.shape[0]]
            '''
            for i in range(data.shape[0]):
                cv2.circle(image, (int(estimated_data[i][0]),
int(estimated_data[i][1])), 5, (121,123,0), 10)
            cv2.imshow('output', image)
            '''
            cv2.imshow('output', image)
            pressedKey2 = cv2.waitKey(100000)
            if pressedKey2 != pressedKey:
                key = [25, 27, 29, 31]
                estimated_data[0, :] = [results.pose_landmarks.landmark[key[0]].x *
image.shape[1],
                                        results.pose_landmarks.landmark[key[0]].y *
image.shape[0]]
                estimated_data[1, :] = [results.pose_landmarks.landmark[key[1]].x *
image.shape[1],
                                        results.pose_landmarks.landmark[key[1]].y *
image.shape[0]]
                estimated_data[2, :] = [results.pose_landmarks.landmark[key[2]].x *
image.shape[1],
                                        results.pose_landmarks.landmark[key[2]].y *
image.shape[0]]
                estimated_data[3, :] = [results.pose_landmarks.landmark[key[3]].x *
image.shape[1],
                                        results.pose_landmarks.landmark[key[3]].y *
image.shape[0]]
        else:
            output[file] = {'detected': 0}
            continue
        # ankle difference calculation
        angle_predict = angle_calc(estimated_data)
        angle_truth = angle_calc(data)
        diff = abs(angle_truth - angle_predict) / angle_truth
        diff_angle_list.append(diff)

        # pck calculation
        thres = 0.2
        head1 = [results.pose_landmarks.landmark[7].x * image.shape[1],
                                        results.pose_landmarks.landmark[7].y *
image.shape[0]]
        head2 = [results.pose_landmarks.landmark[8].x * image.shape[1],
                                        results.pose_landmarks.landmark[8].y *
image.shape[0]]
        head_distance = dis_calc(head1, head2)
        count = 0
        for i in range(4):
            part_dis = dis_calc(estimated_data[i, :], data[i, :])
            if part_dis / head_distance <= thres:
                count += 1
        pck = count / 4.
        pck_list.append(pck)
        output[file] = {'knee': list(estimated_data[0, :]), 'ankle':
list(estimated_data[1, :]),
                        'heel': list(estimated_data[2, :]), 'index':
list(estimated_data[3, :]),
                        'angle_diff': diff, 'pck': pck, 'detected': 1}
print('pck:')
print(np.mean(pck_list))
print('angle_difference: ')
print(np.mean(diff_angle_list))
'''
with open('output.txt', 'w+') as f:
    json.dump(output, f)
```

```
'''
```

file_sum.py

```
import json
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


file_sum = {}
file_numpy = np.zeros((123, 5))
with open('pck.txt', 'r+') as f1:
    pck = json.load(f1)

with open('distance.txt', 'r+') as f2:
    distances = json.load(f2)

with open('types.txt', 'r+') as f3:
    types = json.load(f3)

with open('output.txt', 'r+') as f4:
    outputs = json.load(f4)

num = -1
for file in outputs.keys():

    if outputs[file]['detected'] == 0:
        file_sum[file] = {'detected': 0}

        continue
    if file in ['115.png', '116.png', '117.png']:
        continue
    file_address = os.path.join('landing-foot-recognition/images', file)
    pck_one = pck[file]

    for dis in distances.keys():
        if file_address in distances[dis]:
            break

    for type_one in types.keys():
        if file_address in types[type_one]:
            break

    angle_diff = outputs[file]['angle_diff']

    file_sum[file] = {'pck': pck_one, 'distance': dis, 'type': type_one, 'angle
difference': angle_diff, 'detected': 1}
    num += 1


    file_numpy[num, 0] = float(file.split('.')[0])
    file_numpy[num, 1] = pck_one
    file_numpy[num, 2] = float(dis)
    file_numpy[num, 3] = angle_diff

    if type_one == 'ankle shoes':
        file_numpy[num, 4] = 1
    elif type_one == 'long boots':
        file_numpy[num, 4] = 2
    else:
        file_numpy[num, 4] = 3
```

```
# with open('file_sum.txt', 'w+') as f:
#     json.dump(file_sum, f)
# np.savetxt('frame.csv', file_numpy, delimiter=',')


dis_box_angle = np.array([file_numpy[file_numpy[:, 2] == 1, 3],
file_numpy[file_numpy[:, 2] == 2, 3],
                          file_numpy[file_numpy[:, 2] == 3, 3],
file_numpy[file_numpy[:, 2] == 4, 3]])
dis_box_pck = np.array([file_numpy[file_numpy[:, 2] == 1, 1], file_numpy[file_numpy[:,
2] == 2, 1],
                        file_numpy[file_numpy[:, 2] == 3, 1], file_numpy[file_numpy[:,
2] == 4, 1]]) * 100.

type_box_angle = np.array([file_numpy[file_numpy[:, 4] == 1, 3],
file_numpy[file_numpy[:, 4] == 2, 3],
                           file_numpy[file_numpy[:, 4] == 3, 3]])
type_box_pck = np.array([file_numpy[file_numpy[:, 4] == 1, 1], file_numpy[file_numpy[:,
4] == 2, 1],
                         file_numpy[file_numpy[:, 4] == 3, 1]]) * 100.

plt.figure(figsize=(12, 12))

plt.subplot(221)
plt.boxplot(dis_box_angle)
plt.xticks(range(1, 5), ['Very Close', 'Close', 'Far', 'Very Far'], size=16)
plt.yticks([0.0, 0.2, 0.4, 0.6, 0.8, 1.0], ['0%', '20%', '40%', '60%', '80%', '100%'],
size=18)
plt.xlabel('Distance', size=18)
plt.ylabel('Angle Difference', size=18)

plt.subplot(222)
plt.boxplot(dis_box_pck)
plt.xticks(range(1, 5), ['Very Close', 'Close', 'Far', 'Very Far'], size=16)
plt.yticks(size=16)
plt.xlabel('Distance', size=18)
plt.ylabel('PCK@0.2', size=18)

plt.subplot(223)
plt.boxplot(type_box_angle)
plt.xticks(range(1, 4), ['Ankle Shoes', 'Long Boots', 'Others'], size=16)
plt.yticks([0.0, 0.2, 0.4, 0.6, 0.8, 1.0], ['0%', '20%', '40%', '60%', '80%', '100%'],
size=18)
plt.xlabel('Types of Shoes', size=18)
plt.ylabel('Angle Difference', size=18)

plt.subplot(224)
plt.boxplot(type_box_pck)
plt.xticks(range(1, 4), ['Ankle Shoes', 'Long Boots', 'Others'], size=16)
plt.yticks(size=16)
plt.xlabel('Types of Shoes', size=18)
plt.ylabel('PCK@0.2', size=18)

plt.savefig('1.png', bbox_inches='tight', dpi=1200)
```

output_calc.py

```
import os
import json
import numpy as np


def dis_calc(a, b):
    return np.sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2)
```

```
annotations = 'landing-foot-recognition/annotation'
head_lengths = 'landing-foot-recognition/headlength'
joints = ['knee', 'ankle', 'heel', 'index']
pck_list = []
pck_dict = {}


with open('output.txt', 'r+') as f:
    output = json.load(f)
    for file in output.keys():
        if output[file]['detected'] == 0:
            continue
        with open(os.path.join(annotations, file.replace('png', 'json')), 'r+') as f:
            annotation = json.load(f)

        with open(os.path.join(head_lengths, file.replace('png', 'json')), 'r+') as f:
            head_length_file = json.load(f)

        head_length = dis_calc(head_length_file['shapes'][0]['points'][0],
head_length_file['shapes'][0]['points'][1])

        # calculation distances
        distances = []
        for joint in joints:
            for i in annotation['shapes']:
                if i['label'] == joint:
                    distances.append(dis_calc(output[file][joint], i['points'][0]))
                    continue

        # pck calculation
        thres = 0.2
        count = 0
        for i in range(4):
            part_dis = distances[i]
            if part_dis / head_length <= thres:
                count += 1
        pck = count / 4.
        pck_list.append(pck)
        pck_dict[file] = pck
with open('pck.txt', 'w+') as f:
    json.dump(pck_dict, f)
print(np.mean(pck_list))
```