UNIVERSITY OF TRENTO

Department of Industrial Engineering

Master degree in Mechatronics Engineering

Course of Robotic Perception and Action

Gait recognition with the usage of a Intel© RealSense ToF camera

Daniele Serafini mat. 232273
(daniele.serafini@studenti.unitn.it)

Matteo Mastrogiuseppe mat. 231762
(matteo.mastrogiuseppe@studenti.unitn.it)

Fanni Sara Beleznay mat. xxxxxx
(fanni.beleznay@studenti.unitn.it)

Academic Year 2021-2022

Contents

1 Introduction

Rett Syndrome is a genetic disorder that disrupts brain development, which results in mental and physical disabilities. Difficulty with walking presents itself at the very early stages from when symptoms start to occur. In the end stages of the syndrome, patients lose their ability to walk altogether. Furthermore, this deterioration is accompanied with a side to side movement and gait abnormalities. In order to assist the affected children with walking and improving their gait, an walking aid structure has been developed that focuses on supporting the patients. A requirement of this project is to be able to attach a 3D Camera to the walking aid in order to measure the hip movements of the patients and map out the changes within their gait cycle.

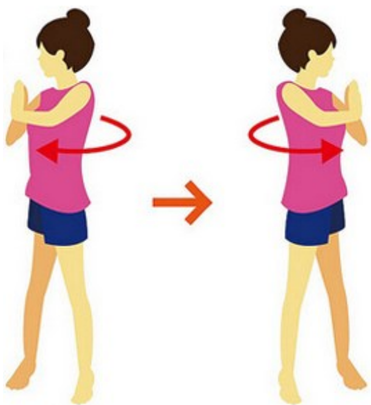
The gait cycle refers to the series of repetitive movements that occur when a person takes a step forward and it measures the time interval between these same series of movements. The starting point of a gait cycle is often the moment a foot comes in contact with the ground. For the purpose of this project, instead of a full gait cycle analysis, we will measure the repetitive hip movements that accompany the walking strides. From the relative positioning of the hips in reference to the camera, we will be able to produce an analysis of the parity of the gait and the changes over time. A clearer variable that can be used in our analysis is the angle subtended by the line approximating the waist, with respect to the camera reference frame.

ToF cameras are ideal for the measurements within this study. Unlike an RGB camera, a ToF camera is able to produce depth analysis of what is picturing, thus recreating a virtual 3D point-cloud of the environment. In our application the camera is fixed on the walking aid, and the distance between the camera and the person is a non flexible constraint. Therefore, it is necessary to use a device that is capable of dealing with objects that are placed in a close range. In order to get a reliable point-cloud, using the Intel Real-Sense D455, the minimum distance from the camera shall be no less than 15/20 cm: a lower value would lead to excessively noisy data or even no data at all.

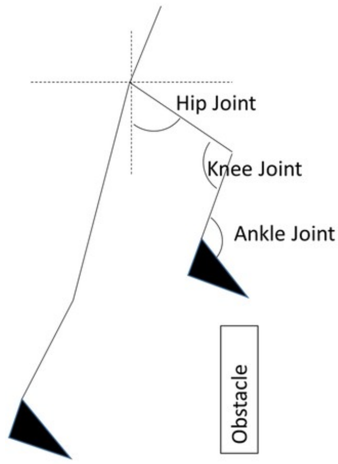
Most of the research in the field of gait analysis consists in fixed cameras that frame a specific environment, where subjects must walk in front of the camera. The approaches tend to limit their attention to the lateral view, while in our case the camera is placed right in front of the person, further increasing the challenge of our application.

2 Gait Analysis

In our work we can't use the usual set of parameters that the literature adopts to describe the gait of the person, since they refer to a side-view approach. Two gait parameters that we can analyze just by looking at the person's waist are the angle of rotation of the waist and the angle that the leg forms with respect to the hip, when stepping forward. These two parameters can be obtained by exploiting the depth information that we get from the camera, so they are suitable for our application.



(a)



(b)

Figure 1: Gait parameters analyzed in our work. Figure (1a) refers to the angle subtended by the waist with respect to the horizontal axis. Figure (1b) shows the "hip joint angle" parameter.

3 Evolution of our model

Our preliminary experiments focused on the estimation of the distance between the person's hips/lower midriff and the camera. Multiple approaches were used in order to be able to estimate the correct position of the waist points.

The identification of the belly, on a first approach, is obtained just by cropping the acquired RGB image, in order to get a close-up of the person's belly. The point-cloud is extracted with the depth camera, making sure that the framing is correctly calibrated. The distance of the waist from the camera is calculated by simply taking the average distance of the single points in the point-cloud. The points of potential objects in the background are filtered to avoid significant errors, simply by setting a range of significance on the axis that refers to the distance from the camera. This approximation requires the subject to position itself to get his waist correctly framed, greatly reducing the flexibility of the system.

A completely different approach is tackling the problem by using Person Detection and Pose Estimation algorithms based on Neural Networks. This tools would allow us to quickly detect the key-points of interest, that are necessary for every type of gait analysis. A Machine Learning approach surely increases the versatility of the technology, minimizing the need for a calibration of the device. As mentioned before, this is not without drawbacks. It is in fact very challenging to find an appropriate network that fits our aim, since most of the pre-trained networks are specialized in identifying the set of joints of the whole body. Training a new network from scratch can't be considered as a feasible route for our application, due to the lack of compatible, pre-existent online data sets. Creating our own data set would require the help of a field expert, and in general an enormous amount of effort.

Another possible route for joint detection is using a belt with markers in correspondence of the key-points of the waist. By processing the RGB feed it is possible to recognize the geometrical shape of the markers, and then translate these positions in the point-cloud. The results that we obtained were not good enough to rely on this tool, since the algorithm often suggested many more points than the ones that we would expect. However, the major constraint is time related: the methods that we found and implemented are very slow. The processing of a single image takes multiple seconds, thus it is not appropriate for a real-time application.

Our final model relies only the information that we get from the point-cloud: intelligently cutting the point-cloud around the person waist, it is possible to find the key-points by referring to the extremities of the body and averaging on regions of points.

4 Real-Time Demo

4.1 Intel Real-Sense Configuration

```matlab
% Make Pipeline object to manage streaming
pipe = realsense.pipeline();

% Create an empty Point Cloud from the RealSense
pointcloud = realsense.pointcloud();

% Start streaming on an arbitrary camera with default settings
config = realsense.config();
config.enable_stream(realsense.stream.depth,640,360,...
    realsense.format.z16,30);
config.enable_stream(realsense.stream.color,640,360,...
    realsense.format.rgb8,30);

pipe.start(config);
```

For a correct communication with the **Intel Real-Sense** is necessary to follow some procedures. At the beginning it is mandatory the creation of a *pipeline* that allows the bidirectional communication from the camera to the computer and vice versa: this pipeline will be used to request frames when needed, that will be sent to the computer when ready. Following the creation of the pipeline it is necessary to define a class object of type *realsense.pointcloud* that will store all the data connected to the depth frame obtained by the depth sensor of the *ToF camera*. Successively the actual configuration of the camera must be done: here the user defines the resolution and the frame rate of both the RGB camera and the depth sensor to his necessities, here we used the specific formats **rgb8** for the color camera and **z16** for the depth camera. After the configuration is saved in a variable, the actual *pipeline* can be started with the command *pipe.start(config)*.

```matlab
% transformation
tform = rigid3d([1 0 0; 0 0 -1; 0 1 0],[0 0 0]);

player = pcplayer([-0.25 0.25], [0.1 0.8], [-0.2 0.2]);
title = player.Axes.Title;

figure(2);
subplot(3,1,1)
grid on
ylim([-0.25 0.25])
li_x = animatedline(gca);

subplot(3,1,2)
grid on
ylim([0.1 0.8])
li_y = animatedline(gca);

subplot(3,1,3)
grid on
ylim([-45 45])
li_th = animatedline(gca);
```

```

    fs = pipe.wait_for_frames();

    % Depth Frame
    depth = fs.get_depth_frame();
```

For a correct representation of the data collected from the camera is necessary to apply a spacial transformation of the point cloud: the function *tform* allows us to apply a rotation of the pointcloud form the reference frame of the depth camera to the static reference frame of the ground. Successively it is necessary to start a specific plot to obtain a fast and responsive representation of the successions of pointclouds: with the command *pcplayer* Matlab cretes a specific plot to show successive pointclouds in real time, closely to a format of a video. For a time varying representation of the data it is necessary instead the initialization of different kind of plot: with *animatedline* we can create a real time representation in terms of a line plot. Finally we can request frames to the camera with the command *pipe.wait for frames()* and store the data contained in the frame with the command *fs.get depth frame()*.

```
    points = pointcloud.calculate(depth);
    vertices = points.get_vertices();
    ptcl = pointCloud(vertices(rem(1:height(vertices),15)==0,:));
```

With these commands we can create and manipulate the point cloud in Matlab. With the command *pointcloud.calculate(depth)* it is possible to get the positions of all the points detected by the depth sensor: we can store this information as a vector of coordinates with the command *points.get vertices()*. To actually create a point-cloud that can be visualized and manipulated in Matlab it is necessary to translate the coordinates: this is done with the command *pointCloud* where we also apply a downsampling to allow fluidity to our code.

```
    ptcl_out = pctransform(ptcl,tform);
    indices = findPointsInROI(ptcl_out,[-0.25 0.25 range -0.15 0.15]);
    ptcl_zone = select(ptcl_out,indices);
    ptcl_zone.Color = lab2uint8(repmat([128 128 128],ptcl_zone.Count,1));
    mean_zone = mean(ptcl_zone.Location);
```

This series of code lines permits to select the points of the point cloud of our interest: we want to analyze only the points of the waist that are in front of the camera, after the proper rotation of the point cloud is applied. To do this it is necessary to use the function *findPointsInROI*, setting the range as `[-0.25 0.25 range -0.15 0.15]`.*Range* is a two dimensional vector that refers to the interval to consider in the y axis (distance with respect to the camera). This range is updated at the end of every acquisition, in order to make this region move coherently with the position of the body. In this way it is possible to filter the background and undesired objects, without the need for a proper calibration depending on the distance from the camera. Successively a defined color is set and the mean over all points is calculated.

```
    indices_base = findPointsInROI(ptcl_zone,[ptcl_zone.XLimits(1)...
       ptcl_zone.XLimits(2) range mean_zone(3)+0.02 mean_zone(3)+0.07]);
    ptcl_base = select(ptcl_zone,indices_base);

    indices_left = findPointsInROI(ptcl_zone,[ptcl_base.XLimits(1)...
        ptcl_base.XLimits(1)+0.05 range mean_zone(3)+0.02 ...
           mean_zone(3)+0.07]);
    ptcl_left = select(ptcl_zone,indices_left);
    ptcl_left.Color = lab2uint8(repmat([255 0 0],ptcl_left.Count,1));
    mean_left = mean(ptcl_left.Location);

    indices_right = ...
       findPointsInROI(ptcl_zone,[ptcl_base.XLimits(2)-0.05 ...
                ptcl_base.XLimits(2) range mean_zone(3)+0.02 ...
                    mean_zone(3)+0.07]);
    ptcl_right = select(ptcl_zone,indices_right);
    ptcl_right.Color = lab2uint8(repmat([200 0 200],ptcl_right.Count,1));
```

```
mean_right = mean(ptcl_right.Location);
```

This section of code represents the final manipulation of the point cloud to determine the angle of the waist of the person in front of the camera. With the definition of the point cloud *ptcl base* we define a zone of the human waist that is 10 centimeters above its mean value: in this way we can estimate better the angle of the waist since we are more confident of actually picking points of the waist and not points of the legs. Successively we define two new point clouds that will show the left portion and right portion of the point cloud on which the calculation of the angle is executed: for an easier representation different colors are attributed to these two portions of the main point cloud. Finally, for a well defined representation, spheres are added to the point cloud in correspondence of the mean values calculated. The result is shown in figure 2.

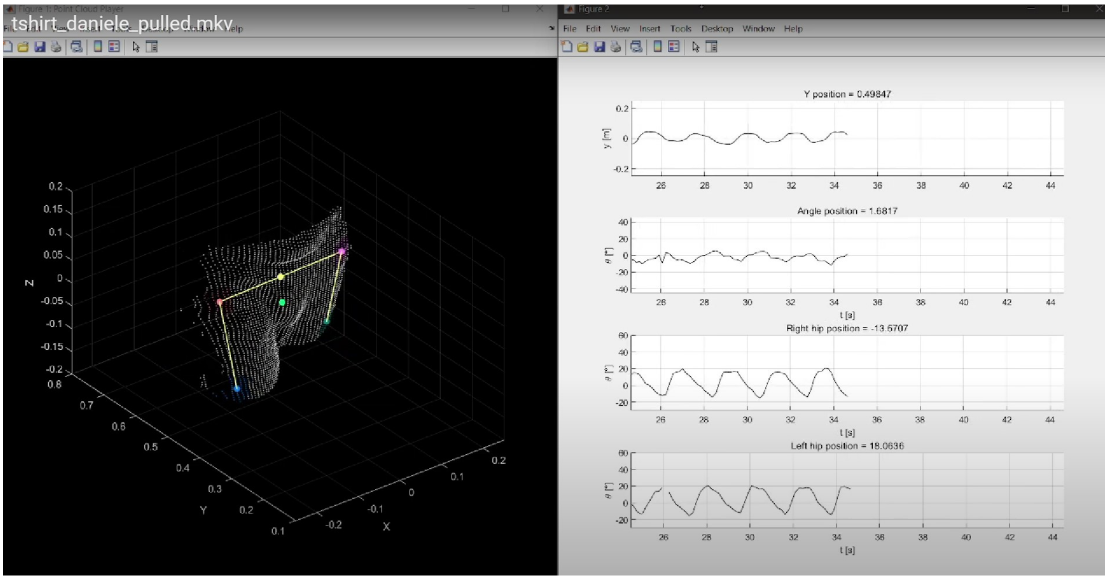


Figure 2: Image showing the demo of the code: on the left the green point shows the mean position of all the point cloud, the red point shows the mean of the left section and the magenta point shows the mean of the right section. The yellow point shows the origin of the line. On the right the representation of the values obtained in time.

For a fully comprehensive understanding of the demo the code used in its totality is listed in the appendix.

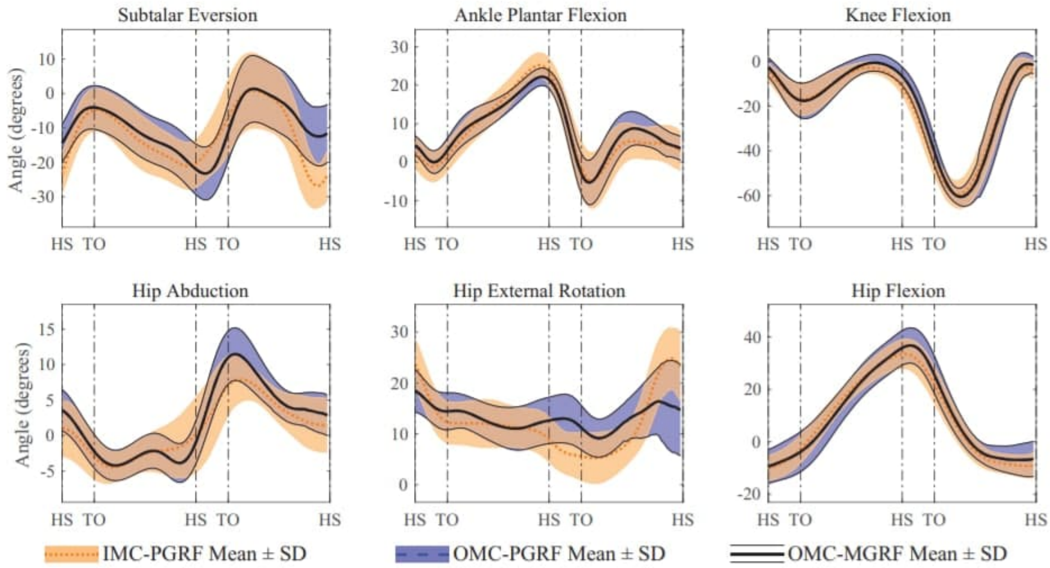


Figure 3: Diagram obtained from the paper *Predicting kinetics using musculoskeletal modeling and inertial motion capture*[1] where we can see the peculiar graph of the hip flex that we can also obtain from the demo. Here **HS** refers to the heel strike of the leg analyzed and **TO** refers to the toe off of the opposite leg.

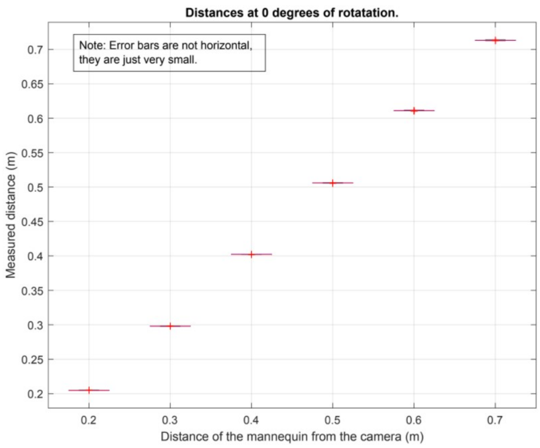
4.2 Static performances

In figure 4 we analyze the behaviour of the model in a static test. A mannequin is fixed in front of the camera, at a certain distance and a relative rotation. Our goal is to determine the accuracy of the program by comparing the belly distance and waist angle that we obtain with the real ones. The tests were executed by moving the mannequin with a step of 10 cm, starting with a distance of 20 cm and stopping at 70 cm. For each step the measurement is performed in two possible configurations of angles, one maintaining the mannequin orthogonal with respect to the camera, one by rotating it by 20 degrees. In each configuration we recorded 1000 camera frames: in the plots showed in figure 4 the deviation off the mean of these measurements is illustrated.

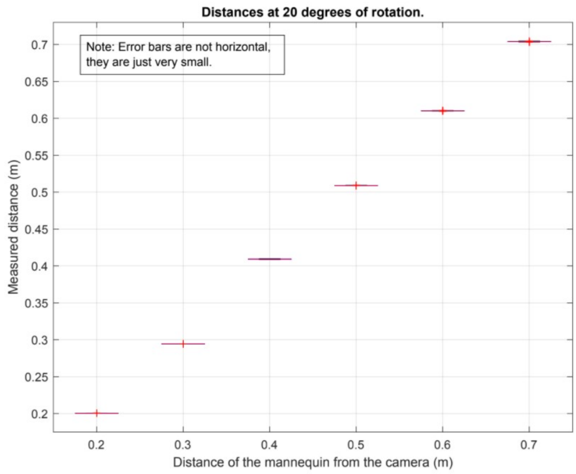
It can be observed that the model is able to measure the distance of the mannequin's waist from the camera quite effectively. Most notably, the repetitivity of the measurement is very high, since the deviation in every frame acquired is very low.

Regarding the waist angle, it can be observed that the model is not stable, even in a static environment. This is because the waist angle measurement relies on the points of the point-cloud close to the end of the waist, and this region is very noisy. We also observed that it is not feasible to estimate accurately the angle when we place the mannequin further from the camera: the point-cloud has fewer points representing the body. On average, the values that we obtain are decent if we want to just focus on the analysis of the gait patterns.

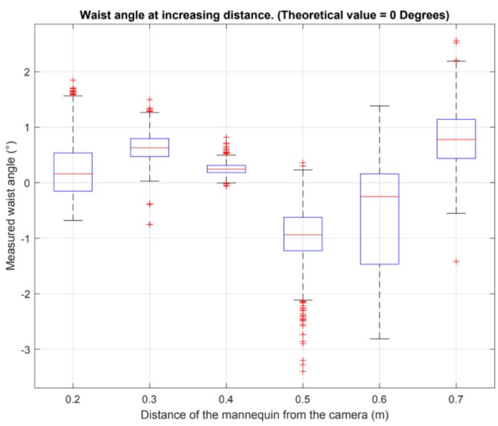
The experimental setup was definitely not accurate enough to analyze the results in a



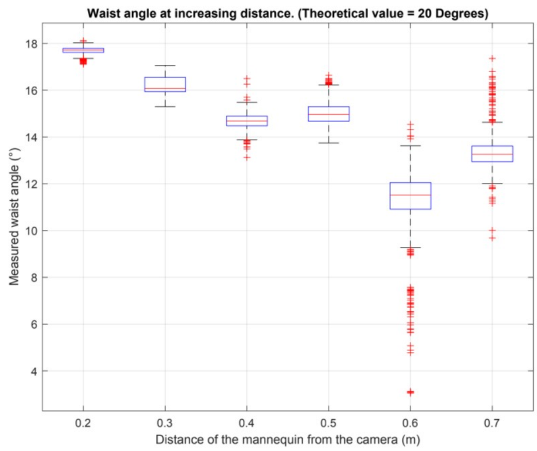
(a)



(b)



(c)



(d)

Figure 4: Statistical results obtained in a well defined environment at **MIRO Lab**.

serious way, but the patterns in the deviation that we identified are valid.

4.3 Real-Time Application

The application we created is intended for a human-friendly selection of the cutting zone of the waist. This is done by implementing the visualization on the RGB feed of the camera of four lines that represent the boundaries of the point cloud selected. The depth, since it is not showing in the RGB frame, is not controllable, so it is set to cover the range of distances from 0.2 to 0.7 meters.

The complete code is added in the appendix. We decided to keep both models (the Matlab function and the executable program), since the former is able to recursively and intelligently cut the point-cloud and because the manual cutting is slightly slower.

The final application represented in figure 5 shows a human friendly interface, where an operator can manually select the region of interest to the analysis.

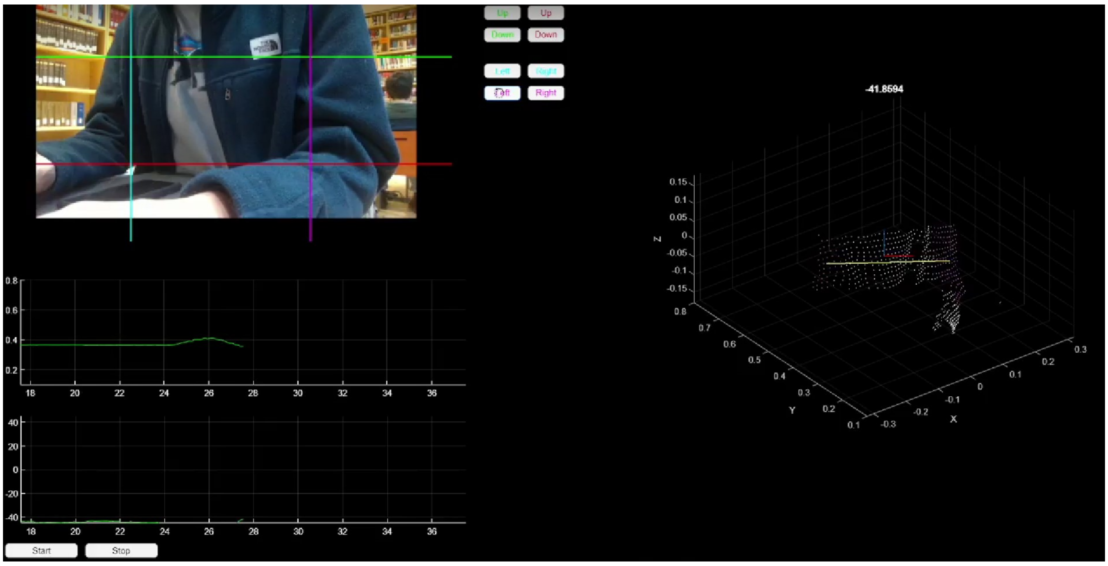


Figure 5: Image showing the application: On the left the RGB feed is shown to understand the region where the cut is made but also two graphs, regarding distance and angle, are shown. On the left the point cloud is shown to better understand what section of the RGB camera is shown.

4.4 Real-time performances

Here are presented two snapshots from a demo where we mounted the camera on top of a cart (figure 6 and figure 7). In the first case the cart is pulled by an external individual, in the second one the person carrying out the gate is pushing the cart by himself. It can be observed how the model is able to provide quite precise information about key parameters of the gait:

- "Y Position" refers to the distance of the person from the camera. The distance remains basically the same, except for small deviations that are due to the fact that the person is not completely synchronized with the cart movement.
- "Angle Position" refers to the angle determined by the two key-points of the waist. As we can expect, its pattern is coherent with the movement of the person, assuming negative and positive values depending on which leg is moving forward.
- "Right"/"Left Hip Position" refers to the angle that the key-points of the right and left leg subtend with the respective waist key-point. A leg moving forward is matched with a positive value for the angle and viceversa. Note that in our model the terminology for "right" and "left" is inverted, since we wanted to make it more intuitive when looking at the mirrored point-cloud.

In our trials we found that the model is not sensible to changes in the individual's clothing, the key-points are correctly identified in a standard setting. The same stands for the environment: the model is capable to filter objects placed behind or in front of the person, but in this second case we would of course lose part of the point-cloud, since the object shadows the body.

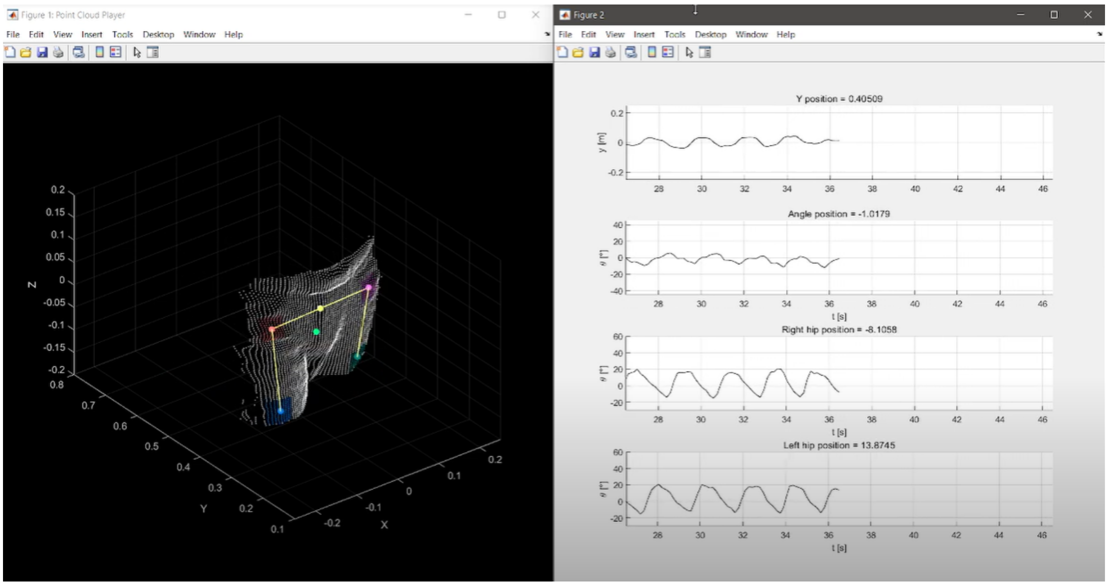


Figure 6: Snapshot of individual 1 moving, cart is pulled.

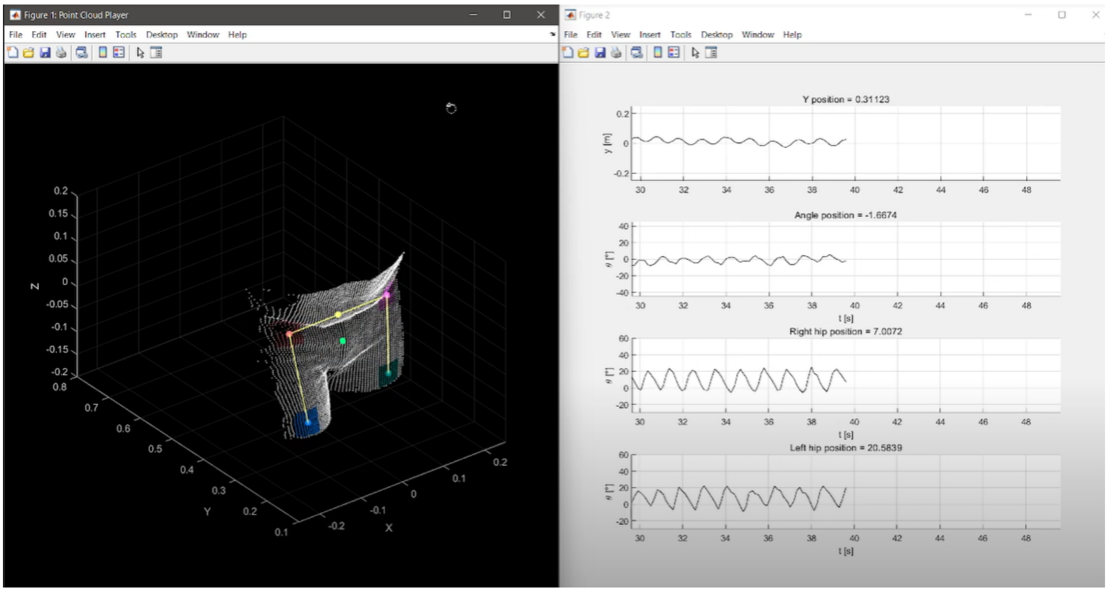


Figure 7: Snapshot of individual 2 moving, cart is pushed. Note how a loose cardigan is not a problem, the model is capable of accurately detecting the key-points of the waist.

5 Skeleton Analysis

A potential solution to the initial problem of identifying the points of the waist from a camera included using previously built learning algorithms and adapt them to recognise the waist and legs from a close range. These methods, however presented with a lot of difficulties therefore were not fit for the purposes of this study.

Xu et al. [2] conducted a study in which they were able to use a Microsoft $Kinect^{TM}$ sensor on a treadmill to measure the gait parameters from a frontal point of view. Within the study, it was concluded that the sensor was not accurate enough for all walking speeds and gaits, it is our aim that using a ToF camera, the results will be thorough and correct. With OpenPose we are attempting to follow this approach and extract only the hip location and angle in order to be able to determine some gait abnormalities.

OpenPose is a system that is able to detect joints within an image and reconstruct a skeleton based on the points. It has a large data-set already, with a trained algorithm that can be used and is designed so that it is relatively easy to apply within a study. Moreover, it is includes facial feature recognition as well as foot recognition. The latter has a key function within gait analysis. Using this for gait analysis is helpful because it would allow for an accurate estimation of the skeletal structure and be able to track the walking movements of the patients to detect any gait abnormalities.

5.1 Matlab Built-in Pose Estimation

First, we applied the Matlab Pose Estimation within trial versions of this project which yielded some mixed results. The pose estimator is based on the use of OpenPose. In terms

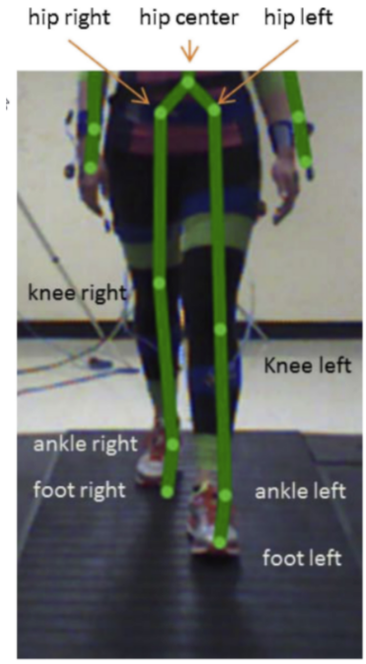


Figure 8: [2] This figure is an example of what was achieved during the study by Xu et al wherein a frontal view of the camera is used to detect hip and leg joint locations and movements.

of application, it was fairly simple, however there is a lack of flexibility when it came to the alteration of the original application which is where the problems occurred.

Matlab has built in usage of the OpenPose algorithm with a pre-trained network that can be integrated within the code for the recognition of multiple people within an image. It is also able to recognise a singular person in a live video feed at a pace that is fast enough for application within the project. However, the key issue that we noticed was that the accuracy of the pose estimation decreased significantly as soon as parts of the body were missing from the image, including if the person was too close to the camera. Due to this, the comprehensive gait analysis was not reliable enough to be able to give an accurate representation.

5.2 OpenPose possible applications

In order to improve upon the Matlab application of OpenPose, we attempted to train the network using the OpenPose algorithm as a basis, however there was an increased number of factors that would prove not to be useful within this application. The usage of the OpenPose algorithm is due to the fact that it is not feasible for our project to design and train our own deep network with the creation of our own dataset.

Initially, there was a slight improvement upon the previous Matlab application, as this version of OpenPose was able to identify partial bodies within a frame. With images, the

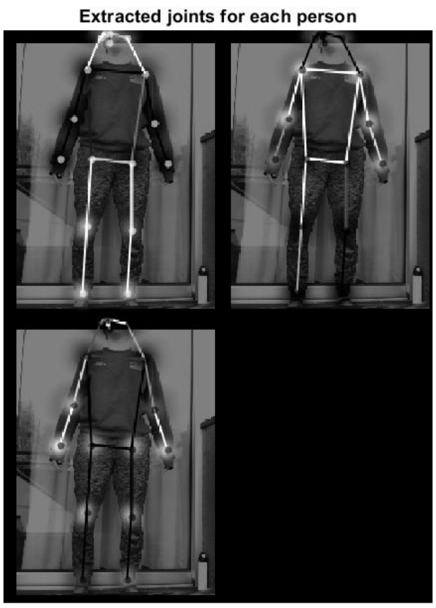


Figure 9: The images represent the last step in pose estimation that the algorithm which attempts to find all the people within a picture. It requires the entire body to be visible within the frame to give the correct representation.

results were good, on the other hand, the recognition from a video feed was not acceptable.

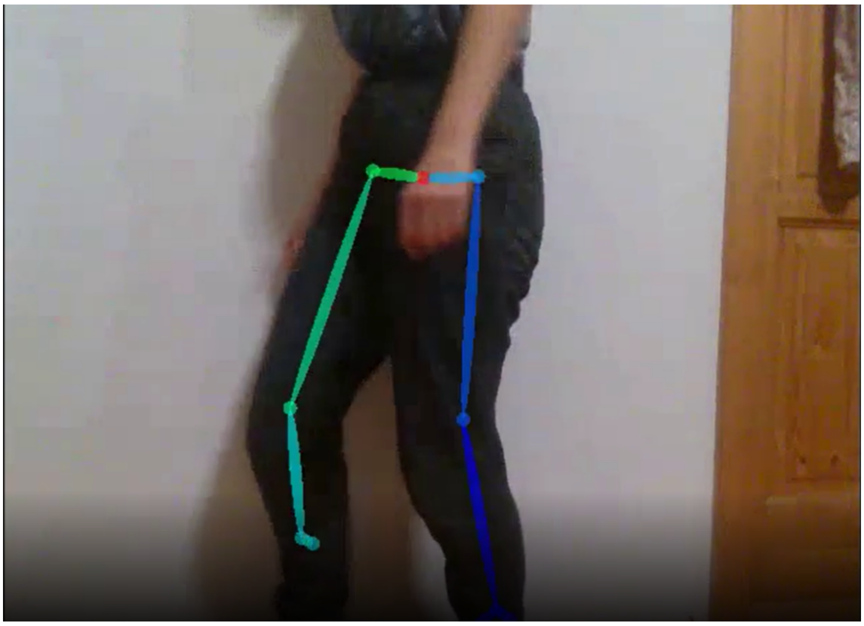


Figure 10: Above is an image captured from a video that is applying the OpenPose technique. The skeleton tracking is partially correct although it does not follow the skeletal structure of the person through all the frames within the video.

The software was not able to keep the frame of the body. Additionally, when attempting to put the camera close (within the 40cm range) and identify the hips, it was not able to so. Some other constraints and problems that affected the usage of this particular algorithm were its heavy reliance on the environmental conditions. Even the slightest change in lighting affected the results, as well as using looser clothing which were the main two that were tested. Overall, the use of OpenPose algorithm to identify a skeletal framework is useful to other applications and has worked in analysing the gait cycle of a person when the right conditions are met, however not useful in our case of study.

bib.bib

6 Appendix

6.1 Application

```
classdef depth_view < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
```

```
        YposAX                      matlab.ui.control.UIAxes
        THposAX                     matlab.ui.control.UIAxes
        ColorAX                     matlab.ui.control.UIAxes
        PointCloudAX                matlab.ui.control.UIAxes
        StartButton                 matlab.ui.control.Button
        StopButton                  matlab.ui.control.Button
        Up1                         matlab.ui.control.Button
        Down1                       matlab.ui.control.Button
        Up2                         matlab.ui.control.Button
        Down2                       matlab.ui.control.Button
        Left1                       matlab.ui.control.Button
        Right1                      matlab.ui.control.Button
        Left2                       matlab.ui.control.Button
        Right2                      matlab.ui.control.Button
    end

    properties (Access = private) %, Hidden = true)
        MeTimer             % Timer object
        DrawFlag            % Boolean
        Cfg                 % Realsense.config
        Pipe                % Realsense.pipeline
        Colorizer           % Realsense.colorizer
        PointCloud          % Realsense.pointcloud
        Profile             % Realsense.profile
        Frameset            % Realsense.frameset
        hhDepth             % Image
        hhColor             % Image
        hhCloud
        cy1
        cy2
        cx1
        cx2
        y1line
        y2line
        x1line
        x2line
        tform
        player
        li_y
        li_th
    end

    % Callbacks that handle component events
    methods (Access = private)

        function MeTimerFcn(app,varargin)

        if ( app.DrawFlag == 1 )

            % lock drawing process
            app.DrawFlag = 0;
```

```

            % Get frameset
            app.Frameset = app.Pipe.wait_for_frames();

            % Color
            color_frame = app.Frameset.get_color_frame();
            color_data = color_frame.get_data();
            color_img = permute(reshape(color_data', ...
                [3,color_frame.get_width(),color_frame.get_height()]), ...
                [3 2 1]);
            [ki,kj] = size(app.hhColor);

            if ki*kj < 1

                app.hhColor = ...
                    imshow(color_img,'Parent',app.ColorAX,'XData', [1 ...
                    app.ColorAX.Position(3)], ...
                                         'YData', [1 ...
                                            app.ColorAX.Position(4)]);
                app.y1line = yline(app.cy1,'Color','green', ...
                    'LineWidth',2,'Parent',app.ColorAX);
                app.y2line = yline(app.cy2,'Color','red', ...
                    'LineWidth',2,'Parent',app.ColorAX);
                app.x1line = xline(app.cx1,'Color','cyan', ...
                    'LineWidth',2,'Parent',app.ColorAX);
                app.x2line = xline(app.cx2,'Color','magenta', ...
                    'LineWidth',2,'Parent',app.ColorAX);

            else

                app.hhColor.CData = color_img;
                app.y1line.Value = app.cy1;
                app.y2line.Value = app.cy2;
                app.x1line.Value = app.cx1;
                app.x2line.Value = app.cx2;

            end

            % PointCloud Graph

            depth_frame = app.Frameset.get_depth_frame();

            if depth_frame.logical()

                points = app.PointCloud.calculate(depth_frame);
                vertices = points.get_vertices();
                ptcl = pointCloud(vertices(rem(1:height(vertices), ...
                    30)==0,:));

                ptcl_out = pctransform(ptcl,app.tform);
```

```
                indices = ...
                    findPointsInROI(ptcl_out,[-(320-app.cx1)/1000 ...
                    (app.cx2-320)/1000 0.1 0.8 -(app.cy2-180)/1000 ...
                    (180-app.cy1)/1000]);
                ptcl_zone = select(ptcl_out,indices);
                ptcl_zone.Color = lab2uint8(repmat([128 128 ...
                    128],ptcl_zone.Count,1));
                mean_zone = mean(ptcl_zone.Location);

                indices_left = ...
                    findPointsInROI(ptcl_zone,[-(320-app.cx1)/1000 ...
                    -((320-app.cx1)/1000)+0.05  0.1 0.8 ...
                    -(app.cy2-180)/1000 (180-app.cy1)/1000]);
                ptcl_left = select(ptcl_zone,indices_left);
                ptcl_left.Color = lab2uint8(repmat([255 0 ...
                    0],ptcl_left.Count,1));
                mean_left = mean(ptcl_left.Location);

                indices_right = ...
                    findPointsInROI(ptcl_zone,[((app.cx2-320)/1000) ...
                    -0.05 (app.cx2-320)/1000 0.1 0.8 ...
                    -(app.cy2-180)/1000 (180-app.cy1)/1000]);
                ptcl_right = select(ptcl_zone,indices_right);
                ptcl_right.Color = lab2uint8(repmat([200 0 ...
                    200],ptcl_right.Count,1));
                mean_right = mean(ptcl_right.Location);

                if ki*kj < 1

                    app.player = pcplayer([-0.320 0.320], [0.1 0.8], ...
                        [-0.18 0.18],'Parent',app.PointCloudAX);
                    view(app.player,pccat([ptcl_zone ptcl_right ...
                        ptcl_left]));

                    % Ypos Graph

                    app.li_y = animatedline(app.YposAX,'Color','g');
                    addpoints(app.li_y,toc,mean_zone(2));
                    drawnow limitrate
                    app.YposAX.XLim = [toc-10 toc+10];


                    % THpos Graph

                    app.li_th = animatedline(app.THposAX,'Color','g');
                    addpoints(app.li_th,toc,0);
                    drawnow limitrate
                    app.THposAX.XLim = [toc-10 toc+10];


```

```
                    elseif length(mean_left) == 3 && length(mean_right) ...
                        == 3

                        u = ...
                            (mean_left-mean_right)/norm(mean_left-mean_right);
                        line = mean_right + ...
                            (0:0.0005:norm(mean_left-mean_right))'*u;

                        ptcl_line = pointCloud(line);
                        ptcl_line.Color = lab2uint8(repmat([255 255 ...
                            0],ptcl_line.Count,1));

                        view(app.player,pccat([ptcl_zone ptcl_line ...
                            ptcl_right ptcl_left]));

                        % Ypos Graph

                        addpoints(app.li_y,toc,mean_zone(2));
                        drawnow limitrate
                        app.YposAX.XLim = [toc-10 toc+10];
                        app.YposAX.Title.String = ['Y Position = ' ...
                            num2str(mean_zone(2))];

                        % THpos Graph

                        angle = real(asind((mean_left(2)-mean_right(2))/ ...
                            (mean_left(1)-mean_right(1))));
                        addpoints(app.li_th,toc, angle);
                        drawnow limitrate
                        app.THposAX.XLim = [toc-10 toc+10];
                        app.THposAX.Title.String = ['Angle Position = ' ...
                            num2str(angle)];

                    end
                end

                % unlock drawing process
                app.DrawFlag = 1;
                pause(0.001);

            end
        end

        % Executes after component creation
        function StartUpFunc(app)
            % Create Realsense items

            app.Cfg = realsense.config();
            app.Cfg.enable_stream(realsense.stream.depth,424,240,...
                realsense.format.z16,30);
            app.Cfg.enable_stream(realsense.stream.color,424,240,...
```

```
                realsense.format.rgb8,30)
            app.Pipe = realsense.pipeline();
            app.Colorizer = realsense.colorizer();
            app.PointCloud = realsense.pointcloud();
            app.Profile = app.Pipe.start(app.Cfg);

            % Create timer object

            kFramePerSecond = 30.0; % Number of frames per second
            Period = double(int64(1000.0 / ...
                kFramePerSecond))/1000.0+0.001; % Frame Rate


            tic
            app.MeTimer = timer('ExecutionMode', 'fixedSpacing', ...
                'Period', Period, 'BusyMode', 'drop', 'TimerFcn', ...
                @app.MeTimerFcn);

            app.DrawFlag = 0;
            app.hhDepth = [];
            app.hhColor = [];
            app.tform = rigid3d([1 0 0; 0 0 -1; 0 1 0],[0 0 0]);

        end

        % Button pushed function: start timer
        function onStartButton(app, event)
            % If timer is not running, start it
            if strcmp(app.MeTimer.Running, 'off')
               app.DrawFlag = 1;
               start(app.MeTimer);
            end
        end

        function onUp1(app, event)

            app.cy1 = app.cy1 - 2;

        end

        function onUp2(app, event)

            app.cy2 = app.cy2 - 2;

        end

        function onDown1(app, event)

            app.cy1 = app.cy1 + 2;

        end
```

```
        function onDown2(app, event)

            app.cy2 = app.cy2 + 2;

        end

        function onRight1(app, event)

            app.cx1 = app.cx1 + 2;

        end

        function onRight2(app, event)

            app.cx2 = app.cx2 + 2;

        end

        function onLeft1(app, event)

            app.cx1 = app.cx1 - 2;

        end

        function onLeft2(app, event)

            app.cx2 = app.cx2 - 2;

        end

        % Button pushed function: stop timer
        function onStopButton(app, event)
            app.DrawFlag = 0;
            stop(app.MeTimer);
        end

        %Close request UIFigure function
        function UIFigureCloseRequest(app,event)
            app.DrawFlag = 0;
            stop(app.MeTimer);
            delete(app.MeTimer);
            app.Pipe.stop();
            delete(app.Profile);
            delete(app.Colorizer);
            delete(app.Pipe);
            delete(app.Cfg);
            delete(app);
        end

    end
```

```
    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = ...
                uifigure('WindowState','maximized','Visible','on');
            app.UIFigure.Name = 'Gait Analysis';
            app.UIFigure.CloseRequestFcn = ...
                createCallbackFcn(app,@UIFigureCloseRequest);
            setAutoResize(app,app.UIFigure,false);

            % Create ColorAX
            app.ColorAX = uiaxes(app.UIFigure);
            app.ColorAX.Position = [10 420 640 360];

            % Create YposAX
            app.YposAX = uiaxes(app.UIFigure);
            app.YposAX.Position = [10 230 640 170];
            app.YposAX.Color = 'k';
            app.YposAX.YColor = 'w';
            app.YposAX.XColor = 'w';
            app.YposAX.XGrid = 'on';
            app.YposAX.YGrid = 'on';
            app.YposAX.GridColor = 'w';
            app.YposAX.Title.String = 'Y Position';
            app.YposAX.Title.Color = [1 1 1];
            app.YposAX.YLabel.String = 'y [m]';
            app.YposAX.YLabel.Color = [1 1 1];
            app.YposAX.YLim = [0.1 0.8];




            % Create THposAX
            app.THposAX = uiaxes(app.UIFigure);
            app.THposAX.Position = [10 40 640 170];
            app.THposAX.Color = 'k';
            app.THposAX.YColor = 'w';
            app.THposAX.XColor = 'w';
            app.THposAX.XGrid = 'on';
            app.THposAX.YGrid = 'on';
            app.THposAX.GridColor = 'w';
            app.THposAX.Title.String = 'Angle Position';
            app.THposAX.Title.Color = [1 1 1];
            app.THposAX.XLabel.String = 't [s]';
            app.THposAX.XLabel.Color = [1 1 1];
            app.THposAX.YLabel.String = '\theta [deg]';
```

```
            app.THposAX.YLabel.Color = [1 1 1];
            app.THposAX.YLim = [-45 45];


            % Create PointCloudAX
            app.PointCloudAX = uiaxes(app.UIFigure);
            app.PointCloudAX.Position = [900 125 600 600];
            app.PointCloudAX.XLim = [-0.5 0.5];
            app.PointCloudAX.YLim = [0.1 0.8];
            app.PointCloudAX.ZLim = [-0.5 1.2];
            app.PointCloudAX.Title.String = 'Point Cloud';
            app.PointCloudAX.Title.Color = [1 1 1];


            % Create StartButton
            app.StartButton = uibutton(app.UIFigure, 'push');
            app.StartButton.ButtonPushedFcn = createCallbackFcn(app, ...
                @onStartButton, true);
            app.StartButton.IconAlignment = 'center';
            app.StartButton.Position = [10 10 100 20];
            app.StartButton.Text = 'Start';

            % Create Up1
            app.Up1 = uibutton(app.UIFigure, 'push');
            app.Up1.ButtonPushedFcn = createCallbackFcn(app, @onUp1, ...
                true);
            app.Up1.IconAlignment = 'center';
            app.Up1.Position = [670 750 50 20];
            app.Up1.Text = 'Up';
            app.Up1.FontColor = 'green';
            app.cy1 = 90;

            % Create Up2
            app.Up2 = uibutton(app.UIFigure, 'push');
            app.Up2.ButtonPushedFcn = createCallbackFcn(app, @onUp2, ...
                true);
            app.Up2.IconAlignment = 'center';
            app.Up2.Position = [730 750 50 20];
            app.Up2.Text = 'Up';
            app.Up2.FontColor = 'red';
            app.cy2 = 270;

            % Create Down1
            app.Down1 = uibutton(app.UIFigure, 'push');
            app.Down1.ButtonPushedFcn = createCallbackFcn(app, ...
                @onDown1, true);
            app.Down1.IconAlignment = 'center';
            app.Down1.Position = [670 720 50 20];
            app.Down1.Text = 'Down';
            app.Down1.FontColor = 'green';

```

```
            % Create Down2
            app.Down2 = uibutton(app.UIFigure, 'push');
            app.Down2.ButtonPushedFcn = createCallbackFcn(app, ...
               @onDown2, true);
            app.Down2.IconAlignment = 'center';
            app.Down2.Position = [730 720 50 20];
            app.Down2.Text = 'Down';
            app.Down2.FontColor = 'red';

            % Create Right1
            app.Right1 = uibutton(app.UIFigure, 'push');
            app.Right1.ButtonPushedFcn = createCallbackFcn(app, ...
               @onRight1, true);
            app.Right1.IconAlignment = 'center';
            app.Right1.Position = [730 670 50 20];
            app.Right1.Text = 'Right';
            app.Right1.FontColor = 'cyan';
            app.cx1 = 160;

            % Create Right2
            app.Right2 = uibutton(app.UIFigure, 'push');
            app.Right2.ButtonPushedFcn = createCallbackFcn(app, ...
               @onRight2, true);
            app.Right2.IconAlignment = 'center';
            app.Right2.Position = [730 640 50 20];
            app.Right2.Text = 'Right';
            app.Right2.FontColor = 'magenta';
            app.cx2 = 480;

            % Create Left1
            app.Left1 = uibutton(app.UIFigure, 'push');
            app.Left1.ButtonPushedFcn = createCallbackFcn(app, ...
               @onLeft1, true);
            app.Left1.IconAlignment = 'center';
            app.Left1.Position = [670 670 50 20];
            app.Left1.Text = 'Left';
            app.Left1.FontColor = 'cyan';

            % Create Left2
            app.Left2 = uibutton(app.UIFigure, 'push');
            app.Left2.ButtonPushedFcn = createCallbackFcn(app, ...
               @onLeft2, true);
            app.Left2.IconAlignment = 'center';
            app.Left2.Position = [670 640 50 20];
            app.Left2.Text = 'Left';
            app.Left2.FontColor = 'magenta';

            % Create StopButton
            app.StopButton = uibutton(app.UIFigure, 'push');
            app.StopButton.ButtonPushedFcn = createCallbackFcn(app, ...
               @onStopButton, true);
```

```
                app.StopButton.IconAlignment = 'center';
                app.StopButton.Position = [120 10 100 20];
                app.StopButton.Text = 'Stop';

                % Show the figure after all components are created
                app.UIFigure.Visible = 'on';
            end
        end

        % App creation and deletion
        methods (Access = public)

            % Construct app
            function app = depth_view

                % Create UIFigure and components
                createComponents(app)

                % Register the app with App Designer
                registerApp(app, app.UIFigure)

                % Set Startup function - after component creation
                runStartupFcn(app,@StartUpFunc);

                if nargout == 0
                    clear app
                end
            end

            % Code that executes before app deletion
            function delete(app)

                % Delete UIFigure when app is deleted
                delete(app.UIFigure)
            end
        end
    end
```

6.2 Matlab function

```
function waistEstimator_automatic
    close all
    clc

    % Make Pipeline object to manage streaming
    pipe = realsense.pipeline();

    % Create an empty Point Cloud from the RealSense
```

```
    pointcloud = realsense.pointcloud();

    % Start streaming on an arbitrary camera with default settings
    config = realsense.config();
    config.enable_stream(realsense.stream.depth,640,360,...
        realsense.format.z16,30);
    config.enable_stream(realsense.stream.color,640,360,...
        realsense.format.rgb8,30);

    pipe.start(config);

    % transformation
    tform = rigid3d([1 0 0; 0 0 -1; 0 1 0],[0 0 0]); % 0.15

    player = pcplayer([-0.25 0.25], [0.1 0.8], [-0.2 0.2]);
    title = player.Axes.Title;

    figure(2);
    subplot(3,1,1)
    grid on
    ylim([-0.25 0.25])
    li_x = animatedline(gca);

    subplot(3,1,2)
    grid on
    ylim([0.1 0.8])
    li_y = animatedline(gca);

    subplot(3,1,3)
    grid on
    ylim([-45 45])
    li_th = animatedline(gca);

    fs = pipe.wait_for_frames();

    %Depth Frame
    depth = fs.get_depth_frame();

    flag = 0;
    while flag == 0
     if (depth.logical())

            points = pointcloud.calculate(depth);
            vertices = points.get_vertices();
            ptcl = pointCloud(vertices(rem(1:height(vertices),30)==0,:));

            ptcl_out = pctransform(ptcl,tform);
            indices = findPointsInROI(ptcl_out,[-0.25 0.25 0.1 0.8 ...
                -0.15 0.15]);
            ptcl_zone = select(ptcl_out,indices);
```

```
            ptcl_zone.Color = lab2uint8(repmat([128 128 ...
                128],ptcl_zone.Count,1));
            mean_zone = mean(ptcl_zone.Location);
            flag = 1;

            range = [mean_zone(2)-0.1 mean_zone(2)+0.2];

     end
    end

    numFaces = 30;
    [x,y,z] = sphere(numFaces);

    % Main loop
    tic
    for i = 1:10000

        % Obtain frames from a streaming device
        fs = pipe.wait_for_frames();

        % Divide in Depth and Color Fram
        depth = fs.get_depth_frame();

        % Produce pointcloud
        if depth.logical()

            points = pointcloud.calculate(depth);
            vertices = points.get_vertices();
            ptcl = pointCloud(vertices(rem(1:height(vertices),15)==0,:));

            ptcl_out = pctransform(ptcl,tform);
            indices = findPointsInROI(ptcl_out,[-0.25 0.25 range ...
                -0.15 0.15]);
            ptcl_zone = select(ptcl_out,indices);
            ptcl_zone.Color = lab2uint8(repmat([128 128 ...
                128],ptcl_zone.Count,1));
            mean_zone = mean(ptcl_zone.Location);

            indices_base = ...
                findPointsInROI(ptcl_zone,[ptcl_zone.XLimits(1)...
                 ptcl_zone.XLimits(2) range mean_zone(3)+0.02 ...
                    mean_zone(3)+0.07]);
            ptcl_base = select(ptcl_zone,indices_base);

            ptcl_zone_mean = ...
                pointCloud(([x(:),y(:),z(:)]*0.005)+[mean_zone(1) ...
                mean_zone(2) mean_zone(3)]);
            ptcl_zone_mean.Color = lab2uint8(repmat([0 255 ...
                0],ptcl_zone_mean.Count,1));

```

```
        indices_left = ...
            findPointsInROI(ptcl_zone,[ptcl_base.XLimits(1)...
            ptcl_base.XLimits(1)+0.05 range mean_zone(3)+0.02 ...
                mean_zone(3)+0.07]);
        ptcl_left = select(ptcl_zone,indices_left);
        ptcl_left.Color = lab2uint8(repmat([255 0 ...
            0],ptcl_left.Count,1));
        mean_left = mean(ptcl_left.Location);

        indices_right = ...
            findPointsInROI(ptcl_zone,[ptcl_base.XLimits(2)-0.05 ...
            ptcl_base.XLimits(2) range mean_zone(3)+0.02 ...
                mean_zone(3)+0.07]);
        ptcl_right = select(ptcl_zone,indices_right);
        ptcl_right.Color = lab2uint8(repmat([200 0 ...
            200],ptcl_right.Count,1));
        mean_right = mean(ptcl_right.Location);

        mean_lrtotal = (mean_right+mean_left)/2;

        if length(mean_left) == 3 && length(mean_right) == 3
            angle = real(asind((mean_left(2)-mean_right(2))/ ...
                (mean_left(1)-mean_right(1))));

            ptcl_left_mean = ...
                pointCloud(([x(:),y(:),z(:)]*0.005)+[mean_left(1) ...
                mean_left(2) mean_left(3)]);
            ptcl_left_mean.Color = lab2uint8(repmat([255 0 ...
                0],ptcl_left_mean.Count,1));

            ptcl_right_mean = ...
                pointCloud(([x(:),y(:),z(:)]*0.005)+[mean_right(1) ...
                mean_right(2) mean_right(3)]);
            ptcl_right_mean.Color = lab2uint8(repmat([200 0 ...
                200],ptcl_right_mean.Count,1));

            u = (mean_left-mean_right)/norm(mean_left-mean_right);
            line = mean_right + ...
                (0:0.0005:norm(mean_left-mean_right))'*u;

            ptcl_line = pointCloud(line);
            ptcl_line.Color = lab2uint8(repmat([255 255 ...
                0],ptcl_line.Count,1));

            ptcl_lrtotal_mean = ...
                pointCloud(([x(:),y(:),z(:)]*0.005)+...
            [mean_lrtotal(1) mean_lrtotal(2) mean_lrtotal(3)]);
            ptcl_lrtotal_mean.Color = lab2uint8(repmat([255 255 ...
                0],ptcl_lrtotal_mean.Count,1));

```

```
                view(player,pccat([ptcl_zone ptcl_zone_mean ptcl_line ...
                   ptcl_lrtotal_mean ptcl_right ptcl_right_mean ...
                   ptcl_left ptcl_left_mean]));
                title.String = num2str(angle);

                addpoints(li_x,toc,mean_zone(1));
                drawnow limitrate
                subplot(3,1,1)
                subtitle(['X position = ' num2str(mean_zone(1))])
                xlim([toc-10 toc+10])
                ylabel('x [m]')


                addpoints(li_y,toc,mean_zone(2));
                drawnow limitrate
                subplot(3,1,2)
                subtitle(['Y position = ' num2str(mean_zone(2))])
                xlim([toc-10 toc+10])
                ylabel('y [m]')


                addpoints(li_th,toc,angle);
                drawnow limitrate
                subplot(3,1,3)
                subtitle('Angle position')
                subtitle(['Angle position = ' num2str(angle)])
                xlim([toc-10 toc+10])
                xlabel('t [s]')
                ylabel('\theta [deg]')

                range = [mean_zone(2)-0.15 mean_zone(2)+0.15];

            end
        end
    end
end
```

References

[1] Angelos Karatsidis et al. “Predicting kinetics using musculoskeletal modeling and inertial motion capture”. In: *arXiv preprint arXiv:1801.01668* (2018).

[2] Xu Xu et al. “Accuracy of the Microsoft KinectTM for measuring gait parameters during treadmill walking”. In: *Gait Posture* 34 (May 2015). DOI: 10.1016/j.gaitpost.2015.05.002.