INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE BIOTECNOLOGÍA

DEPARTAMENTO DE BIOINGENIRÍA

BIOINSTRUMENTACIÓN IV

PRÁTICA 2: Videografía para análisis de marcha

7MV4

CERVANTES LÓPEZ VICTOR ANDRÉS

CRUZ OLIVARES LUIS ROBERTO

LOZANO GARCÍA EDUARDO ALEJANDRO

PEREZ LÓPEZ NORA FERNANDA

RAMIREZ MIRANDA JUNIOR ISAIAS

DRA. MARIANA FELISA BALLESTEROS

DR. DAVID CRUZ ORTIZ

FECHA DE ASIGNACIÓN: 14/10/2020

FECHA DE ENTREGA: 05/11/2020

INTRODUCCIÓN

La búsqueda interminable acerca del diagnóstico oportuno de patologías o malformaciones que pueden ser congénitas o causadas por otros factores, han hecho que el desarrollo de nuevas técnicas y tecnologías sea una carrera contra el tiempo. Para ello contamos con la biomecánica que como no lo define: Becerra A. (2007) “es la disciplina dedicada al estudio del cuerpo humano, considerando este como una estructura que funciona según las leyes mecánicas de Newton y las leyes de la biología”; por ello nos basamos en esta disciplina para verificar mediante estudios biomecánicos los movimientos naturales del ser humano. Pero sin duda se requiere de un análisis de la marcha humana, y el propósito de ello radica en el análisis de la misma, pero si realizamos un proyecto de esta índole ¿por qué es importante analizar una actividad tan común como lo es el caminar?, definamos primero la marcha, la cual argumentan Cifuentes C., Martínez y Romero E., (2010), que: “La marcha humana se describe como un conjunto de movimientos alternantes y rítmicos de las extremidades inferiores y del tronco, que permite el desplazamiento de cuerpo a través de la acción coordinada de cada uno de los componentes que conforman el sistema locomotor humano”. Este movimiento tan común y fácil de realizar requiere de un gran esfuerzo por parte del organismo representando una de las tantas actividades complejas tanto en movimiento como para su análisis y diagnóstico.

Con las nuevas herramientas tecnológicas se pueden realizar distintos tipos de análisis, para el caso de este reporte se presenta un análisis de marcha con marcadores y un procesamiento por imagen, sin embargo, es importante definir este termino ya que como lo presenta Mehmet K. (2019). “Las pruebas de diagnóstico por la imagen ofrecen una imagen interior del organismo, ya sea de su totalidad o solo de una parte.” Esto es importante ya que la propuesta de esta técnica para análisis de marcha es una técnica por imágenes, pero no se emplean ninguna que emplee rayos X, tomografía o alguna prueba clínica de diagnóstico por imagenología. Una vez mencionado esto se realiza la obtención de los ángulos por las diferentes articulaciones en el cuerpo específicamente de la cadera, rodilla y tobillo, correspondientes al miembro inferior del ser humano y en concreto de ambas extremidades.

La toma de vídeo de un ciclo de marcha normal con marcadores hechos con materiales comunes como lo son papel lleva un preprocesamiento para eliminar el ruido que se pueda

adquirir en la imagen por la toma del vídeo y los factores ambientales que influyen directamente complicando la obtención de cada marcador con el procesamiento normal o filtrado digital. Para corregir esto se propuso la propuesta de valor en este trabajo es utilizar colores para los marcadores haciendo un seguimiento de punto a punto para delimitar una mejor trayectoria del ciclo y sea más perceptible el cambio de ángulos al momento de realizar el movimiento de manera natural, reflejando en este algunas variaciones que pueden ser por un ciclo de marcha afectado por una patología o afectación física lo cual se pondrá a debate con los resultados obtenidos con este procesamiento de video para el análisis de marcha humana.

OBJETIVOS

Objetivo General

- Realizar un análisis de marcha de miembros inferiores por un método de imagen.

Objetivos Particulares

- Analizar la marcha por imagen en el plano sagital.
- Analizar la marcha por imagen en el plano frontal.
- Obtener una animación del ciclo de marcha con los datos obtenidos.
- Obtener la animación tridimensional con interfaz gráfica, que permita seleccionar a cada integrante del equipo.

METODOLOGÍA Y RESULTADOS

- **EXPERIMENTO I: GONIOMETRO POR PROCESAMIENTO DE IMAGEN**

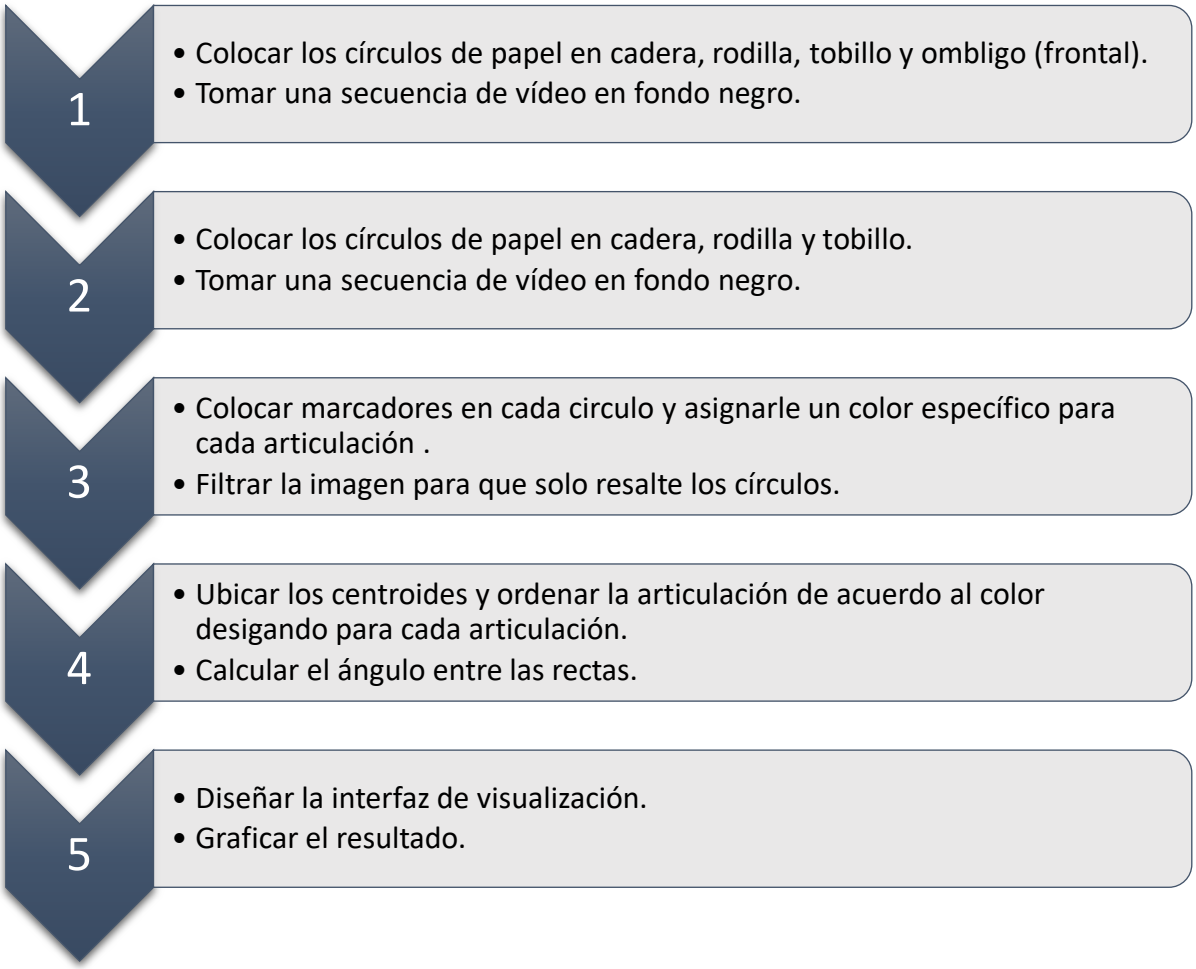


Figura 1. Diagrama con los pasos a seguir para la realización del goniómetro por procesamiento de imágenes.

MATERIALES

- 1 cámara web
- 1 equipo de computo
- Software MATLAB
- Fondo negro
- 12 círculos de papel blanco (1.5cm, 2.5cm, 3.5cm) de radio 3 de cada medida

EXPERIMENTO I Análisis de marcha por imagen en el plano sagital.

Adquisición de imagen

En el caso del plano sagital se tomaron 5 referencias articulares una para cadera (para ambas piernas), para rodilla y tobillo de cada pierna (en la parte externa de una e interna de la otra) como se observa en la Figura 2.

Figura 2. Marcha plano sagital con 5 referencias circulares color blanco colocadas en las articulaciones de interés.

Preprocesamiento

Se llevó acabo un tratamiento previo al procesamiento de imagen con el objetivo de facilitar la obtención de las referencias articulares, así como del centro masa y con ello las trayectorias particulares de cada integrante del equipo.

Dicho tratamiento se realizó mediante el software de edición gráfica Adobe After Effects, en el cual se implantaron *rastreadores de movimiento,* herramienta que permite crear trayectorias en los planos “X” y “Y” e implementarlos en este caso a una máscara de recorte, eliminando de esta forma todo el fondo del video como se observa en la Figura 3.

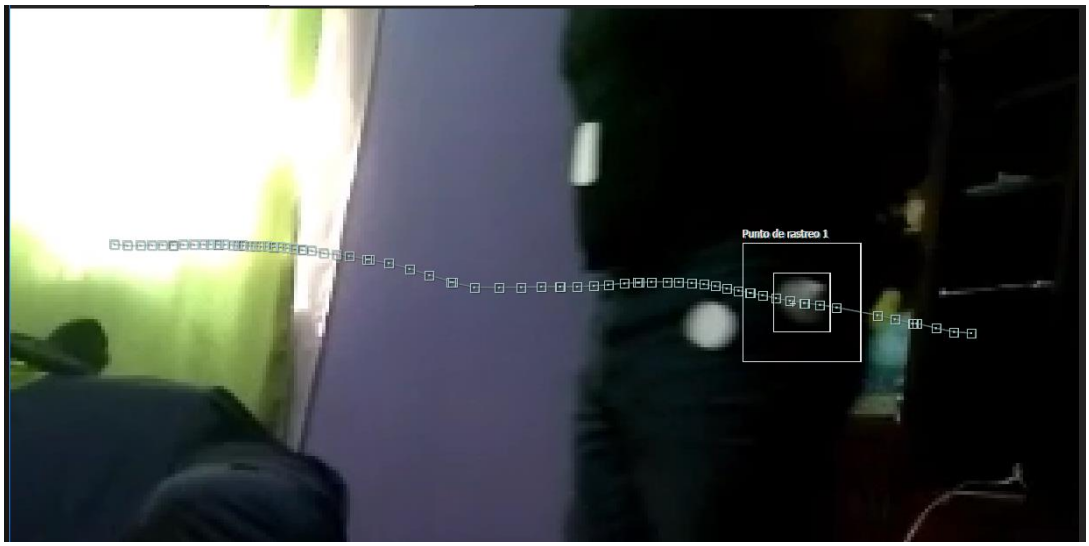


Figura 3. Mascara de recorte con seguimiento de trayectoria de la cadera en Adobe After Effects

Finalmente, para optimizar aún más el procesamiento se coloreo el área de interés articular con blanco, rojo, azul y verde para el centro de masa, cadera, rodilla y tobillo respectivamente como se puede observar en la Figura 2, esto para la implementación de un filtro RGB. Este proceso se llevó a cabo para el plano de interés (sagital) de cada integrante

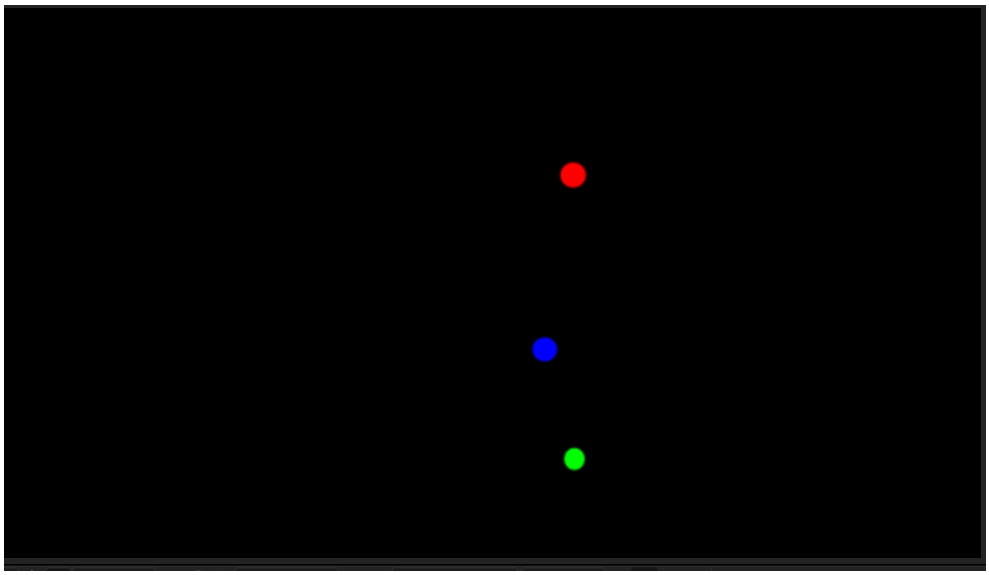


Figura 4. Resultado preprocesamiento de marcha en plano sagital donde se observa la referencia de la articulación de cadera en color rojo, en azul la de rodilla y en verde la del tobillo.

Procesamiento

Para este punto, se propone inicialmente un algoritmo enfocado en la detección de figuras geométrica circulares de diferentes tamaños para la localización de los puntos de interés (cadera rodilla, talón) y posteriormente unir estos puntos para generar de manera animada y en 2D al miembro inferior.

```
clc; clear all; close all;

contador=[];

contadorc=[];

contadorr=[];

contadort=[];

contado=[];

t=[];

conp=[];

conxa=[];

    video=VideoReader('S3.mp4');

    Cuadros=video.NumberofFrames;



for j=1:Cuadros

imagen=read(video,j); % Segmentación del vídeo en imágenes.

gc1=rgb2gray(imagen); %Escala de grises.

u=0.4; %.40-S3  F-0.06; %Umbral para binarización;

salida=imbinarize(gc1,u);

morf=strel('disk',10);

cerra=imclose(salida,morf); %Serie de operaciones morfológicas;

dila=bwareaopen(cerra,30);

Ta=bwlabel(dila);

Taa=label2rgb(Ta);

ce= regionprops(Ta);
```

```
C=cat(1,ce.Centroid);%Localización de centroides;

A=cat(1,ce.Area); %Área obtenida en pixeles;

P=sort(A); % Se acomodan de mayor a menor área de figuras.

Tal=P(1);

Rod=P(2);

Cad=P(3);

r=[C A];

ra=find(r==Cad);

rb=find(r==Rod);

rc=find(r==Tal);

[aa,bb]=ind2sub([3,3],ra);

Cade=r(aa,:);

Cade=Cade(1,1:2);

[cc,dd]=ind2sub([3,3],rb);

Rodi=r(cc,:);

Rodi=Rodi(1,1:2);

[ee,ff]=ind2sub([3,3],rc);

Talo=r(ee,:);

Talo=Talo(1,1:2);

ju=[Cade; Rodi; Talo];

cr=[Cade; Rodi];

xa = ju(:,1);

ya = ju(:,2);

hold on

plot(xa,ya,'m','linewidth',2)

End
```

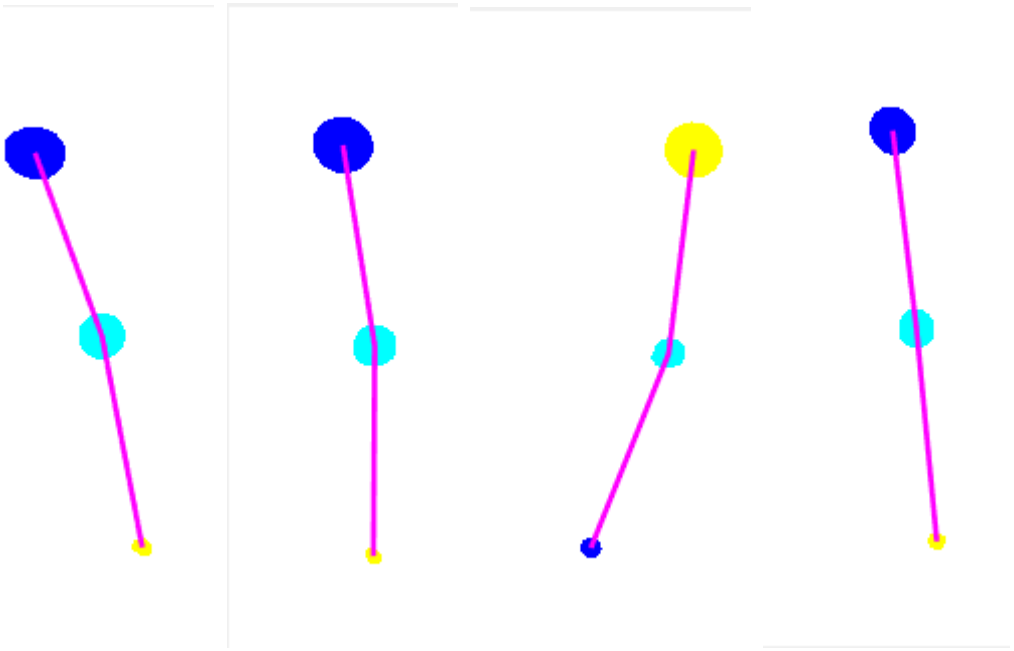


Figura 5. Visualización del código implementado en MATLAB para la obtención del ciclo de la marcha en plano sagital obteniendo la gráfica por cada miembro.

Implementación y mejora del código en Python.

El procesamiento de imagen para este programa se enfoca en la detección de colores, (RGB) para cada articulación, de los miembros inferiores.

```
import numpy as np
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
import cv2
import time
import xlrd       # pip install xlrd
import xlsxwriter # pip install xlscwriter
```

```
### Basic operations ###


def hilight(img,Morph=True,Plot=False):
    img = cv2.cvtColor(img , cv2.COLOR_BGR2RGB)     # Convert from BGR to RGB
    Th = 240
    R,G,B = img[:,:,0],img[:,:,1],img[:,:,2]        # Separate img into RGB
    MR =   1*(R >= Th)                              # Red Mask
    MG =   1*(G >= Th)                              # Green Mask
    MB =   1*(B >= Th)                              # Blue Mask

    M = (MR * MG * MB).astype(np.uint8)              # Get total mask

    ### Apply Morphological operations ###
    kernel = np.ones((1,1),np.uint8)                 # Initialize kernel
    if Morph:
        M = cv2.morphologyEx(M, cv2.MORPH_OPEN, kernel)  # Apply Openning
        M = cv2.morphologyEx(M, cv2.MORPH_CLOSE, kernel) # Apply Closing
    img2 = np.copy(img)                              # Copy image so we can␣
↪modify it

    img2[:,:,0] = img[:,:,0]*M                       # Multiply Red   Channel␣
↪por total Mask
    img2[:,:,1] = img[:,:,1]*M                       # Multiply Green Channel␣
↪por total Mask
    img2[:,:,2] = img[:,:,2]*M                       # Multiply Blue  Channel␣
↪por total Mask
```

```
    if Plot:
        plot_operation(img2,new,title)

    new = cv2.cvtColor(img2, cv2.COLOR_RGB2BGR)  # Convert from BGR to RGB
    return new

def dynamic_treshold(frame,offset,lower):
    S = frame.shape
    N = S[0]
    M = S[1]
    Maximo = -1
    for n in range(0,N):
        for m in range(0,M):
            v = frame[n,m]
            if v > Maximo:
                Maximo = frame[n,m]

    if Maximo < lower:
        th = lower
        print(th)

    else:
        th =  Maximo - offset
        print(th)

    return th
```

```
def hilight_gray(img,Th,Morph=True,Plot = False):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


    M =   1*(img >= Th)                          # Red Mask
    M = M.astype(np.uint8)                       # Get Mask as unsigned intiger

    ### Apply Morphological operations ###
    kernel1 = np.ones((5,5),np.uint8)                # Initialize kernel
    kernel2 = np.ones((7,7),np.uint8)                # Initialize kernel
    if Morph:
        M = cv2.morphologyEx(M, cv2.MORPH_OPEN, kernel1)  # Apply Openning
        M = cv2.morphologyEx(M, cv2.MORPH_CLOSE, kernel2) # Apply Closing

    img2 = img*M                                     # Multiply Red   Channel␣
↪por total Mask

    if Plot:
```

```
        plt.imshow(img2)
    return  img2
```

```
def angle_between_vectors(v1,v2):
    v1_norm = np.linalg.norm(v1)           # compute norm of vector 1
    v2_norm = np.linalg.norm(v2)           # Compute norm of vector 2
    intern = v1.dot(v2)/(v1_norm*v2_norm)  # Divide dot product over␣
↪mulitplication of norms
    angle = np.arccos(intern)              # Find angle with definition of dot␣
↪product
    angle = angle*(180/np.pi)              # Convert radians to degrees
    return angle


def find_centroids(cnts):
    """ Finds centroids of countours:
        cnts = Countours
        How it works: we use  10 and 00 Hue moments, we know the 00 moment is␣
↪the area of the image

    """
    """ Finds centroids"""
    CX   = []                               # List to store x coordinates of␣
↪centroids
    CXo  = []                               # List to store x coordinates of␣
↪centroids in order (from Max area to min area)
    CY   = []                               # List to store y coordinates of␣
↪centroids
```

```
    CYo  = []                                # List to store y coordinates of␣
↪centroids in order (from Max area to min area)
    Area = []                                # List to store Area of contours␣
↪found
    for c in cnts:
    ### compute the center of the contour ##
        M = cv2.moments(c)                   # Compute Hue moments
        cX = int(M["m10"] /( M["m00"]))      # Get cX by definition
        cY = int(M["m01"] / (M["m00"]))      # Get cY by definition
        CX.append(cX)                        # Append x component of centroid
        CY.append(cY)                        # Append y component of centroid
        Area.append(M["m00"])                # Append area of blob
    idx = order_3_max(Area)                  # Obtain index order (from bigest␣
↪to smallest)
    for i in idx:                            # Iterate over idx
        CXo.append(CX[i])                    # append centroids in order (x␣
↪component)
```

```
        CYo.append(CY[i])                    # append centroids in order (y␣
↪component)

    return CXo,CYo

def find_centroids(cnts):
    """ Finds centroids of countours:
        cnts = Countours
        How it works: we use  10 and 00 Hue moments, we know the 00 moment is␣
↪the area of the image

    """
    """ Finds centroids"""
    CX   = []                                # List to store x coordinates of␣
↪centroids
    CY   = []                                # List to store y coordinates of␣
↪centroids
    for c in cnts:
    ### compute the center of the contour ##
        M = cv2.moments(c)                   # Compute Hue moments
        cX = int(M["m10"] /( M["m00"]))      # Get cX by definition
        cY = int(M["m01"] / (M["m00"]))      # Get cY by definition
        CX.append(cX)                        # Append x component of centroid
        CY.append(cY)                        # Append y component of centroid

    return CX,CY
```

```
def digital_goniomertry(video_path,th,scale_percent=80,version=␣
↪"sticky",line="conitnues"):
    global frame
    global positionx
    global width
    global height
    global positiony
    angles = []
    positionx = []
    positiony = []
    capture = cv2.VideoCapture(video_path)  # load video
    kernel = np.ones((6,6),np.uint8)        # Kernel for Openning

    ### stethic stuff ###
    font                   = cv2.FONT_HERSHEY_SIMPLEX
    bottomLeftCornerOfText = (20,40)
    fontScale              = 1
    fontColor              = (255,255,255)
```

```
   lineType               = 2
   ######################

   status = True                           # Variable to iterate over frames
   initialization = True
   CXM = np.array([0,0,0])
   CYM = np.array([0,0,0])
   while status:                           # While loop
       status,frame = capture.read()       # Reads video, returns␣
↪(status,frame)
       if status == True:                  # If there is a frame
           width = int(frame.shape[1] * scale_percent / 100)
           height = int(frame.shape[0] * scale_percent / 100)
           dim = (width, height)
           frame =  cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)
           new = hilight_gray(frame,th)
           new = new.astype(np.uint8)
           cv2.normalize(frame, frame, 0, 255, cv2.NORM_MINMAX)
           contours, hierarchy = cv2.findContours(new,cv2.RETR_TREE,cv2.
↪CHAIN_APPROX_NONE) # Get contours
           try:
               CX,CY = find_centroids(contours)

               angle = round(get_angle(CX,CY),2)
               if version == "sticky":
                   frame = frame*0 + 255
                   fontColor = (0,0,0)
```

```
                elif version == "normal":
                    pass

                ### Draw centroids ###                 #B    G   R
                cv2.circle(frame , (CX[0], CY[0]), 6, (233, 101, 0), -1)
                ######### Compute center of mass  ######
                CMassX = int((CX[1] + CX[0])/2)
                CMassY = int(CY[1]+CY[0]*0.4)
                positionx.append(CMassX)
                positiony.append(CMassY)
                ########################################
                cv2.circle(frame , (CMassX,CMassY), 9, (0, 233, 255), -1)   #␣
↪Centro de masa
                cv2.circle(frame , (CX[1], CY[1]), 6, (89, 71, 255), -1)
                cv2.circle(frame , (CX[2], CY[2]), 6, (20, 157, 15), -1)


                ### Draw lines
```

```
                cv2.line(frame, (CX[0], CY[0]),(CX[1], CY[1]), (219, 236, 50),␣
↪4)
                cv2.line(frame, (CX[0], CY[0]),(CX[2], CY[2]), (219, 236, 50),␣
↪4)

                angles.append(angle)
                if initialization:                # This part of code only runs␣
↪during intialization (First Frame)
                    CXM = CX                      # first element of list of X␣
↪cordiantes is the first centroid recorded
                    CYM = CY                      # first element of list of Y␣
↪cordiantes is the first centroid recorded
                    initialization = False        # Set initalization to False,␣
↪this will never run agian

                elif  initialization == False:    # This part of the code runs␣
↪if initializon was done correctly
                    CX = np.array(CX)             # Create a numpy.array from␣
↪python list
                    CY = np.array(CY)             # Create a numpy.array from␣
↪python list
                    CXM = np.vstack((CXM,CX))     # Stack horizontally the␣
↪values of centroids
                    CYM = np.vstack((CYM,CY))     # Stack horizontally the␣
↪values of centroids

                    ### Draw line of movment of center of mass ###
                    if line == "conitnues":
```

```
                    cv2.line(frame, (positionx[0],␣
↪positiony[0]),(positionx[-1], positiony[-1]), (0, 233, 255), 1)
                elif line == "discrete":
                    for px,py in zip(positionx,positiony):                ␣
↪# iterate over points collected
                        cv2.circle(frame , (px, py), 2, (0, 233, 255), -1)␣
↪# Draw all of them in current frame

            #print(CX,CY)


            ### Draw text ###
            cv2.putText(frame,str(angle), bottomLeftCornerOfText,font,␣
↪fontScale,fontColor,lineType)
            cv2.imshow("Video",frame)


        except:
```

```
            pass
                                    # Display it on Window


    if cv2.waitKey(1) & 0xFF == ord('q'): # If q is pressed
        break                             # Break from while loop

cv2.destroyAllWindows()                   # Destroy window
capture.release()                         # Release video (free memory)

############## Compute movement of center of mass in X and Y axis #########
Dx = 180          # The camara recorded an area that has 180 cm of distance
Dy = 120          # The camare recorded an area that has 120 cm of heigth
CFX = Dx/width    # Convertion from pixel to cm for x axis
CFY = Dy/height   # Convertion from pixel to cm for y axis
Xmovement = (np.max(positionx) - np.min(positionx))*CFX            # Get␣
↪range of x axis
Ymovement = (np.max(positiony) - np.min(positiony))*CFY            # Get␣
↪range of y axis
```

```
    ### Print summarys ###

    print(f"Movement  of center of mass in X: {round(Xmovement,2)} cm ,␣
↪Movement in Y: {round(Ymovement,2)} cm")
    MAX = np.max(angless)
    MIN = np.min(angless)
    print(f"ROM {MIN} - {MAX}")

    return angles,CXM,CYM,
```

```
def color_mask(frame,th,color):
    """ BGR """
    #### Select appropiate chanel for color selection ###
    if color == "blue":
        c =  0
    elif color == "green":
        c = 1
    elif color == "red":
        c =  2

    Mask = 1*(frame[:,:,c] >= th).astype(np.uint8)  # Create mask (only selects␣
↪pixles that are greater than threshold)
    return Mask,c
```

```
def digital_goniomertry(video_path,th,scale_percent=80,version=␣
 ↪"sticky",line="conitnues"):
    global frame
    global positionx
    global width
    global height
    global positiony
    angles_r = []
    angles_c = []
    angles_t = []
    positionx = []
    positiony = []
    capture = cv2.VideoCapture(video_path)  # load video
    ### stethic stuff ###
    font                   =  0
    bottomLeftCornerOfText = (20,20)
    bottomLeftCornerOfText2= (20,40)
    bottomLeftCornerOfText3= (20,60)
    fontScale              = 0.4
    fontColor              = (255,255,255)
    lineType               = 1
```

```
    status = True                               # Variable to iterate over frames
    initialization = True
    CXM = np.array([0,0,0])
    CYM = np.array([0,0,0])
    while status:                               # While loop
        status,frame = capture.read()           # Reads video, returns␣
↪(status,frame)
        if status == True:                      # If there is a frame
            if initialization:
                ### Obtain shapes in order to do scaling ###
                width = int(frame.shape[1]*scale_percent / 100)
                height = int(frame.shape[0]*scale_percent / 100)
                dim = (width, height)

            frame =  cv2.resize(frame, dim, interpolation = cv2.INTER_AREA) #␣
↪Rescale video

            ### Obtain frames by channel (BGR) ###
            M_cadera,rc  =  color_mask(frame,th,color="red")     # Obtain Mask␣
↪for Cadera (Red channel)
            M_rodilla,bc =  color_mask(frame,th,color="blue")    # Obtain Mask␣
↪for Rodilla (Blue channel)
            M_tobillo,gc =  color_mask(frame,th,color="green")   # Obtain Mask␣
↪for tobillo (Green Channel)
```

```
            cadera  =  M_cadera*frame[:,:,rc]                    # Obtain frame␣
↪for cadera
            rodilla =  M_rodilla*frame[:,:,bc]                   # Obtain frame␣
↪for rodilla
            tobillo =  M_tobillo*frame[:,:,gc]                   # Obtain frame␣
↪for tobillo

            ### Obtian the countrors of every image ###
            contours_c, hierarchy = cv2.findContours(cadera,cv2.RETR_TREE,cv2.
↪CHAIN_APPROX_NONE) # Get contours
            contours_r, hierarchy = cv2.findContours(rodilla,cv2.RETR_TREE,cv2.
↪CHAIN_APPROX_NONE) # Get contours
            contours_t, hierarchy = cv2.findContours(tobillo,cv2.RETR_TREE,cv2.
↪CHAIN_APPROX_NONE) # Get contours
            contours=  [contours_c[0],contours_r[0],contours_t[0]]

            ### Obtian Centroids ###
            CX,CY = find_centroids(contours)
            Centroids_cadera  = np.array([CX[0],CY[0]])                  #␣
↪From image
            Centroids_ref_cadera = np.array([CX[0],CY[0]-20])
            Centroids_rodilla = np.array([CX[1],CY[1]])                  #␣
↪From image
            Centroids_ref_rodilla = np.array([CX[2] ,CY[1]-12])
            Centroids_tobillo = np.array([CX[2],CY[2]])                  #␣
↪From image
```

```
            Centroids_masa =    np.array([CX[0],int(CY[0] - .50*CY[0])])   #␣
↪Created from data
            Centroids_feet =    np.array([CX[2],int(CY[2] + .040*CY[2])])   #␣
↪Created from data

            if version == "sticky":
                frame = frame*0 + 255
                fontColor = (0,0,0)

            elif version == "normal":
                pass


            ### Draw centroids ###                                          ␣
↪ #B    G   R
            cv2.circle(frame , (Centroids_cadera[0],  Centroids_cadera[1]),  3,␣
↪(0,   0, 255), -1) # Cadera
```

```
            cv2.circle(frame , (Centroids_rodilla[0], Centroids_rodilla[1]), 3,␣
↪(255, 0,   0), -1) # Rodilla
            cv2.circle(frame , (Centroids_tobillo[0], Centroids_tobillo[1]), 3,␣
↪(0,  255,  0), -1) # tobillo

            ### Mass ceneter (sagital plane) ###
            cv2.circle(frame , (Centroids_masa[0],  Centroids_masa[1]),  4,␣
↪(255,   0, 255), -1)          # Centro de masa
            cv2.circle(frame , (Centroids_feet[0], Centroids_feet[1]),␣
↪3,fontColor, -1)


            ### Draw lines ###
            cv2.line(frame,(Centroids_masa[0],␣
↪Centroids_masa[1]),(Centroids_cadera[0], Centroids_cadera[1]),fontColor, 1)
            cv2.line(frame,(Centroids_cadera[0],␣
↪Centroids_cadera[1]),(Centroids_rodilla[0], Centroids_rodilla[1]),␣
↪fontColor, 1)
```

```
            #### Linea parlalela
            cv2.line(frame,(Centroids_cadera[0],␣
↪Centroids_cadera[1]),(Centroids_ref_cadera[0],Centroids_ref_cadera[1]),␣
↪fontColor, 1)
            cv2.line(frame,(Centroids_rodilla[0],␣
↪Centroids_rodilla[1]),(Centroids_ref_rodilla[0],Centroids_ref_rodilla[1]),␣
↪fontColor,1)
            cv2.line(frame,(Centroids_rodilla[0],␣
↪Centroids_rodilla[1]),(Centroids_tobillo[0], Centroids_tobillo[1]),␣
↪fontColor,1)
            cv2.line(frame,(Centroids_tobillo[0],␣
↪Centroids_tobillo[1]),(Centroids_feet[0], Centroids_feet[1]), fontColor, 1)

            ### Draw movement of center of mass ###
            positionx.append(Centroids_masa[0])
            positiony.append(Centroids_masa[1])
            for px,py in zip(positionx,positiony):            # iterate over␣
↪points collected
                cv2.circle(frame , (px, py), 1,fontColor, -1) # Draw all of␣
↪them in current frame
```

```
    ### Tell user about angles ###
    ################### ANGLE IN HIPPS ####################
    v1 = Centroids_masa    - Centroids_cadera
    v2 = Centroids_rodilla -  Centroids_cadera
    angle_cadera = angle_between_vectors(v1,v2)
```

```
            angles_c.append(angle_cadera)
            msg_cadera = f"Hipp angle: {round(angle_cadera,2) }"

            ################### ANGLE IN Knee ####################

            v1 = Centroids_cadera  -  Centroids_rodilla
            v2 = Centroids_tobillo -  Centroids_rodilla
            angle_rodilla =180 - angle_between_vectors(v1,v2)
            angles_r.append(angle_rodilla)
            msg_rodilla = f"Knee angle: {round(angle_rodilla,2) }"

            ################### ANGLE IN Ankle ####################

            v1 = Centroids_rodilla - Centroids_tobillo
            v2 = Centroids_feet    - Centroids_tobillo
            angle_tobillo = angle_between_vectors(v1,v2)
            angles_t.append(angle_tobillo)
            msg_tobillo = f"Ankle angle: {round(angle_tobillo,2) }"
```

```
            ###########################################################
            time.sleep(0.9)
            cv2.putText(frame,str(msg_rodilla), bottomLeftCornerOfText,font,␣
↪fontScale,fontColor,lineType)
            cv2.putText(frame,str(msg_cadera),  bottomLeftCornerOfText2,font,␣
↪fontScale,fontColor,lineType)
            cv2.putText(frame,str(msg_tobillo ),bottomLeftCornerOfText3,font,␣
↪fontScale,fontColor,lineType)
            cv2.imshow("Video",frame)


            if  initialization:
                 initialization  = False


        if cv2.waitKey(1) & 0xFF == ord('q'): # If q is pressed
            break                             # Break from while loop

    cv2.destroyAllWindows()                   # Destroy window
    capture.release()                         # Release video (free memory)
```

Se declaran las variables correspondientes al rango articular máximo y mínimo, para que pueda visualizarse al momento de correr la animación.

```
    ### MESSAGE FOR USERS
    MAX = round(np.max(angles_c),2)
    MIN = round(np.min(angles_c),2)
    print(f"Hip ROM {MIN} - {MAX}")
    MAX = round(np.max(angles_r),2)
    MIN = round(np.min(angles_r),2)
    print(f"Knee ROM {MIN} - {MAX}")
    MAX = round(np.max(angles_t),2)
```

Se importan los vectores que se usaran como referencia en el análisis de la marcha, cabe resaltar que estos vectores describen los rangos articulares para una marcha normal.

```
    MIN = round(np.min(angles_t),2)
    print(f"Ankle ROM {MIN} - {MAX}")

    return angles_c,angles_r,angles_t
```

```
def plot_real_data_sagital():
    ###### REAL DATA ####
    a = np.array([19.33,18.92,18.45,17.94,17.3,16.4,15.18,13.67,11.97,10.21,8.
↪48,6.74,4.94,3.13,1.42,-0.13,-1.54,-2.87,-4.12,-5.3,-6.4,-7.43,-8.39,-9.
↪27,-10.02,
    -10.61,-10.95,-10.91,-10.31,-9,-6.95,-4.25,-1.05,2.42,5.93,9.22,12.11,14.
↪55,16.53,18.13,19.45,20.54,21.38,21.84,21.87,21.5,20.84,20.09,19.5,19.18,19.
↪01,])


    c = np.array([3.97,7,10.59,14.12,17.38,19.84,21.27,21.67,21.22,20.2,18.
↪86,17.35,15.73,14.08,12.5,11.09,9.91,8.97,8.28,
    7.86,7.72,7.94,8.6,9.76,11.5,13.86,16.97,20.96,26,32.03,38.74,45.6,52.05,57.
↪54,61.66,64.12,64.86,63.95,61.59,57.97,
    53.27,47.58,40.94,33.46,25.38,17.27,9.94,4.31,1.12,0.54,2.21
    ])
    b = np.array([38.12,44,49.78,55.9,60.43,64.34,67.9,74,78,82,84,82.65,78,70.
↪09,64.78,58.98,52.34,46.94,40.62,34.98,28.76,22.12,
    16.97,14,13.45,15.9,19.14,21.79,24.
↪05,28,30,30,30,30,30,28,26,24,22,19,15,13,12,11,11,12,15,19,22,27,30])
```

```
   d = np.array([4.19,4.58,4.79,4.98,5.11,5.12,5.17,5.34,5.61,5.85,5.95,5.85,5.
↪57,5.18,4.84,4.69,4.64,4.66,4.74,4.86,4.95,4.98,4.97,
   4.96,4.97,5.05,5.22,5.49,5.86,6.2,6.3,6.05,5.53,4.99,4.75,4.93,5.41,5.99,6.
↪51,6.93,7.22,7.42,7.55,7.54,7.29,6.69,5.73,4.54,3.55,3.28,3.6,])

   d = np.array([0.02,-2.06,-3.88,-4.6,-3.98,-2.4,-0.45,1.45,3.04,4.27,5.13,5.
↪71,6.1,6.43,6.76,7.12,7.54,7.99,8.44,8.86,9.23,9.51,9.62,
   9.43,8.7,7.2,4.69,1.15,-3.26,-8.17,-13.05,-17.13,-19.52,-19.77,-18.12,-15.
↪29,-12.04-8.85,-5.96,-3.51,-1.64,
   -0.5,-0.07,-0.16,-0.42,-0.52,-0.26,0.36,1,1.2,0.58])


   e = np.array([0.02,-2.06,-3.88,-4.6,-3.98,-2.4,-0.45,1.45,3.04,4.27,5.13,5.
↪71,6.1,6.43,6.76,7.12,7.54,7.99,8.44,8.86,9.23,9.51,
   9.62,9.43,8.7,7.2,4.69,1.15,-3.26,-8.17,-13.05,-17.13,-19.52,-19.77,-18.
↪12,-15.29,-12.04,-8.85,-5.96,-3.51,-1.64,-0.5,-0.07,-0.16,
   -0.42,-0.52,-0.26,0.36,1,1.2,0.58,])
```

```
    figure, ax = plt.subplots(1,5,figsize=(20,4))
    ax[0].plot(np.array(a))
    ax[0].set_title("Cadera")
    ax[0].grid("on")
    ax[1].plot(c)
    ax[1].set_title("Rodilla Derecha")
    ax[1].grid("on")
    ax[2].plot(b)
    ax[2].set_title("Rodilla Izquierda")
    ax[2].grid("on")
    ax[3].plot(d)
    ax[3].set_title("Pie Derecho")

    ax[3].grid("on")
    ax[4].plot(e)
    ax[4].set_title("Pie izquierdo")
    ax[4].grid("on")
```

```
def plot_real_data_frontal():
    a = np.array([0,-3.3,-5,-4.05,-1.9,0,0.6,5.1,7,6.45,3.5,1,0.5])
    b = np.array([0,3,5,4,4,3,3,3,3,3,5,5,4,6,7,9,12,11,3,3,3])
    c = np.array([5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,3,5,3,1,1,1,1,1,2,5,3])
    figure, ax = plt.subplots(1,5,figsize=(20,4))
    ax[0].plot(np.array(a))
    ax[0].set_title("Cadera")
    ax[0].grid("on")
    ax[1].plot(b)
    ax[1].set_title("Rodilla Derecha")
    ax[1].grid("on")
    ax[2].plot(b)
    ax[2].set_title("Rodilla Izquierda")
    ax[2].grid("on")
    ax[3].plot(c)
    ax[3].set_title("Pie Derecho")

    ax[3].grid("on")
    ax[4].plot(c)
    ax[4].set_title("Pie izquierdo")
    ax[4].grid("on")
```

```
def digital_goniomertry(video_path,video_path2,th,scale_percent=80,version=␣
 ↪"sticky",line="conitnues"):
    global frame
    global height
    global positionx
    global positionyf
    ### X POSITION ##
```

```
global k_rx
global h_rx
global a_rx
global k_lx
global h_lx
global a_lx
## Y position ##
global k_ry
global h_ry
global a_ry
global k_ly
global h_ly
global a_ly


## ANGLES ##
angles_r  = []  # Store trayectories of angles for right knee
angles_c  = []  # Store trayectories of angles for right Hip
angles_t  = []  # Store trayectories of angles for right  Ankle
angles_r2 = []  # Store trayectories of angles for left knee
angles_c2 = []  # Store trayectories of angles for left Hip
angles_t2 = []  # Store trayectories of angles for left  Ankle
```

```
# X POSITION ##
k_rx  = []  # Store trayectories of angles for right knee
h_rx  = []  # Store trayectories of angles for right Hip
a_rx  = []  # Store trayectories of angles for right  Ankle
k_lx  = []  # Store trayectories of angles for left knee
h_lx  = []  # Store trayectories of angles for left Hip
a_lx  = []  # Store trayectories of angles for left  Ankle

## Y position ##
k_ry  = []  # Store trayectories of angles for right knee
h_ry  = []  # Store trayectories of angles for right Hip
a_ry  = []  # Store trayectories of angles for right  Ankle
k_ly  = []  # Store trayectories of angles for left knee
h_ly  = []  # Store trayectories of angles for left Hip
a_ly  = []  # Store trayectories of angles for left  Ankle

## SPECIAL (FOR CENTER OF MASS) ##
positionx = []  # Store x trayectories o center of mass
positiony = []  # Store y trayectories o center of mass
positionyf = []
### stethic stuff ###
font                   =  0
bottomLeftCornerOfText  = (20,20)
bottomLeftCornerOfText2 = (20,40)
```

```
bottomLeftCornerOfText3 = (20,60)
bottomRightCornerOfText = (250,20)
bottomRightCornerOfText2= (250,40)
bottomRightCornerOfText3= (250,60)
fontScale               = 0.4
fontColor               = (255,255,255)
lineType                = 1

#####################
status = True                           # Variable to iterate over frames
initialization = True
capture  = cv2.VideoCapture(video_path)  # load video for right
capture2 = cv2.VideoCapture(video_path2) # Load video for left
while status:                           # While loop
    status,frame = capture.read()       # Reads video, returns␣
(status,frame) right
    status,frame2 = capture2.read()       # Reads video,returns␣
(status,frame) left
    if status == True:                  # If there is a frame
        if initialization:
            ### Obtain shapes in order to do scaling ###
            width = int(frame.shape[1]*scale_percent / 100)
            height = int(frame.shape[0]*scale_percent / 100)
            dim = (width, height)

        frame =  cv2.resize(frame, dim, interpolation = cv2.INTER_AREA) #␣
↪Rescale video
        frame2=  cv2.resize(frame2, dim, interpolation = cv2.INTER_AREA) #␣
↪Rescale
        ### Obtain frames by channel (BGR) Right  ###
        M_cadera,rc  =  color_mask(frame,th,color="red")     # Obtain Mask␣
↪for Cadera (Red channel)
        M_rodilla,bc =  color_mask(frame,th,color="blue")    # Obtain Mask␣
↪for Rodilla (Blue channel)
        M_tobillo,gc =  color_mask(frame,th,color="green")   # Obtain Mask␣
↪for tobillo (Green Channel)
        cadera  =  M_cadera*frame[:,:,rc]                    # Obtain frame␣
↪for cadera
        rodilla =  M_rodilla*frame[:,:,bc]                   # Obtain frame␣
↪for rodilla
        tobillo =  M_tobillo*frame[:,:,gc]                   # Obtain frame␣
↪for tobillo

        ### Obtain frames by channel (BGR) Leftt  ###
        M_cadera2,rc2  =  color_mask(frame2,th,color="red")     # Obtain␣
↪Mask for Cadera (Red channel)
```

```
            M_rodilla2,bc2 =  color_mask(frame2,th,color="blue")    # Obtain␣
↪Mask for Rodilla (Blue channel)
            M_tobillo2,gc2 =  color_mask(frame2,th,color="green")   # Obtain␣
↪Mask for tobillo (Green Channel)
            cadera2  =  M_cadera2*frame2[:,:,rc2]                    # Obtain␣
↪frame for cadera
            rodilla2 =  M_rodilla2*frame2[:,:,bc2]                   # Obtain␣
↪frame for rodilla
            tobillo2 =  M_tobillo2*frame2[:,:,gc2]                   # Obtain␣
↪frame for tobillo

            ### Obtian the countrors of every image  Right ###
            contours_c, hierarchy = cv2.findContours(cadera,cv2.RETR_TREE,cv2.
↪CHAIN_APPROX_NONE) # Get contours
            contours_r, hierarchy = cv2.findContours(rodilla,cv2.RETR_TREE,cv2.
↪CHAIN_APPROX_NONE) # Get contours
            contours_t, hierarchy = cv2.findContours(tobillo,cv2.RETR_TREE,cv2.
↪CHAIN_APPROX_NONE) # Get contours
            contours=  [contours_c[0],contours_r[0],contours_t[0]]
```

```
            ### Obtian the countrors of every image  Left ###
            contours_c2, hierarchy = cv2.findContours(cadera2,cv2.RETR_TREE,cv2.
↪CHAIN_APPROX_NONE) # Get contours
            contours_r2, hierarchy = cv2.findContours(rodilla2,cv2.
↪RETR_TREE,cv2.CHAIN_APPROX_NONE) # Get contours
            contours_t2, hierarchy = cv2.findContours(tobillo2,cv2.
↪RETR_TREE,cv2.CHAIN_APPROX_NONE) # Get contours
            contours2=  [contours_c2[0],contours_r2[0],contours_t2[0]]

            ### Obtian Centroids Right  ###
            CX,CY = find_centroids(contours)
            Centroids_cadera  = np.array([CX[0],CY[0]])                  #␣
↪From image
            Centroids_rodilla = np.array([CX[1],CY[1]])                  #␣
↪From image
            Centroids_tobillo = np.array([CX[2],CY[2]])                  #␣
↪From image
            ### Obtain cetnroids using anatomic truth ###
            Centroids_masa =    np.array([CX[0],int(CY[0] - .40*CY[0])])   #␣
↪Created from data
            Centroids_feet =    np.array([CX[2],int(CY[2] + .040*CY[2])])   #␣
↪Created from data



            ### Obtian Centroids Left ###
```

```
            CX2,CY2 = find_centroids(contours2)
            Centroids_cadera2  = np.array([CX2[0],CY2[0]])                    #␣
↪From image
            Centroids_rodilla2 = np.array([CX2[1],CY2[1]])                    #␣
↪From image
            Centroids_tobillo2 = np.array([CX2[2],CY2[2]])                    #␣
↪From image
            ### Obtain cetnroids using anatomic truth ###
            Centroids_feet2 =    np.array([CX2[2],int(CY2[2] + .040*CY2[2])])  ␣
↪# Created from data


            ### append centroid coordiantes for 3d plot ###
            ### X POSITION ###
            k_rx.append(Centroids_rodilla[0])  # Store trayectories of angles␣
↪for right knee
            h_rx.append(Centroids_cadera[0])   # Store trayectories of angles␣
↪for right Hip
            a_rx.append(Centroids_tobillo[0])  # Store trayectories of angles␣
↪for right  Ankle
            k_lx.append(Centroids_rodilla2[0])   # Store trayectories of angles␣
↪for left knee
            h_lx.append(Centroids_cadera2[0])   # Store trayectories of angles␣
↪for left Hip
            a_lx.append(Centroids_tobillo2[0])   # Store trayectories of angles␣
↪for left  Ankle
```

```
            ### Y position ###
            k_ry.append(-1*Centroids_rodilla[1] + height) # Store trayectories␣
↪of angles for right knee
            h_ry.append(-1*Centroids_cadera[1]  + height) # Store trayectories␣
↪of angles for right Hip
            a_ry.append(-1*Centroids_tobillo[1] + height) # Store trayectories␣
↪of angles for right  Ankle
            k_ly.append(-1*Centroids_rodilla2[1]+ height) # Store trayectories␣
↪of angles for left knee
            h_ly.append(-1*Centroids_cadera2[1] + height) # Store trayectories␣
↪of angles for left Hip
            a_ly.append(-1*Centroids_tobillo2[1]+ height) # Store trayectories␣
↪of angles for left  Ankle


            if version == "sticky":
                frame = frame*0 + 255
                frame2 = frame2*0 + 255
                fontColor = (0,0,0)
```

```
            elif version == "normal":
                pass


            ### Draw centroids Ritght ###                                    ␣
↪#B    G   R
            cv2.circle(frame , (Centroids_cadera[0],  Centroids_cadera[1]),  3,␣
↪(0,   0, 255), -1) # Cadera
            cv2.circle(frame , (Centroids_rodilla[0], Centroids_rodilla[1]), 3,␣
↪(255, 0,   0), -1) # Rodilla
            cv2.circle(frame , (Centroids_tobillo[0], Centroids_tobillo[1]), 3,␣
↪(0,  255,  0), -1) # tobillo
            cv2.circle(frame , (Centroids_feet[0], Centroids_feet[1]),␣
↪3,fontColor, -1)
            ### Draw centroids Left ###                                      ␣
↪   #B    G   R
            cv2.circle(frame , (Centroids_cadera2[0],  Centroids_cadera2[1]), ␣
↪3, (0,   0, 255), -1) # Cadera
            cv2.circle(frame , (Centroids_rodilla2[0], Centroids_rodilla2[1]),␣
↪3, (255, 0,   0), -1) # Rodilla
            cv2.circle(frame , (Centroids_tobillo2[0], Centroids_tobillo2[1]),␣
↪3, (0,  255,  0), -1) # tobillo
            cv2.circle(frame , (Centroids_feet2[0], Centroids_feet2[1]),␣
↪3,fontColor, -1)
```

```
            ### Mass ceneter (sagital plane) ###
            cv2.circle(frame , (Centroids_masa[0],  Centroids_masa[1]),  4,␣
↪(255,   0, 255), -1)         # Centro de masa



            ### Draw lines  Right ###
            cv2.line(frame,(Centroids_masa[0],␣
↪Centroids_masa[1]),(Centroids_cadera[0], Centroids_cadera[1]),fontColor, 1)
            cv2.line(frame,(Centroids_cadera[0],␣
↪Centroids_cadera[1]),(Centroids_rodilla[0], Centroids_rodilla[1]),␣
↪fontColor, 1)
            cv2.line(frame,(Centroids_rodilla[0],␣
↪Centroids_rodilla[1]),(Centroids_tobillo[0], Centroids_tobillo[1]),␣
↪fontColor,1)
            cv2.line(frame,(Centroids_tobillo[0],␣
↪Centroids_tobillo[1]),(Centroids_feet[0], Centroids_feet[1]), fontColor, 1)

            ### Draw lines  Left ###
```

```
            cv2.line(frame,(Centroids_masa[0],␣
↪Centroids_masa[1]),(Centroids_cadera2[0], Centroids_cadera2[1]),fontColor, 1)
            cv2.line(frame,(Centroids_cadera2[0],␣
↪Centroids_cadera2[1]),(Centroids_rodilla2[0], Centroids_rodilla2[1]),␣
↪fontColor, 1)
            cv2.line(frame,(Centroids_rodilla2[0],␣
↪Centroids_rodilla2[1]),(Centroids_tobillo2[0], Centroids_tobillo2[1]),␣
↪fontColor,1)
            cv2.line(frame,(Centroids_tobillo2[0],␣
↪Centroids_tobillo2[1]),(Centroids_feet2[0], Centroids_feet2[1]), fontColor,␣
↪1)


            ### Draw movement of center of mass ###
            positionx.append(Centroids_masa[0])
            positiony.append(Centroids_masa[1])
            positionyf.append(-1*Centroids_masa[1]+height)
            for px,py in zip(positionx,positiony):            # iterate over␣
↪points collected
                cv2.circle(frame , (px, py), 1,fontColor, -1) # Draw all of␣
↪them in current frame
```

```
        ### Tell user about angles  RIght ###
        ################### ANGLE IN HIPPS ####################
        v1 = Centroids_masa    - Centroids_cadera
        v2 = Centroids_rodilla -  Centroids_cadera
        s=-1
        if Centroids_rodilla[0] > Centroids_cadera[0]:
            s=1
        angle_cadera = s*(180 -  angle_between_vectors(v1,v2))
        angles_c.append(angle_cadera)
        msg_cadera = f"Right Hipp angle: {round(angle_cadera,2) }"

        ################### ANGLE IN Knee ####################
        v1 = Centroids_tobillo - Centroids_rodilla
        v2 = Centroids_cadera  - Centroids_rodilla
        angle_rodilla = 180 - angle_between_vectors(v1,v2)
        angles_r.append(angle_rodilla)
        msg_rodilla = f"Right Knee angle: {round(angle_rodilla,2) }"

        ################### ANGLE IN Ankle ####################
        v1 = Centroids_rodilla - Centroids_tobillo
        v2 = Centroids_feet    - Centroids_tobillo
        s=1
```

```
    if Centroids_rodilla2[0] > Centroids_tobillo2[0]:
        s=-1
    angle_tobillo =s*(180 -  angle_between_vectors(v1,v2))
    angles_t.append(angle_tobillo)
    msg_tobillo = f"Right Ankle angle: {round(angle_tobillo,2) }"
    ##########################################################

    ### Tell user about angles  left ###
    ################## ANGLE IN HIPPS ###################
    v1 = Centroids_masa    - Centroids_cadera2
    v2 = Centroids_rodilla2 -  Centroids_cadera2
    angle_cadera2 =180 - angle_between_vectors(v1,v2)
    angles_c2.append(angle_cadera2)
    msg_cadera2 = f"Left Hipp angle: {round(angle_cadera2,2) }"

    ################## ANGLE IN Knee ###################
    v1 = Centroids_tobillo2 - Centroids_rodilla2
    v2 = Centroids_cadera2  - Centroids_rodilla2
    angle_rodilla2 =180 - angle_between_vectors(v1,v2)
    angles_r2.append(angle_rodilla2)
    msg_rodilla2 = f"Left Knee angle: {round(angle_rodilla2,2) }"
```

```
        ################## ANGLE IN Ankle ###################
        v1 = Centroids_rodilla2 - Centroids_tobillo2
        v2 = Centroids_feet2    - Centroids_tobillo2
        s=-1
        if Centroids_rodilla2[0] > Centroids_tobillo2[0]:
            s=1
        angle_tobillo2 =s*(180 - angle_between_vectors(v1,v2))
        angles_t2.append(angle_tobillo2)
        msg_tobillo2 = f"Left Ankle angle: {round(angle_tobillo2,2) }"
        ##########################################################

        time.sleep(0.2)
        cv2.putText(frame,str(msg_rodilla), bottomLeftCornerOfText,font,␣
→fontScale,fontColor,lineType)
        cv2.putText(frame,str(msg_cadera),  bottomLeftCornerOfText2,font,␣
→fontScale,fontColor,lineType)
        cv2.putText(frame,str(msg_tobillo ),bottomLeftCornerOfText3,font,␣
→fontScale,fontColor,lineType)
        cv2.putText(frame,str(msg_rodilla2), bottomRightCornerOfText,font,␣
→fontScale,fontColor,lineType)
        cv2.putText(frame,str(msg_cadera2),  bottomRightCornerOfText2,font,␣
→fontScale,fontColor,lineType)
        cv2.putText(frame,str(msg_tobillo2),bottomRightCornerOfText3,font,␣
→fontScale,fontColor,lineType)
```

```
            cv2.imshow("Video",frame)



            if  initialization:
                 initialization  = False # change state of initialization (only␣
↪happens for first frame )



    #### This section of the code runs once we have porcessed all the video ␣
↪####
        if cv2.waitKey(1) & 0xFF == ord('q'): # If q is pressed
            break                             # Break from while loop

    cv2.destroyAllWindows()                   # Destroy window
    capture.release()                         # Release video (free memory)
```

```
### MESSAGE FOR USERS RIght
MAX = round(np.max(angles_c),2)
MIN = round(np.min(angles_c),2)
print(f"Right Hip ROM:  {MIN} - {MAX}")
MAX = round(np.max(angles_r),2)
MIN = round(np.min(angles_r),2)
print(f"Right Knee ROM:  {MIN} - {MAX}")
MAX = round(np.max(angles_t),2)
MIN = round(np.min(angles_t),2)
print(f"Right Ankle ROM:  {MIN} - {MAX}")
print()
### MESSAGE FOR USERS LEft
MAX = round(np.max(angles_c2),2)
MIN = round(np.min(angles_c2),2)
print(f"Left Hip ROM:  {MIN} - {MAX}")
MAX = round(np.max(angles_r2),2)
MIN = round(np.min(angles_r2),2)
print(f"Left Knee ROM:  {MIN} - {MAX}")
MAX = round(np.max(angles_t2),2)
MIN = round(np.min(angles_t2),2)
print(f"Left Ankle ROM:  {MIN} - {MAX}")
print()
MAX = round(np.max(positionx),2)
MIN = round(np.min(positionx),2)
displacement_of_mass = (MAX-MIN)*(132/width)
print(f"Center of mass x movement:  {displacement_of_mass} cm ")

return angles_c,angles_r,angles_t,angles_c2,angles_r2,angles_t2
```

```
def digital_goniomertry2(video_path,video_path2,th,scale_percent=80,version=␣
↪"sticky",line="conitnues"):
    global frame
    global positionx2
    global positiony2
    ### Z POSITION ##
    global k_rz
    global h_rz
    global a_rz
    global k_lz
    global h_lz
    global a_lz

    ## ANGLES ##
    angles_r  = []  # Store trayectories of angles for right knee
    angles_c  = []  # Store trayectories of angles for right Hip
    angles_t  = []  # Store trayectories of angles for right  Ankle
    angles_r2 = []  # Store trayectories of angles for left knee
    angles_c2 = []  # Store trayectories of angles for left Hip
    angles_t2 = []  # Store trayectories of angles for left  Ankle
    positionx2 = []  # Store x trayectories o center of mass
    positiony2 = []  # Store y trayectories o center of mass
```

```
  # Z POSITION ##
  k_rz  = []  # Store trayectories of angles for right knee
  h_rz  = []  # Store trayectories of angles for right Hip
  a_rz  = []  # Store trayectories of angles for right  Ankle
  k_lz  = []  # Store trayectories of angles for left knee
  h_lz  = []  # Store trayectories of angles for left Hip
  a_lz  = []  # Store trayectories of angles for left  Ankle

  ### stethic stuff ###
  font                   =  0
  bottomLeftCornerOfText  = (20,20)
  bottomLeftCornerOfText2 = (20,40)
  bottomLeftCornerOfText3 = (20,60)
  bottomRightCornerOfText = (250,20)
  bottomRightCornerOfText2= (250,40)
  bottomRightCornerOfText3= (250,60)
  fontScale              = 0.4
  fontColor              = (255,255,255)
  lineType               = 1
  ######################
  status = True                           # Variable to iterate over frames
  initialization = True
  capture  = cv2.VideoCapture(video_path)  # load video for right
  capture2 = cv2.VideoCapture(video_path2) # Load video for left
```

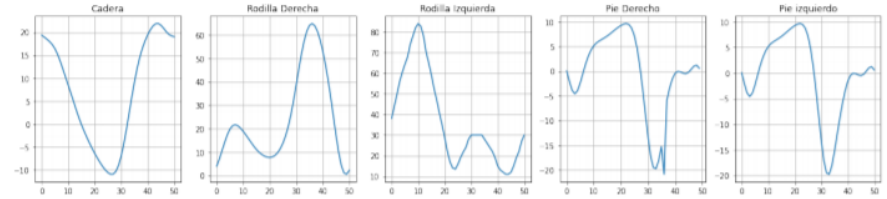
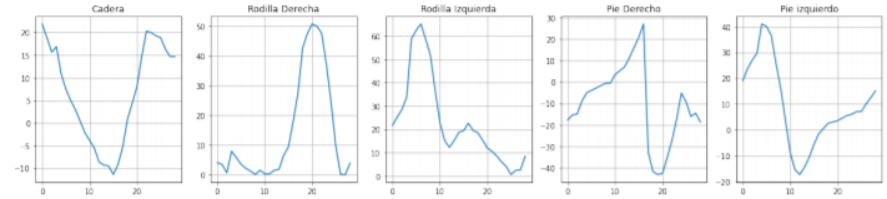
Se desarrolla el orden de las gráficas de referencia, para poder ser comparadas con los datos de cada miembro del equipo en el plano sagital.

Plano sagital						
ROM						
Sujeto	Derecho			Izquierdo		
	Cadera	Rodilla	Tobillo	Cadera	Rodilla	Tobillo
Luis	-11.4 a 21.88	0.1 a 50.74	-43.18 a -27.08	0.0 a 35.64	0.61 a 65.17	-17.32 a 41.03
Alejandro	-18.9 a 16.81	0.07 a 40.44	-44.23 a 28.65	0.0 a 19.05	2.66 a 49.09	-23.91 a 50.27
Junior	-32.39 a 6.2	0.1 a 34.93	-40.5 a 39.56	0.6 a 20.08	0.08 a 45.43	-40.45 a 18.43
Nora	-13.93 a 16.78	0.02 a 34.72	-45.39 a 28.71	0.89 a 79.69	0.15 a 3.14	-5.38 a 5.38
Víctor	-14.55 a 13.15	0.02 a 43.26	-43.71 a 22.97	nan a nan	7.94 a 48.13	-14.04 a 43.51

Figura 6. Rango articular del miembro inferior de cada integrante del equipo obtenido plano sagital

Movimiento del centro de marcha eje x	
Sujeto	Plano Sagital (cm)
Luis	58.05555556
Alejandro	54.3125
Junior	42.98148148
Nora	49.328125
Víctor	50.94375

Figura 7. Movimiento de COM en el eje x de cada integrante del equipo plano sagital

Plano Sagital comparación					
	Cadera	Rodilla D	Rodilla I	Pie D	Pie I
Bibliográfico					
Luis					

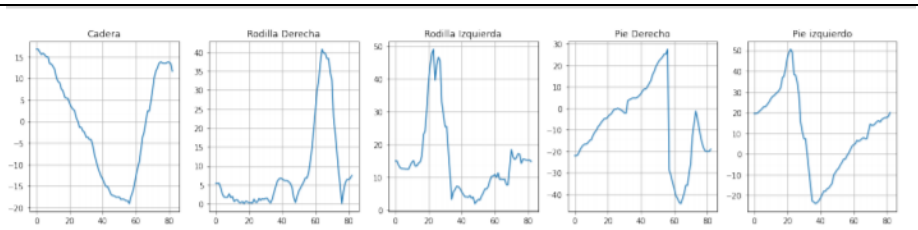
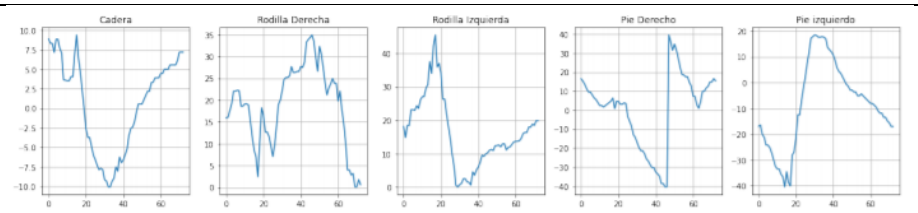
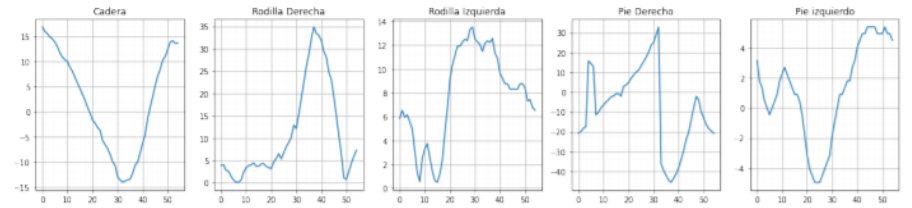
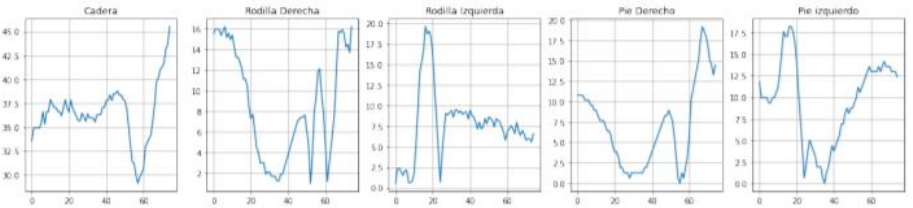
Alejandro	
Junior	
Nora	
Víctor	

Figura 8. Comparación de los datos obtenidos por integrante a la bibliografía plano sagital

EXPERIMENTO II Análisis de marcha por imagen en el plano frontal.

Adquisición de imagen

Para el plano frontal se colocaron 7 referencias (3 para cadera, rodilla y tobillo de cada pierna y una para el centro de masa del cuerpo). Como se observa en la Figura 9.

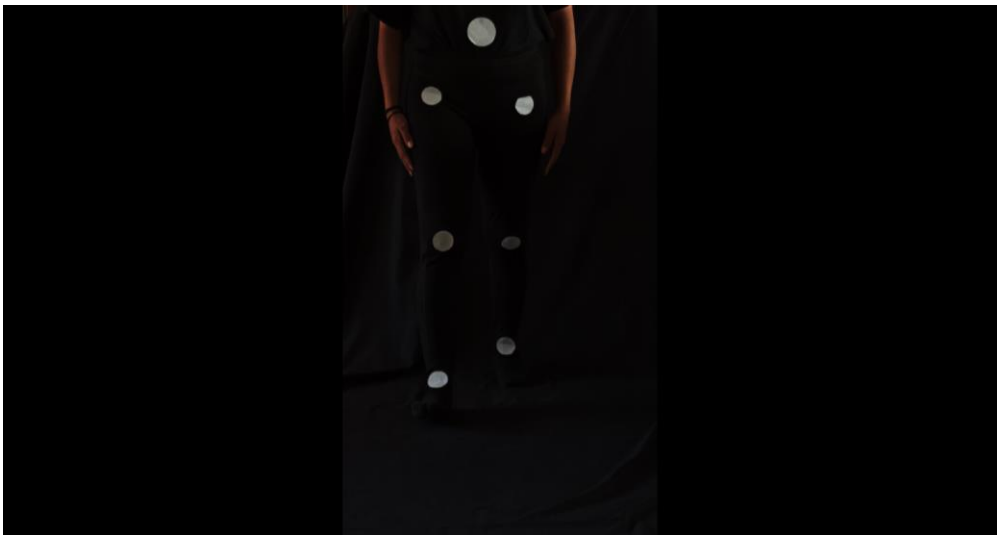


Figura 9. Marcha plano frontal

Preprocesamiento

Se llevó acabo un tratamiento previo al procesamiento de imagen con el objetivo de facilitar la obtención de las referencias articulares, así como del centro masa y con ello las trayectorias particulares de cada integrante del equipo.

Dicho tratamiento se realizó mediante el software de edición gráfica Adobe After Effects, en el cual se implantaron *rastreadores de movimiento,* herramienta que permite crear trayectorias en los planos “X” y “Y” e implementarlos en este caso a una máscara de recorte, eliminando de esta forma todo el fondo del video como se observa en la Figura 10.

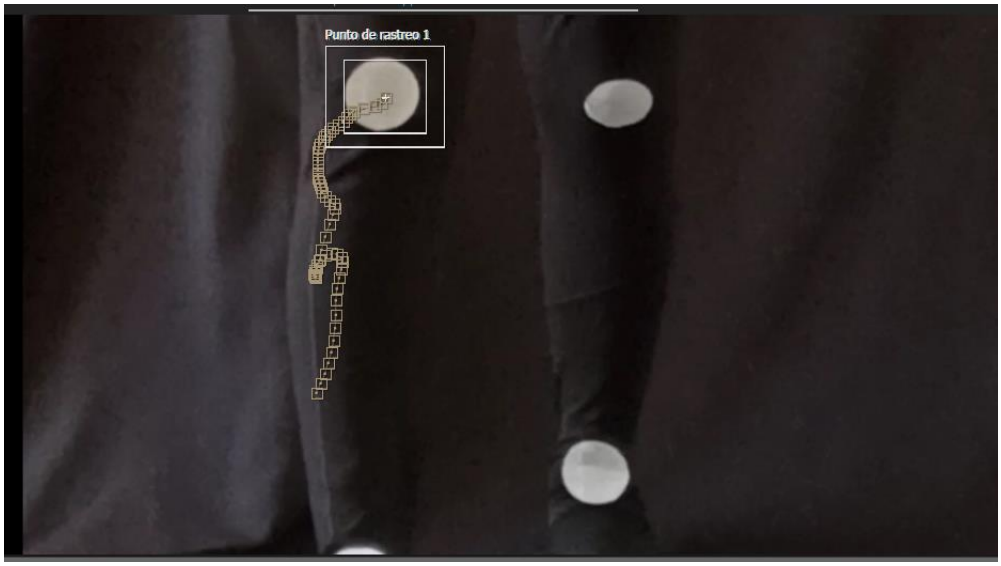


Figura 10. Mascara de recorte con seguimiento de trayectoria en la rodilla derecha en Adobe After Effects

Finalmente, para optimizar aún más el procesamiento se coloreo el área de interés articular con blanco, rojo, azul y verde para el centro de masa, cadera, rodilla y tobillo respectivamente como se puede observar en la Figura 11, esto para la implementación de un filtro RGB. Este proceso se llevó a cabo para el plano de interés (frontal) de cada integrante. Se generaron 3 videos a partir de este plano, uno para cada pierna con las 3 articulaciones (cadera, rodilla y tobillo) y otro solo del centro de masa en color blanco. En la Figura 11 se muestra el resultado conjunto de estos 3 videos generados para su procesamiento.

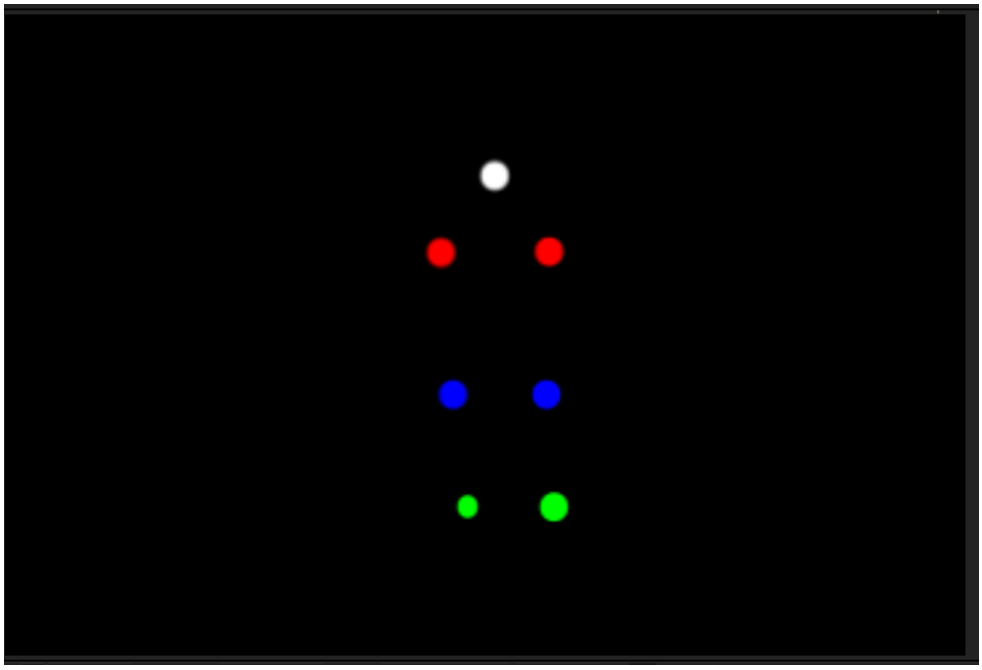


Figura 11. Resultado conjunto de preprocesamiento de marcha en plano frontal donde se observa con blanco, rojo, azul y verde para el centro de masa, cadera, rodilla y tobillo respectivamente.

Procesamiento

```
def plot_real_data_frontal():

    a = np.array([0,-3.3,-5,-4.05,-1.9,0,0.6,5.1,7,6.45,3.5,1,0.5])

    b = np.array([0,3,5,4,4,3,3,3,3,3,5,5,4,6,7,9,12,11,3,3,3])

    c = np.array([5,-1,-1,-1,-1,-1,-1,-1,-1,-1,3,5,3,1,1,1,1,1,2,5,3])

    figure, ax = plt.subplots(1,5,figsize=(20,4))

    ax[0].plot(np.array(a))

    ax[0].set_title("Cadera")

    ax[0].grid("on")

    ax[1].plot(b)
```

```
ax[1].set_title("Rodilla Derecha")

ax[1].grid("on")

ax[2].plot(b)

ax[2].set_title("Rodilla Izquierda")

ax[2].grid("on")

ax[3].plot(c)

ax[3].set_title("Pie Derecho")


ax[3].grid("on")

ax[4].plot(c)

ax[4].set_title("Pie izquierdo")

ax[4].grid("on")
```

Del mismo modo que en el plano sagital, se forman los vectores de referencia para el plano frontal, y se asigna un orden a las graficas para poder ser comparadas con los datos obtenidos de cada miembre del equipo en este plano.

Plano frontal						
ROM						
Sujeto	Derecho			Izquierdo		
	Cadera	Rodilla	Tobillo	Cadera	Rodilla	Tobillo
Luis	22.85 a 43.62	0.88 a 15.15	0.0 a 14.16	22.88 a 52.83	0.06 a 11.64	0.0 a 10.01
Alejandro	46.88 a 53.38	2.09 a 41.51	0.0 a 38.47	37.66 a 64.14	0.01 a 21.68	0.0 a 23.12
Junior	20.06 a 34.25	0.02 a 11.52	0.0 a 8.97	16.49 - 33.12	0.09 a 11.8	0.0 a 6.75
Nora	27.11 a 40.77	0.35 a 8.31	0.36 a 9.64	26.21 a 41.62	0.49 a 13.52	0.0 a 5.4
Víctor	29.13 a 45.53	1.05 a 16.21	0.0 a 19.2	28.98 a 61.51	0.55 a 19.59	0.0 a 18.25

Figura 12. Rango articular del miembro inferior de cada integrante del equipo obtenido plano frontal

Movimiento del centro de marcha eje x		
Sujeto	Plano frontal (cm)	Bibliografía
Luis	5.907407407	Debe moverse un promedio de 5cm ±1cm
Alejandro	8.59375	
Junior	5.5	
Nora	5.614583333	
Víctor	7.21875	

Figura 13. Movimiento de COM en el eje x de cada integrante del equipo en el plano frontal

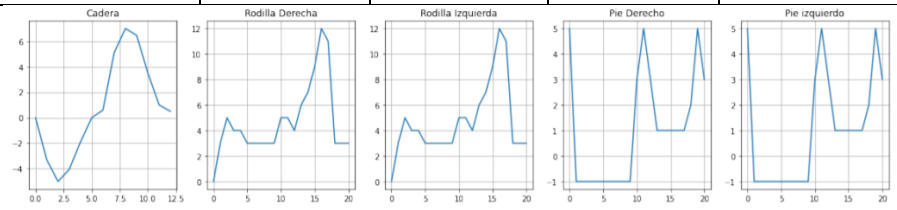
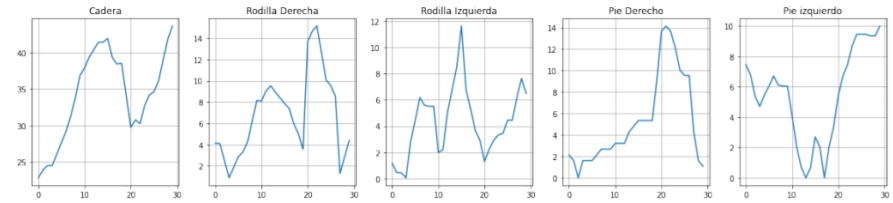
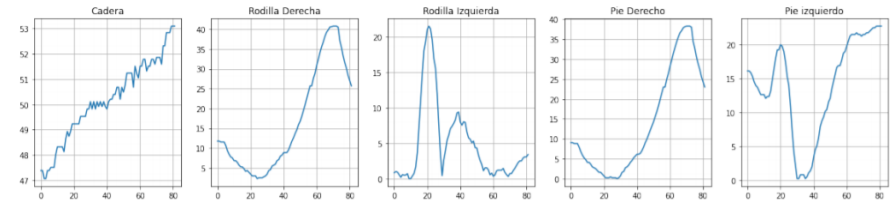
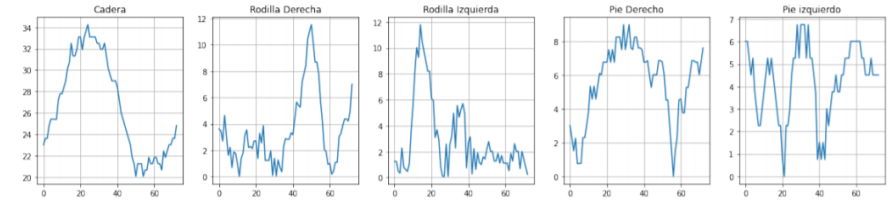
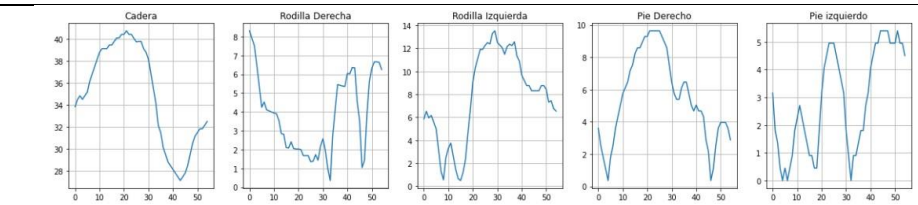
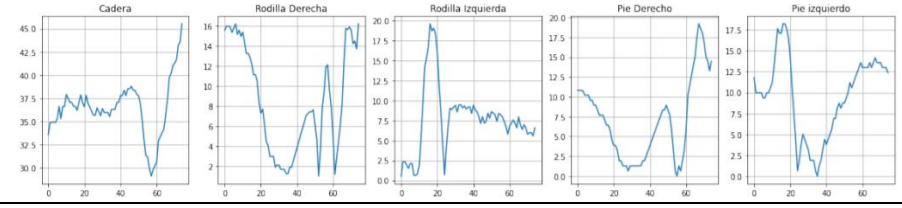
Plano Frontal comparación					
	Cadera	Rodilla D	Rodilla I	Pie D	Pie I
Bibliográfico					
Luis					
Alejandro					
Junior					
Nora					
Víctor					

Figura 14. Comparación de los datos obtenidos por integrante a la bibliografía plano frontal

EXPERIMENTO II| Animación del ciclo de marcha con los datos obtenidos.

Para este apartado la visualización que se despliega a partir del código presentado con anterioridad hecho en Python se creó a partir del preprocesamiento y el filtro empleado para extraer los marcadores, como se puede observar en las páginas 17-18, extraemos las figuras que en este caso son círculos y a cada articulación se le asigna un color principal en RGB y un blanco para el caso de centro de masa, este último solo para el análisis del plano frontal. Como se puede apreciar la asignación de un color a los marcadores nos ayuda a obtener solo esas figuras morfológicas con el fondo en negro es procesamiento de este filtro digital es más rápido ya que no tiene ruido que pueda afectar a esta operación.

Por consiguiente se toman los centroides de cada figura y se traza una recta entre las articulaciones que tiene una intersección con otra, tomando un solo punto para la cadera en el plano sagital y tomar las referencias de rodilla y tobillo, anudado a esto se realiza el procedimiento matemático presentado en las paginas 18-19 donde se aprecia la obtención del ángulo entre dos rectas, a continuación se presenta la visualización 2D obtenida para la animación del ciclo de marcha en el plano sagital y frontal.

Procesamiento

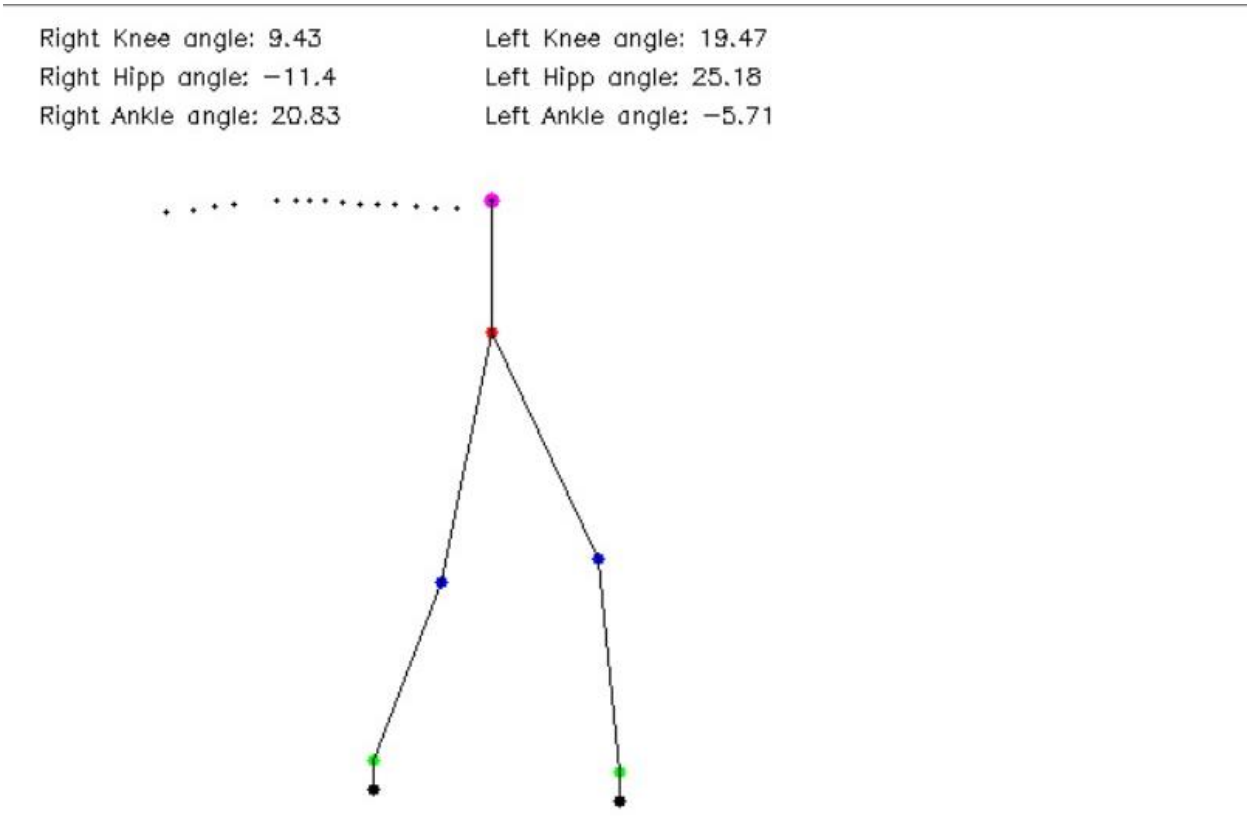


Figura 15. Animación 2D del ciclo de marcha sagital, visualización Python, se observan los marcadores con asignación de colores rojo, azul y verde para cadera, rodilla y tobillo respectivamente, el punto resaltado en violeta es una representación del centro de masa

es meramente un elemento para facilitar la visualización del movimiento de los miembros inferiores al graficar los ángulos de cada articulación.

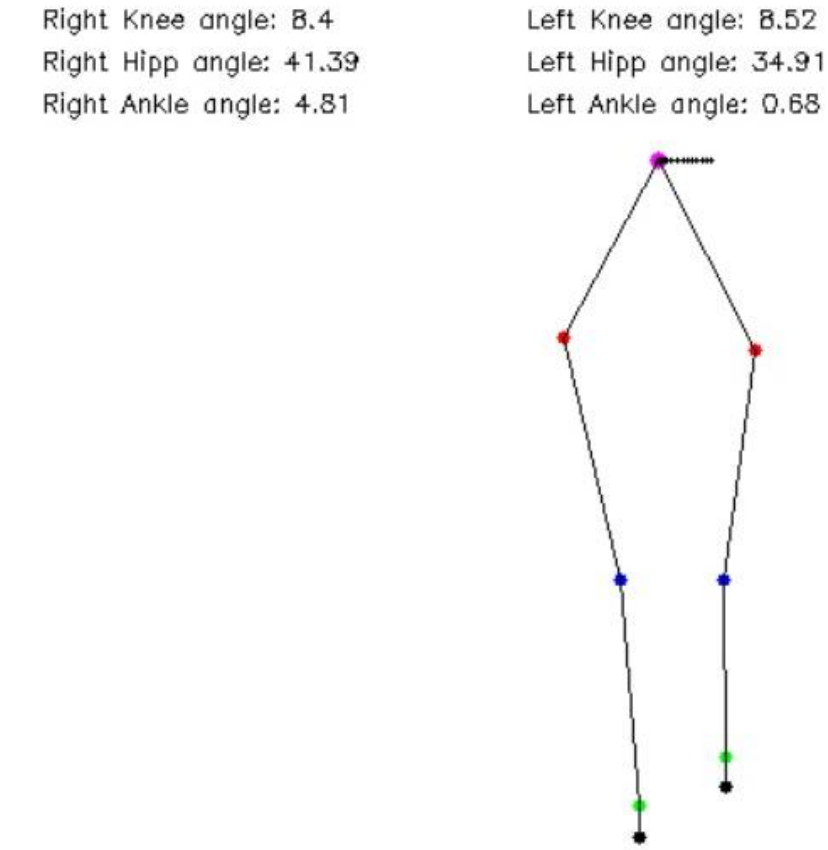


Figura 16. Animación 2D del ciclo de marcha frontal, visualización Python, se observan los marcadores con asignación de colores violeta, rojo, azul y verde para el centro de masa (referencia ombligo), cadera, rodilla y tobillo respectivamente.

Para el caso de esta Figura 16., a diferencia de la Figura 15., en esta si es importante tomar como una referencia el centro de masa para notar la desviación que se tiene en la cadera y en las demás articulaciones ya que a diferencia del plano sagital en este se observa en el plano frontal y se visualizan las dos articulaciones por lo que debemos tener un punto de referencia para unir las rectas y visualizar de una mejor manera el movimiento en el plano frontal.

EXPERIMENTO IV Animación tridimensional con interfaz gráfica, que permita seleccionar a cada integrante del equipo.

Se exportaron las trayectorias de Python como archivos .xlsx tablas de Excel, mismos que se importaron a Matlab y así poder proceder con la animación tridimensional de las trayectorias obtenidas previamente en los planos sagital y frontal para cada integrante.

Para ello se realizó el siguiente código

```
if app.ListBox.Value == "Alejandro"
    path = "C:\Users\Eduardo\Documents\CARRERA\8vo_semestre\BIO_4\Lab\2_Analiss_marcha\position_vectors\Alejandro.xlsx";

elseif  app.ListBox.Value == "Luis"
    path = "C:\Users\Eduardo\Documents\CARRERA\8vo_semestre\BIO_4\Lab\2_Analiss_marcha\position_vectors\Luis.xlsx";

elseif  app.ListBox.Value == "Junior"
    path = "C:\Users\Eduardo\Documents\CARRERA\8vo_semestre\BIO_4\Lab\2_Analiss_marcha\position_vectors\Junior.xlsx";

elseif  app.ListBox.Value == "Victor"
    path = "C:\Users\Eduardo\Documents\CARRERA\8vo_semestre\BIO_4\Lab\2_Analiss_marcha\position_vectors\Victor.xlsx";

elseif  app.ListBox.Value == "Nora"
    path = "C:\Users\Eduardo\Documents\CARRERA\8vo_semestre\BIO_4\Lab\2_Analiss_marcha\position_vectors\Nora.xlsx";

end


Data = xlsread(path);
MassX          = Data(:,1);
MassY          = Data(:,2);
MassZ          = Data(:,3);
RightHipX  = Data(:,4);
RightHipY  = Data(:,5);
RightHipZ  = Data(:,6);
RightKneeX = Data(:,7);
RightKneeY = Data(:,8);
RightKneeZ = Data(:,9);
RightAnkleX= Data(:,10);
RightAnkleY= Data(:,11);
RightAnkleZ= Data(:,12);
LeftHipX   = Data(:,13);
LeftHipY   = Data(:,14);
LeftHipZ   = Data(:,15);
LeftKneeX  = Data(:,16);
LeftKneeY  = Data(:,17);
LeftKneeZ  = Data(:,18);
LeftAnkleX = Data(:,19);
LeftAnkleY = Data(:,20);
LeftAnkleZ = Data(:,21);

%%%% 3D plot %%%%

for i=1:length(MassZ)
    cla(app.UIAxes);

    %%% Right %%%
    plot3(app.UIAxes,[MassX(i), RightHipX(i)], [MassY(i), RightHipY(i)], [MassZ(i), RightHipZ(i)], 'b-');
    hold(app.UIAxes, 'on' )
    plot3(app.UIAxes,[RightHipX(i), RightKneeX(i)], [RightHipY(i), RightKneeY(i)], [RightHipZ(i), RightKneeZ(i)], 'b-');
    hold(app.UIAxes, 'on' )
    plot3(app.UIAxes,[RightKneeX(i), RightAnkleX(i)], [RightKneeY(i), RightAnkleY(i)], [RightKneeZ(i), RightAnkleZ(i)], 'b-');
    hold(app.UIAxes, 'on' )
    %%% Left %%%
    plot3(app.UIAxes,[MassX(i), LeftHipX(i)], [MassY(i), LeftHipY(i)], [MassZ(i), LeftHipZ(i)], 'r-');
    hold(app.UIAxes, 'on' )
    plot3(app.UIAxes,[LeftHipX(i),LeftKneeX(i)], [LeftHipY(i), LeftKneeY(i)], [LeftHipZ(i),LeftKneeZ(i)], 'r-');
    hold(app.UIAxes, 'on' )
    plot3(app.UIAxes,[LeftKneeX(i), LeftAnkleX(i)], [LeftKneeY(i), LeftAnkleY(i)], [LeftKneeZ(i), LeftAnkleZ(i)], 'r-');
    set(app.UIAxes,'XLim',[0,1000],'YLim',[0,1000],'ZLim',[0,1000])
    xlabel(app.UIAxes,"X");
    ylabel(app.UIAxes,"Y");
    zlabel(app.UIAxes,"Z");
    view(app.UIAxes,10,70);
    grid(app.UIAxes,'on');
    drawnow
    pause(0.1);


end
```

Figura 17. Código implementado para el diseño de la API en MATLAB, para la visualización de la marcha graficando con visualización en 3D.

La interfaz gráfica cuenta con un menú selector “List Box”, en la cual como se puede apreciar en la primera sección del código anterior en la Figura 17, obtiene las trayectorias del archivo correspondiente para posteriormente graficarlo en los ejes “X”, “Y” y “Z”.

Cabe mencionar que en la etapa de preprocesamiento de imagen mencionado en el Experimento I y Experimento II para el plano sagital se obtuvieron datos en 2 dimensiones (desplazamiento vertical y horizontal), mientras que en el plano frontal se procesaron datos solo en una (desplazamiento horizontal), esto con la finalidad de obtener una animación síncrona.

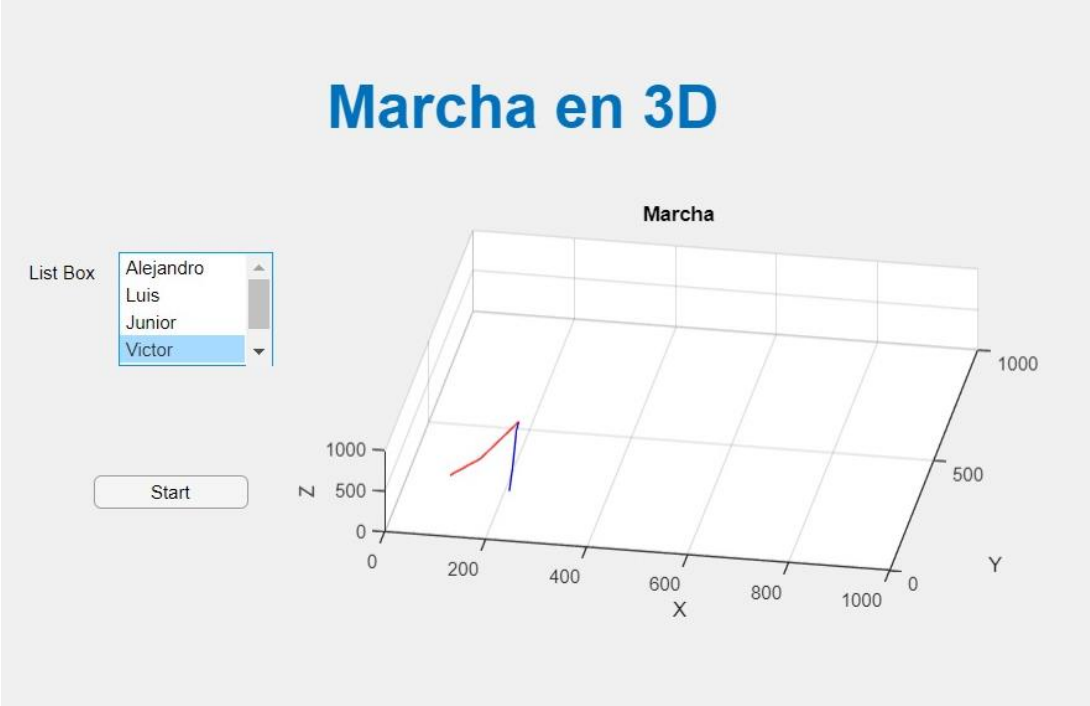


Figura 18. Interfaz gráfica de marcha en 3 dimensiones con selector para cada integrante del equipo y botón de inicio.

ANÁLISIS DE RESULTADOS

Es importante que analicemos las trayectorias obtenidas con la referencia bibliográfica, porque este tipo de análisis son realizados por médicos para la identificación de alguna anormalidad como lo argumentan Cifuentes C., Martínez y Romero E., (2010), “Con la información obtenida, el clínico formula un tratamiento y sigue la evolución del paciente, basándose en la información estadística, obtenida de estudios poblacionales y de su propia experiencia. Esta metodología es por supuesto altamente subjetiva y depende del conocimiento del especialista”; aclarando que nosotros no somos especialistas ni médicos, los resultados y el análisis brindado es meramente subjetivo y sin validez de un diagnóstico clínico solo es un indicador con base en las trayectorias obtenidas, afectaciones conocidas en el cuerpo de los sujetos de prueba y la bibliografía consultada.

Para el caso de la Figura 8, donde puede observarse los datos de cada participante graficados contra la bibliografía en el plano sagital.

- Se tuvo una mayor complicación el obtenerlos datos de la pierna más alejada de la cámara, ya que se llegan a perder los centroides durante el proceso de marcha. En la mayoría de los análisis esta pierna fue la izquierda con excepción del participante Junior en cuyo caso la pierna más alejada es la derecha.
- En el caso del participante Víctor sufre de una lesión en la cadera lo cual afecta toda la forma de su marcha y sus piernas tienen longitudes distintas.

Para la Figura 13, donde se tiene el desplazamiento de los centros de marcha comparados con la bibliografía.

- El movimiento del centro de masa de la mayoría de los participantes es alrededor de 5 cm lo cual es esperado, con la excepción del participante Víctor, por las características individuales ya mencionadas, y el participante Alejandro no camino recto hacia la cámara, si no que camino diagonalmente lo que provocó un mayor movimiento de su centro de masa.
- El resto de los participantes también sobrepaso los 5cm que se tiene como ideal, esto puede ser debido a que se camina hacia la cámara por lo que el programa lo analiza como un mayor desplazamiento.

Para la Figura 14, donde puede observarse los datos de cada participante graficados contra la bibliografía en el plano frontal.

- Los datos bibliográficos obtenidos no son exclusivos de una sola fuente, a diferencia en los propuestos en sagital, esto es debido a la poca información y análisis que se tiene del plano frontal en análisis de marcha.
- Para el caso del participante Alejandro en la gráfica de cadera se puede deber a que la caminata la hizo en forma diagonal.
- Nuevamente puede verse en la primeras tres graficas del participante Víctor, estas son afectadas por la lesión de cadera y por la diferencia de longitud de las piernas.

MATERIAL DE APOYO

MARCHA NORMAL PLANO SAGITAL						
% Ciclo de Marcha	Cadera		Rodilla		Tobillo	
	Promedio	%DE	Promedio	%DE	Promedio	%DE
0	19.33	5.64	3.97	4.19	0.02	3.93
2	18.92	5.79	7	4.58	-2.06	4.36
4	18.45	5.77	10.59	4.79	-3.88	4.32
6	17.94	5.64	14.12	4.98	-4.6	4.02
8	17.3	5.56	17.38	5.11	-3.98	3.95
10	16.4	5.63	19.84	5.12	-2.4	4.05

12	15.18	5.81	21.27	5.17	-0.45	4.01
14	13.67	5.94	21.67	5.34	1.45	3.81
16	11.97	5.9	21.22	5.61	3.04	3.56
18	10.21	5.73	20.2	5.85	4.27	3.36
20	8.48	5.57	18.86	5.95	5.13	3.24
22	6.74	5.47	17.35	5.85	5.71	3.19
24	4.94	5.38	15.73	5.57	6.1	3.18
26	3.13	5.36	14.08	5.18	6.43	3.18
28	1.42	5.51	12.5	4.84	6.76	3.21
30	-0.13	5.75	11.09	4.69	7.12	3.3
32	-1.54	5.93	9.91	4.64	7.54	3.44
34	-2.87	6.14	8.97	4.66	7.99	3.6
36	-4.12	6.34	8.28	4.74	8.44	3.79
38	-5.3	6.58	7.86	4.86	8.86	4
40	-6.4	6.86	7.72	4.95	9.23	4.25
42	-7.43	7.14	7.94	4.98	9.51	4.51
44	-8.39	7.4	8.6	4.97	9.62	4.75
46	-9.27	7.68	9.76	4.96	9.43	4.98
48	-10.02	7.97	11.5	4.97	8.7	5.26
50	-10.61	8.25	13.86	5.05	7.2	5.65
52	-10.95	8.51	16.97	5.22	4.69	6.12
54	-10.91	8.71	20.96	5.49	1.15	6.56
56	-10.31	8.81	26	5.86	-3.26	6.87
58	-9	8.72	32.03	6.2	-8.17	6.93
60	-6.95	8.39	38.74	6.3	-13.05	6.64
62	-4.25	7.84	45.6	6.05	-17.13	6.19
64	-1.05	7.17	52.05	5.53	-19.52	5.91
66	2.42	6.47	57.54	4.99	-19.77	5.81
68	5.93	5.8	61.66	4.75	-18.12	5.57
70	9.22	5.22	64.12	4.93	-15.29	5.06
72	12.11	4.75	64.86	5.41	-12.04	4.45
74	14.55	4.44	63.95	5.99	-8.85	3.99
76	16.53	4.35	61.59	6.51	-5.96	3.76
78	18.13	4.43	57.97	6.93	-3.51	3.63
80	19.45	4.59	53.27	7.22	-1.64	3.49
82	20.54	4.76	47.58	7.42	-0.5	3.27
84	21.38	4.84	40.94	7.55	-0.07	2.94
86	21.84	4.83	33.46	7.54	-0.16	2.65
88	21.87	4.75	25.38	7.29	-0.42	2.71
90	21.5	4.68	17.27	6.69	-0.52	3.09
92	20.84	4.72	9.94	5.73	-0.26	3.45

94	20.09	4.84	4.31	4.54	0.36	3.53
96	19.5	5.02	1.12	3.55	1	3.4
98	19.18	5.23	0.54	3.28	1.2	3.33
100	19.01	5.43	2.21	3.6	0.58	3.52

Tabla 1. Ángulos de articulaciones durante ciclo de marcha acorde a Winter, D. A. [7]

MARCHA NORMAL PLANO FRONTAL			
%CM	Cadera	Rodilla	Tobillo
0	0	0	5
5	-1.75	3	-1
10	-3.3	5	-1
15	-4.41	4	-1
20	-5	4	-1
25	-4.83	3	-1
30	-4.05	3	-1
35	-3.09	3	-1
40	-1.9	3	-1
45	-0.72	3	-1
50	0.6	5	3
55	2.91	5	5
60	5.1	4	3
65	6.6	6	1
70	6.45	7	1
75	4.95	9	1
80	3.5	12	1
85	2.13	11	1
90	1	3	2
95	0.6	3	5
100	0	3	3

Tabla 2. Ángulos de articulaciones durante ciclo de marcha acorde a Neumann, D. A. [6] y S.H. Cho [9]

Movimiento del centro de marcha (cm)		
%CM	Plano frontal	Plano sagital
0	0.48	1.2
5	0	0
10	-0.48	1.2
15	-1.01	2.5
20	-1.49	3.7

25	-1.86	4.65
30	-2	5
35	-1.86	4.5
40	-1.49	3.2
45	-1.01	1.7
50	-0.48	0.5
55	0	0
60	0.48	0.55
65	1.01	1.8
70	1.49	3.25
75	1.86	4.5
80	2	5
85	1.86	4.6
90	1.49	3.7
95	1.01	2.5
100	0.48	1.2

Tabla 3. Movimiento del centro de marcha según Neumann, D. A.[6]

CONCLUSIONES

- Al momento de desarrollar un programa de este tipo es indispensable y medular como se graba el ciclo de marcha en cuanto a los aspectos físicos, ya que estos representan un gran reto al momento de procesar el video, es necesario tener un fondo y ropa neutra, para que los marcadores resalten y sean apreciables, por ello el preprocesamiento nos ayuda bastante a optimizar el código ya que retiramos todo el ruido.
Además, con base en los resultados obtenidos y comparándolos con las referencias bibliográficas no podemos emitir un diagnóstico clínico con validez, ya que nosotros no somos especialistas en la salud ni mucho menos médicos, sin embargo, al ver los resultados de las trayectorias podemos decir que efectivamente este programa permite resaltar las diferencias entre marchas en el caso de mi análisis se nota las diferencias en las trayectorias, debido a afectaciones que tuve en la cadera y por pie plano, claramente esto no es objetivo pero como se mencionó en el análisis de resultados es meramente subjetivo y con lo obtenido podemos decir que este método puede ser una herramienta útil para el diagnóstico de afectaciones en el ciclo de marcha siempre y cuando sea realizado con un equipo y ambiente controlado, siempre y cuando sea realizado por un profesional en el área.
Cervantes López Victor Andrés

- Se llevó a cabo el análisis de la marcha en los planos sagitales y frontales, donde se presentó como principal problemática la adquisición de imagen por lo cual se implementó un preprocesamiento en el cual se le asigno un color

a cada articulación y al centro de masa para identificarlos posteriormente con un filtro RGB.

Se recabó información en diversas fuentes para realizar la comparación con los resultados obtenidos, en el cual para el plano frontal pocos estudios lo contemplan, centrándose en estudios pediátricos o con patologías. Los resultados obtenidos presentan trayectorias similares con los de la literatura, atribuyendo la diferencia a factores como la falta de estandarización en la adquisición de imagen.

La implementación de Python en la adquisición de datos mejoró de forma importante la velocidad de procesamiento en la animación tridimensional.

Cruz Olivares Luis Roberto

- A comparación del goniómetro digital realizado en Matlab, se observó que la implementación en Python mediante Open CV no solo empleó menos recursos, sino fue capaz de realizar un mayor número de instrucciones en un menor tiempo, dando un procesamiento cercano a tiempo real.

Las trayectorias de la mayoría de los integrantes (especialmente en el plano sagital) coinciden con las trayectorias descritas por Winter, D. A. (1998). Es difícil realizar una comparación punto por punto, ya que los puntos que se obtienen dependen de la cámara con la cual se adquirió la imagen (la cual fue diferente para cada integrante), por esto último es difícil obtener un error y evaluarlo.

El cambio de un algoritmo que clasifica los centroides por tamaño a uno que clasifica los centroides por color, lo cual solo fue posible aplicando un preprocesamiento al video, elimino el error de la obtención de ángulos incorrectos a la hora de realizar una inclinación en el plano sagital. Esto mejoró la adquisición de datos.

Lozano García Eduardo Alejandro

Para la facilitar la forma de obtención de los datos ya que los videos no se encontraban generalizados debido a que cada participante los hizo con diferentes equipos fue necesario el preprocesamiento de los videos para facilitar la obtención de los ángulos de las articulaciones, así como el centro de masa y su movimiento. De esta forma se generalizaron los videos de mejor forma, sin embargo, la falta de datos bibliográficos en el plano frontal dificulta el análisis de una marcha normal, así como no se pudo realizar la caminata en una caminadora.

Pérez López Nora Fernanda

- La importancia de crear herramientas para la adquisición de información, como lo fue la videografía expuesta en esta grafica para analizar los rangos articulares de la marcha humana nos presenta grandes retos en el transcurso del desarrollo, tuvimos que pasar por una serie de etapas para obtener una muestra adecuada, en este caso el vídeo; inicialmente teníamos que usar un

fondo totalmente negro, y círculos como indicadores de diferente tamaño, el problema era que si había un poco de luz, la muestra se alteraba drásticamente, además de que el procesamiento en Matlab fue demasiado lento debido a la demanda de operaciones morfológicas aplicadas a cada imagen, la ventaja, es que si se hacía de manera adecuada, la adquisición de las diferentes articulaciones, se consiguió de manera adecuada, y la obtención de los rangos articulares también, pero este proceso mejoro drásticamente con la implementación de su desarrollo en python, y la localización de las articulaciones por un filtro de color, esto lo vuelve más preciso, una facil trazabilidad para la animación, y una mejor obtención de los rangos articulares en comparación de las referencias.

Ramírez Miranda Junior Isaías

REFERENCIAS

[1] Becerra A. (2007) *"Biomecánica".* 1ª Ed. Pp. 2-5; ISBN: 959-261-245-5. Ciudad de la Habana: CUBA.

[2] Mehmet K. (2019) *"Introducción a las pruebas de diagnóstico por la imagen"*. Disponible en línea: https://www.msdmanuals.com/es-mx/hogar/temas-especiales/pruebas-de-diagn%C3%B3stico-por-la-imagen-habituales/introducci%C3%B3n-a-las-pruebas-de-diagn%C3%B3stico-por-la-imagen. MSD y los Manuales MSD: USA.

[3] Mondragón W. (2012). *"BENEFICIOS OBTENIDOS EN GRADOS DE MOVIMIENTO ARTICULAR EN MIEMBROS PÉLVICOS CON RUTINA DE ESTIRAMIENTO MÚSCULOTENDINOSO EN ATLETAS DE FONDO DEL CLUB ATLÉTICO MONDRAGÓN DEL PRIMERO DE JUNIO AL PRIMERO DE AGOSTO DEL 2011.*"; p. 6; Vol 1. Tesis de licenciatura. UAEM: México.

[4] Benítez J. y Hueso J. (sin fecha) Introducción a MATLAB; disponible en línea: http://personales.upv.es/jbenitez/data/matlab.pdf; consultado el 16/10/2020; Universidad politécnica de Valencia: España

[5] Zamora F. (sin editorial). Capitulo 3: Fundamentos de morfología matemática. En "Procesamiento morfológico de imágenes a color. Aplicación a la reconstrucción geodésica". (42-44). España.

[6] Neumann, D. A. (2002). Kinesiology of the Musculoskeletal System: Foundations for Physical Rehabilitation (1.a ed.). Mosby-Year Book.

[7] Winter, D. A. (1998). The Biomechanics and Motor Control of Human Gait (2.a ed.). Amsterdam University Press.

[8] Cifuentes C., Martínez y Romero E., (2010), *"ANÁLISIS TEÓRICO Y COMPUTACIONAL DE LA MARCHA NORMAL Y PATOLÓGICA: UNA REVISIÓN".* Recuperado de*:* *http://www.scielo.org.co/pdf/med/v18n2/v18n2a05.pdf.* Vol 18. No. 02. Pp. 186-192., Revista Med: Centro de Telemedicina, U. Nacional de Colombia. Bogotá: Colombia.

[9] S.H. Cho, J.M. Park, O.Y. Kwon (2004) Gender differences in three dimensional gait analysis data from 98 healthy Korean adults. Department of Physical Therapy, Graduate School of Rehabilitation Therapy, Research Institute of Health Science, Yonsei University, Wonju-si, Kangwon-do, South Korea.