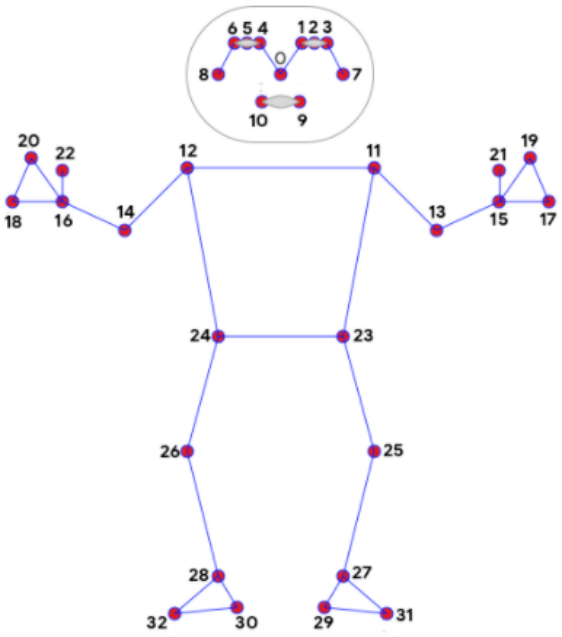
Pose_Estimation

April 4, 2023

#

Pose_Estimation

0.1 Opciones de Configuracion



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

- TATIC_IMAGE_MODE: Si se establece en false, la solución trata las imágenes de entrada como una transmisión de video. Intentará detectar a la persona más prominente en las primeras imágenes, y en una detección exitosa localiza aún más los puntos de referencia de la pose. En imágenes posteriores, simplemente rastrea esos puntos de referencia sin invocar otra detección hasta que pierde el rastro, reduciendo la computación y la latencia. Si se establece en true, la detección de persona ejecuta cada imagen de entrada, ideal para procesar un lote de imágenes estáticas, posiblemente no relacionadas. Predeterminado a false.
- MODEL_COMPLEXITY: Complejidad del modelo de punto de referencia de pose: `0, 1 o 2`. La precisión del marcador, así como la latencia de inferencia, generalmente aumentan con la complejidad del modelo. Predeterminado a `1`.
- SMOOTH_LANDMARKS: Si se establece en true, los filtros de solución plantean puntos de referencia en diferentes imágenes de entrada para reducir el fluctuación de fase, pero se ignoran si static_image_mode también está configurado para true. Predeterminado a true.

- ENABLE_SEGMENTATION: Si se establece en true, además de los puntos de referencia de la pose, la solución también genera la máscara de segmentación. Predeterminado a false.
- SMOOTH_SEGMENTATION: Si se establece en `true`, la solución filtra las máscaras de segmentación en diferentes imágenes de entrada para reducir el jitter. Ignorado si habilitar_segmentación es `false` o static_image_mode es `true`. Predeterminado a `true`.
- MIN_DETECTION_CONFIDENCE: Valor mínimo de confianza `([0.0, 1.0])` del modelo de detección de persona para que la detección se considere exitosa. Predeterminado a `0.5`.
- MIN_TRACKING_CONFIDENCE: Valor mínimo de confianza `([0.0, 1.0])` del modelo de seguimiento de puntos de referencia para que los puntos de referencia de pose se consideren rastreados con éxito, o de lo contrario se invocará automáticamente la detección de persona en la siguiente imagen de entrada. Establecerlo en un valor más alto puede aumentar la robustez de la solución, a expensas de una latencia más alta. Ignorado si static_image_mode es true, donde la detección de personas simplemente se ejecuta en cada imagen. Predeterminado a `0.5`.

0.1.1 Salida

- POSE_LANDMARKS: Una lista de puntos de referencia de pose. Cada punto de referencia consta de lo siguiente:
  - x y y: Coordenadas del marcador normalizadas a `[0.0, 1.0]` por el ancho y la altura de la imagen respectivamente.
  - z: Representa la profundidad de referencia con la profundidad en el punto medio de las caderas como origen, y cuanto menor sea el valor, más cerca estará el punto de referencia de la cámara. La magnitud de z usa aproximadamente la misma escala que x.
  - visibility: Un valor en `[0.0, 1.0]` indicando la probabilidad de que el punto de referencia sea visible (presente y no ocluido) en la imagen.
- POSE_WORLD_LANDMARKS: Otra lista de puntos de referencia de pose en coordenadas mundiales. Cada punto de referencia consta de lo siguiente:
  - x, y y z: Coordenadas 3D del mundo real en metros con el origen en el centro entre las caderas.
  - visibility: Idéntico al definido en el correspondiente Pose_landmarks.

Fuente

0.2 Explicacion del codigo!!

[2]:
```
#importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
```

[3]:
```
# creamos las instancias necesarias de para realizar la inferencia
mp_pose = mp.solutions.pose  # solucion que nos permite inferir la posee
mp_drawing = mp.solutions.drawing_utils # utilidades para dibujar
```

```
mp_drawing_styles = mp.solutions.drawing_styles # estilos para dibujar
```

[4]:
```
# confihuramos los parametros de nuestro detector
pose = mp_pose.Pose( static_image_mode=True, # le decimos que trabajaremos␣
 ↪sobre imagenes estaticas
                     min_detection_confidence=0.5, # un minimo de confianza de␣
 ↪50% en la deteccion
                     min_tracking_confidence=0.5)  # unminimo de confianza del␣
 ↪50% en el tracking
```

[6]:
```
# haciendo uso de Ipython.display.Image mostramos la imagen
path="yoga2.jpg"
Image(path)
```

[6]:

[7]:
```
img = cv.imread(path) # cargamos la imagen con la cual vamos a trabajar
img = cv.cvtColor(img, cv.COLOR_BGR2RGB) # realizamos un cambio de espacio de␣
 ↪color a RGB
img_pro = img.copy() # creamos una copia de la imagen
```

[8]:
```
results = pose.process(img) # realizamos la prediccion
```

[13]:
```
# si nuestra prediccion tiene resultados ingresara al if
if results.pose_landmarks:
```

```
    land_marks = results.pose_landmarks # extraemos los puntos importantes
    for id, mark in enumerate(land_marks.landmark): # enumeramos e imprimimos␣
↪los las cordenadas de los puntos importantes
        print(mark)
```

```
x: 0.45557141304016113
y: 0.5883466005325317
z: -0.22248272597789764
visibility: 0.9999887943267822

x: 0.44931501150131226
y: 0.5756958723068237
z: -0.23850305378437042
visibility: 0.9999946355819702

x: 0.44968748092651367
y: 0.5722119808197021
z: -0.23847368359565735
visibility: 0.9999969005584717

x: 0.44993847608566284
y: 0.5684333443641663
z: -0.23857422173023224
visibility: 0.9999945163726807

x: 0.4481392800807953
y: 0.5789586305618286
z: -0.20838508009910583
visibility: 0.9999783039093018

x: 0.4477941393852234
y: 0.5780285596847534
z: -0.20840123295783997
visibility: 0.9999855756759644

x: 0.44729241728782654
y: 0.5767556428909302
z: -0.2082982361316681
visibility: 0.9999839067459106

x: 0.4559192657470703
y: 0.5488487482070923
z: -0.2746198773384094
visibility: 0.9999896287918091

x: 0.45218390226364136
```

```
y: 0.5578610301017761
z: -0.13753552734851837
visibility: 0.9999908208847046

x: 0.4671911299228668
y: 0.5801762938499451
z: -0.2332431524991989
visibility: 0.9999192953109741

x: 0.46567147970199585
y: 0.5863727927207947
z: -0.19413872063159943
visibility: 0.999954104423523

x: 0.5122780203819275
y: 0.517123281955719
z: -0.3299248218536377
visibility: 0.9998914003372192

x: 0.49200373888015747
y: 0.5345454216003418
z: -0.04547976702451706
visibility: 0.9999053478240967

x: 0.5206464529037476
y: 0.6390525102615356
z: -0.42812782526016235
visibility: 0.9968903660774231

x: 0.49957937002182007
y: 0.6425914764404297
z: -0.028117496520280838
visibility: 0.6315256357192993

x: 0.5126832127571106
y: 0.741550862789154
z: -0.4898722171783447
visibility: 0.9886748790740967

x: 0.4947667717933655
y: 0.7463598847389221
z: -0.09141349792480469
visibility: 0.8523831963539124

x: 0.5109248161315918
y: 0.7589132785797119
z: -0.5541337728500366
visibility: 0.97918701171875
```

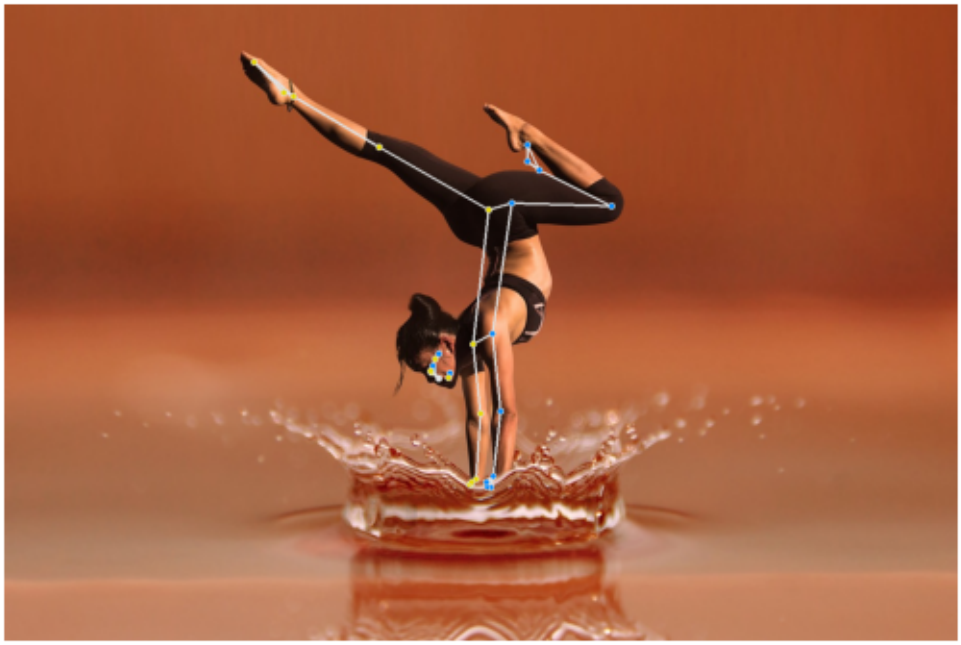
```
x: 0.48942941427230835
y: 0.7557462453842163
z: -0.10044588148593903
visibility: 0.8511770963668823

x: 0.506310760974884
y: 0.7579829692840576
z: -0.5434064865112305
visibility: 0.975131094455719

x: 0.48795270919799805
y: 0.7543599605560303
z: -0.12882989645004272
visibility: 0.8581571578979492

x: 0.5061345100402832
y: 0.7513277530670166
z: -0.4913550317287445
visibility: 0.9568849802017212

x: 0.49031487107276917
y: 0.7509795427322388
z: -0.10603611171245575
visibility: 0.8519642949104309

x: 0.5325754880905151
y: 0.31285083293914795
z: -0.09451388567686081
visibility: 0.999914288520813

x: 0.5084764957427979
y: 0.3226236402988434
z: 0.09415336698293686
visibility: 0.9999802112579346

x: 0.6370707750320435
y: 0.3175014853477478
z: -0.07427604496479034
visibility: 0.9994537234306335

x: 0.3931005597114563
y: 0.2246125191450119
z: 0.19823944568634033
visibility: 0.9796093106269836

x: 0.5612906217575073
y: 0.26076531410217285
```

```
z: 0.1603175401687622
visibility: 0.9383107423782349

x: 0.30390238761901855
y: 0.14481046795845032
z: 0.21948333084583282
visibility: 0.911787748336792

x: 0.548602283000946
y: 0.24635164439678192
z: 0.19103041291236877
visibility: 0.8777211904525757

x: 0.29372772574424744
y: 0.13890069723129272
z: 0.22263869643211365
visibility: 0.8540935516357422

x: 0.5489404797554016
y: 0.2208191454410553
z: 0.2707515060901642
visibility: 0.8978449702262878

x: 0.26216360926628113
y: 0.09068168699741364
z: 0.24898478388786316
visibility: 0.8115478157997131
```

[14]:
```
#dibujamos los puntos y sus conexiones en la imagen
mp_drawing.draw_landmarks(img_pro,     # imagen a dibujar
                          results.pose_landmarks, # puntos
                          mp_pose.POSE_CONNECTIONS, # conexiones
                          landmark_drawing_spec=mp_drawing_styles.
 ↪get_default_pose_landmarks_style()) # estilo
```

[16]:
```
# mostramos la imagen dibujado las posees
import matplotlib.pyplot as plt
plt.imshow(img_pro)
plt.axis("off")
```

[16]: (-0.5, 1279.5, 852.5, -0.5)



0.3 Deteccion de Pose en imagenes Estaticas

[19]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt
#instanciamos los objetos necesarios
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
# configuramos nuestro detector
pose = mp_pose.Pose( static_image_mode=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5)
```

[20]:
```
# creamos una funcion que nos permite realizar la inferencia y dibujar nuestros␣
 ↪resultados
def pose_estimation(img_res):

    results = pose.process(img_res)
    if results.pose_landmarks:
        mp_drawing.draw_landmarks(img_res,
                          results.pose_landmarks,
```

```
                        mp_pose.POSE_CONNECTIONS,
                        landmark_drawing_spec=mp_drawing_styles.
↪get_default_pose_landmarks_style())
    plt.imshow(img_res)
    plt.axis("off")
    plt.show()
```

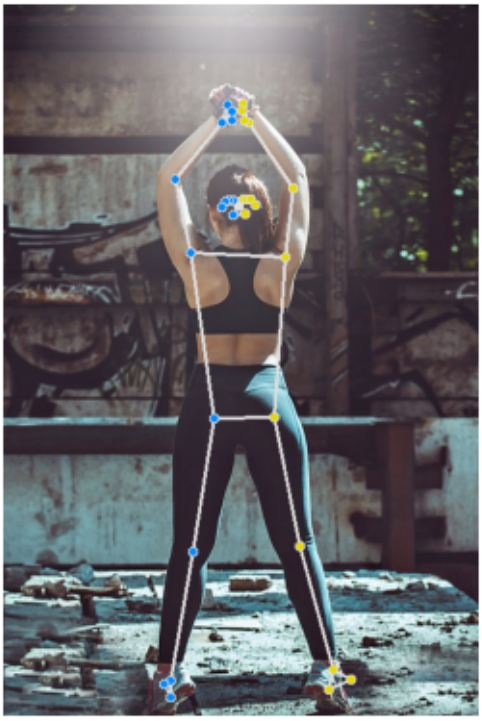
0.3.1 Probamos con algunas Imagenes

[21]:
```
path= "karate.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_estimation(img)
```

[22]:
```
path= "man.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_estimation(img)
```

[23]:
```
path= "sports.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_estimation(img)
```

[24]:
```
path= "sports2.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_estimation(img)
```



[25]:
```
path= "yoga.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_estimation(img)
```

[ ]:

0.4 Pose estimation en Video

[28]:
```
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt

mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

pose = mp_pose.Pose( static_image_mode=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5)



def pose_estimation(img_res):

    img_res = cv.cvtColor(img_res, cv.COLOR_BGR2RGB)
    results = pose.process(img_res)
    if results.pose_landmarks:
        mp_drawing.draw_landmarks(img_res,
                          results.pose_landmarks,
                          mp_pose.POSE_CONNECTIONS,
                          landmark_drawing_spec=mp_drawing_styles.
 ↪get_default_pose_landmarks_style())

    img_res = cv.cvtColor(img_res, cv.COLOR_RGB2BGR)
    return img_res

##----------PRINCIPAL CODDE -------------------
cv.namedWindow("Pose Estimation", cv.WINDOW_NORMAL)

cap = cv.VideoCapture(0)

while cap.isOpened():

    ret, frame= cap.read()
    if ret:
```

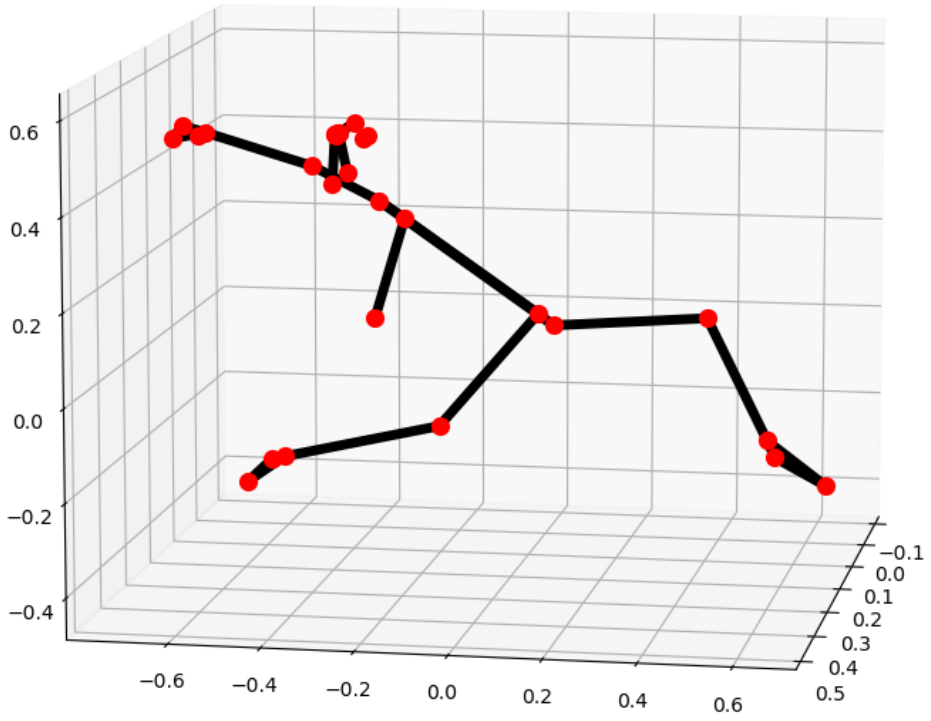
```
        img_ret = pose_estimation(frame)
        cv.imshow("Pose Estimation", img_ret)

        key = cv.waitKey(5)
        if key==27:
            break
    else:
        break

cap.release()
cv.destroyAllWindows()
```

0.5 Graficación en 3d de los puntos de interes

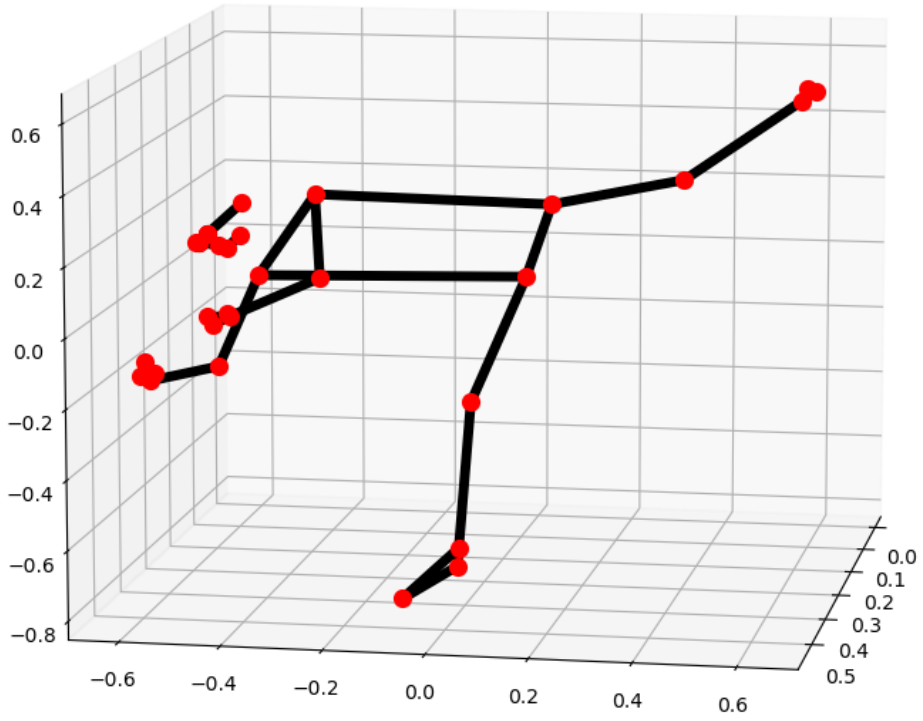
[4]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt
#instanciamos los objetos necesarios
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
# configuramos nuestro detector
pose = mp_pose.Pose( static_image_mode=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5)
```

[7]:
```
# creamos una funcion que nos permite realizar la inferencia y graficar en 3d␣
 ↪los puntos de interes y sus conexiones
def plot_landmark_pose(img_res):

    results = pose.process(img_res)
    if results.pose_landmarks:
        mp_drawing.plot_landmarks(results.pose_world_landmarks, mp_pose.
 ↪POSE_CONNECTIONS)
```

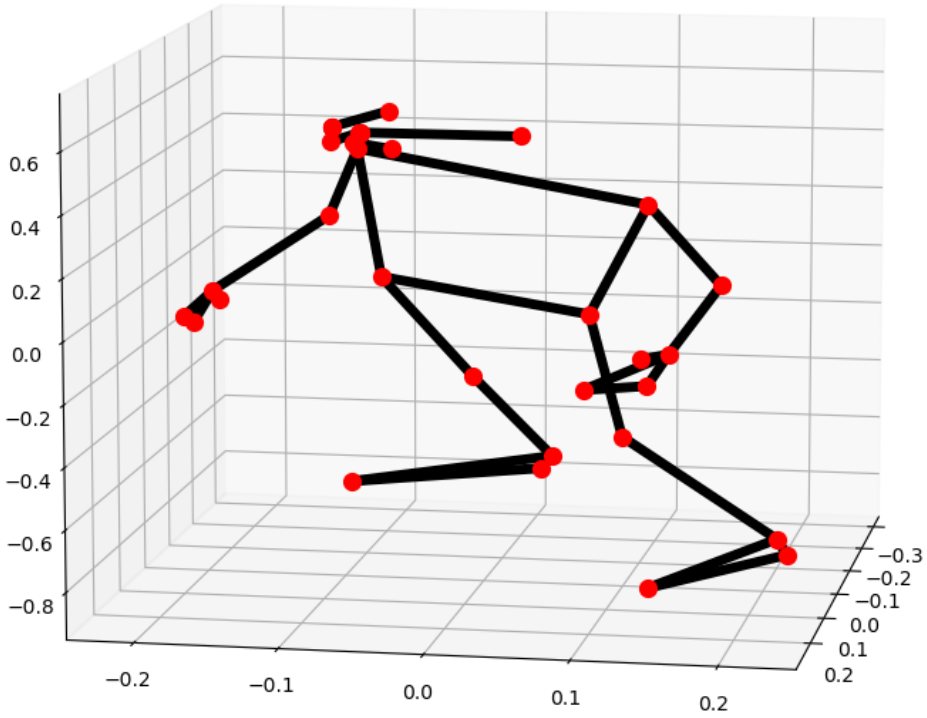
[8]:
```
path= "yoga.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
plot_landmark_pose(img)
```



[9]:
```
path= "Karate.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
plot_landmark_pose(img)
```



[12]:
```
path= "man.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
plot_landmark_pose(img)
```



0.6 Segmentación en Imagenes

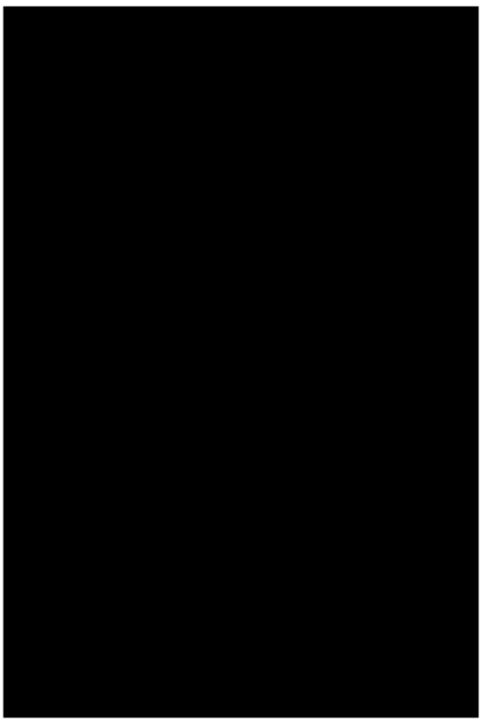
0.6.1 Explicación

[96]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt
#instanciamos los objetos necesarios
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
```

```
mp_drawing_styles = mp.solutions.drawing_styles
# configuramos nuestro detector
pose = mp_pose.Pose( static_image_mode=True,
                     model_complexity=2,
                     enable_segmentation=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5,
                   )

path= "man.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)

pros_img = img.copy()
results = pose.process(img)

#if results.pose_landmarks:
```

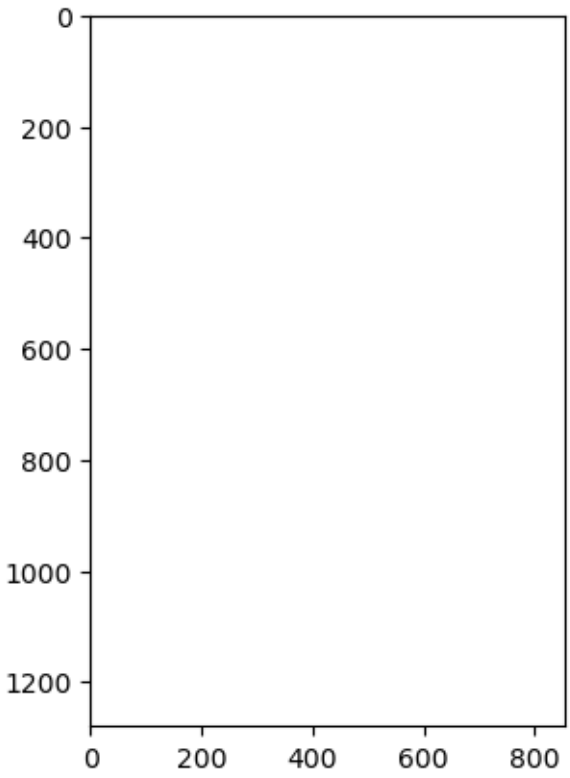
[130]:
```
#usando np.zeros_like creamos una mascara con las mismas dimensiones de la␣
 ↪imagen original de tipo enteto de 8bits
red_img = np.zeros_like(pros_img, dtype=np.uint8)
print(type(red_img))
print(red_img.shape)
plt.imshow(red_img)
plt.axis("off")
plt.show()
```

```
<class 'numpy.ndarray'>
(1280, 853, 3)
```



[131]:
```
# llenamos la mascara con valores de 255 en todos los pixeles de todos los␣
  ↪canales
red_img[:,:] = (255,255,255)

print(type(red_img))
print(red_img.shape)
plt.imshow(red_img)
#plt.axis("off")
plt.show()
```

```
<class 'numpy.ndarray'>
(1280, 853, 3)
```

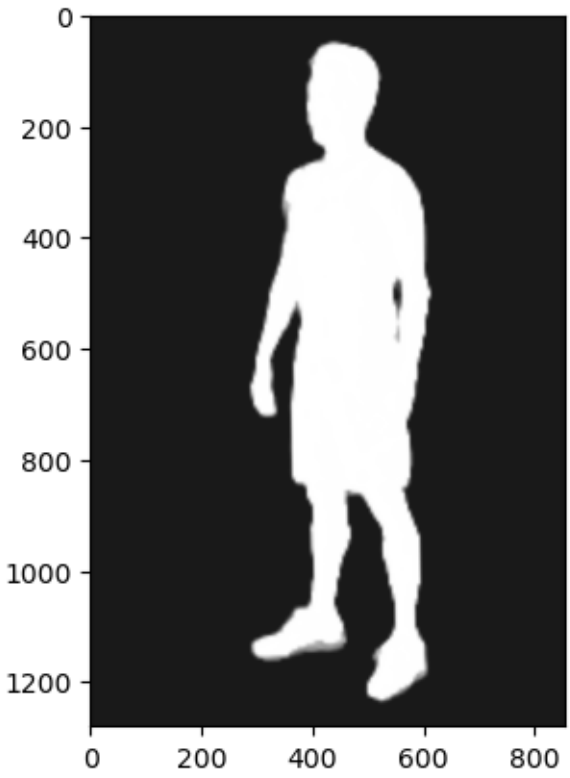


[138]:
```
segm_img = 0.1 + 0.9 * results.segmentation_mask

print(type(segm_img))
print(segm_img.shape)
#print(segm_img)
plt.imshow(segm_img, cmap="gray")
plt.axis("off")
plt.show()
```

```
<class 'numpy.ndarray'>
(1280, 853)
```

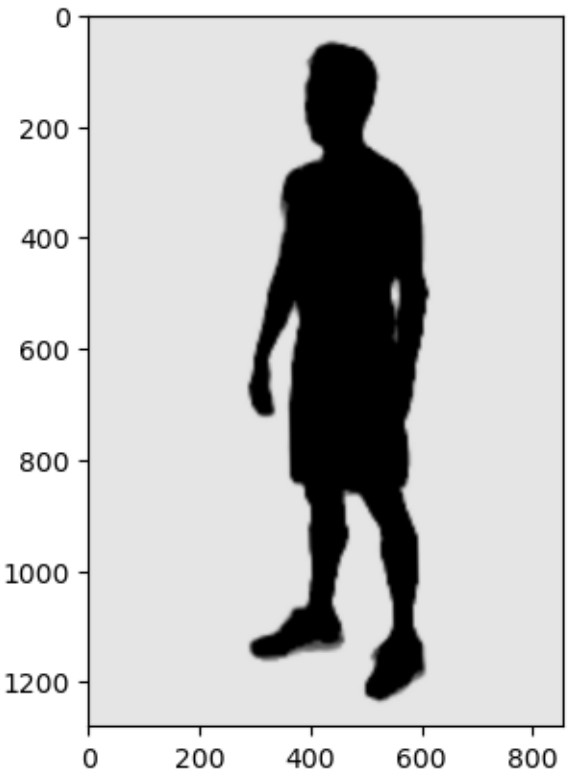
[139]:
```
# la funcion repeat de numpy(np.repeat), nos ayuda a repetir los elementos de
  ↪un array y a expandir esta repeticion
# en otras dimensioniones numpy.repeat(a, repeats, axis=None)[source]

segm_img = np.repeat(segm_img[..., np.newaxis], 3, axis=2)

print(type(segm_img))
print(segm_img.shape)
#print(segm_img)
plt.imshow(segm_img)
#plt.axis("off")
plt.show()
```

```
<class 'numpy.ndarray'>
(1280, 853, 3)
```



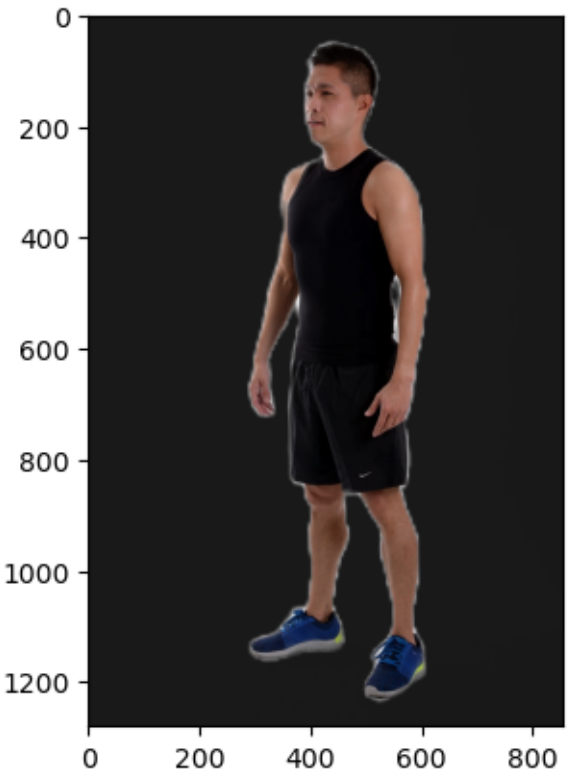
[141]:
```
res_img1 = red_img * (1 - segm_img)
res_img1 = np.array(res_img1, dtype='uint8')
print(type(res_img1))
print(res_img1.shape)
#print(res_img1)
plt.imshow(res_img1, cmap="gray")
#plt.axis("off")
plt.show()
```

```
<class 'numpy.ndarray'>
(1280, 853, 3)
```



[142]:
```
res_img2 = pros_img * segm_img

res_img2 = np.array(res_img2, dtype='uint8')
print(type(res_img2))
print(res_img2.shape)

plt.imshow(res_img2, cmap="gray")
#plt.axis("off")
plt.show()
```

```
<class 'numpy.ndarray'>
(1280, 853, 3)
```



[143]:
```
pros_img = res_img1 + res_img2
```

[144]:
```
pros_img = np.array(pros_img, dtype='uint8')

plt.imshow(pros_img)
plt.axis("off")
plt.show()
```

0.6.2 Segmentación en Imagene Estaticas codigo completo

[145]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt
#instanciamos los objetos necesarios
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
# configuramos nuestro detector
pose = mp_pose.Pose( static_image_mode=True,
                     model_complexity=2,
                     enable_segmentation=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5,
                   )
```

[146]:
```
# creamos una funcion que nos permite realizar la inferencia y dibujar nuestros␣
 ↪resultados
def pose_segmentation(img_res):
```

```
    pros_img = img_res.copy()
    results = pose.process(img_res)
    if results.pose_landmarks:
        red_img = np.zeros_like(pros_img, dtype=np.uint8)
        red_img[:, :] = (255,255,255)
        segm_img = 0.1 + 0.9 * results.segmentation_mask
        segm_img = np.repeat(segm_img[..., np.newaxis], 3, axis=2)
        pros_img = pros_img * segm_img + red_img * (1 - segm_img)
        pros_img = np.array(pros_img, dtype='uint8')

    plt.imshow(pros_img)
    plt.axis("off")
    plt.show()
```

0.6.3 Pruebas

[147]:
```
path= "karate.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_segmentation(img)
```

[148]:
```
path= "man.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
```

```
pose_segmentation(img)
```

[149]:
```
path= "sports.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_segmentation(img)
```

[150]:
```
path= "sports2.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_segmentation(img)
```

[151]:
```
path= "yoga.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_segmentation(img)
```

[152]:
```
path= "yoga2.jpg"
img = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)
pose_segmentation(img)
```

0.6.4 Cambiando el fondo de una imagen usando segmentación

[153]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt
#instanciamos los objetos necesarios
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
# configuramos nuestro detector
pose = mp_pose.Pose( static_image_mode=True,
                     model_complexity=2,
                     enable_segmentation=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5,
                   )
```

[168]:
```
# creamos una funcion que nos permite realizar la inferencia y dibujar nuestros␣
 ↪resultados
def pose_segmentation(img_res, fondo):
    #igualamos las dimensiones de las imagenes
    hf,wf,c = fondo.shape
    print(fondo.shape)
    hs,ws,c = img_res.shape
    print(img_res.shape)
    img_res = cv.resize(img_res,(int(hf/hs*ws),hf))
    hs,ws,c = img_res.shape
    print(img_res.shape)
    #**********************

    pros_img = img_res.copy()
    results = pose.process(img_res)
    if results.pose_landmarks:
        red_img = fondo[0:hs,0:ws]

        segm_img = 0.1 + 0.9 * results.segmentation_mask
        segm_img = np.repeat(segm_img[..., np.newaxis], 3, axis=2)
        img_res1 = red_img * (1 - segm_img)
        img_res2 = pros_img * segm_img
        pros_img = img_res1 + img_res2
        pros_img = np.array(pros_img, dtype='uint8')
        fondo[0:hs,0:ws]=pros_img
    plt.imshow(fondo)
    plt.axis("off")
    plt.show()
```

[172]:
```
path1 = "sports.jpg"
path2 = "luna.jpg"
img1 = cv.cvtColor(cv.imread(path1), cv.COLOR_BGR2RGB)
img2 = cv.cvtColor(cv.imread(path2), cv.COLOR_BGR2RGB)

pose_segmentation(img1, img2)
```

```
(718, 1280, 3)
(1280, 853, 3)
(718, 478, 3)
```

[171]:
```
path1 = "sports2.jpg"
path2 = "pasto.jpg"
img1 = cv.cvtColor(cv.imread(path1), cv.COLOR_BGR2RGB)
img2 = cv.cvtColor(cv.imread(path2), cv.COLOR_BGR2RGB)

pose_segmentation(img1, img2)
```

```
(1043, 1280, 3)
(640, 427, 3)
(1043, 695, 3)
```

0.6.5 Moviendo nuestra imagen

[153]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt
#instanciamos los objetos necesarios
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
# configuramos nuestro detector
pose = mp_pose.Pose( static_image_mode=True,
                     model_complexity=2,
                     enable_segmentation=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5,
                   )
```

[179]:
```
# creamos una funcion que nos permite realizar la inferencia y dibujar nuestros␣
 ↪resultados
def pose_segmentation(img_res, fondo):
```

```
    #igualamos las dimensiones de las imagenes
    hf,wf,c = fondo.shape
    print(fondo.shape)
    hs,ws,c = img_res.shape
    print(img_res.shape)

    # La dimesion en x a cambiar debe tener la misma relacion de cambio de y␣
 ↪por lo tanto para obtener esa relacion
    # de divide el alto de la imagen de fondo / sobre la altura de la imagen a␣
 ↪segmentar y  el tamaño a modificar en x se obtiene
    # multiplicando esta relacion por el ancho de la imagen a segmentar

    img_res = cv.resize(img_res,(int(hf/hs*ws),hf))
    hs,ws,c = img_res.shape
    print(img_res.shape)
    #**********************

    pros_img = img_res.copy()
    results = pose.process(img_res)
    dx = 100
    if results.pose_landmarks:
        red_img = fondo[0:hs,dx:ws+dx]

        segm_img = 0 + 1 * results.segmentation_mask
        segm_img = np.repeat(segm_img[..., np.newaxis], 3, axis=2)
        img_res1 = red_img * (1 - segm_img)
        img_res2 = pros_img * segm_img
        pros_img = img_res1 + img_res2
        pros_img = np.array(pros_img, dtype='uint8')
        fondo[0:hs,dx:ws+dx]=pros_img
    plt.imshow(fondo)
    plt.axis("off")
    plt.show()
```

[180]:
```
path1 = "sports2.jpg"
path2 = "pasto.jpg"
img1 = cv.cvtColor(cv.imread(path1), cv.COLOR_BGR2RGB)
img2 = cv.cvtColor(cv.imread(path2), cv.COLOR_BGR2RGB)

pose_segmentation(img1, img2)
```

```
(1043, 1280, 3)
(640, 427, 3)
(1043, 695, 3)
```

0.6.6 Utilizando el GUI de OpenCv

[11]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt

#instanciamos los objetos necesarios
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
# configuramos nuestro detector
pose = mp_pose.Pose( static_image_mode=True,
                     model_complexity=2,
                     enable_segmentation=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5,
                   )
def nothing(x):
    pass
```

```
# creamos una funcion que nos permite realizar la inferencia y dibujar nuestros␣
 ↪resultados
def pose_segmentation(img_res, fondo,dx):
    #igualamos las dimensiones de las imagenes
    hf,wf,c = fondo.shape
    hs,ws,c = img_res.shape
    img_res = cv.resize(img_res,(int(hf/hs*ws),hf))
    hs,ws,c = img_res.shape
    #**********************

    pros_img = img_res.copy()
    pros_fondo = fondo.copy()
    results = pose.process(img_res)
    #dx = 100
    if results.pose_landmarks:

        red_img = fondo[0:hs,dx:ws+dx]

        segm_img = 0 + 1 * results.segmentation_mask
        segm_img = np.repeat(segm_img[..., np.newaxis], 3, axis=2)
        img_res1 = red_img * (1 - segm_img)
        img_res2 = pros_img * segm_img
        pros_img = img_res1 + img_res2
        pros_img = np.array(pros_img, dtype='uint8')
        pros_fondo[0:hs,dx:ws+dx]=pros_img
    return pros_fondo

##*************************************************************
##---------------Codigo Principal--------------------------

#Creamos la ventana donde se mostrara el resultado
cv.namedWindow("Salida", cv.WINDOW_NORMAL)
# leemos las iamgenes aprocesar
path1 = "karate.jpg"
path2 = "pasto.jpg"
img = cv.cvtColor(cv.imread(path1), cv.COLOR_BGR2RGB)
img_f = cv.cvtColor(cv.imread(path2), cv.COLOR_BGR2RGB)

hf,wf,c = img_f.shape
h,w,c = img.shape

## Creamos el trackbar para el desplazamiento en X
cv.createTrackbar("pos_x","Salida",0,wf-w,nothing)

while True:
    dx = cv.getTrackbarPos("pos_x","Salida")
```

```
    img = cv.cvtColor(cv.imread(path1), cv.COLOR_BGR2RGB)
    img_f = cv.cvtColor(cv.imread(path2), cv.COLOR_BGR2RGB)

    img = pose_segmentation(img,img_f,dx)
    img = cv.cvtColor(img, cv.COLOR_RGB2BGR)

    cv.imshow("Salida", img)
    if cv.waitKey(1) & 0xFF==ord("q"):
        break

cv.destroyAllWindows()
```

0.6.7 Resultado desplazando la imagen en el GUI

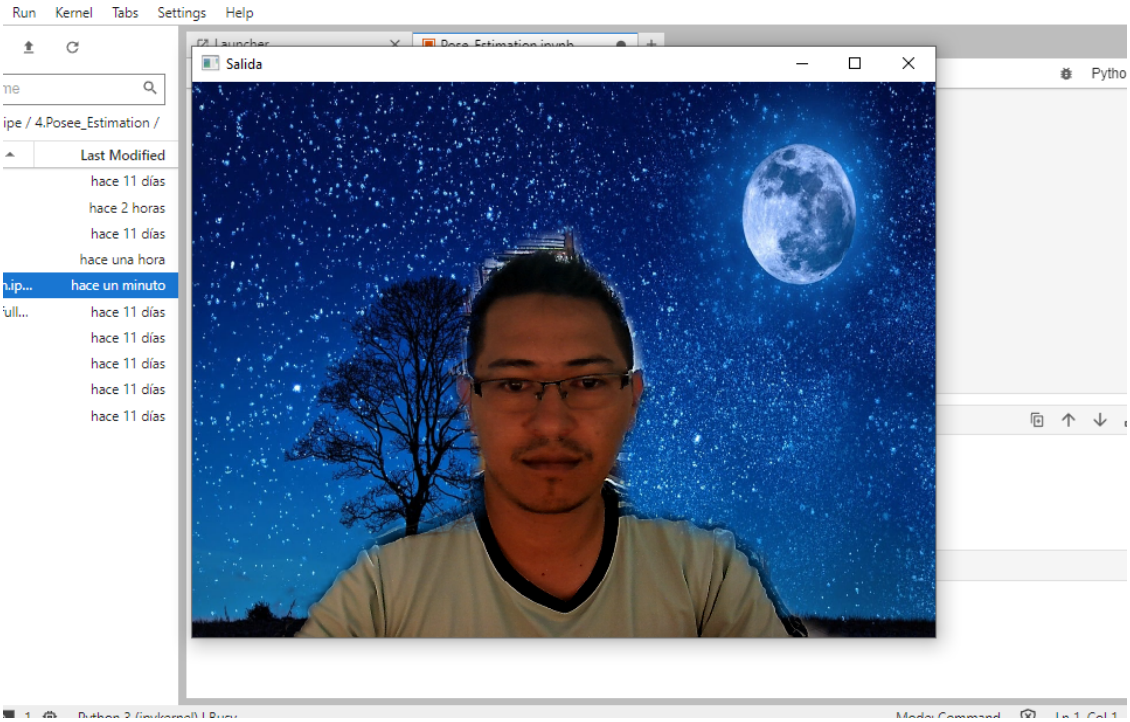
0.7 Segmentación desde la Camara web

[12]:
```
# importamos librerias
import cv2 as cv
import numpy as np
import mediapipe as mp
from IPython.display import Image
import matplotlib.pyplot as plt

#instanciamos los objetos necesarios
mp_pose = mp.solutions.pose
```

```
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
# configuramos nuestro detector
pose = mp_pose.Pose( static_image_mode=True,
                     model_complexity=2,
                     enable_segmentation=True,
                     min_detection_confidence=0.5,
                     min_tracking_confidence=0.5,
                    )

# creamos una funcion que nos permite realizar la inferencia y dibujar nuestros␣
 ↪resultados
def pose_segmentation(img_res, fondo):
    #igualamos las dimensiones de las imagenes

    pros_img = img_res.copy()

    results = pose.process(img_res)
    #dx = 100
    if results.pose_landmarks:

        red_img = fondo.copy()

        segm_img = 0 + 1 * results.segmentation_mask
        segm_img = np.repeat(segm_img[..., np.newaxis], 3, axis=2)
        img_res1 = red_img * (1 - segm_img)
        img_res2 = pros_img * segm_img
        pros_img = img_res1 + img_res2
        pros_img = np.array(pros_img, dtype='uint8')

    return pros_img

##*************************************************************
##---------------Codigo Principal--------------------------

#Creamos la ventana donde se mostrara el resultado
cv.namedWindow("Salida", cv.WINDOW_NORMAL)
path = "luna.jpg"
img_f = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)

cap = cv.VideoCapture(0)
while cap.isOpened:
    img_f = cv.cvtColor(cv.imread(path), cv.COLOR_BGR2RGB)

    ret, frame = cap.read()
    h,w,c = frame.shape
    img_f = cv.resize(img_f,(w,h))
```

```
    if ret:
        img = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        img = pose_segmentation(img,img_f)
        img = cv.cvtColor(img, cv.COLOR_RGB2BGR)

        cv.imshow("Salida", img)
        if cv.waitKey(1) & 0xFF==ord("q"):
            break
    else:
        break
cap.release()
cv.destroyAllWindows()
```

0.7.1 Resultado en Video



1 FIN

[ ]: