Tratamiento de Imágenes por Computadora

Detección del contacto bebé-objeto

En colaboración con CICEA

Estudiante:
Matías Fernández Lakatos

Profesor:
Álvaro Gómez

Facultad de Ingeniería
Universidad de la República
24 de julio de 2019

Índice

1. Introducción

Este proyecto se encuentra enmarcado en un proyecto del Centro Interdisciplinario en Cognición para la Enseñanza y el Aprendizaje (CICEA)[1]. En CICEA se propusieron crear un espacio que permita el estudio ecológico del comportamiento sensoriomotriz del niño a través del uso de múltiples cámaras de alta resolución temporal, micrófonos y algoritmos de visión por computadora; apostando a su vez, al trabajo y desarrollo de proyectos interdisciplinarios como éste.

2. Objetivos

En esta instancia se desea automatizar una tarea dentro del estudio de las reacciones del bebé. Se intenta obtener sin necesidad de intermediarios los instantes en los que el bebé toca al objeto, por cuánto tiempo lo hace, si lo llega a sujetar y por cuánto tiempo.

3. Equipos y Materiales

3.1. Espacio Físico

La obtención de los datos se realiza en el Laboratorio de Aprendizaje en Primera Infancia CICEA (LabAPI-CICEA) situado en José Enrique Rodó 1839 bis. Este laboratorio puede apreciarse en la figura 1, y en la figura 2 está una vista ampliada de la cámara a utilizarse.

Figura 1: Laboratorio de Aprendizaje en Primera Infancia CICEA (LabAPI-CICEA).

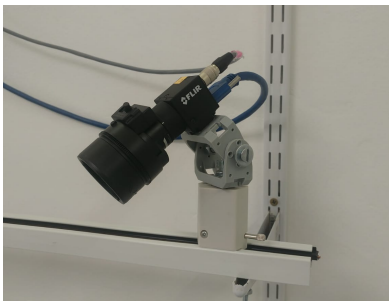

Figura 2: Montaje de la cámara.

3.2. Equipo disponible para el proyecto

El laboratorio cuenta con tres cámaras Flir Blackfly S, ver figura 3:

- Resolución 1280x1024px.
- Máximo frame rate 170fps (manteniendo tamaño imagen).
- Sensor a color.
- Conector datos y alimentación USB3.1

[1] `https://www.cicea.ei.udelar.edu.uy/`

- Conector analógico
- Tamaño del buffer 240 MB
- Totalmente configurables mediante el uso de la librería Spinnaker de Flir.
- Cuenta con un interfaz de visualización de nombre Sinview.

Figura 3: Cámara Flir Blackfly S.

La computadora que procesa los datos tiene las siguientes características:

- Tarjeta de Video NVidia GTX1060
- Tarjeta USB 3.1 - 4 puertos, bus independientes
- 32GB RAM
- SSD 256GB - con SO Ubuntu 18
- HDD WD Blue 1TB (Procesamiento)
- HDD WD Purple 4TB (Guardado)
- HDD WD Purple 4TB (Respaldo)

4. Montaje Experimental

No debe haber presentes objetos de color similar, si bien es capaz de distinguir entre tonalidades distintas mejor no forzar al código. El bebé deberá situarse a la derecha de la madre, considerando la visión de las cámaras. La cámara que se encuentra enfrentada a la puerta de entrada (cámara número 10) tendrá una perspectiva que tomará al bebé como debajo de la madre. Todas las cámaras se encuentran fijas y no se mueven para distintas tomas.

5. Código base

Para el análisis de los videos se utilizó como base el código proporcionado por Adrian Rosebrock presente en su página web[2]. Más específicamente el código para el *Ball Tracking with OpenCV*[3]. El mismo fue modificado para que se adaptara a nuestra situación pero utiliza la misma idea.

6. Objetos

Los objetos serán simplificados a un punto, el centroide. A continuación vemos cómo extraer este punto del video. Es por esto que las distancias al objeto serán medidas desde este punto.

[2] `https://www.pyimagesearch.com/author/adrian/`
[3] `https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/#`

6.1. Espacio de color de los objetos

Previo al análisis se definen los rangos de colores en el espacio de color HSV de los distintos objetos que participarán en los videos. Puede realizarse esto incluso con una imagen extraída del video a analizar. Para esto se utilizó el programa de *Range-Detector.py* disponible en github [4]. El mismo consiste en ir variando los límites en el espacio de color HSV (permite también hacerlo en el espacio RGB) hasta quedarse sólo con el objeto. Luego, estos límites son introducidos en el algoritmo de seguimiento de objetos y serán los que definan al objeto a seguir. De esta manera podremos separarlo del resto del video. Es por esta razón que no deben aparecer elementos que se encuentren dentro del subespacio de color asociado a cualquiera de los objetos que se desean seguir. Por lo menos no deben aparecer de forma tal que ocupen un lugar mayor al objeto en cuestión.

6.2. Máscara

Una vez aplicada la primera máscara de color haremos una sucesión de funciones que erosionarán y dilatarán los espacios que hayan caído dentro del subespacio de color del objeto. Primero se erosiona para eliminar cualquier tipo de ruido pequeño que se encuentre en el video, después se dilata para que el objeto llegue a un tamaño típico del objeto.
La máscara de color del objeto debe presentar un tamaño considerable para no ser eliminado por la erosión.
Con el fin de visualizar lo que estamos realizando se despliega la máscara en otra ventana.
A partir de la máscara obtenemos el centroide que vamos a considerar como el centro del objeto. Otra razón más para no introducir elementos con un código de color similar al objeto seguido.
El centroide es desplegado como un punto rojo en el video y también se genera una figura amarilla que envuelve el objeto si sus dimensiones son mayores a un cierto tamaño. Esto último se hace con la función cv2.minAreaRect para los objetos con forma de prismas ó cv2.minEnclosingCircle [5] para aquellos que son más similares a una esfera o una dona.
A su vez, contamos con una cola de puntos rojos que siguen al centroide del objeto. Mientras haya registro del objeto la cola se irá diluyendo con el tiempo.

7. Esqueleto de las personas

Una vez obtenido el punto que representará la posición del objeto, el centroide, debemos encontrar aquel punto que represente la mano del bebé y así poder medir la distancia entre él y el objeto.
Utilizando el programa de OpenPose [6] podemos obtener el esqueleto de las personas. En otras palabras, tenemos puntos clave, como son las muñecas del sujeto de estudio, disponibles en todo el video como podemos ver en la figura 4.

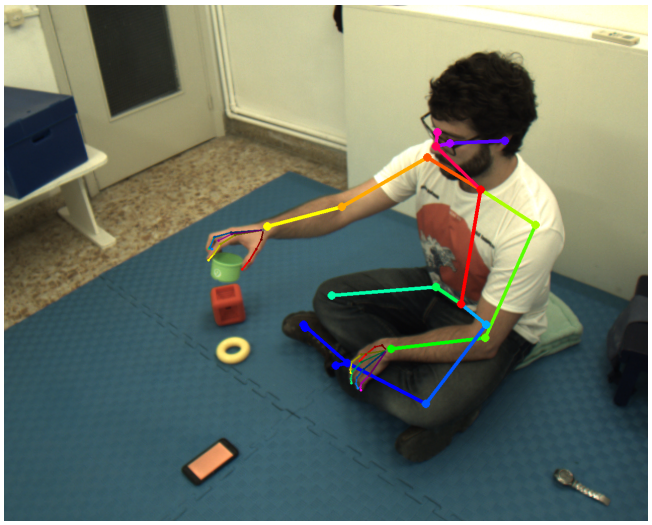

Figura 4: Captura de video con el esqueleto superpuesto.

Podemos apreciar los puntos de la mano que también son otorgados por OpenPose, la adquisición de estos datos analizando el esqueleto de un bebé es más complicado ya que presentan una confianza menor. Por esta

[4] https://github.com/jrosebr1/imutils/blob/master/bin/range-detector
[5] https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html
[6] https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/output.md

razón, utilizaremos los datos de sus respectivas muñecas. La visualización del algoritmo de seguimiento será de sólo los puntos de las muñecas, el objeto (su centroide) y el trazo que hace este último como podemos ver en la figura 5.

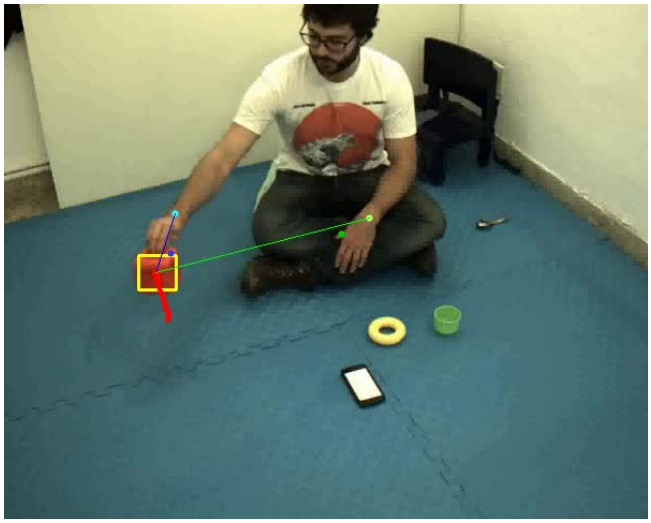

Figura 5: Captura de video con distancia muñeca derecha-objeto (azul) y distancia muñeca izquierda-objeto (verde). Datos de esqueleto sólo para muñecas y pulgares.

Debido a que en los videos originales aparece la madre con su bebé el programa OpenPose guardará los dos esqueletos. La información de los esqueletos fue otorgada por el laboratorio y se encuentra en archivos .json. Al momento de importar esta información hay que ser precavido. El bebé estará situado a la derecha de la madre, por lo que de haber dos elementos en el .json que guarda la información de los esqueletos tomaremos aquel en el que la distancia en el eje x de la nariz sea mayor. Esto vale para las cámaras 09 y 11, para la 10, debido a la perspectiva, tomaremos la información del esqueleto que tenga una coordenada y mayor. En caso de que haya información de un sólo esqueleto no imponemos condición. Tal vez en un trabajo posterior se pueda imponer una restricción para ver si en estos casos el esqueleto es el de la madre o del bebé.

8. Distancia objeto - muñeca del bebé

Se estudiará cada video por separado pero siempre tomando en cuenta que son diferentes perspectivas de la misma escena. Sabemos que de las tres cámaras, dos se encuentran enfrentadas, y todas distan unos 90^o de la más próxima, ver figura 1. Todas están aproximadamente a una altura entre 1,3 y 1,4 metros del piso. Los sujetos a estudiar estarán sentados por debajo de las cámaras. Esta disposición será importante debido a que si registramos un contacto bebé-objeto en al menos dos cámaras, tendremos un contacto efectivamente.
Cada vez que la distancia, μ, entre el objeto y la muñeca del bebé (izquierda y derecha) sea menor a un cierto valor fijo, η, se guardará la información de los segundos en que sucedió. Consideramos que hay contacto con el objeto si la distancia es menor a η. Tomamos como $\eta_{\max}$ a la distancia desde la esquina superior izquierda a la inferior derecha. De esta manera, medimos todas las distancias de forma tal que $\eta_{\max} = 100$ y logrando que se mantengan invariante las distancias frente al reescaleo de la pantalla. Es decir,

$$\mu = 100 \times \frac{\sqrt{\big(\text{centroide}[1] - \text{muneca}[1]\big)^2 + \big(\text{centroide}[0] - \text{muneca}[0]\big)^2}}{\sqrt{\text{length} + \text{width}}} \tag{1}$$

El valor de la distancia límite debe de ser mejorado ya que no toma en cuenta la profundidad. De todas formas, el problema no es tan grande ya que el espacio en donde se encuentra el bebé será similar en todas las grabaciones. Una vez que la restricción en la distancia se calibra, ésta será una muestra muy buena de lo que uno considera que el bebé se encuentra en contacto con el objeto. La forma de calibrarla es ayudada al mostrar en el video la distancia entre las muñecas y el objeto siendo seguido segundo a segundo. De esta manera, podemos medir la distancia cuando el bebé toque efectivamente al objeto en los videos guardados.
Si tomáramos la información de una sola cámara podríamos encontrar falsos positivos debido a que no considera la profundidad de los elementos. Por esta razón consideraremos que hay contacto solamente cuando quede registrado para dos cámaras. Siguiendo esta línea de razonamiento, tomaremos el inicio y final del contacto para los momentos en que no haya registro para al menos dos cámaras en el frame anterior y posterior, respectivamente.

Es necesario aclarar que de no modificar el programa los intervalos de tiempo en que se considera el contacto pueden aparecer diferenciados por milésimas de segundo. Como esto no es posible se define el intervalo de tiempo

intervalo_t de forma tal que si la diferencia entre el inicio del contacto siguiente y el final del contacto actual es menor a este valor, se unen los dos lapsos de tiempo. En este programa se tomó intervalo_t= 0,3.

9. Ejecución, entrada y salida

9.1. Ejecución y entrada

Para llamar al código se deben primero tener todos los paquetes necesarios (ver código). El código deberá estar en una carpeta en donde se encuentren:

- Una carpeta conteniendo los archivos .json
- Tantas carpetas como objetos a seguir con nombres siguiendo el siguiente estilo: imagenes_"color del objeto".
- Los tres videos de las cámaras con nombre vid_cam_<X> con X = 09, 10 y 11.

Luego, desde la terminal se debe escribir el siguiente código adaptado a la situación de cada uno: python <nombre del código.py> –video <link a la carpeta donde se encuentran los videos> –json <link a donde se encuentran los .json> –confianza 0.5 –escala 600 –buffer 42 –guardar si –objeto verde

Se agregaron varias opciones para modificar de la menor manera posible el código. Las opciones son las siguientes:

- –video ó -v: link a donde se encuentran los videos, éstos deben tener el siguiente formato: vid_cam_<X>.mp4 con X = 09, 10 y 11.
- –json ó -j: link a donde se encuentran los .json. Muy seguramente deban ser modificados en el código cómo llamarlos, en el actual código los .json tienen una estructura como la siguiente: 000000016300_rendered_182855<X>_keypoints.json, en este caso comienzan por el número 16300 y luego es rellenado de ceros hasta llegar a los doce dígitos.
- –confianza ó -c: es la confianza permitida para tomar los puntos tomados del código de OpenPose.
- –escala ó -e: es el reescaleo que se le hace al video con el único fin de visualizar mejor los videos.
- –buffer ó -b: El buffer son la cantidad de puntos guardados del centroide. Éstos sirven para ver el haz que sigue al objeto, en este código de color rojo.
- –guardar ó -g: Si uno pone la opción "imgs" salva todos los frames de los videos que se despliegan; si uno pone la opción "vid" salva los videos que aparecen. Debido a que puede haber un enlentecimiento debido al procesamiento de datos consideré mejor tener las dos opciones, una para luego generar un video fidedigno con el tiempo que transcurre y otro más visual e inmediato.
- –objeto ó -o: el objeto a seguir: "verde", "amarillo" ó "rojo".

9.2. Salida

Como se menciona en la subsección anterior, si se elije guardar los videos se guardarán con el siguiente formato: "outputcam<cámara>_e-<escala>_b-<buffer>_c-<confianza*10>_o-<objeto>.avi".
Ejemplo: "outputcam09_e-600_b-42_c-03_o-verde.avi" . Uno por cada cámara.
También se creará un archivo .dat con la información de los contactos para cada mano. Cuándo se inició, cuándo finalizó y cuánto duró. Un ejemplo de un archivo de nombre "data_objeto_verde.dat" me dio como resultado el siguiente contenido:

Para el objeto verde, con escala 600, buffer 42 y confianza 0.3:

Para la distancia objeto - muñeca derecha:
inicio,final,duración
7.560 8.600 1.040
10.440 10.560 0.120
14.680 16.640 1.960
17.040 20.000 2.960

Para la distancia objeto - muñeca izquierda:
inicio,final,duración
18.960 19.000 0.040

10. Pasaje de conjunto de imágenes a video

Al tener un conjunto de imágenes podemos generar un video utilizando el programa FFMPEG.
La línea de código para realizarlo en la terminal de linux es:

```
ffmpeg -framerate 60 -pattern_type glob -i 'outimg_cam<Y>_*.png' -c:v libx264 -pix_fmt yuv420p
outvid_cam_<X>.mp4
```

con X = 09, 10 u 11 e Y = 0, 1 ó 2 correspondiente a cada cámara. La opción "framerate" corresponde a los frames por segundo de la imagen. La toma de datos en el laboratorio corresponde a 60fps.

Si previamente se instaló conda, una error común suele aparecer con el comando x264, en ese caso es necesario colocar en la terminal de linux el siguiente comando:

```
conda install x264=='1!152.20180717' ffmpeg=4.0.2 -c conda-forge
```

11. Posibles mejoras

A lo largo del informe mencioné algunas mejoras que pueden hacerse al código actual. Aquí menciono también la posibilidad de eliminar el fondo de una manera inteligente. Una forma sería adaptar el código de Adrian Rosebrock titulado Basic motion detection and tracking with Python and OpenCV [7].
De esta manera en la pantalla se mostrarían solo los elementos que estén en movimiento, haciendo que los objetos aparezcan sólo al ser movidos, simplificando el análisis.

Debido a que no contamos con la información tridimensional de la posición de los objetos nuestras distancias serán euclideanas proyectadas sobre un plano. Una mejora sería contar con esta información, pero sin ésta este problema intenta ser solucionado con la condición de que al menos dos cámaras observen un contacto con el objeto. Observando los resultados resulta evidente que las distancias deben ser modificadas de acuerdo a la cámara que estemos observando.

[7] https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/ba
subtraction

12. Código

Proyecto_Matias_Fernandez_Lakatos.py

```
### INFORMACIÓN A TENER EN CUENTA:

# Los videos no deben poseer más de un objeto con igual color. Si bien es posible diferenciar diferentes tonalidades de un color es mejor no llevar al extremo el código.
# Una posible mejora sería combinar este trackeo de color con otro que use la textura del objeto.
# Al ser objetos sencillos no se pueden sacar muchos descriptores y por ende keypoints, por eso se trabaja con el color.
# Un perfeccionamiento para lograr obtener un mejor trackeo de los objetos sería entrenar a un algoritmo con cada objeto en diferentes posiciones y con incidencias de luz variadas, mostrándole cuál es el objeto en cada instante. Deep Learning.


#1# Los videos de las tres cámaras deben presentarse como vid_cam_09 / 10 / 11
#2# Los límites de los colores en el espacio HSV de los objetos fueron obtenidos con el archivo range-detector.py
#3# Los archivos .json tienen la siguiente estructura: '12 dígitos conteniendo el número de la imagen'+ '_rendered_18285509_keypoints' 'nada/_1/_2'.json Ejemplo: '000000000321_rendered_18285509_keypoints.json'
#4# En la terminal se deben poner el siguiente link y adaptarlo a cada sitio donde es arrancado. Abrir la terminal desde la carpeta donde se encuentra el archivo "tracker_objetos_varias_camaras_color-centroide.py":

# python Proyecto_Matias_Fernandez_Lakatos.py --video <link a la carpeta donde estan los videos> --json <link a la carpeta donde estan los archivos .json> --confianza 0.5 --escala 600 --buffer 42 --guardar imgs --objeto verde

# Las opciones para el objeto (o) son: 'rojo' 'amarillo' y 'verde'
# La confianza (c) es el valor que arroja los .json del OpenPose con respecto a una determinada predicción.
# La escala (e) es con el fin de mejorar la visualización para distintas máquinas
# El buffer (b) corresponde a cuán larga es la curva que sigue al centroide, en mi caso, de color roja.
# Opción disponible para guardar (g) o no los videos. "si": guarda.
###

# Paquetes necesarios

from collections import deque
from imutils.video import VideoStream
import numpy as np
import argparse
import cv2
import imutils
import time
import urllib.request as request
import json

# Llamar a las variables necesarias

ap = argparse.ArgumentParser()
ap.add_argument("-o","--objeto", type=str, required = True, help="objeto a trackear")
ap.add_argument("-v","--video", required = True, help="camino a la carpeta de videos")
ap.add_argument("-j","--json", required = True,help="path to the json's files")
ap.add_argument("-c","--confianza",type=float,default=0.2, help="parámetro de confianza para OpenPose")
ap.add_argument("-e","--escala", type=int, default=600, help="parámetro de re-escaleo")
ap.add_argument("-b","--buffer" , type=int, default=64 , help="max buffer size")
```

```
ap.add_argument("-g","--guardar", type=str, required = True, help="guarda video si se coloca la opción 'vid', guarda las imágenes de los frames si la opción es 'imgs'")
args = vars(ap.parse_args())


# Definimos los límites de color, en el espacio de color HSV, para cada objeto
# Lo hacemos con el archivo: "range-detector.py"

if args["objeto"]=='rojo':
  colorLower = (0, 187,0)   #RedLower
  colorUpper = (6, 244,255) #RedUpper
elif args["objeto"]=='amarillo': #v2
  colorLower = (22, 126,0)  #YellowLower
  colorUpper = (24,154,255) #YellowUpper
elif args["objeto"]=='verde': #v2
  colorLower = (46,82,0)  #VerdeLower
  colorUpper = (65,142,255) #VerdeUpper
elif args["objeto"]=='azul':
  colorLower = (92,39,84)   #VerdeLower
  colorUpper = (120,85,167) #VerdeUpper

# Puntos de la traza de centroides detectados. Uno para cada video.

pts = [ deque(maxlen=args["buffer"]), deque(maxlen=args["buffer"]), deque(maxlen=args["buffer"]) ]

# Importo los videos de las tres cámaras:

vs = [cv2.VideoCapture(args["video"]+'/vid_cam_09.mp4'), cv2.VideoCapture(args["video"]+'/vid_cam_10.mp4'), cv2.VideoCapture(args["video"]+'/vid_cam_11.mp4')]

# Descriptores de los videos (tomo cam09 como la representativa)

length = int(vs[0].get(cv2.CAP_PROP_FRAME_COUNT))
width  = int(vs[0].get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(vs[0].get(cv2.CAP_PROP_FRAME_HEIGHT))
fps    = vs[0].get(cv2.CAP_PROP_FPS)
print('cantidad de frames:',length,'largo del video:',width,'ancho del video:',height,'fps:',fps)

# Genero los elementos que harán la escala y el video que guardaré

Factor_escala =  args["escala"]
imgScale      =  Factor_escala/width
newX,newY     =  width*imgScale, height*imgScale
if args["guardar"]=='vid':
  fourcc        =  cv2.VideoWriter_fourcc(*'MP4V')   #esto sirve para que salga un video, pero más lento. Palabra clave: videoout
  out           =  [
cv2.VideoWriter(args["video"]+'/outputcam09_e-'+repr(args["escala"])+'_b-'+repr(args["buffer"])+ '_c-0'+repr(int(10*args["confianza"]))+'_o-'+args["objeto"]+ '.mp4',fourcc, 20.0, (int(newX),int(newY))),
cv2.VideoWriter(args["video"]+'/outputcam10_e-'+repr(args["escala"])+'_b-'+repr(args["buffer"])+ '_c-0'+repr(int(10*args["confianza"]))+'_o-'+args["objeto"]+ '.mp4',fourcc, 20.0, (int(newX),int(newY))),
cv2.VideoWriter(args["video"]+'/outputcam11_e-'+repr(args["escala"])+'_b-'+repr(args["buffer"])+ '_c-0'+repr(int(10*args["confianza"]))+'_o-'+args["objeto"]+ '.mp4', fourcc, 20.0, (int(newX),int(newY)))]    #esto sirve para que salga un video, pero más lento. Palabra clave: videoout


#iniciar conteos

tiempo_R_toca = np.zeros((length+1,3))
tiempo_L_toca = np.zeros((length+1,3))
```

```
97  cambia = 'si'     # Utilizo esta variable a modo de switch para graficar o no los
        puntos de OpenPose, la prendo al grabar nueva info en niño, la pago al finalizar
        el loop
98  restriccion_distancia = 8   # Para distancias menores a ésta considero que el bebé
        toca el objeto.
99
100 # Loop para las tres cámaras
101 for j in range(3):
102   counter_frames = 0
103   while True:
104     # Toma el frame del video
105     oriframe= vs[j].read()
106     oriframe = oriframe[1]
107
108     # Si es el final del video, que rompa el while
109     if oriframe is None:
110       break
111
112     # Sumamos un frame más ya que continuamos en el loop
113     counter_frames += 1
114
115     # (resize the frame)
116     frame = cv2.resize(oriframe,(int(newX),int(newY)))
117     # Borroneamos para hacer un pasa alto en las frecuencias: blur
118     blurred = cv2.GaussianBlur(frame, (11, 11), 0)
119     # Pasamos al espacio de color HSV
120     hsv     = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
121
122     # Construimos una máscara con los límites impuestos en el espacio de color
123     mask= cv2.inRange(hsv, colorLower, colorUpper)
124
125     # a series of dilations and erosions to remove any small
126     # blobs left in the mask
127 # Estas dos líneas son análogas a cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel).
128 # Mediante dilataciones y erosiones removemos las pequeñas burbujas que quedan en la
        máscara
129 # Aquí se presenta la solución al problema de los puntos falsos positivos de la
        máscara de color que es distinto para cada objeto.
130 # Es más artesanal este paso, y adaptado a cada objeto.
131
132     if args["objeto"]   == 'rojo':
133       mask = cv2.erode(mask,  None, iterations=3)
134       mask = cv2.dilate(mask, None, iterations=3)
135     elif args["objeto"] == 'amarillo' or args["objeto"] == 'verde':
136       mask = cv2.erode(mask,  None, iterations=3)
137       mask = cv2.dilate(mask, None, iterations=3)
138     #Para visualizar lo que ve el algorítmo como máscara:
139     cv2.imshow("Mask", mask)
140
141     # Encuentra los contornos de la máscara
142     cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
143     # SIMPLE (guarda menos elementos, más rápida) / NONE
144     # Toma la versión correcta de OpenCV
145     cnts = imutils.grab_contours(cnts)
146     center = None
147     radius = None
148     rect   = None
149
150     # only proceed if at least one contour was found
151     if len(cnts) > 0:
152       # Encuentra el controno más grande de la máscara y lo utiliza para encontrar
            la mínima figura correspondiente al objeto y su respectivo centroide. Por
            ejemplo, un rectángulo o un círculo
153       c = max(cnts, key=cv2.contourArea)
154       # Sabemos la figura que se adapta a cada objeto
155       if args["objeto"]=='rojo':
156         rect = cv2.minAreaRect(c)
```

```
157       elif args["objeto"]=='amarillo' or args["objeto"]=='verde':
158         ((x, y), radius) = cv2.minEnclosingCircle(c)
159
160       # moments trae el centroide:
161       # cx = int(M['m10']/M['m00']) & cy = int(M['m01']/M['m00'])
162       M = cv2.moments(c)
163       center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
164       # Dibujo el centroide de color Rojo (0,0,255) [BGR]
165       cv2.circle(frame, center, int(10*imgScale), (0, 0, 255), -1)
166
167       # Procede si radio mayor a un valor,
168       if radius is not None:
169         if radius > 10*imgScale:
170           # Dibuja un círculo amarillo (0, 255,255) centrado en el centroide
171           cv2.circle(frame, (int(x), int(y)), int(radius),(0, 255,255), 2)
172       # o altura AND ancho mayor a un valor.
173       if rect is not None:
174         if rect[1][0]>10*imgScale and rect[1][1]>10*imgScale:
175           box = cv2.boxPoints(rect)
176           box = np.int0(box)
177           # Dibuja un rect amarillo (0, 255,255) centrado en el centroide
178           cv2.drawContours(frame,[box],0,(0,255,255),2)
179
180
181       # Actualiza los puntos de trackeo (queue). Lo hace tal que los va colocando a
              la izquierda, por eso la defincion de thickness más adelante.
182       pts[j].appendleft(center)
183
184     # Loop en los puntos de trackeo. Acá hago el haz que se va desintegrando
185     for i in range(1, len(pts[j])):
186
187       # Si alguno de los dos últimos en None, los ignoro
188       if pts[j][i - 1] is None or pts[j][i] is None:
189         continue
190
191       # Si no, defino el ancho de la línea que conecta los puntos de forma tal que
192       # cuanto más "viejos" sean los puntos más se achiquen.
193       thickness = 1+int(np.sqrt(args["buffer"] / float(i + 1)) * 2*imgScale)
194       cv2.line(frame, tuple(pts[j][i-1]),tuple(pts[j][i]), (0,0,255), thickness)
195
196
197     ##################  JASON FILES ##################
198
199     # Archivos: 000000000 número de archivo _rendered_18285509_keypoints  _1 _2 .json
200
201     # Hay 12 dígitos de números, pongo el número de la imagen y lleno de ceros hasta
          completar los 12 dígitos
202     num_archivo = repr(counter_frames-1).zfill(12)
203     if j==0:
204       with open(args["json"]+'/'+num_archivo+'_rendered_18285509_keypoints.json') as
            f:
205         data = json.load(f)
206     elif j==1:
207       with open(args["json"]+'/'+num_archivo+'_rendered_18285509_keypoints_1.json')
            as f:
208         data = json.load(f)
209     else:
210       with open(args["json"]+'/'+num_archivo+'_rendered_18285509_keypoints_2.json')
            as f:
211         data = json.load(f)
212
213     # Extraigo valores del json:
214     persona_1 = data['people'][0]
215     pose_keypoints_2d_1 = persona_1["pose_keypoints_2d"]
216     pos_cabeza = 1 # posición en el archivo .json
217     xc1 = pose_keypoints_2d_1[int((pos_cabeza-1)*3)]
218     yc1 = pose_keypoints_2d_1[int((pos_cabeza-1)*3+1)]
```

```
    # Prueba ver si hay datos de otro esqueleto y compara con el que ya está para ver si el segundo es el niño o es simplemente el primero
    try:
      persona_2 = data['people'][1]
      pose_keypoints_2d_2 = persona_2["pose_keypoints_2d"]
      xc2 = pose_keypoints_2d_2[int((pos_cabeza-1)*3)]
      yc2 = pose_keypoints_2d_2[int((pos_cabeza-1)*3+1)]
      # En la cámara 10, la opuesta a la puerta, el niño tiene la condición de que está a una y menor, no una x mayor.
      if (xc1-xc2<0) and j != 1:  # Si xc1 < xc2 -> me quedo con xc2 (niño derecha)
        niño   = data['people'][1]
      elif (yc1-yc2<0) and j == 1:  # Si yc1 < yc2 -> me quedo con yc2 (niño abajo)
        niño   = data['people'][1]
    except:
      niño   = data['people'][0]

    pose_keypoints_2d = niño["pose_keypoints_2d"]
    hand_left     = niño["hand_left_keypoints_2d"]
    hand_right    = niño["hand_right_keypoints_2d"]
    if cambia == 'si':
      # Cabeza:
      xc = pose_keypoints_2d[int((pos_cabeza-1)*3)]
      yc = pose_keypoints_2d[int((pos_cabeza-1)*3+1)]
      cc = pose_keypoints_2d[int((pos_cabeza-1)*3+2)]
      cabeza_pos = (int(xc*imgScale),int(yc*imgScale))
      cv2.circle(frame, cabeza_pos, int(8*imgScale), (255,255,255), -1) #Blanco

    # Pulgar de la mano derecha:
    pos_pulgar_RHand = 4 # posición en el archivo .json
    xpRH = hand_right[int((pos_pulgar_RHand-1)*3)]
    ypRH = hand_right[int((pos_pulgar_RHand-1)*3+1)]
    cpRH = hand_right[int((pos_pulgar_RHand-1)*3+2)]
    # Junto con la imagen, las posiciones se ajustan a la nueva escala.
    #Factor_escala = 600
    #imgScale = Factor_escala/width
    RHand_pulgar_pos = (int(xpRH*imgScale),int(ypRH*imgScale))
    cv2.circle(frame, RHand_pulgar_pos, int(8*imgScale), (255,51,51), -1)     #azul claro

    # Pulgar de la mano izquierda:
    pos_pulgar_LHand = 4
    xpLH = hand_left[int((pos_pulgar_LHand-1)*3)]
    ypLH = hand_left[int((pos_pulgar_LHand-1)*3+1)]
    cpLH = hand_left[int((pos_pulgar_LHand-1)*3+2)]
    LHand_pulgar_pos = (int(xpLH*imgScale),int(ypLH*imgScale))
    cv2.circle(frame, LHand_pulgar_pos, int(8*imgScale), (0 ,204, 0), -1)     #verde claro

    # Muñeca derecha:
    pos_RWrist = 5
    xRW = pose_keypoints_2d[int((pos_RWrist-1)*3)]
    yRW = pose_keypoints_2d[int((pos_RWrist-1)*3+1)]
    cRW = pose_keypoints_2d[int((pos_RWrist-1)*3+2)]
    RWrist_pos = (int(xRW*imgScale),int(yRW*imgScale))
    cv2.circle(frame, RWrist_pos, int(8*imgScale), (255,255,51), -1)    #celeste

    # Muñeca izquierda:
    pos_LWrist = 8
    xLW = pose_keypoints_2d[int((pos_LWrist-1)*3)]
    yLW = pose_keypoints_2d[int((pos_LWrist-1)*3+1)]
    cLW = pose_keypoints_2d[int((pos_LWrist-1)*3+2)]
    LWrist_pos = (int(xLW*imgScale),int(yLW*imgScale))
    cv2.circle(frame, LWrist_pos, int(8*imgScale), (102,255,178), -1)     #verde agua

    ############################################################
```

```
281     # Defino la distancia de forma tal que si estan los objetos lo más alejado
            posible da 100.
282     if center is not None and center != (0,0):
283
284       if cRW > args["confianza"]:
285         cambia == 'si'
286         # Distancia objeto-muñeca derecha
287         dist_obj_Rwrist = 100*np.sqrt((RWrist_pos[0]-center[0])**2 +
                (RWrist_pos[1]-center[1])**2 )/(np.sqrt(newX**2+newY**2))
288         cv2.line(frame, RWrist_pos,(int(center[0]),int(center[1])), (255,0,0),
                int(4*imgScale))  #En azul
289
290         # Guardo los puntos potenciales al contacto bebé-objeto:
291         if dist_obj_Rwrist < restriccion_distancia:
292           tiempo_R_toca[ counter_frames][j] = counter_frames/fps
293           cv2.putText(frame, "TOCA", (int(newX*0.02),  int(newY*0.8)),
294           cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
295
296         info_R = ["Dist MD-O", dist_obj_Rwrist]
297         text_R = "{}: {}".format(info_R[0],info_R[1])
298         cv2.putText(frame, text_R, (int(newX*0.02),  int(newY*0.9)),
299         cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
300
301       if cLW > args["confianza"]:
302         cambia == 'si'
303         # Distancia objeto-muñeca izquierda
304         dist_obj_Lwrist = 100*np.sqrt((LWrist_pos[0]-center[0])**2 +
                (LWrist_pos[1]-center[1])**2 )/(np.sqrt(newX**2+newY**2))
305         cv2.line(frame, LWrist_pos,(int(center[0]),int(center[1])), (0,255,0),
                int(4*imgScale))  #En verde
306
307         if dist_obj_Lwrist < restriccion_distancia:
308           tiempo_L_toca[counter_frames][j] = counter_frames/fps
309
310         info_L = ["Dist MI-O", dist_obj_Lwrist]
311         text_L = "{}: {}".format(info_L[0],info_L[1])
312         cv2.putText(frame, text_L, (int(newX*0.02),  int(newY*0.95)),
313         cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
314
315       else:
316         cambia == 'no'
317
318     # Muestra el video y la máscara.
319     cv2.imshow("Frame", frame)
320
321     # Exporto el frame si la opción guardar es 'imgs'
322     if args["guardar"]=='imgs':
323       outfile = args["video"]+'/imagenes_'+args["objeto"]+'/outimg_cam'+repr(j)+'_'+
              repr(counter_frames).zfill(12)+'.png'
324       cv2.imwrite(outfile, frame)
325       outmask =
              args["video"]+'/imagenes_'+args["objeto"]+'/outmask_cam'+repr(j)+'_'+
              repr(counter_frames).zfill(12)+'.png'
326       cv2.imwrite(outmask, mask)
327
328     # Escribo el video de salida si se pone la opción 'vid'
329     if args["guardar"]=='vid':
330       out[j].write(frame) #esto sirve para que salga un video, pero más lento.
              Palabra clave: videoout
331
332     cv2.moveWindow('Frame',  0  ,0) # x horizontal(izq-der), y vertical(arr-aba)
333     cv2.moveWindow('Mask' , 700 ,0)
334     key = cv2.waitKey(1) & 0xFF
335
336     # Si la tecla 'q' es presionada, termina el loop de esta cámara
337     if key == ord("q"):
338       print('cantidad de frames hasta ahora:',counter_frames)
```

```
      break

vs[2].release()
# Cierra todas las ventanas
cv2.destroyAllWindows()

# Supongo que el video no empieza con el bebé tocando un objeto
# tiempo_RL_toca es una variable auxiliar para simplificar código, el for que recorre las 'k' es para tomar tanto la muñeca derecha como la izquierda en un sólo pedazo de código.
tiempo_RL_toca= [tiempo_R_toca,tiempo_L_toca]

# contacto[0] será para la muñeca derecha, contacto[1] para la izquierda
contacto = [[],[]]
for k in range(2):
  tiempo_contacto = tiempo_RL_toca[k]
  for i in range(int(length)):
    # Todos tendrán la misma información, el tema es ver si son diferentes a cero
    # A continuación vemos si dos para al menos dos cámaras el bebé toca el objeto
    a, b, c     = tiempo_RL_toca[k][i-1,0], tiempo_RL_toca[k][i-1,1], tiempo_RL_toca[k][i-1,2] #pasado
    #print(a,b,c)
    aa, bb, cc  = tiempo_RL_toca[k][i+0,0], tiempo_RL_toca[k][i+0,1], tiempo_RL_toca[k][i+0,2] #presente
    aaa,bbb,ccc = tiempo_RL_toca[k][i+1,0], tiempo_RL_toca[k][i+1,1], tiempo_RL_toca[k][i+1,2] #futuro
    # Si la cámara 'a' y la cámara 'b' detecta contacto, entonces...
    if (aa*bb != 0):
      # Si en el paso anterior no hubo contacto para UNA o NINGUNA CÁMARA entonces estamos en el inicio del contacto (un contacto no es considerado contacto)
      if  (sum([a==0,b==0,c==0])>=2):
        inicio = aa # puede ser bb también.
      # Si en el próximo ínidice termina el video, entonces
      if i+1 == int(length):
        # Si no detecta ningún contacto en alguna cámara, se termina ya
        if (aaa+bbb+ccc == 0):
          final = aa
        # De lo contrario, termina el video tocando el objeto
        else:
          if (aaa !=0):
            final = aaa
          elif (bbb !=0):
            final = bbb
          else:
            final = ccc
        contacto[k].append([inicio,final,final-inicio])
        break
      elif (sum([aaa==0,bbb==0,ccc==0])>=2):
        final = aa
        contacto[k].append([inicio,final,final-inicio])
    elif (aa*cc != 0):
      # Si en el paso anterior no hubo contacto en NINGUNA CÁMARA entonces estamos en el inicio del contacto
      if  (sum([a==0,b==0,c==0])>=2):
        inicio = aa # puede ser bb también.
      # Si en el próximo ínidice termina el video, entonces
      if i+1 == int(length):
        # Si no detecta ningún contacto en alguna cámara, se termina ya
        if (aaa+bbb+ccc == 0):
          final = aa
        # De lo contrario, termina el video tocando el objeto
        else:
          if (aaa !=0):
            final = aaa
          elif (bbb !=0):
            final = bbb
          else:
```

```
            final = ccc
        contacto[k].append([inicio,final,final-inicio])
        break
      elif (sum([aaa==0,bbb==0,ccc==0])>=2):
        final = aa
        contacto[k].append([inicio,final,final-inicio])
    elif (bb*cc != 0):
      # Si en el paso anterior no hubo contacto en NINGUNA CÁMARA entonces estamos en el inicio del contacto
      if  (sum([a==0,b==0,c==0])>=2):
        inicio = bb # puede ser bb también.
      # Si en el próximo ínidice termina el video, entonces
      if i+1 == int(length):
        # Si no detecta ningún contacto en alguna cámara, se termina ya
        if (aaa+bbb+ccc == 0):
          final = bb
        # De lo contrario, termina el video tocando el objeto
        else:
          if (aaa !=0):
            final = aaa
          elif (bbb !=0):
            final = bbb
          else:
            final = ccc
        contacto[k].append([inicio,final,final-inicio])
        break
      elif (sum([aaa==0,bbb==0,ccc==0])>=2):
        final = bb
        contacto[k].append([inicio,final,final-inicio])

# Si el final de un contacto y el inicio del siguiente se diferencian en un 'intervalo_t' entonces uno los dos segmentos.
intervalo_t = 0.3 #segundos
contacto_mod = [ [] , [] ]
for k in range(2):
  contacto[k].append([length/fps+10,length/fps+10,length/fps+10]) #Esto lo hago por un problema de programación que no encuentro. De no hacerlo no me toma los últimos puntos cuando junto los tiempos que distan menos de intervalo_t
  B = np.zeros(len(contacto[k])-1)
  for i in range(len(contacto[k])-1):
    B[i] = contacto[k][i+1][0]-contacto[k][i][1] < intervalo_t #Relaiso la resta entre el inicio del frame i+1 con el final del frame i
  i = 0
  inicio = contacto[k][0][0]
  final  = contacto[k][0][1]
  if len(contacto[k])==2:
    contacto_mod[k].append([inicio,final,final-inicio]) #Si hay dos veces que toca, una será el elemento agregado:
  else:
    while i < len(B):
      i+=1
      if B[i-1]==1:
        final =  contacto[k][i][1] #Voy a tomar el siguiente por si es falso B
      else:
        contacto_mod[k].append([inicio,final,final-inicio])
        inicio = contacto[k][i][0] #Voy a tomar el siguiente por si es falso B
        final =  contacto[k][i][1] #Voy a tomar el siguiente por si es falso B


# Imprimo los valores en la terminal
print('Para la mano derecha:')
if len(contacto_mod[0])==0:
  print('No toca la mano derecha el objeto')
for i in range(len(contacto_mod[0])):
  print('inicio:',contacto_mod[0][i][0], 'final:', contacto_mod[0][i][1], 'duración:', contacto_mod[0][i][2])

```

```
print('Para la mano izquierda:')
if len(contacto_mod[1])==0:
  print('No toca la mano izquierda el objeto')
for i in range(len(contacto_mod[1])):
  print('inicio:',contacto_mod[1][i][0], 'final:', contacto_mod[1][i][1],
      'duración:', contacto_mod[1][i][2])

# Genero el archivo .dat
# Guardo los inicios, finales y duraciones
header = "Para el objeto "+args["objeto"]+", con escala " +repr(args["escala"])+", buffer "+repr(args["buffer"])+" y confianza 0."+repr(int(10*args["confianza"]))+":"

f = open('data_objeto_'+args["objeto"]+'.dat', 'wb')
np.savetxt(f, [], header=header)

# Defino los textos a colocar en el archivo
string  = ["Para la distancia objeto - muñeca derecha:"]
string2 = ["Para la distancia objeto - muñeca izquierda:"]
infidu  = ["inicio,final,duración"]
espacio = [" "]

np.savetxt(f,espacio,fmt="%s")
np.savetxt(f,string,fmt="%s")
np.savetxt(f,infidu,fmt="%s")
for i in range(len(contacto_mod[0])):
    data = np.column_stack((contacto_mod[0][i][0],
        contacto_mod[0][i][1],contacto_mod[0][i][2]))
    np.savetxt(f, data,delimiter='  ',fmt='%1.3f')

np.savetxt(f,espacio,fmt="%s")
np.savetxt(f,string2,fmt="%s")
np.savetxt(f,infidu,fmt="%s")
for i in range(len(contacto_mod[1])):
    data = np.column_stack((contacto_mod[1][i][0],
        contacto_mod[1][i][1],contacto_mod[1][i][2]))
    np.savetxt(f, data,delimiter='  ',fmt='%1.3f')

f.close()
```