“PROGRAMACION DE UN ROBOT COLABORATIVO PARA USO COMO COMPAÑERO DE JUEGOS DE MESA”

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Curso 2018-2019

Trabajo Fin de Grado

AUTOR:

Mario Salvador Muñoz

TUTOR:

Carlos Ricolfe Viala

ÍNDICE

ÍNDICE DE IMÁGENES

1-OBJETIVO

El objetivo de este proyecto consiste en la programación e implementación de un sistema autónomo capaz de jugar contra una persona a un juego de mesa, siendo este juego el Jenga.

El sistema deberá poder reconocer las piezas de la torre y las piezas que se han extraído mediante una cámara, además tendrá que poder extraer las piezas mediante un brazo robot, de forma que se pueda jugar una partida contra él sin necesidad de la presencia de otro jugador.

Palabras Clave: visión artificial, brazo robot, Python, juegos.

2-MEMORIA

ANTECEDENTES

La motivación para el desarrollo de este proyecto es por el hecho está aumentando el uso de sistemas robóticos utilizados en campo de la salud, en el caso de los hospitales estos sistemas se pueden usar como asistentes a la hora de realizar operaciones, como puede ser el robot daVinci [1], usado en distintos tipos de operaciones, cuyo uso permite realizar operaciones menos invasivas, reduciendo el dolor postoperatorio y reduciendo el tiempo de hospitalización.

a b

Imagen 1 Robot daVinci (a) y su control (b), Hospital General Universitario de Valencia [2]

También hay sistemas que se están usando para la rehabilitación de los pacientes, de forma que al usarlos se puede mejorar la calidad de la rehabilitación y reducir el tiempo que se le dedica a la misma, ya que el paciente puede dedicar más tiempo de rehabilitación sin la necesidad de otra persona que sea la que lleve a cabo la rehabilitación, un ejemplo puede ser el robot IMBiO3R [3], que están desarrollando investigadores del Instituto ai2, con el instituto de Biomecánica de Valencia y el centro de Investigación de ingeniería Mecánica de la UPV, donde dicho robot realizará una rehabilitación de tobillo y rodilla permitiendo personalizar los tratamientos y adaptarse a las características del paciente y a su evolución.

Imagen 2 Robot IMBiO3R desarrollado por la UPV [4]

En el caso de la gente que padece enfermedades mentales, ya sean enfermedades como el Alzheimer, la demencia senil, etc. Se suelen usar tratamientos en los que se usan pasatiempos, juegos de estrategia, cuadernos de ejercicios, o terapias que implicar que el paciente esté acompañado, todo ello con la finalidad de estimular el cerebro evitando la atrofia de este, y que dichas enfermedades se agraven.

JUSTIFICACIÓN

Ya que para jugar a la gran mayoría de juegos de mesa se necesita la participación de un mínimo dos jugadores, ya sean juegos como: el ajedrez, el parchís, el dominó, etc. El paciente sólo podrá jugar en el caso de que haya un enfermero/a u otro paciente que juegue contra él, además, en el caso de que el paciente tenga movilidad reducida por lo cual tenga una mayor dificultad para manipular dichos juegos, los pacientes también deberán de depender de otras personas para que estas realicen los movimientos, por lo que en el caso de que no pueda haber una persona que les ayude, no podrán jugar, o jugarán con mayor dificultad.

Esto se puede solucionar con el uso de ordenadores, teléfonos, tabletas, etc. Cualquier plataforma donde de forma virtual se puede jugar a casi cualquier juego, eliminado la necesidad de jugar contra otra persona. Pero uno de los problemas que se encuentran, es que, de esta forma las personas de edades avanzadas que, al no tener mucha soltura con las nuevas tecnologías, les puede ser difícil el empezar a usar estos juegos, por lo que puede ser más probable que pierdan motivación para jugar a dichos juegos, además siempre es más gratificante poder jugar a un juego de este tipo de forma real por la interactuación con el juego.

Con este sistema se pretende dar mayor accesibilidad a los juegos para la rehabilitación de los pacientes con problemas de memoria, ya sea tanto en hospitales o en residencias, de forma que son ellos los que jueguen sin necesidad de otra persona, pero manteniendo la interacción con el juego. Este programa se va a desarrollar para jugar al Jenga, pero se podría adaptar fácilmente para jugar a la mayoría de los juegos de mesa, de forma que con un solo sistema se pueda cambiar entre varios juegos.

SOLUCIÓN ADOPTADA

Para la realización de este proyecto se utilizará un brazo robot colaborativo, ya que la persona que juegue contra él va a estar dentro del radio de acción del brazo robot, por lo que es necesario el uso de este tipo de robot. Además, este robot nos ofrece un control por fuerza, que se usará para saber que fuerza se está aplicando para retirar las piezas.

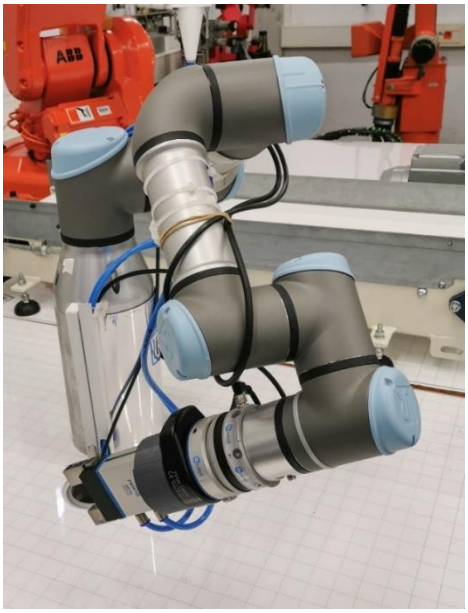

Imagen 3 Robot Colaborativo UR3

Imagen 4 Modulo HEX-E para el control de fuerza de la marca Onrobot

Para manipular las piezas se utilizará la pinza neumática DHPS-20-A de la marca FESTO, que lleva instalado el robot, ya que tiene el tamaño adecuado para poder empujar y agarrar las piezas de madera, ya que cuando la pinza está cerrada la distancia entre los dedos es de 17mm, y al abrir la pinza es de 30mm, por lo que la pieza cabe perfectamente, además la dimensión de los dedos es de 17,5x12mm, mientras que la pieza mide 25x15mm, por lo que puede empujar la pieza sin tocar las piezas colindantes. Para usar esta pinza se necesita un compresor de aire, elementos de mantenimiento (secador, filtro y lubricador), tubos para conectar los distintos elementos y una válvula electrónica.

Imagen 5 Pinza FESTO

Para el sistema de visión se procederá a usar la cámara de la marca ROBOTIQ, que lleva incorporado el propio brazo robot en su muñeca, ya que esto significa el uso de menos componentes externos y menor montaje que el usar una cámara externa. Esta cámara se conectará directamente al ordenador, ya que dispone de una conexión USB. La cámara permite usar varias resoluciones, vamos a usar una resolución de 640x480 pixeles, que permite tener una correcta definición de las piezas sin tener una carga muy pesada a la hora de analizar las imágenes.

Imagen 6 Cámara UR3

Para la parte de programación se utilizará el lenguaje Python, ya que es más sencillo para programar a la hora de trabajar con listas de vectores, por parte del robot, dado que el utilizado es el UR3 de Universal Robots se programará en su código propio, el URScript.

La comunicación entre el robot y el ordenador se realizará con una comunicación socket, en la que el ordenador actuará como servidor y el robot como cliente, ambos deberán estar conectados a la misma red.

Para simplificar el sistema de visión se hará uso de un fondo negro, de forma que sea más fácil identificar las piezas.

Como torre se usará un Jenga adquirido en una juguetería.

Imagen 7 Caja del Jenga [5]

DESARROLLO DE LA SOLUCIÓN ADOPTADA

Programación en Python:

Primero se ha de desarrollar el software que reconocerá el Jenga, para ello se ha optado por reconocer las piezas que quedan en posición horizontal a la vista de la cámara, ya que, a diferencia de las otras piezas estas son prácticamente lisas, es decir, las betas de la madera no se aprecian a comparación de las otras piezas que son muy irregulares. El robot tomará una posición fija a la hora de capturar las imágenes, esta posición se llamará "P_cam". Para obtener esta posición se han probado varias posiciones para la torre, en las que se ha comprobado que el robot pueda acceder a todas las piezas de Jenga, tanto por la parte delantera como por la trasera, además de que pueda realizar los movimientos sin colisionar con la torre, para ello ha sido necesario el uso de una base, como se ve en la imagen 8, para que la cámara pueda situarse en dicha posición si colisionar con la mesa a la hora de tomar la posición "P_cam" además de dar accesibilidad a la pinza para extraer las piezas inferiores. Para que se coordine con el robot se espera a recibir un valor desde el UR3 como confirmación de que esté situado en "P_Cam"

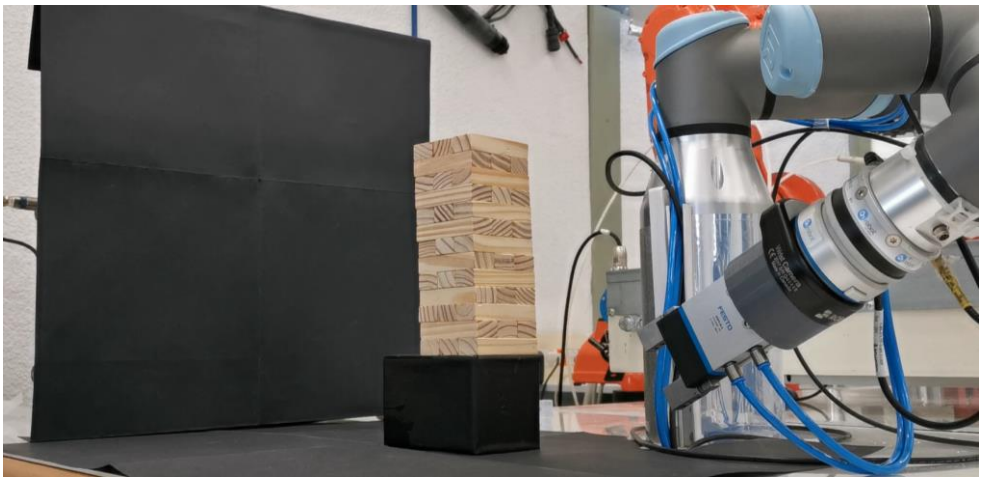

Imagen 8 UR3 situado en P_Cam

Se obtiene la siguiente imagen a partir de la cual se detectarán las piezas:

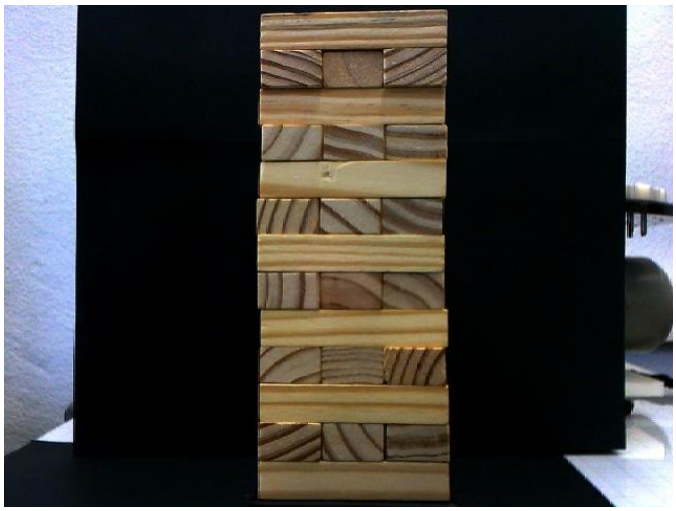

Imagen 9 Vista de la cámara

Para reconocer estas piezas primero se segmentará la imagen, con la función threshold() de forma que los valores de los pixeles sean 0 o 254, con un valor de umbral de 62, para las condiciones de iluminación del laboratorio.

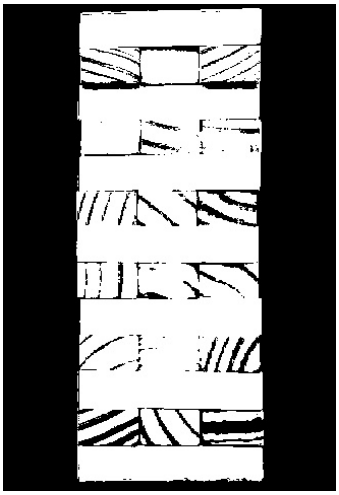

Imagen 10 Jenga segmentado

-Esta parte del programa corresponde a la función: detec_piezas_horizontales():

Una vez segmentada la imagen se buscarán líneas horizontales, las cuales empezarán cuando el valor de los pixeles pase de 0 a 254, es decir, pasen de negro a blanco, y se mantendrá hasta que el valor de los pixeles pase de 254 a 0, de blanco a negro, usando un valor de corte para evitar las líneas pequeñas, dicho valor se ha establecido en 150 pixeles. También obtendremos la distancia media de todas las líneas, y se almacenará en la variable llamada media, para después utilizarla en otras funciones, como en la conversión de pixeles a milímetros. El programa empieza a analizar desde el origen de coordenadas de la imagen, situado la esquina superior izquierda, representado en la imagen 11, estando el eje X representado en rojo y eje Y en verde.

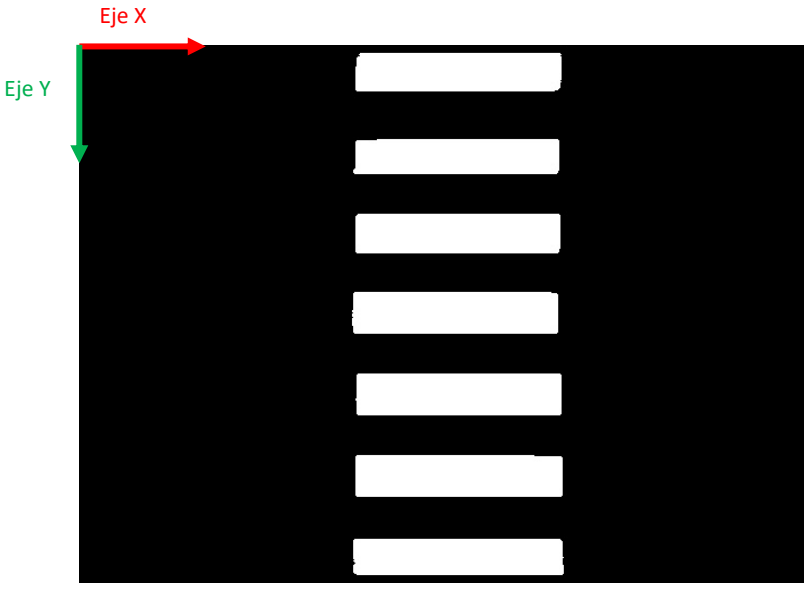

Imagen 11 Piezas horizontales con los ejes X e Y

-Esta parte del programa corresponde a la función: detectar_centroides():

Ahora que ya tenemos las piezas horizontales se buscará el centro de dichas piezas, llamado centroide, dichos centroides se guardaran en dos listas, cada lista corresponde a la posición X e Y del centroide en pixeles, teniendo en cuenta que el valor 0,0 corresponde con el origen de coordenadas de la imagen 11.

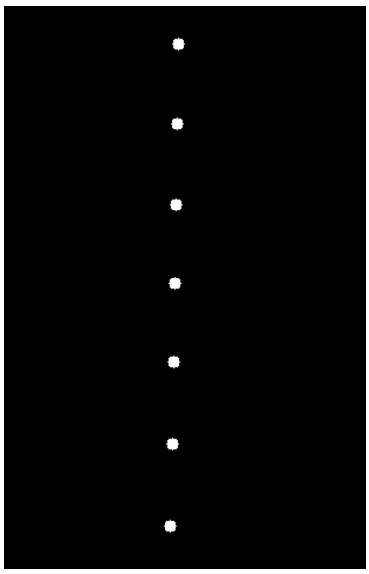

Imagen 12 Centroides detectados

Obtenemos los puntos correspondientes a los centroides, coordenadas X e Y en pixeles: Centroides [334.0, 451.0, 334.0, 380.0, 334.0, 308.0, 331.0, 236.0, 333.0, 165.0, 333.0, 98.0, 334.0, 23.0].

Se observa que los centroides se han guardado de forma que el primer valor de la lista corresponde al último centroide, es el que mayor valor tiene en la coordenada Y (451.0). Esto se debe tener en cuenta a la hora de trabajar con los centroides.

-Esta parte del programa corresponde a la función: dePixAmm():

Una vez obtenidos los centroides de las piezas horizontales hay que obtener los centros que corresponden a las piezas restantes del Jenga en la imagen, para ello hay que convertir la distancia que hay entre las piezas de la torre, en milímetros, que es la distancia que podemos obtener midiendo la torre física, a pixeles. Como en la función detec_piezas_horizontales() hemos obtenido la longitud media de las líneas, llamada media, que corresponde a la longitud de las piezas en pixeles, por lo que solo hay que convertir esta distancia al resto de dimensiones de la pieza.

Sabiendo que la longitud de una pieza es tres veces su anchura, la longitud es de 75mm, y la anchura es de 25mm. Si se divide el valor de la longitud media de las piezas en pixeles por tres, obtenemos la anchura media de las piezas en pixeles, por lo que la distancia en el eje X de cada pieza a los centroides es:

$$d\_lat = media/3$$

Mientras que para obtener la altura con la longitud necesitamos saber que la altura de las piezas es de 15mm, por lo que, si multiplicamos la longitud media por la altura y luego dividimos por la longitud en milímetros, obtenemos la altura en pixeles en el eje Y de cada pieza:

$$d\_vert = 15mm * media/75mm$$

De esta forma se obtienen los valores de anchura "d_lat" y altura "d_vert" de cada pieza.

-Esta parte del programa corresponde a la función: puntosAuxiliares():

Ahora, sabiendo que cara de la torre se está detectando se pueden aproximar los centros del resto de las piezas. Como los centroides se han guardado desde abajo hasta arriba la función que obtendrá los centros empezará con el ultimo valor de la lista de centroides, y acabará en el primer valor de la lista.

En el caso de que el lado sea el que tiene siete centroides, se suma la altura en pixeles a cada centroide para obtener la posición Y, luego para obtener la posición X de cada punto se ha de sumar, restar, o dejar igual el valor, según la pieza, empezando por el ultimo valor de la lista de centroides.

En la imagen se ha sumado la d_vert (representado como una flecha verde) a la posición del primer centroide, (punto blanco) y seguidamente se han creado tres puntos auxiliares(puntos verdes), el primer valor se ha obtenido restando el valor de la d_lat (representado como una flecha verde), el central manteniendo el valor y el derecho sumándolo.

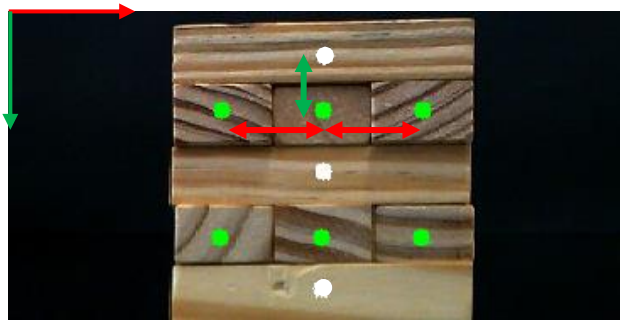

Imagen 13 Obtención de los puntos auxiliares lado 7

Repitiendo el proceso seis veces se obtiene el resto de los puntos auxiliares de la torre (véase imagen 16).

El caso con la cara con seis centroides similar al de siete, pero en este caso los valores se obtienen a partir del centroide siguiente, ya que no hay un centroide al inicio a partir del cual se puedan obtener los puntos auxiliares, por lo que al primer valor de centroide se le resta el valor d_vert para obtener el valor Y de dichos puntos, seguidamente se procede como en el caso anterior sumando, restando o dejando sin modificar el valor de la coordenada X. El problema es que de esta forma el último conjunto de puntos no se puede obtener, ya que no hay un centroide debajo de él para obtener los valores, por lo que dichos valores se obtienen fuera del bucle usando el centroide anterior. De esta forma se obtienen los puntos auxiliares (véase imagen 15)

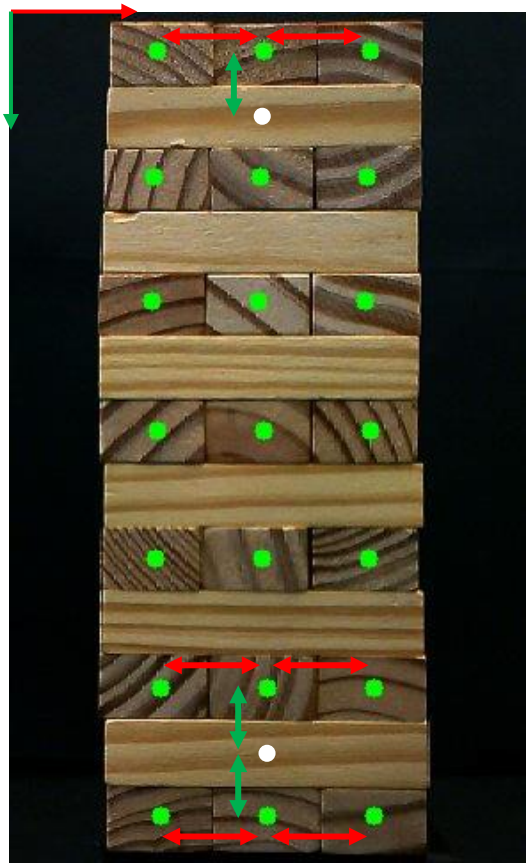

Imagen 14 Obtención de los puntos auxiliares lado 6

Con esto se obtienen los valores de los centros aproximados de las piezas según el lado de la torre:

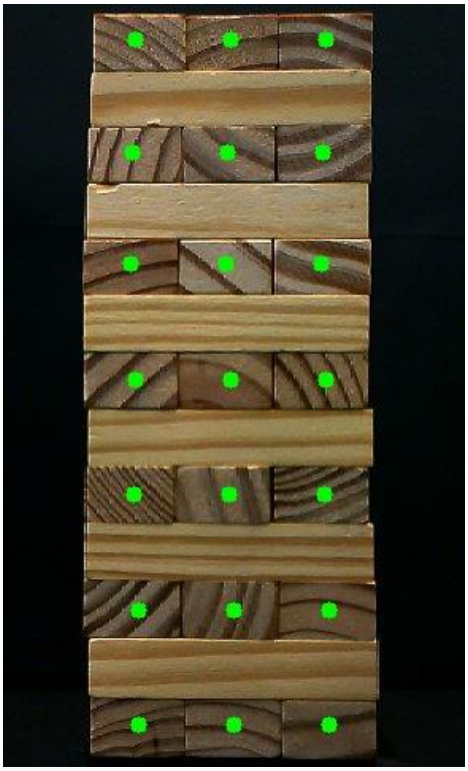

Imagen 15 Puntos auxiliares lado 6

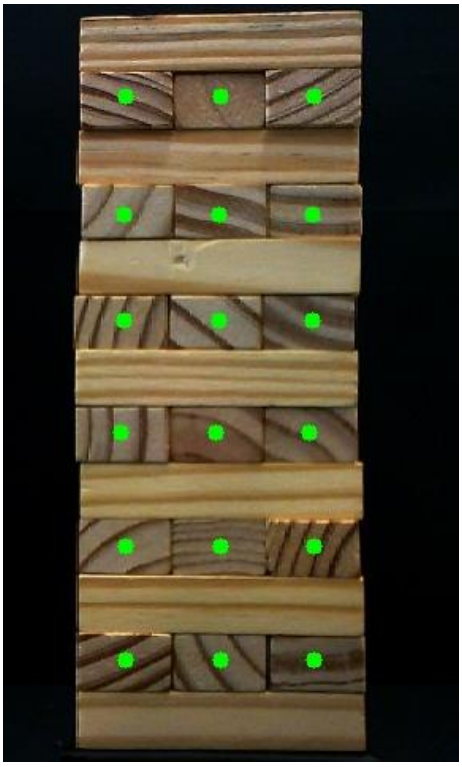

Imagen 16 Puntos auxiliares lado 7

-Esta parte del programa corresponde a la función: piezasRetiradas():

Una vez obtenidos los centros de cada pieza y haciendo uso de otra imagen segmentada, con un valor mucho menor de segmentación, a fin de obtener la forma de la pieza evitando las betas de la madera en las mismas.

Esta imagen se analizará para comprobar que en la posición de cada punto auxiliar hay o no una pieza, esto se hace creando un roi que tiene como centro dicho punto, y como valores de altura y anchura las dimensiones en pixeles de la pieza, d_lat y d_vert (aunque se han fijado los valores de la acnhura a 60 pixeles y de la altura a 40 pixeles, para trabajar números más exactos).

De esta forma se pueden contar los pixeles que hay dentro de cada roi para saber si hay pieza, este caso es afirmativo si hay una cierta cantidad de pixeles sabiendo que la dimesion del roi es de 60*40 pixeles, se obtiene un area total de 2400 pixeles, el valor de corte para decidir si hay una pieza se ha establecido en 2200 pixeles blancos, por lo que si el numero de pixeles blancos es menor se toma como que no hay pieza. Este valor se a obtenido mediante pruebas, ya que es dificil obtener un valor de corte más preciso, debido a que se está usando el mismo valor para todas las piezas, habiendo piezas situadas en el centro que dan mayor número de pixeles negros que las piezas situadas en las partes superior e inferior, que por la cercanía de la cámara no se obtiene el hueco de la pieza con claridad.

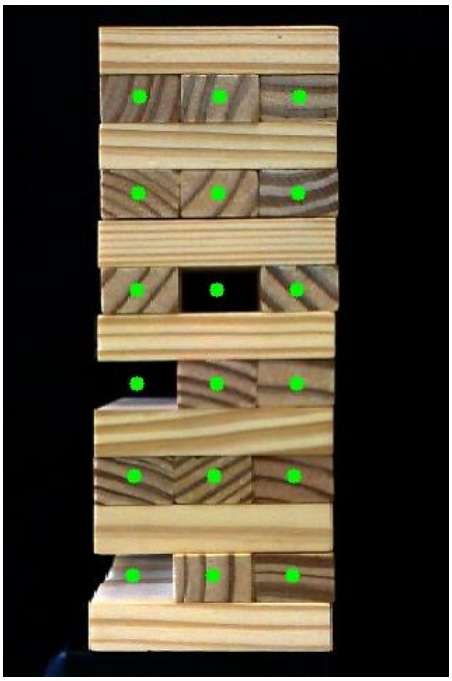

Imagen 17 Jenga con tres piezas retiradas

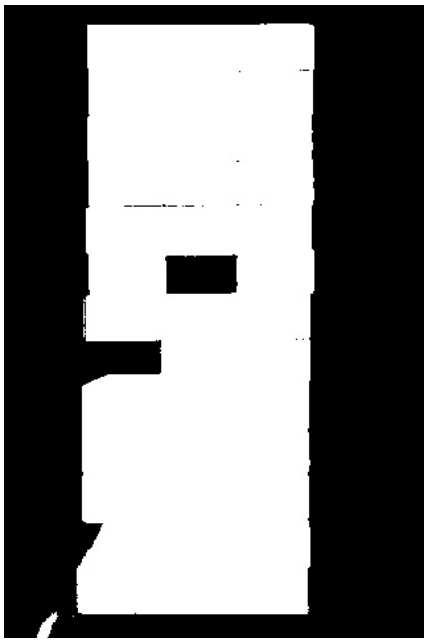

Imagen 18 Segmentación de la imagen 17

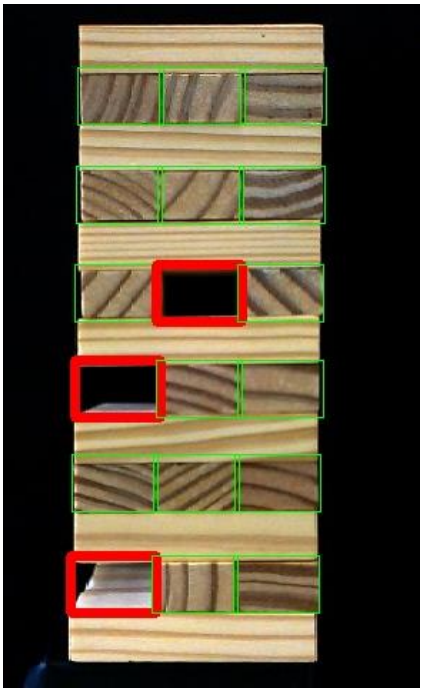

Imagen 19 Detección de la imagen 17

Valores de cada lista:

listaPuntosX [251.0, 250.0, 249.0, -1, 247.0, -1, 308.0, 307.0, -1, 306.0, 304.0, 303.0, 365.0, 364.0, 363.0, 363.0, 361.0, 360.0]

listaPuntosY [65.0, 134.0, 203.0, -1, 336.0, -1, 65.0, 134.0, -1, 270.0, 336.0, 407.0, 65.0, 134.0, 203.0, 270.0, 336.0, 407.0]

La cantidad de pixeles blancos en cada posición es (Teniendo en cuenta que el primer valor de la lista se considera en la posición 0):

Pixeles blancos [2322, 2254, 2252, 839, 2358, 2077, 2399, 2374, 859, 2351, 2400, 2400, 2315, 2362, 2328, 2311, 2359, 2398]

Posición 3 de la lista 839 pixeles blancos.
Posición 5 de la lista 2077 pixeles blancos.
Posición 8 de la lista 859 pixeles blancos.

En la quinta posición de la lista se observa, que incluso sin haber una pieza en el hueco, se ve parte de la pieza inferior por la posición de la camara, por lo que el número de pixeles aumenta mucho respecto a los otros dos valores, aparecen más del boble de pixeles blancos que en las otras dos posiciones, y sin embargo, en la posición 1 de la lista, la cantidad de pixeles blancos es de 2254 habiendo una pieza,

-Esta parte del programa corresponde a la función: seleccionPunto():

Para seleccionar la pieza que se va a retirar se han tenido en cuenta las tres posibilidades:

La primera, que el jugador no ha solicitado la ayuda del robot, por lo cual es el quien debe retirar la pieza, por lo que solo hace falta detectar la pieza que se ha retirado mediante una detección.

La segunda posibilidad es que el jugador haya solicitado la ayuda del robot, por lo que este será el ecnargado de quitar la pieza, por lo que el jugado debe señalar la pieza que quiere extraer haciendo doble clic con el botón izquierdo del ratón en la pantalla sobre la pieza deseada, seguidamente ha de pulsar enter para confirmar, dicho punto se comparará con los puntos auxiliares mediante la función puntoMasCercano para obtener el punto mas cercano a la posición del clic del ratón, que será el que se tome como pieza para extraer.

Finalmente el ultimo caso es que el robot haya de elegir un punto, este se elige de forma aleatoria entre los posibles puntos de la listas. Estos puntos se tomarán como validos siempre que su valor sea distinto de -1, es decir, que haya una pieza en dicha posición.

Ahora debemos saber si las piezas se pueden quitar o no, para ello primero analizaremos cada fila de piezas con la finalidad de encontrar las piezas que realmente se puedan extraer, sin que exista la posibilidad de derribar la torre por extraer una pieza que comprometa la integridad estructural de la torre, para ello existen cuatro casos de posiciones para las piezas:

El primero es que estén las tres piezas en la fila, por lo que a priori se puede extraer cualquiera de las piezas. (Fila de color verde en la imagen 20)

El segundo es cuando ya se ha retirado la pieza central, por lo que retirar cualquiera de las otras dos piezas provocaría que se derribará la torre. (Filas de color rojo en la imagen 20)

El tercer caso es cuando quedan dos piezas, la central y una de las dos laterales, en cuyo caso solo se podrá extraer la pieza situada en el lateral, ya que retirar la central provocaría el derrumbe de la torre. (Filas de color azul en la imagen 20)

El último caso es cuando solo resta la pieza central, la cual no se deberá de poder quitar, ya que esto implicaría la caída de la torre. (Fila de color amarillo en la imagen 20)

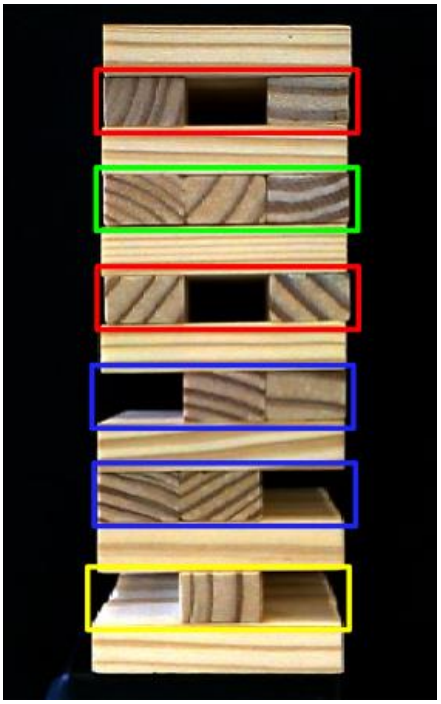

Imagen 20 Posibles agrupaciones de piezas

Finalmente, una vez elegido se procederá a enviar dicho punto al robot con la función "quitarPieza", este se envía al robot de dos formas, una para que el robot la empuje, a este punto se le suma o resta una distancia de 23mm en el eje X, (valor ajustado a mano) ya que como la pinza tiene dos dedos, y se toma como origen el centro de la pinza, para poder extraer la pieza está solo puede empujarla con uno de los dedos, por lo que se tiene que modificar dicha distancia, si el punto que se quiere quitar está a la izquierda se le restará esa cantidad haciendo que se use el dedo derecho para extraer la pieza, en el caso de que esté en lado derecho, se quitaría con el dedo izquierdo, y si la pieza está en el centro la empujará con el dedo derecho, por elegir uno de los dos, mientras que el punto para extraerla por la parte posterior sigue siendo el mismo, ya que la pieza queda en el centro de la pinza.

En este punto se inicia la parte del código del robot, ya que el ordenador una vez mandado el punto espera a recibir una respuesta por parte del robot, que puede ser que ha quitado la pieza, recibe un 1, o no ha quitado la pieza, recibe un 2. En el primer caso, se realiza una detección para comprobar que realmente se ha quitado la pieza, ya que el control de fuerza puede fallar y tomar como que ha empujado la pieza cuando no, o se ha podido soltar la pieza a la hora de extraerla. Si en el primer caso se ha retirado la pieza el ordenador enviará un 1 como confirmación al robot de que paletice la pieza, en caso contrario envía un 0 para que no paletice. En los casos que no se ha retirado la pieza se devuelve el turno al jugador correspondiente, para que seleccione otra pieza.

El programa se repite hasta que se activa la función "condicionFin". Esta función va asociada a la función de detección, donde en cada detección realizada se analiza la torre para saber si hay más piezas extraíbles, contando las filas no tienen más piezas extraíbles, y por cada fila que no tiene más piezas para retirar suma uno a la variable "val" y cuando valor de la variable es igual al número de dichas filas significa que no quedan más piezas por retirar, por lo que la variable fin tona el valor 1 y el programa salta directamente al final y espera a que el jugador cierre el programa pulsando la tecla enter.

Ejemplo de la función “condicionFin” con el lado con seis piezas horizontales:

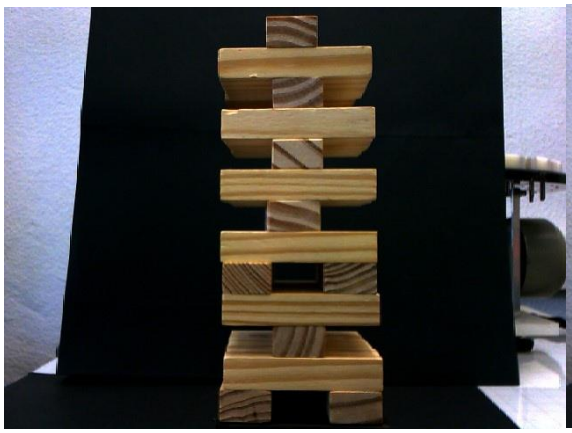

Imagen 21: Jenga (lado 6) sin piezas extraíbles

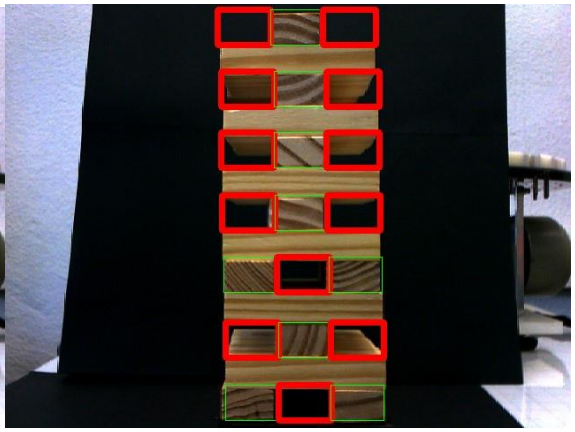

Imagen 22: Detección de la imagen 21

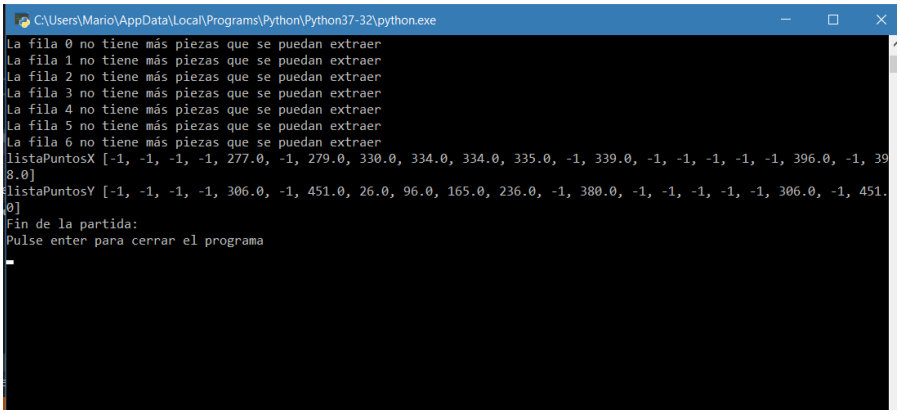

Imagen 23: Consola del programa para la imagen 22

listaPuntosX [-1, -1, -1, -1, 277.0, -1, 279.0, 330.0, 334.0, 334.0, 335.0, -1, 339.0, -1, -1, -1, -1, -1, 396.0, -1, 398.0]

listaPuntosY [-1, -1, -1, -1, 306.0, -1, 451.0, 26.0, 96.0, 165.0, 236.0, -1, 380.0, -1, -1, -1, -1, -1, 306.0, -1, 451.0]

Ejemplo de la función “condicionFin” con el lado con siete piezas horizontales:

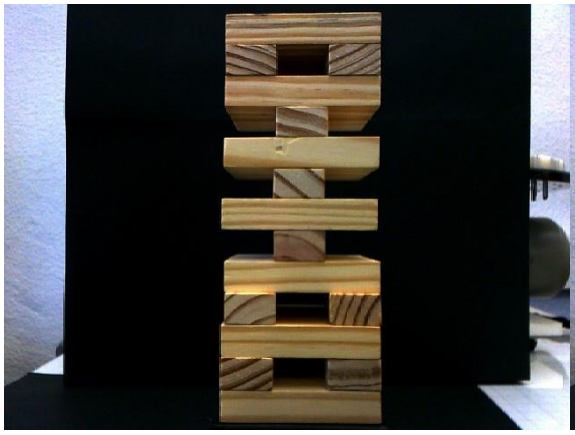

Imagen 24: Jenga (lado 7) sin piezas extraíbles

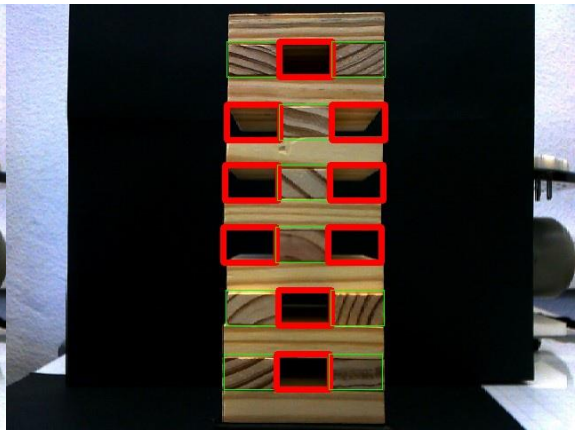

Imagen 25: Detección imagen 24

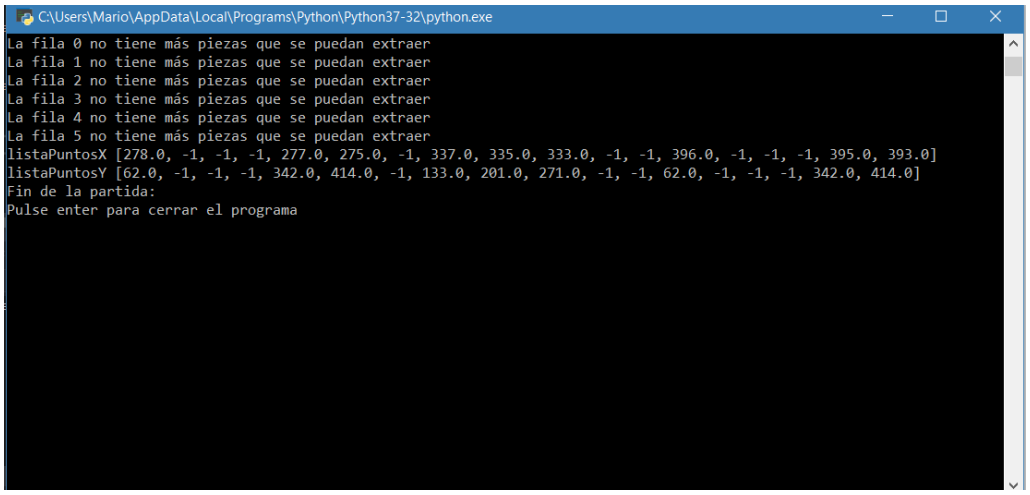

Imagen 26: Consola del programa para la imagen 25

listaPuntosX: [278.0, -1, -1, -1, 277.0, 275.0, -1, 337.0, 335.0, 333.0, -1, -1, 396.0, -1, -1, -1, 395.0, 393.0]

listaPuntosY: [62.0, -1, -1, -1, 342.0, 414.0, -1, 133.0, 201.0, 271.0, -1, -1, 62.0, -1, -1, -1, 342.0, 414.0]

En ambos casos, como se aprecia en las imágenes 23 y 26, el sistema detecta que no hay más piezas restantes, por lo que se muestra el mensaje “Pulse enter para cerrar el programa”.

Programación del UR3

Al iniciar el programa el robot abrirá la pinza y se ubicará en la posición "P_Colocar", posición en la cual una vez esté posicionado el robot colocaremos el Jenga de forma este que toque la pinza del robot, estando centrada respecto al Jenga, tal como se aprecia en la imagen 27. (En el código la función Colocar_Jenga es la encargada de llevar al robot a la posición, esta se invoca más adelante en el programa, pero cada vez que se ejecuta el programa obliga a llevar al robot a la primera posición dada, que es la que se da en Colocar_Jenga, por lo que incluso antes de realizar la conexión con el ordenador se posiciona en "P_Colocar")

Imagen 27: Robot situado en la posición Colocar_Jenga

En las imágenes 27, 28 y 29 se observa que a diferencia de la imagen 8, la base del Jenga se encuentra en posición vertical, esto se debe a que la imagen tomada es anterior a la versión final del programa, por lo que en esta versión la base se situaba de tal forma.

Una vez alcanzada esta posición se iniciará la conexión con el servidor, es decir, el ordenador y cuando se establezca dicha conexión el robot se moverá a la posición de detección, una vez llegue a dicha posición enviara al ordenador el valor 5 como confirmación de que ya ha llegado a dicho punto.

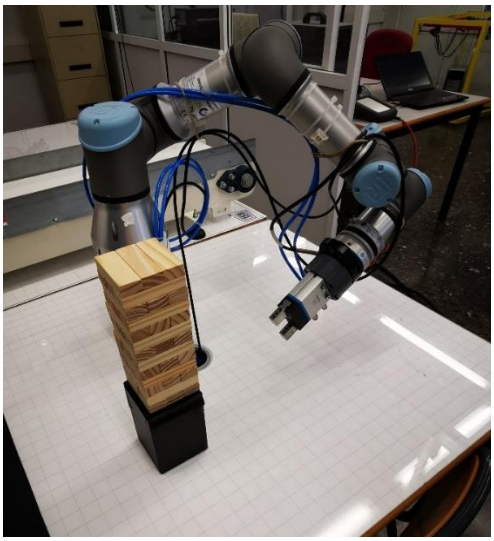

Imagen 28: Vista lateral de la posición P_Cam

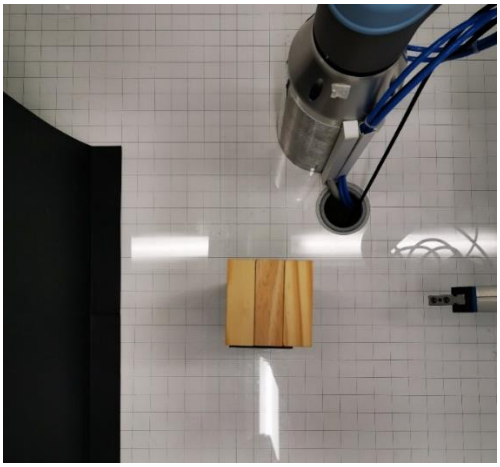

Imagen 29: Vista superior de la posición P_Cam

Una vez llegado a este punto, el robot espera a recibir una cadena de 4 valores que corresponden, o bien al punto de la pieza que se quiere extraer, o al valor [-1,-1,-1,-1], dicho valor activa el fin de la partida, por lo que al recibir este valor el robot finalizará el programa. En el programa se observa que en el valor de la variable coord aparece el valor 4 al inicio de la lista, esto es debido a que las listas en dicho programa tienen como primer valor la longitud de la lista, por lo que al haber 4 puntos aparece el 4, quedando el valor correspondiente al fin de la partida, por ejemplo, de la forma: [4,-1,-1,-1,-1]

En el caso que el robot reciba un punto, primero este se ha de transformar a las coordenadas del brazo, ya que se busca un desplazamiento respecto a un punto. Con el origen de coordenadas y la dirección de los ejes en la base del robot y en sus mismas unidades, por lo que primero se ha de dividir el valor entre 1000, ya que le estamos enviando los valores en milímetros y el robot trabaja con los puntos en metros, después lo que en la pantalla consideramos el eje X positivo, para el robot es el eje Y negativo, y el eje Y positivo del ordenador es el eje Z negativo para el robot.

Una vez transformados los puntos se ubicará en la posición "P_Empujar", que es la misma que "P_Colocar", pero separada cinco milímetros de la torre en el eje X, seguidamente el robot se desplazará en los ejes Z e Y la cantidad indicada, respecto a dicho punto.

puntoEmp:= p[0,-coord[1]/1000,-coord[2]/1000,0,0,0]

puntoQuit:= p[0,-coord[3]/1000,-coord[4]/1000,0,0,0]

Una vez alcanzada dicha posición se inicia el control por fuerza, donde el robot se moverá a lo largo del eje Z con los ejes y el origen de coordenadas del TCP una cantidad de 30mm, y si llega a dicha posición sin exceder los 5N indicados tomará como que ha empujado correctamente la pieza, por lo recorrerá 30mm hacia atrás para iniciar la extracción de esta. En el caso de que no detecte que ha empujado la pieza el robot vuelve a la posición de detección para esperar a que se elija otro punto.

Para extraer la pieza el robot se situará en la parte posterior del Jenga, y de la misma forma que para empujar la pieza se sitúa frente a la pieza que va a extraer, se acerca 33mm, cierra la pinza y la retira moviéndose hacia atrás 80mm, después recorrerá una serie de puntos fijos para poder situarse en la posición de detección sin colisionar con el Jenga. Una vez en este punto espera a recibir la confirmación de paletizar del ordenador, por lo que en caso afirmativo se invoca la función de Palé, donde se paletizan las piezas usando dos posiciones de referencia. La posición va cambiando según el valor de la variable j, que va aumentando cada vez que se retira una pieza.

Una vez paletizada la pieza, se da el valor [0,0,0,0,0] a la variable coord y el programa espera un nuevo valor para la variable.

CONCLUSIONES:

El objetivo inicial, jugar una partida contra el robot como si fuera un compañero de juego, se ha podido lograr.

El Jenga se ha tenido que modificar, ya que se han quitado cinco filas de piezas para que el robot pudiera acceder a todas las piezas, ya que al aumentar el número de filas el control de fuerza daba error por la existencia de puntos singulares, en los cuales no funcionaba, además el robot no podía acceder a todas las piezas. Todo esto se pude solucionar usando un robot de mayor tamaño, como el UR5.

Por la misma razón de que el robot no podía acceder a la torre completa se ha optado por hacer que las piezas extraídas no se sitúen en la parte superior de la torre como dicta el juego, ya que a medida que aumentaría la altura de la torre el robot dejaría de poder acceder a dichas piezas.

Se ha podido realizar un buen análisis de la imagen, aunque este podría ser mucho mejor, de forma que usando inteligencia artificial se podría hacer que se reconocieran directamente las piezas sin necesidad de tener que obtenerlas a partir de otras, y sin necesidad de tener un entorno tan controlado.

Se podría implementar el uso de una base giratoria motorizada, de forma que permitiera la extracción de las piezas de ambas caras, siendo el juego más completo.

El sensor de fuera tiene una sensibilidad muy baja para este uso, ya que las piezas inferiores si ofrecen una mayor resistencia y si se pueden detectar, pero en parte de los casos empuja las piezas situadas más arriba, ya que no ofrecen tanta resistencia como las inferiores, por lo que acaba empujando todas las piezas, esto se solucionaría con un sensor de fuerza más sensible.

Todos los movimientos se han realizado usando el origen de coordenadas situado en la base, hubiera sido más correcto realizar los movimientos respecto a la herramienta o TCP, pero esto no ha sido así debido a que dichos ejes de esta están mal configurados, y a la hora de realizar movimientos a lo largo de los ejes X e Y, la herramienta se desplazaba en diagonal, cosa que al desplazarse respecto a la base no ocurre.

3-Presupuesto

Material	**Modelo**	**Cantidad**	**Precio (€)**	**Total (€)**
Robot	UR3	1	19.750	19.750
Cámara	RWC-UR-KIT	1	5.400	5.400
Pinza	DHPS-20-A	1	565,86	565,86
Válvula	VUVG-LK14-M52-AT-G18-1H2L-S	1	58,16	58,16
Kit neumático	RB-And-48	1	262,01	262,01
Modulo F.R.L	AC20A-N02G-3CZ	1	56,70	56,70
Jenga	---	1	10	10
Fondo negro	---	2	1	2
TOTAL				26.104,73

Tabla 1 Precio de los materiales

"Precios de los materiales de tabla 1 obtenidos según referencias [6] [7] [8] [9] [10] [11]"

Concepto	**Descripción**	**Cantidad (h)**	**Precio (€)**	**Total (€)**
Estudio	Estudio de la viabilidad del proyecto	10	30	300
Desarrollo	Desarrollo del software	150	30	4500
Pruebas	Pruebas del software en la estación robotizada	30	30	900
TOTAL				5700

Tabla 2 Precio de la mano de obra

Coste total del proyecto:	31.804,73 €

4-ANEXOS

Flujogramas:

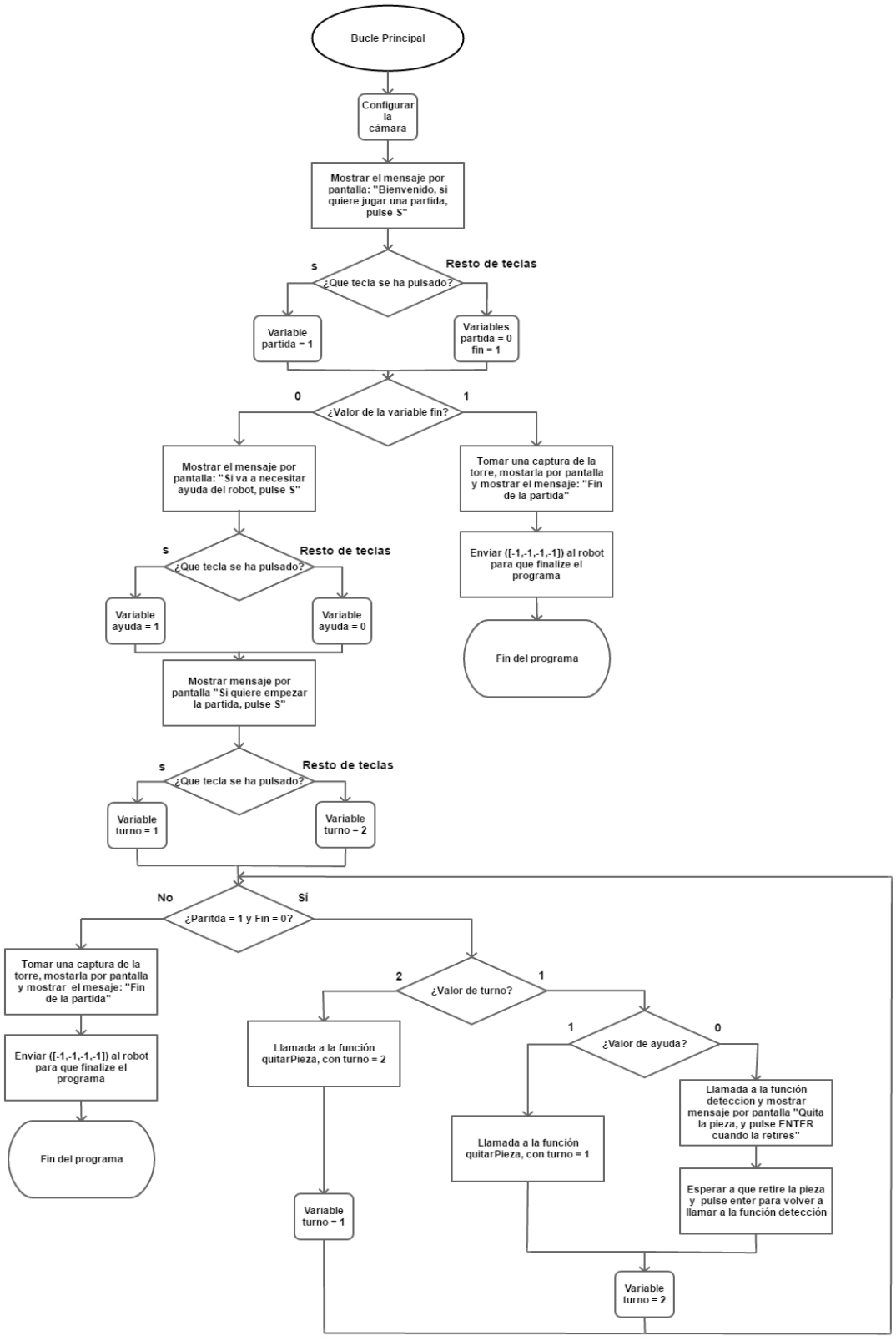

Imagen 30 Flujograma del bucle principal

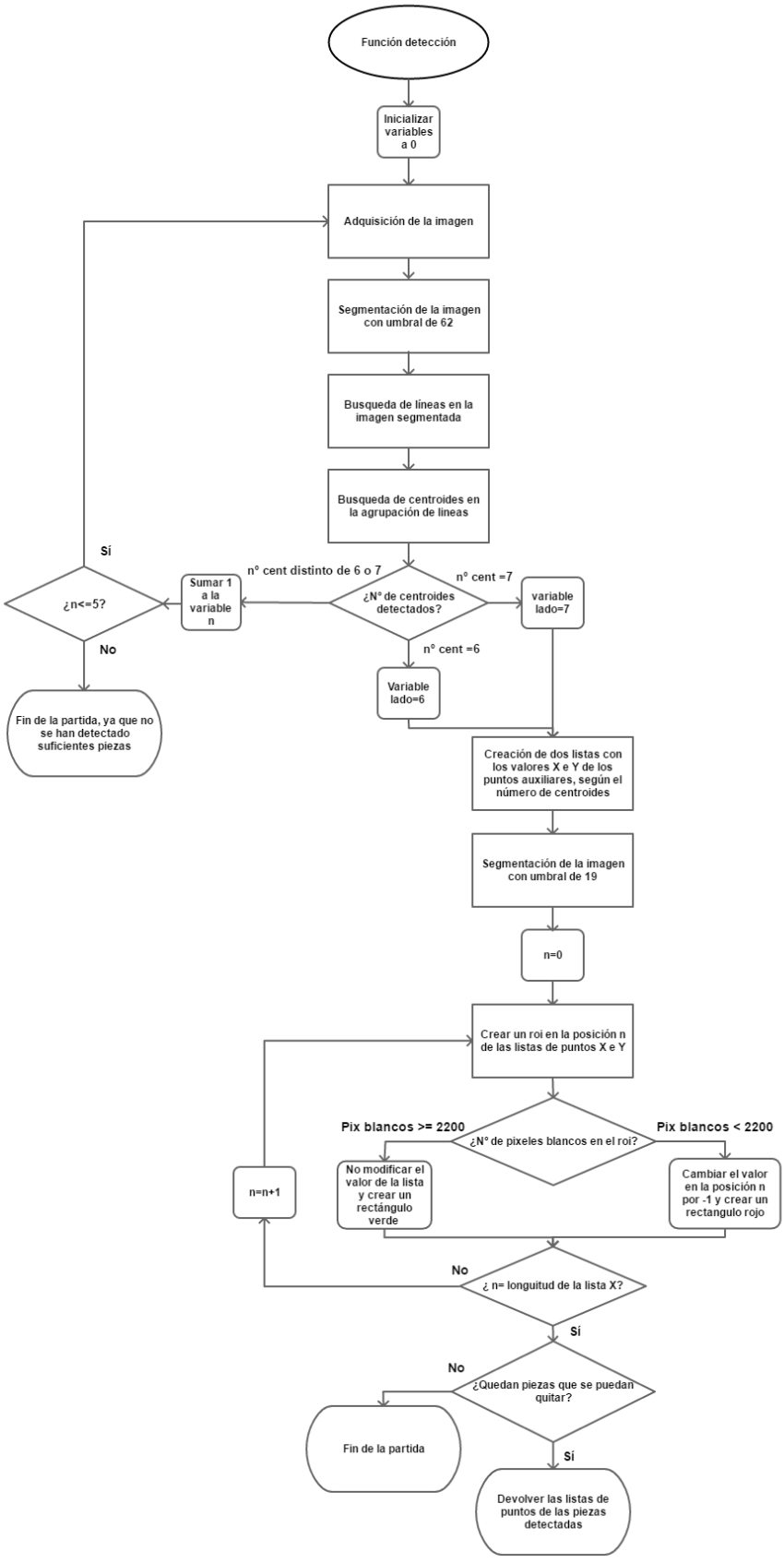

Imagen 31 Flujograma de la función detección

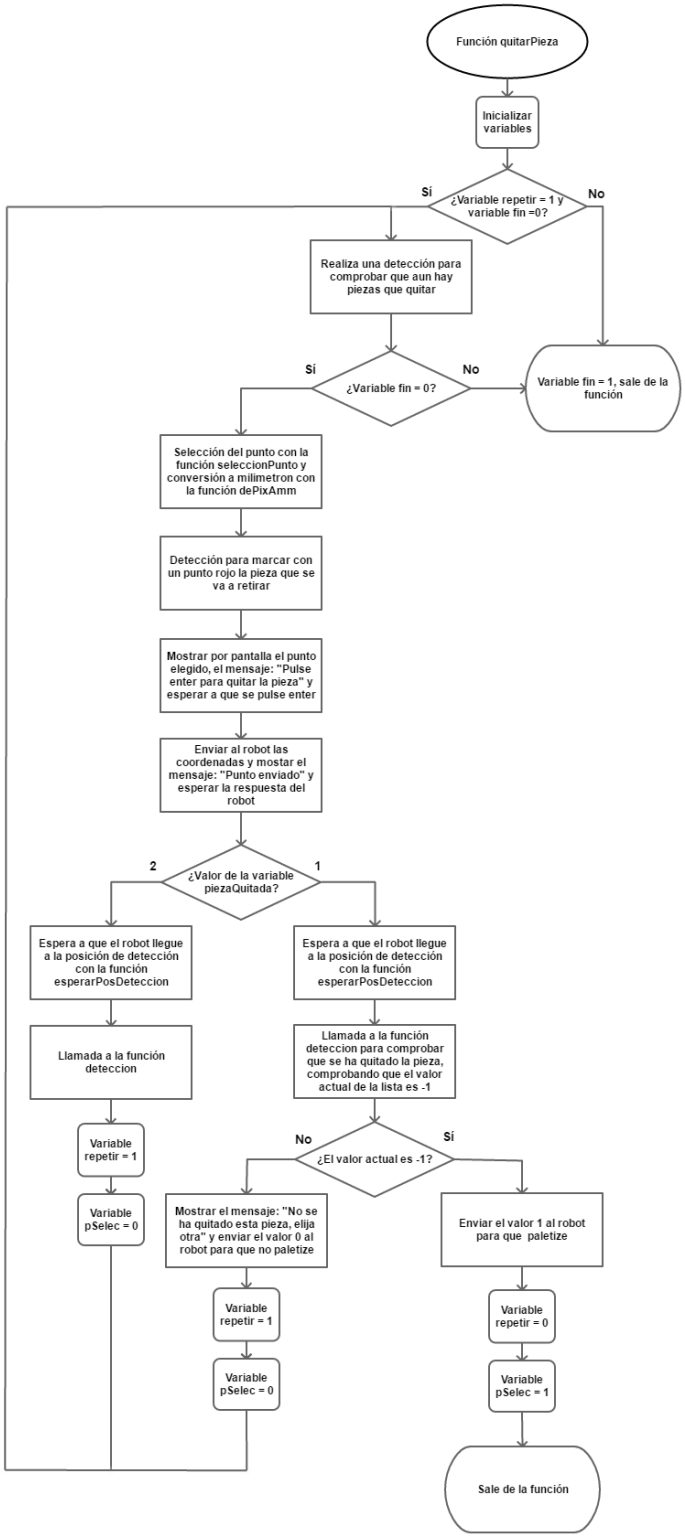

Imagen 32 Flujograma de la función quitarPieza

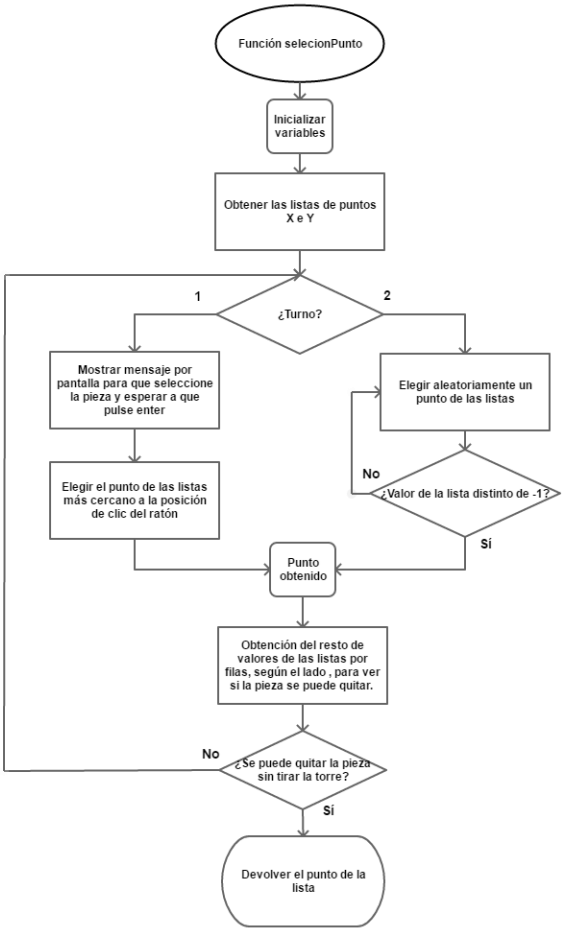

Imagen 33 Flujograma de la función seleccionPunto

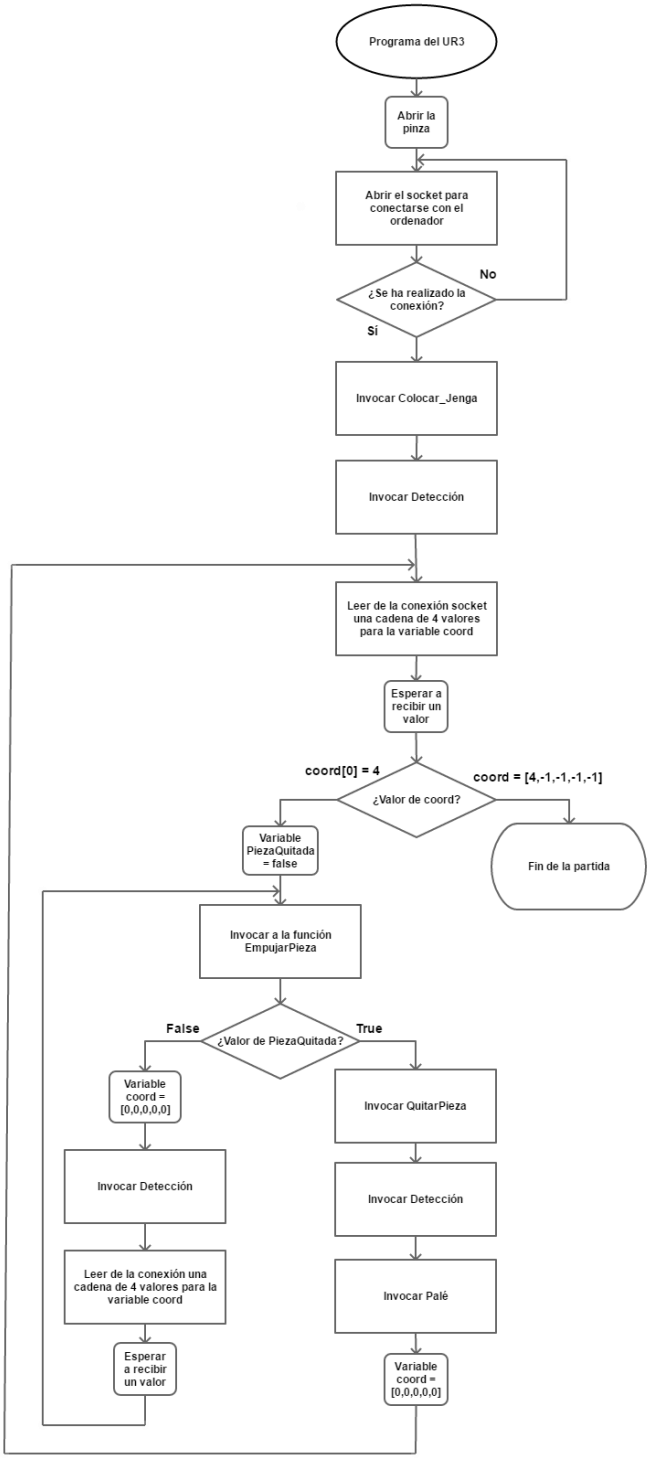

Imagen 34 Flujograma del Programa del UR3

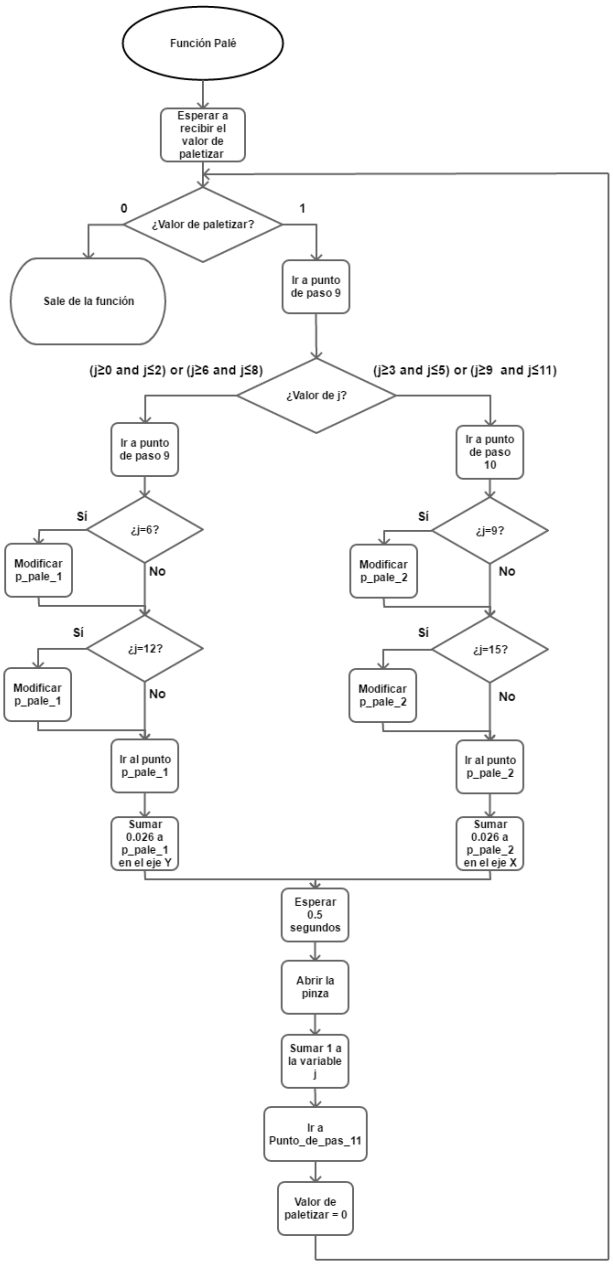

Imagen 35 Flujograma de la función palé

Manual de usuario:

Primero hay que preparar la zona de juego, para ello se debe colocar el fondo y la base, después se montará la torre del Jenga y de situará encima de la base (véase imagen 36).

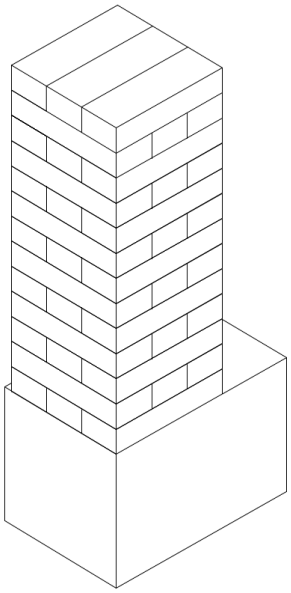

Imagen 36 Jenga situado sobre la base

Ahora se deben iniciar ambos sistemas, el ordenador y el UR3. Por la parte del ordenador hay que conectarlo a la misma red que la del robot, para poder establecer la comunicación. Seguidamente hay que conectar la cámara mediante el cable USB situado dentro de la caja del UR3. Par encender el UR3 hay que encender el sistema con el botón correspondiente y esperar a que se inicie el sistema. Una vez iniciado hay que conectar el almacenamiento USB que contiene el programa y ejecutarlo.

Una vez ejecutado el programa en el UR3 se situará en la posición P_Colocar, una vez situado el robot en dicha posición hay que colocar el Jenga en la posición adecuada. (véase imagen 25). Una vez posicionado se puede iniciar el programa del ordenador, este realizará la pregunta de si se quiere jugar una partida, en el caso afirmativo se iniciará la conexión del robot y realizará varias preguntas para la forma de desarrollar la partida.

En este punto ya se inicia la partida, por lo que solo se tienen que seguir las indicaciones de la pantalla del ordenador hasta que finalice la partida.

Una vez finalizada la partida, o en el caso de que se derribe la torre se puede volver a jugar reiniciando ambos equipos y colocando de nuevo la torre.

Ejemplos del proceso de Detección:

Jenga con la cara con seis piezas horizontales sin piezas retiradas:

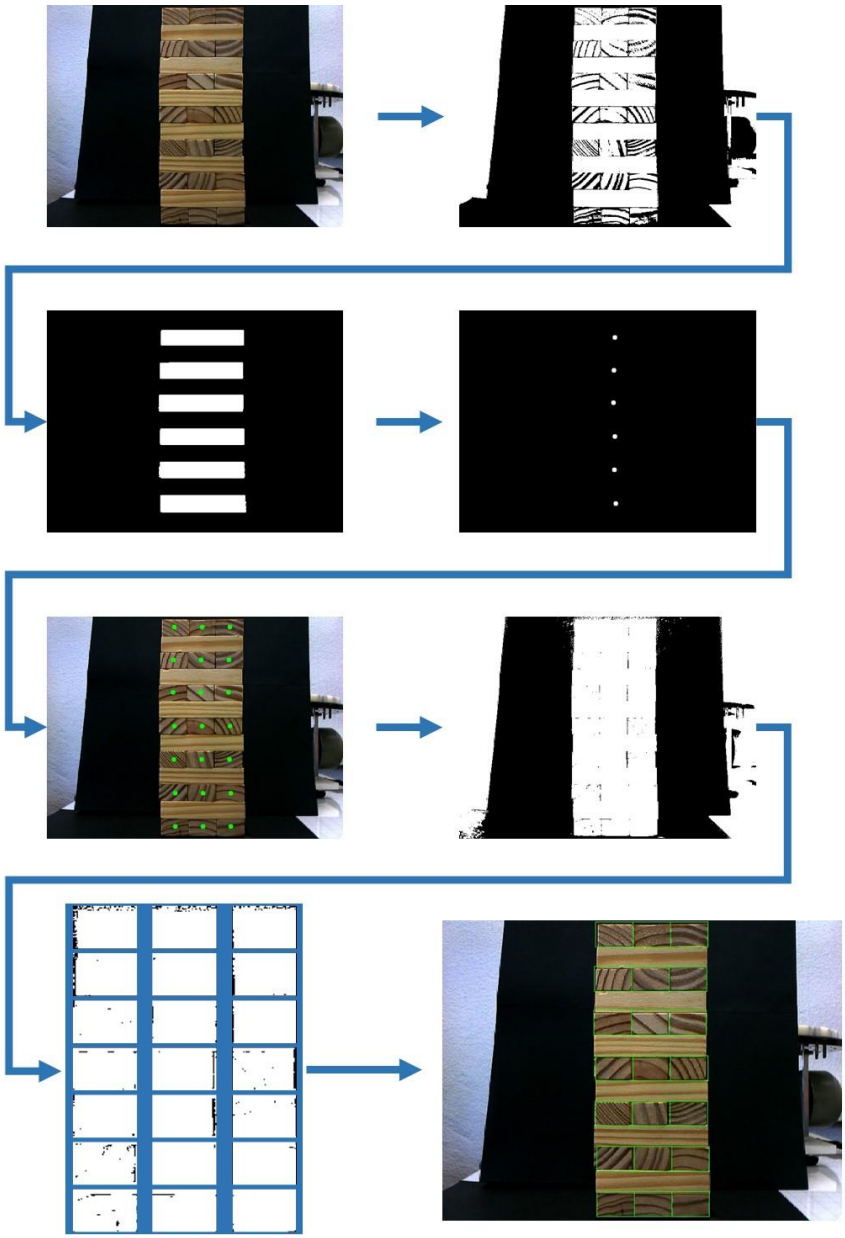

Imagen 37 Ejemplo detección lado 6 con todas las piezas

Centroides: [337.0, 417.0, 334.0, 344.0, 335.0, 272.0, 332.0, 199.0, 333.0, 129.0, 335.0, 58.0]

Pixeles blancos: [2306, 2327, 2390, 2377, 2389, 2342, 2340, 2360, 2397, 2380, 2362, 2317, 2397, 2376, 2360, 2354, 2371, 2280, 2354, 2395, 2387]

listaPuntosX [275.0, 273.0, 272.0, 275.0, 274.0, 277.0, 277.0, 335.0, 333.0, 332.0, 335.0, 334.0, 337.0, 337.0, 395.0, 393.0, 392.0, 395.0, 394.0, 397.0, 397.0]

listaPuntosY [22.0, 93.0, 163.0, 236.0, 308.0, 381.0, 453.0, 22.0, 93.0, 163.0, 236.0, 308.0, 381.0, 453.0, 22.0, 93.0, 163.0, 236.0, 308.0, 381.0, 453.0]

Jenga con la cara con seis piezas horizontales con varias piezas retiradas:

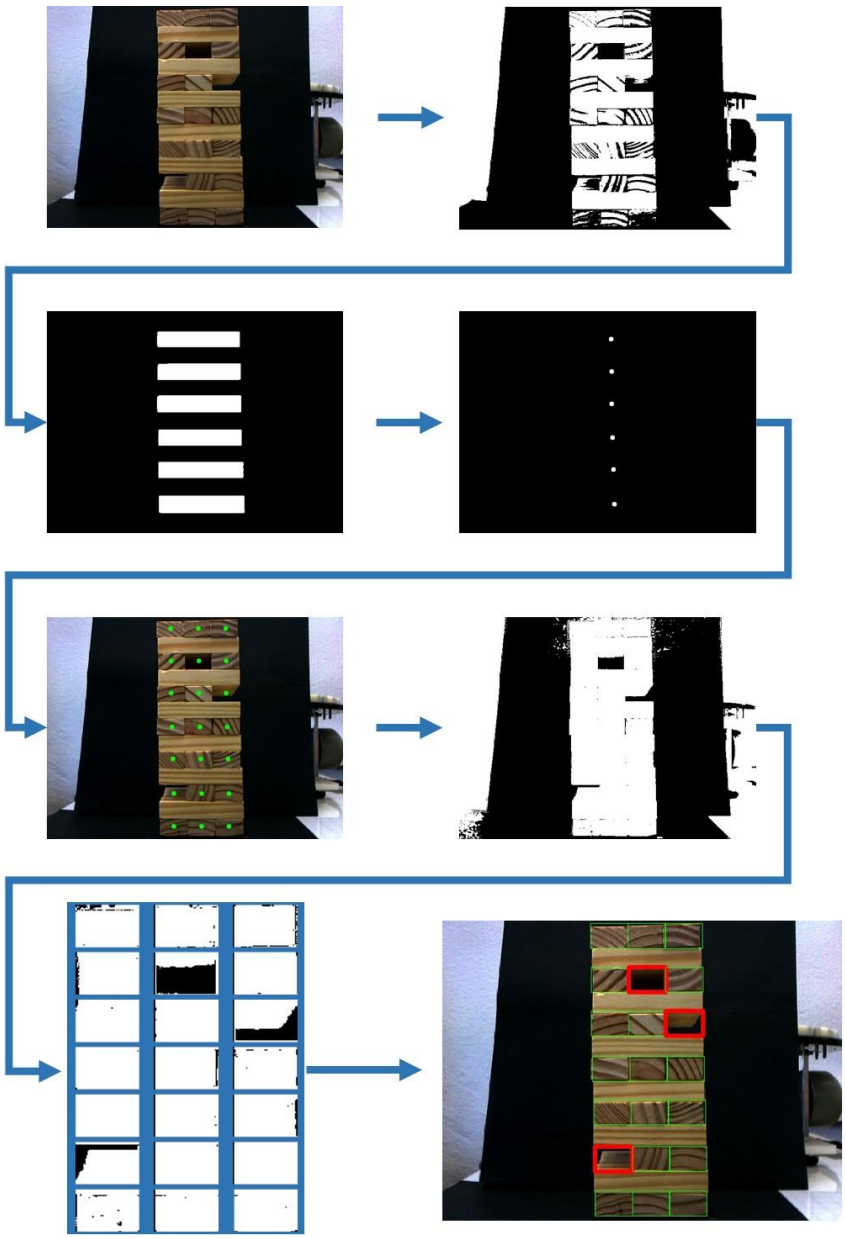

Imagen 38 Ejemplo detección lado 6 con piezas retiradas

Centroides: [334.0, 417.0, 332.0, 342.0, 330.0, 273.0, 328.0, 200.0, 328.0, 130.0, 327.0, 60.0]

Pixeles blancos: [2289, 2310, 2385, 2397, 2400, 1909, 2324, 2361, 941, 2393, 2335, 2397, 2398, 2386, 2270, 2355, 1556, 2330, 2308, 2400, 2383]

listaPuntosX: [267.0, 268.0, 268.0, 270.0, 272.0, -1, 274.0, 327.0, -1, 328.0, 330.0, 332.0, 334.0, 334.0, 386.0, 387.0, -1, 389.0, 391.0, 393.0, 393.0]

listaPuntosY: [24.0, 94.0, 164.0, 237.0, 306.0, -1, 452.0, 24.0, -1, 164.0, 237.0, 306.0, 381.0, 452.0, 24.0, 94.0, -1, 237.0, 306.0, 381.0, 452.0]

Jenga con la cara con siete piezas horizontales sin piezas retiradas:

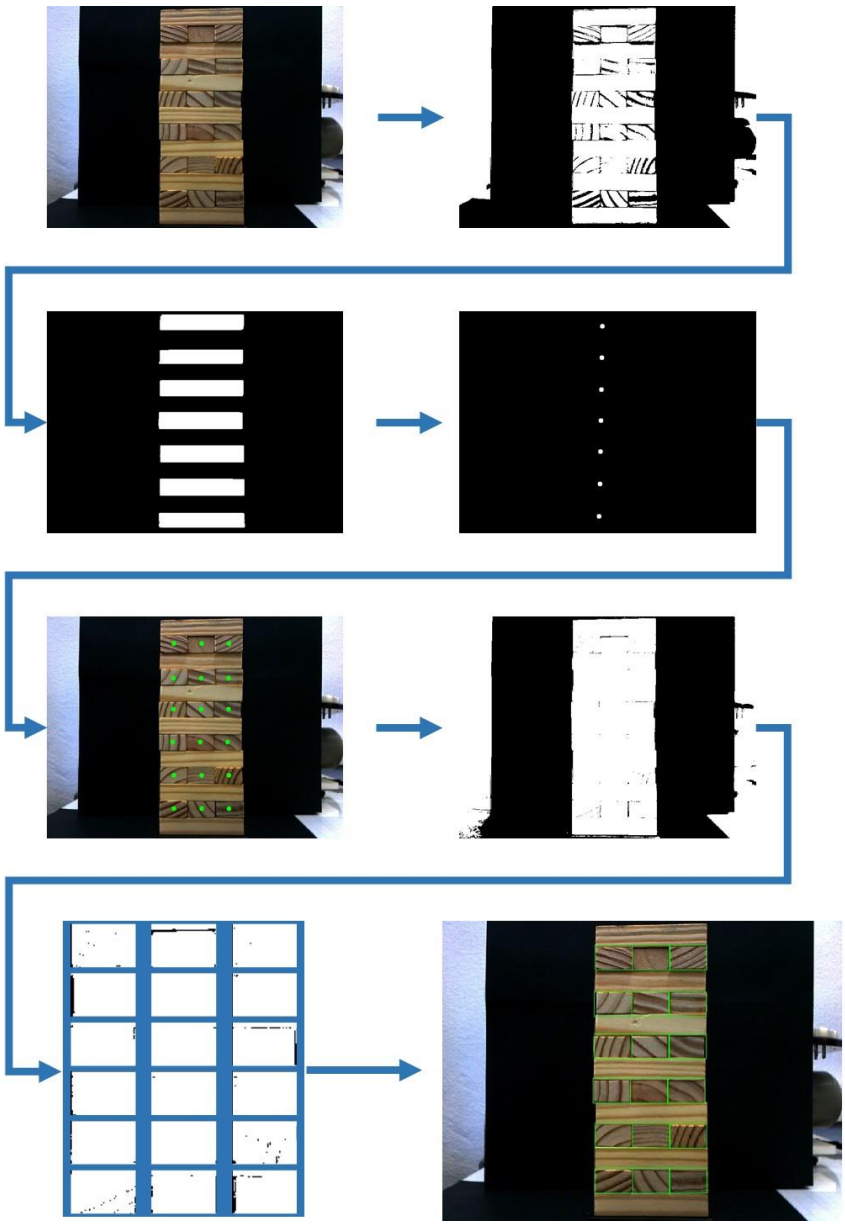

Imagen 39 Ejemplo detección lado 7 con todas las piezas

Centroides: [334.0, 451.0, 334.0, 380.0, 334.0, 308.0, 331.0, 236.0, 333.0, 165.0, 333.0, 98.0, 334.0, 23.0]

Pixeles blancos: [2348, 2290, 2388, 2348, 2370, 2377, 2301, 2400, 2397, 2398, 2397, 2353, 2398, 2397, 2287, 2397, 2356, 2314]

listaPuntosX: [274.0, 273.0, 273.0, 271.0, 274.0, 274.0, 334.0, 333.0, 333.0, 331.0, 334.0, 334.0, 393.0, 392.0, 392.0, 390.0, 393.0, 393.0]

listaPuntosY: [58.0, 133.0, 200.0, 271.0, 343.0, 415.0, 58.0, 133.0, 200.0, 271.0, 343.0, 415.0, 58.0, 133.0, 200.0, 271.0, 343.0, 415.0]

Jenga con la cara con siete piezas horizontales con varias piezas retiradas:

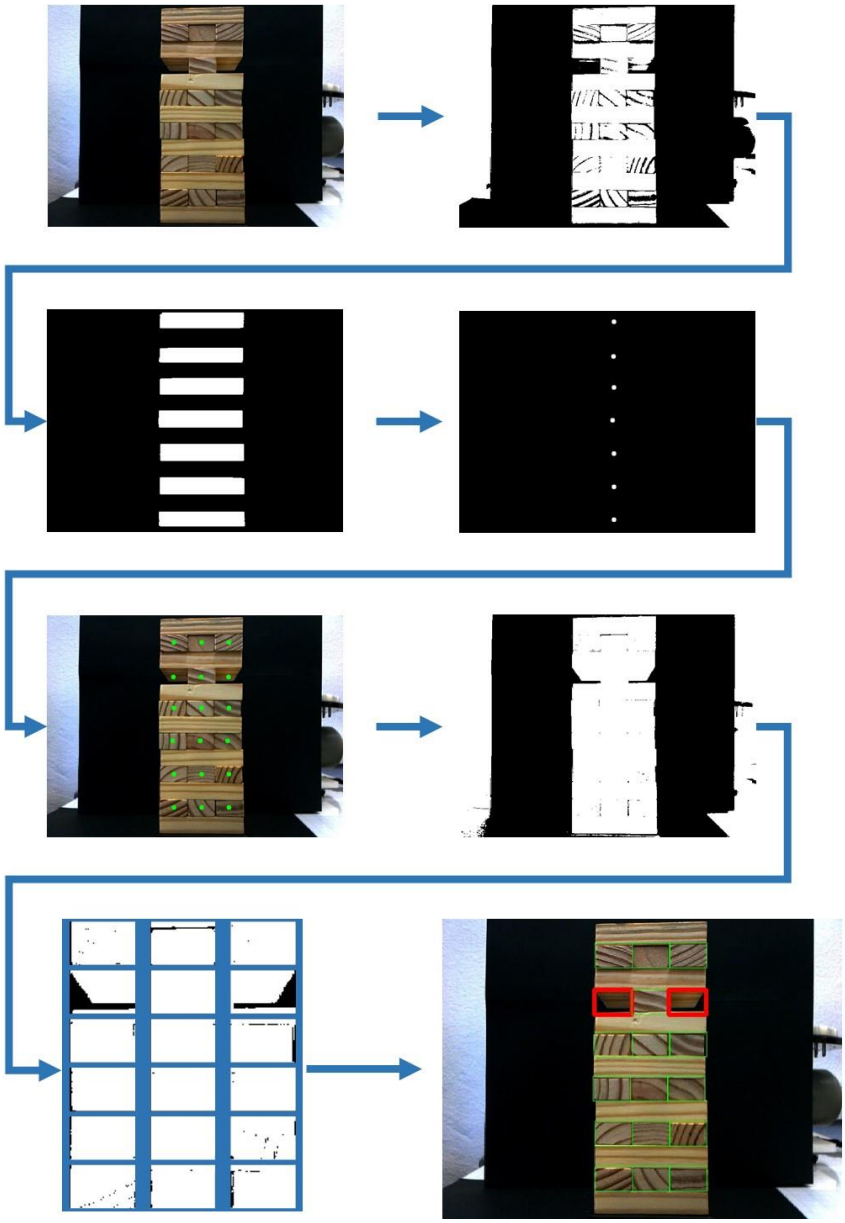

Imagen 40 Ejemplo detección lado 7 con piezas retiradas

Centroides: [334.0, 451.0, 334.0, 380.0, 334.0, 308.0, 331.0, 236.0, 334.0, 165.0, 333.0, 98.0, 334.0, 23.0]

Pixeles blancos: [2346, 1741, 2388, 2350, 2373, 2384, 2302, 2392, 2398, 2398, 2397, 2354, 2399, 1873, 2269, 2397, 2366, 2327]

listaPuntosX: [274.0, -1, 274.0, 271.0, 274.0, 274.0, 334.0, 333.0, 334.0, 331.0, 334.0, 334.0, 393.0, -1, 393.0, 390.0, 393.0, 393.0]

listaPuntosY: [58.0, -1, 200.0, 271.0, 343.0, 415.0, 58.0, 133.0, 200.0, 271.0, 343.0, 415.0, 58.0, -1, 200.0, 271.0, 343.0, 415.0]

Código:

Código en Python:

```
import argparse
import numpy as np
import cv2
import time
import array
import socket
from random import randint
from _thread import *
import statistics
import math

#Crear las ventanas donde se mostrarán las imágenes
cv2.namedWindow('Thres')
cv2.moveWindow('Thres', 0,525)
cv2.namedWindow('Piezas')
cv2.moveWindow('Piezas', 0, 0)
cv2.namedWindow('Deteccion')
cv2.moveWindow('Deteccion', 645, 0)
cv2.namedWindow('Thres2')
cv2.moveWindow('Thres2', 1290, 0)

#Iniciar las ventanas en negro con dimensión 640x480 px
frame_copia = np.zeros((480,640),np.uint8)
frame_copia2 = np.zeros((480,640),np.uint8)
frame_puntos = np.zeros((480,640),np.uint8)

#Dirección y puerto del robot para su conexión
TCP_IP = '158.42.206.16'
TCP_PORT = 1034
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
BUFFER_SIZE = 1024

#Inicializar variables
thres_max=255
X_ant=255
fin=0
ladoIni=0
posRatonX,posRatonY= -1,-1
pSelec=0
pX=0
pY=0

#Asigna la cámara que va a capturar las imágenes
camara = cv2.VideoCapture(1)

def camaraSet():
#Configura los parámetros de la cámara

    camara.set(10,0)        #Brillo
    camara.set(11,100)      #Contraste
    camara.set(12,30)       #Saturación
    camara.set(14,60)       #Ganancia
    camara.set(15,3)        #Exposición
    camara.set(17,5)        #Balance de blancos
    camara.set(cv2.CAP_PROP_AUTOFOCUS, 290)
    return None
```

```
def threshold(img,thres):
#Convierte la imagen (img) a escala de grises para poder segmentarla con un
umbral dado (thres)

    img_gris = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    t,img_thres= cv2.threshold(img_gris,thres,255,cv2.THRESH_BINARY)
    return img_thres

def detec_piezas_horizontales(imagen):
#A partir de la imagen (imagen) segmentada obtiene conjuntos de puntos que forman
una línea para detectar las piezas horizontales

    imagen_blanco = np.zeros((480,640),np.uint8)
    alto,ancho=np.shape(imagen)
    lin_list=[]
    lin_list_acept=[]
    primX=0
    primY=0
    primXant=0
    primYant=0
    ultX=0
    ultY=0
    ultXant=0
    ultYant=0
    xant=0
    yant=0
    ultXant=0
    ultYant=0
    prim=0
    global media

    for Y in range(0,alto):
        for X in range(0,ancho):

            if ((imagen[Y,X]==255) and (imagen[Y,xant]==0)):
                primX=X
                primY=Y

            elif ((imagen[Y,X]==0) and (imagen[Y,xant]==255)):
                ultX=X
                ultY=Y
                l = ultX-primX

                if ((l>150)and (l<200)):

                    lin_list_acept=list(np.append(lin_list_acept,l))

                    if((primXant==0) and (primX!=0)):
                        primXant=primX
                        primYant=primY
                        ultXant=ultX
                        ultYant=ultY
                        esquinaSupIzq=(primXant,primYant)
                        esquinaSupDer=(ultXant,ultYant)

                    cv2.line(imagen_blanco, (primX,primY),(ultX,ultY),
                    (255,255,255),2)

            xant=X
        cv2.imshow("Deteccion",imagen_blanco)
    yant=Y
    media=statistics.median(lin_list_acept)
    return imagen_blanco
```

```
def detectar_centroides(imagen):
#Agrupa las líneas de la imagen (imagen) anterior para crear las piezas y obtener
su centroide, además de contar el número de piezas detectadas

    centros_list=[]
    contours, _ = cv2.findContours(imagen,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    piezasDetectadas=0
    global lado
    lado=0

    for c in contours:
        area = cv2.contourArea(c)
        if(area>=3000):
            piezasDetectadas+=1

            M = cv2.moments(c)
            cX =int(M["m10"] / M["m00"])
            cY= int(M["m01"] / M["m00"])
            centros_list=list(np.append(centros_list,(cX,cY)))

    if (piezasDetectadas==7):
        lado=7
        return centros_list, lado
    elif(piezasDetectadas==6):
        lado=6
        return centros_list, lado
    else:
        print("No se han detectado suficientes piezas")
        lado=0
        centros_list=[0]
        return centros_list, lado

def puntosAuxiliares(lista):
#Con la lista de centroides (lista) y con el lado (variable global) se obtiene la
posición aproximada del resto de las piezas

    puntosX_1=[]
    puntosY_1=[]
    puntosX_2=[]
    puntosY_2=[]
    puntosX_3=[]
    puntosY_3=[]
    puntosX=[]
    puntosY=[]
    d_lat=media/3
    d_vert=15*media/75

    if (lado==7):
        j=12
        k=13
        for i in np.shape(lista):
            while(j>0):
                puntosX_1=list(np.append(puntosX_1,int(lista[j]-d_lat)))
                puntosX_2=list(np.append(puntosX_2,int(lista[j])))
                puntosX_3=list(np.append(puntosX_3,int(lista[j]+d_lat)))
                j-=2
            while(k>1):
                puntosY_1=list(np.append(puntosY_1,int(lista[k]+d_vert)))
                puntosY_2=list(np.append(puntosY_2,int(lista[k]+d_vert)))
                puntosY_3=list(np.append(puntosY_3,int(lista[k]+d_vert)))
                k-=2
            puntosX=list(np.append(puntosX,(puntosX_1,puntosX_2,puntosX_3)))
            puntosY=list(np.append(puntosY,(puntosY_1,puntosY_2,puntosY_3)))
```

```
    if (lado==6):
        j=10
        k=11
        for i in np.shape(lista):

            while(j>=0):
                puntosX_1=list(np.append(puntosX_1,int(lista[j]-d_lat)))
                puntosX_2=list(np.append(puntosX_2,int(lista[j])))
                puntosX_3=list(np.append(puntosX_3,int(lista[j]+d_lat)))
                j-=2
            puntosX_1=list(np.append(puntosX_1,int(lista[0]-d_lat)))
            puntosX_2=list(np.append(puntosX_2,int(lista[0])))
            puntosX_3=list(np.append(puntosX_3,int(lista[0]+d_lat)))

            while(k>=1):
                puntosY_1=list(np.append(puntosY_1,int(lista[k]-d_vert)))
                puntosY_2=list(np.append(puntosY_2,int(lista[k]-d_vert)))
                puntosY_3=list(np.append(puntosY_3,int(lista[k]-d_vert)))
                k-=2
            puntosY_1=list(np.append(puntosY_1,int(lista[1]+d_vert)))
            puntosY_2=list(np.append(puntosY_2,int(lista[1]+d_vert)))
            puntosY_3=list(np.append(puntosY_3,int(lista[1]+d_vert)))

            puntosX=list(np.append(puntosX,(puntosX_1,puntosX_2,puntosX_3)))
            puntosY=list(np.append(puntosY,(puntosY_1,puntosY_2,puntosY_3)))

    return puntosX,puntosY


def clickRaton(event,X,Y,flags,param):
#Devuelve la posición del ratón (X,Y) al hacer doble clic con el botón izquierdo
(event)

    global posRatonX
    global posRatonY
    if event==cv2.EVENT_LBUTTONDBLCLK:
        posRatonX=X
        posRatonY=Y
    return posRatonX,posRatonY


def puntoMasCercano(listaPuntosX,listaPuntosY,posRatonX,posRatonY):
#Obtiene la posición de la pieza más cercana de los puntos auxiliares
(listaPuntosX, listaPuntosY) al clic del ratón(posRatonX, posRatonY), mediante la
ecuación euclídea.

    val=0
    for i in range(len(listaPuntosX)):
        X=listaPuntosX[i]
        Y=listaPuntosY[i]
        dX=X-posRatonX
        dY=Y-posRatonY
        dist=math.sqrt(dX**2+dY**2)
        distMin=20       #Si la distancia es de más de 20 pixeles ignora el punto
        if dist<distMin:
            distMin=dist
            val=i
    valX=listaPuntosX[val]
    valY=listaPuntosY[val]
    return valX, valY, val
```

```
def seleccionPunto(listaPuntosX,listaPuntosY):
#Con las listas de puntos (listaPuntosX, listaPuntosY) y el turno (variable
global) se selecciona la pieza, ya sea el jugador eligiendo la pieza (turno==1) o
el ordenador de forma aleatoria (turno==2), y según el lado se crean las filas de
piezas para ver si se puede quitar la pieza o no.

    repetir=1
    n=0
    p=0

    while repetir==1:
        if turno==1:
            print("Selecciona la pieza y pulse ENTER")
            cv2.waitKey(0)
            pX,pY,p=puntoMasCercano(listaPuntosX,listaPuntosY,
            posRatonX,posRatonY)

        if turno==2:
            max=len(listaPuntosX)
            max-=1
            p=randint(0, (max))

            while listaPuntosX[p]==-1:
                max=len(listaPuntosX)
                max-=1
                p=randint(0, max)

        if lado==7:
            if 0<= p <=5:
                p0=p
                p1=p+6
                p2=p+12
            if 6<= p <= 11:
                p0=p-6
                p1=p
                p2=p+6
            if 12<= p <= 17:
                p0=p-12
                p1=p-6
                p2=p

        if lado==6:
            if 0<= p <=6:
                p0=p
                p1=p+7
                p2=p+14
            if 7<= p <= 13:
                p0=p-7
                p1=p
                p2=p+7
            if 14<= p <= 20:
                p0=p-14
                p1=p-7
                p2=p

        if listaPuntosX[p0]==-1 and listaPuntosX[p2]==-1:
            print('No se puede quitar esta pieza, ya que va a tirar la torre')
            repetir=1

        elif listaPuntosX[p1]==-1:
            print('No se puede quitar esta pieza, ya que va a tirar la torre')
            repetir=1
```

```
            elif listaPuntosX[p0]==-1 and listaPuntosX[p1]!=-1 and
            listaPuntosX[p2]!=-1:
                if p==p1:
                    print('No se puede quitar esta pieza, ya que va a tirar la
                    torre')
                    repetir=1
                else:
                    repetir=0

            elif listaPuntosX[p0]!=-1 and listaPuntosX[p1]!=-1 and
            listaPuntosX[p2]==-1:
                if p==p1:
                    print('No se puede quitar esta pieza, ya que va a tirar la
                    torre')
                    repetir=1
                else:
                    repetir=0

            else:
                repetir=0

        puntoXelegido=listaPuntosX[p]
        puntoYelegido=listaPuntosY[p]
        listaPuntosX[p]=-1
        listaPuntosY[p]=-1

        return puntoXelegido, puntoYelegido,p


    def dePixAmm(pX,pY):
    #Convierte la posición en pixeles de la pieza (pX,pY) respecto a origen de la
    ventana, a la posición den milímetros con origen en el centro de la torre

        pXmm=(pX-320)*(75/media)
        pYmm=(pY-240)*(14/(14*media/75))

        pXmmQuit=pXmm
        pYmmQuit=pYmm

        if pXmm < 10:
            pXmmEmp=pXmm-23 #Si la pieza está en la parte izq o en el centro resta
            pYmmEmp=pYmm    20mm para que la quite con el dedo derecho

        elif pXmm >= 10:
            pXmmEmp=pXmm+23 #Si la pieza está en la parte der resta 20mm para que la
            pYmmEmp=pYmm    quite con el dedo izquierdo

        return pXmmEmp, pYmmEmp, pXmmQuit, pYmmQuit
```

```
def deteccion():
#Utiliza otras funciones para realizar la detección de las piezas, realiza hasta
un máximo de cinco capturas y espera a que se detecten las piezas horizontales,
en caso de no detectar las piezas se toma como que se ha acabado la partida.

    listaPuntosX=[]
    listaPuntosY=[]
    lado=0
    fin=0
    n=0

    while lado==0 and fin==0:
        (siguiente, frame) = camara.read() #Captura la imagen
        frame_copia = frame.copy()
        frame_copia2 = frame.copy()
        imagen_thres=threshold(frame,62)
        cv2.imshow("Thres",imagen_thres)
        imagen_blanco=detec_piezas_horizontales(imagen_thres)
        centros_list,lado=detectar_centroides(imagen_blanco)
        listaPuntosX,listaPuntosY=puntosAuxiliares(centros_list)
        imagen_thres2=threshold(frame,19)
        cv2.imshow("Thres2",imagen_thres2
        listaPuntosX,listaPuntosY=piezasRetiradas(listaPuntosX,listaPuntosY,
        imagen_thres2,frame_copia2)
        finPiezas=condicionFin(listaPuntosX, listaPuntosY)
        n+=1
        if n==5:
            fin=1
        fin=fin or finPiezas
    return listaPuntosX,listaPuntosY,fin


def quitarPieza(turno):
#Manda al robot la posición de la pieza que se va a retirar y espera a que el
robot devuelva un 1 como confirmación de que ha retirado a pieza, o un 2 como que
no ha podido quitar la pieza, en cuyo caso repite la operación.

    global pSelec
    global pX
    global pY
    global fin
    fin=0
    repetir=1
    pSelec=0
    p=0
    fin=0

    while repetir==1 and fin==0:

        listaPuntosX,listaPuntosY,fin=deteccion()

        if fin==0:
            pX,pY,p=seleccionPunto(listaPuntosX,listaPuntosY)
            pXmmEmp,pYmmEmp,pXmmQuit,pYmmQuit=dePixAmm(pX,pY)
            pSelec=1
            listaPuntosX,listaPuntosY,fin=deteccion()
            print('Punto elegido en pixeles:',pX,pY,',en
            milimetros:',pXmmQuit,pYmmQuit)
            print("Pulse enter para quitar la pieza")
            cv2.waitKey(0)
            punto=str([pXmmEmp,pYmmEmp,pXmmQuit,pYmmQuit])
            conn.send(punto.encode())
            print('Punto enviado')
```

```
            piezaQuitada=0
            while piezaQuitada==0:
                res = conn.recv(1024)
                piezaQuitada=int(res)

            if piezaQuitada==1:
                esperarPosDeteccion()
                listaPuntosX,listaPuntosY,fin=deteccion()
                if fin==0:

                    if listaPuntosX[p]==-1 and listaPuntosY[p]==-1:
                        repetir=0
                        pSelec=1
                        conn.send("(1)".encode())
                    else:
                        repetir=1
                        print("No se ha quitado esta pieza, elija otra")
                        pSelec=0
                        conn.send("(0)".encode())

            if piezaQuitada==2:
                print("No se puede quitar esta pieza, elija otra")
                esperarPosDeteccion()
                listaPuntosX,listaPuntosY,fin=deteccion()
                repetir=1
                pSelec=0

    return None


def piezasRetiradas(listaPuntosX,listaPuntosY,imagen_thres2,frame_copia2):
#Con la lista de puntos (listaPuntosX, listaPuntosY) va mirando cada posición de
cada pieza, y si hay una cierta cantidad de pixeles blancos en la imagen
segmentada (imagen_thres2) se toma como que hay pieza en dicha posición, en caso
contrario se toma como que se ha retirado y se inserta un -1 en la posición de la
lista de puntos. Finalmente se muestran los rectángulos en una copia de la torre
(frame_copia2)

    for n in range(len(listaPuntosX)):
        x=int(listaPuntosX[n])
        y=int(listaPuntosY[n])
        cv2.rectangle(frame_copia2,(x-30,y-20),(x+30,y+20),(0,254,0),1)
        #Crea un rectángulo verde a partir del centro de la pieza
        rect_pieza=imagen_thres2[(y-20):(y+20),(x-30):(x+30)]
        #Exporta el rectángulo
        pixBlancos = cv2.countNonZero(rect_pieza)

        if pixBlancos<2200:
            listaPuntosX[n]=-1
            listaPuntosY[n]=-1
            cv2.rectangle(frame_copia2,(x-30,y-20),(x+30,y+20),(0,0,254),5)
            #Crea un rectángulo rojo a partir del centro de la pieza

        if listaPuntosX[n]!=-1:
            if pSelec==1:
                if listaPuntosX[n]==pX and listaPuntosY[n]==pY:
                    cv2.circle(frame_copia2,
(int(listaPuntosX[n]),int(listaPuntosY[n])), 10, (0, 0, 254), -1)

    cv2.imshow("Piezas",frame_copia2)
    return listaPuntosX,listaPuntosY
```

```
def esperarPosDeteccion():
#Espera a que el robot envíe un 5 (número elegido al azar) como confirmación de
que está en la posición de detección, para evitar que detecte sin estar en esa
posición.

    val=0
    while val!=5:
        val=conn.recv(1024)
        val=int(val)
    conn.send("(2)".encode())



def condicionFin(listaPuntosX, listaPuntosY):
#Con las listas de puntos (listaPuntosX, listapUntosY) analiza cada fila de
piezas, si en las filas no hay piezas que se puedan retirar, aumenta el valor de
la variable val, y si este es igual al número de filas con piezas extraíbles
significa que ya no se pueden extraer más piezas, por lo que la partida finaliza.

    columna1=[]
    columna2=[]
    columna3=[]
    val=0
    sum=0
    finPiezas=0

    if lado==7:
        for n in range(len(listaPuntosX)):
            if 0<=n<=5:
                columna1=list(np.append(columna1,listaPuntosX[n]))
                columna2=list(np.append(columna2,listaPuntosX[n+6]))
                columna3=list(np.append(columna3,listaPuntosX[n+12]))
                if columna1[n]==-1 and columna2[n]!=-1 and columna3[n]==-1:
                    val=val+1
                if columna1[n]!=-1 and columna2[n]==-1 and columna3[n]!=-1:
                    val=val+1
                if columna1[n]==-1 and columna2[n]==-1 and columna3[n]==-1:
                    val=val+1
                else:
                    val=val
    if lado==6:
        for n in range(len(listaPuntosX)):
            if 0<=n<=6:
                columna1=list(np.append(columna1,listaPuntosX[n]))
                columna2=list(np.append(columna2,listaPuntosX[n+7]))
                columna3=list(np.append(columna3,listaPuntosX[n+14]))
                if columna1[n]==-1 and columna2[n]!=-1 and columna3[n]==-1:
                    val=val+1
                if columna1[n]!=-1 and columna2[n]==-1 and columna3[n]!=-1:
                    val=val+1
                if columna1[n]==-1 and columna2[n]==-1 and columna3[n]==-1:
                    val=val+1
    if (lado==7 and val==6) or (lado==6 and val==7):
        finPiezas=1

    return finPiezas
```

```
cv2.setMouseCallback('Piezas',clickRaton)
#Bucle principal
while (fin==0):

    camaraSet()
    img_ini = np.zeros((480,640),np.uint8)
    cv2.imshow("Piezas",img_ini)
    cv2.imshow("Thres2",img_ini)
    print("Bienvenido, si quiere jugar una partida, pulse S")
    tecla = cv2.waitKey(0)

    if tecla & 0xFF == ord('s'):
        partida=1
        print('Conectando el robot....')
        conectarRobot(TCP_IP,TCP_PORT)
        print("Robot conectado")
        pos=0
        print("Coloque el jenga en posición y pulse ENTER para continuar")
        cv2.waitKey(0)
    else:
        camara.release()
        cv2.destroyAllWindows()
        fin=1

    if fin!=1:
        print("Si va a necesitar ayuda del robot, pulse S")
        tecla = cv2.waitKey(0)
        if tecla & 0xFF == ord('s'):
            ayuda=1
        else:
            ayuda=0

        print("Si quiere empezar la partida pulse S")
        tecla = cv2.waitKey(0)
        if tecla & 0xFF == ord('s'):
            turno=1
        else:
            turno=2

        conn.send("(1)".encode())
        esperarPosDeteccion()
        listaPuntosX,listaPuntosY,fin=deteccion()

    while partida==1 and fin==0:

        if turno==1 and fin==0:
            print("Es su turno")

            if ayuda==0 and fin==0:
                listaPuntosX,listaPuntosY,fin=deteccion()
                print("Quita la pieza, y pulse ENTER cuando la retires")
                cv2.waitKey(0)
                listaPuntosX,listaPuntosY,fin=deteccion()

            if ayuda==1 and fin==0:
                quitarPieza(turno)

            if fin!=1:
                turno=2
                print('Pulsa ENTER para pasarle el turno al robot')
                cv2.waitKey(0)
```

```
            if turno==2 and fin!=1:
                print("Es el turno del Robot")
                quitarPieza(turno)
                if fin!=1:
                    turno=1

        if fin == 1:
            (siguiente, frame) = camara.read()
            frame_copia = frame.copy()
            cv2.imshow("Piezas",frame_copia)
            print('Fin de la partida:')
            punto=str([-1,-1,-1,-1])
            conn.send(punto.encode())
            print('Pulse enter para cerrar el programa')
            cv2.waitKey(0)

    punto=str([-1,-1,-1,-1])
    conn.send(punto.encode())
    camara.release()
    cv2.destroyAllWindows()
```

UR:

```
Programa
   Inic. variables
   BeforeStart
     Ajustar pinza=Apagar
     Ajustar
     Bucle open ≟ False
       open:=socket_open("158.42.206.16",1034)
   Programa de robot
     Invocar Colocar_Jenga
     Invocar Deteccion
     Bucle
       coord:=socket_read_ascii_float(4)
       If coord≟[4,-1,-1,-1,-1]
         fin:= True
         coord:=[0,0,0,0,0,0]
       If coord[0]≟4
         PiezaQuitada:= False
         Bucle PiezaQuitada≟ False
           Invocar EmpujarPieza
           If PiezaQuitada≟ True
             Invocar QuitarPieza
             Invocar Deteccion
             Invocar Pale
             coord:=[0,0,0,0,0,0]
           Else
             coord:=[0,0,0,0,0,0]
             Invocar Deteccion
             Bucle coord[0]≟0
               coord:=socket_read_ascii_float(4)
       coord:=[0,0,0,0,0,0]
       If fin≟ True
         Aviso
         MoveJ
           P_colocar
         Detener
   Pale
     Bucle paletizar[0]≟0
       paletizar:=socket_read_ascii_float(1)
       If paletizar[1]≟1
         MoveL
           Punto_de_paso_9
         If (j≥0 and j≤2) or (j≥6 and j≤8)
           MoveL
             Punto_de_paso_9
           If j≟6
             p_pale_1:=pose_add(p_pale_1, p[0,-0.078,0.02,0,0,0])
           If j≟12
             p_pale_1:=pose_add(p_pale_1, p[0,-0.078,0.04,0,0,0])
           MoveL
             'P_pale_inf'
             p_pale_1
           p_pale_1:=pose_add(p_pale_1, p[0,0.026,0,0,0,0])
         If (j≥3 and j≤5) or (j≥9  and j≤11)
           MoveL
             Punto_de_pas_10
           If j≟9
             p_pale_2:=pose_add(p_pale_2, p[-0.078,0,0.02,0,0,0])
           If j≟15
             p_pale_2:=pose_add(p_pale_2, p[-0.078,0,0.04,0,0,0])
           MoveL
```

```
            'P_pale_sup'
            p_pale_2
          p_pale_2:=pose_add(p_pale_2, p[0.026,0,0,0,0,0])
        Esperar: 0.5
        Ajustar pinza= False
        Esperar: 0.5
        j:=j+1
        MoveL
          Punto_de_pas_11
          P_cam
  Ajustar pinza=Apagar
  paletizar:=[0,0]
Colocar_Jenga
  Permiso:=[0,0]
  MoveJ
    P_colocar
  Bucle Permiso[0]≟0
    Permiso:=socket_read_ascii_float(1)
QuitarPieza
  despl:=pose_add(P_emp, p[0,0,0.15,0,0,0])
  MoveL
    despl
  MoveL
    Punto_de_paso_1
    Punto_de_paso_2
    Punto_de_paso_3
    Punto_de_paso_4
  despl:=pose_add(puntoQuit,Punto_de_paso_4)
  MoveL
    despl
  despl:=pose_add(despl, p[-0.033,0,0,0,0,0])
  MoveL
    despl
  Ajustar pinza=Encender
  despl:=pose_add(despl, p[0.08,0,0,0,0,0])
  Esperar: 0.5
  MoveL
    despl
  despl:=pose_add(despl, p[0,0,0.11,0,0,0])
  MoveL
    Punto_de_paso_5
    Punto_de_paso_6
    Punto_de_paso_7
    Punto_de_paso_8
EmpujarPieza
  puntoEmp:=p[0,-coord[1]/1000,-coord[2]/1000,0,0,0]
  puntoQuit:=p[0,-coord[3]/1000,-coord[4]/1000,0,0,0]
  MoveJ
    P_empujar
    P_emp:=P_empujar
  despl:=pose_add(puntoEmp, P_empujar)
  MoveL
    despl
  Esperar: 0.5
  Herramienta:=Herram
  force_mode(Herramienta,[0,0,1,0,0,0],[0,0,5,0,0,0],2,[0.025,0.025,0.1,
  0.025,0.025,0.025])
  Esperar: 0.2
  sync()
  pos:=get_actual_tcp_pose()
  Esperar point_dist(pos,get_actual_tcp_pose())≥0.03  or
  (get_target_tcp_speed()<p[0.001,0.001,0.0001,0.001,0.001,0.001])
```

```
    end_force_mode()
    If point_dist(pos,get_actual_tcp_pose())≥0.03
      PiezaQuitada:= True
      MoveL
        pos
      MoveL
        P_emp
      socket_send_string(1)
    Else
      PiezaQuitada:= False
      MoveL
        pos
      socket_send_string(2)
  Deteccion
    Permiso:=[0,0]
    MoveL
      P_cam
      Esperar: 1.0
      socket_send_string(5)
    Bucle Permiso[1]≠2
      Permiso:=socket_read_ascii_float(1)
```

Valores de las posiciones del programa:

```
P_Colocar=[0.045,0.214,-0.031,0.304,1.553,0.294]
P_Empujar=[0.040,0.214,-0.031,0.304,1.553,0.294]
P_Cam=[-0.1227,0.229,-0.1316,1.902,1.276,1.123]
P_pale_inf=[-0.2916,0.409,-0.1716,0.66,3.071,0.008]
P_pale_sup=[-0.3538,0.429,-0.156,2.614,1.728,0.015]
Punto_de_paso_1=[0.0149,0.2137,0.150,0.413,2.488,0.169]
Punto_de_paso_2=[0.092,0.2177,0.1769,0.52,3.106,0.045]
Punto_de_paso_3=[0.1737,0.2087,0.166,0.608,3.61,-0.105]
Punto_de_paso_4=[0.1837,0.2178,-0.0119,0.656,3.435,-0.128]
Punto_de_paso_5=[0.206,0.4177,-0.008,0.699,3.473,-0.081]
Punto_de_paso_6=[0.1077,0.4889,-0.029,0.671,3.051,-0.002]
Punto_de_paso_7=[-0.047,0.4889,-0.029,0.671,3.051,-0.002]
Punto_de_paso_8=[-0.147,0.4117,-0.029,0.671,3.051,-0.002]
Punto_de_paso_9=[-0.264,0.434,0.0267,0.678,3.086,-0.037]
Punto_de_paso_10=[-0.329,0.403,0.0267,2.666,1.716,-0.006]
Punto_de_paso_11=[-0.138,0.367,-0.034,0.649,3.072,0.048]
```

Medidas:

En este apartado se muestran las medidas de los elementos usados en este caso, ya que dependiendo del Jenga pueden variar las medidas de las piezas, por lo que se tendrían que modificar las medidas en el programa, además de tener que cambiar la pinza en el caso que fuese necesario.

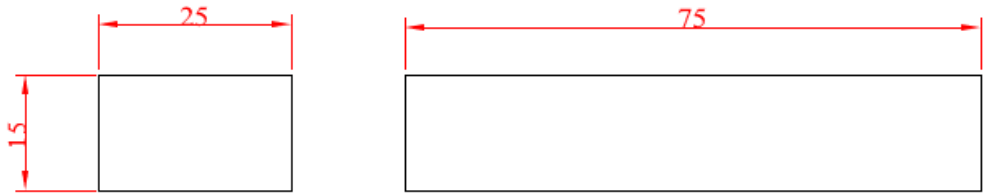

Imagen 41 Medidas de las piezas del Jenga en milímetros

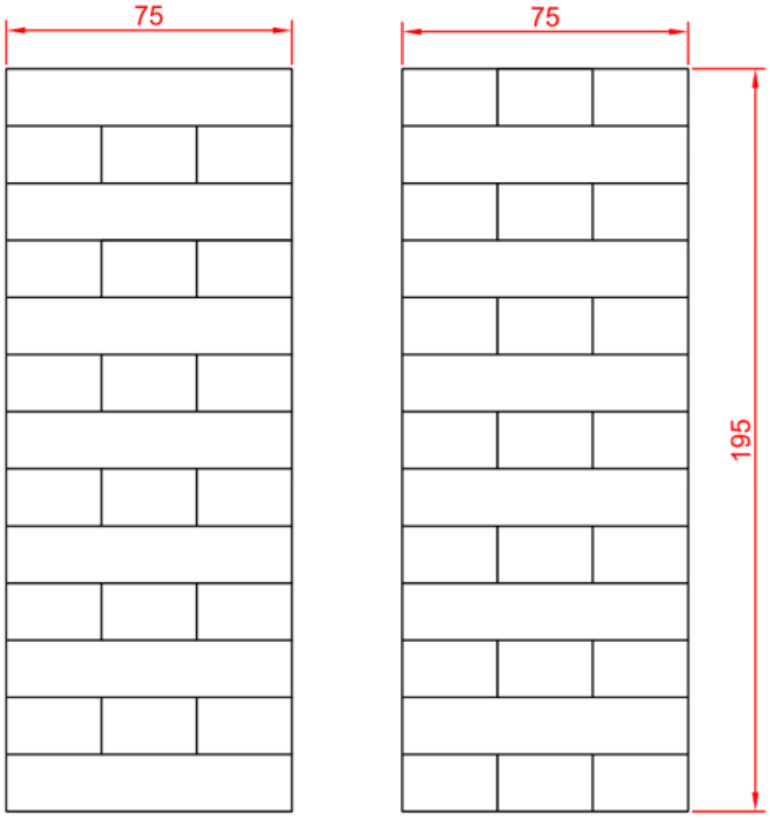

Imagen 42 Medidas de la torre del Jenga en milímetros

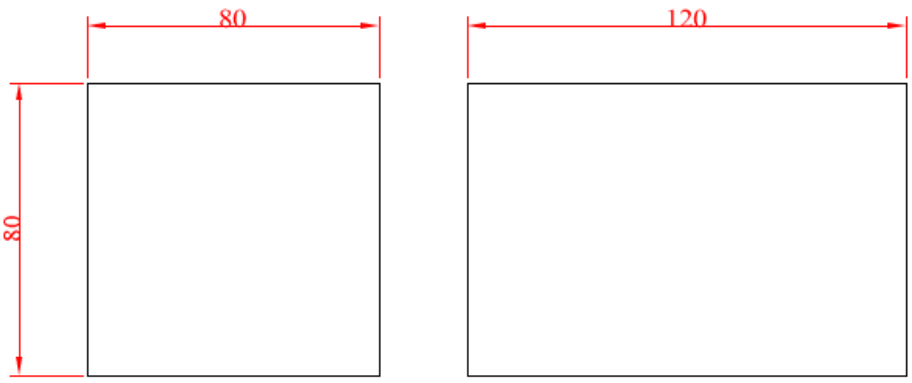

Imagen 43 Medidas de la base en milímetros

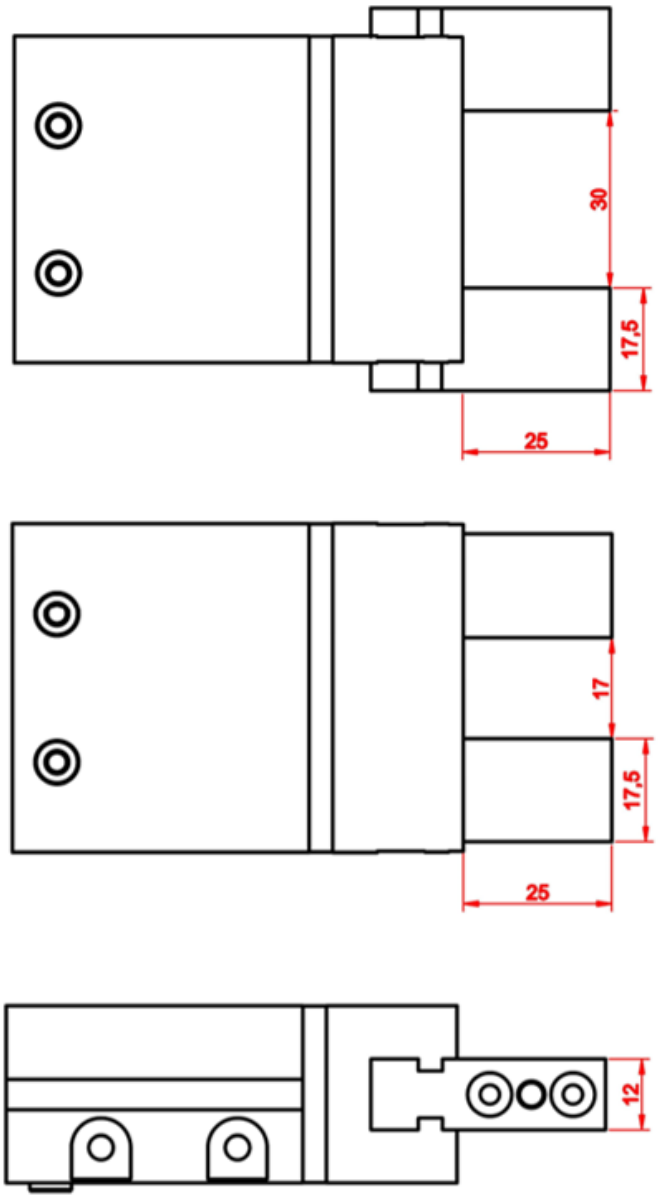

Imagen 44 Medidas de la pinza FESTO DHPS-20-A [12]

5-Bibliografía:

[1] Gabinete de comunicación, "El General de Valencia adquiere el robot Da Vinci - Blog de comunicación Valencia - Hospital General", *Blog de comunicación Valencia - Hospital General*, 2019. [En línea]. [Consulta: 9 de agosto 2019]. Disponible: http://blog.general-valencia.san.gva.es/2018/07/06/el-general-de-valencia-adquiere-el-robot-da-vinci/

[2] Gabinete de comunicación, Sin título, [En línea]. [Consulta: 9 de agosto 2019]. Disponible: http://blog.general-valencia.san.gva.es/2018/07/06/el-general-de-valencia-adquiere-el-robot-da-vinci/

[3] Universitat Politècnica de València, "Integración de Modelos Biomecánicos en el desarrollo y Operación de Robots Rehabilitadores Reconfigurables". [En línea]. [Consulta: 9 de agosto 2019]. Disponible: https://imbio3r.ai2.upv.es/content/integraci%C3%B3n-de-modelos-biomec%C3%A1nicos-en-el-desarrollo-y-operaci%C3%B3n-de-robots-rehabilitadores

[4] Universidat Politècnica de València, Sin título, [En línea]. [Consulta: 9 de agosto 2019]. Disponible: https://imbio3r.ai2.upv.es/sites/imbio3r.ai2.upv.es/themes/idecona/images/slideshows/banner_2.jpg

[5] Amazon, "Torre de madera" , [En línea]. [Consulta: 28 de agosto 2019] Disponible: https://images-na.ssl-images-amazon.com/images/I/61tl-n9ngwL._SX425_.jpg

[6] Robot, [En línea]. [Consulta: 20 de agosto 2019]. Disponible: https://cobots.se/produkt/ur3-robot/

[7] Cámara, [En línea]. [Consulta: 20 de agosto 2019]. Disponible: https://automationdistribution.com/robotiq-camera-universal-robots-installation-kit-rwc-ur-kit/#targetText=With%20Robotiq's%20Camera%2C%20you%20can,CB3.1%20or%20higher%20only.

[8] Pinza, [En línea]. [Consulta: 20 de agosto 2019]. Disponible: https://www.landefeld.com/artikel/en/dhps-20-a-1254046-parall-gripper/OT-FESTO032826

[9] Válvula, [En línea]. [Consulta: 20 de agosto 2019]. Disponible: https://es.rs-online.com/web/p/valvulas-de-control-accionado-por-solenoide-piloto-neumaticas/1215889/

[10] Kit neumático, [En línea]. [Consulta: 20 de agosto 2019]. Disponible: https://www.robotshop.com/es/es/kit-basico-robot-neumatico.html

[11] Modulo F.R.L, [En línea]. [Consulta: 20 de agosto 2019]. Disponible: https://es.rs-online.com/web/p/conjuntos-frl/8254312/

[12] Festo.com. Catálogo pinzas paralelas 2019.[En línea]. [Consulta: 20 de agosto 2019]. Disponible: https://www.festo.com/cat/en-gb_gb/data/doc_ES/PDF/ES/DHPS_ES.PDF