UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Prototipo de un controlador para un brazo robótico KUKA utilizando visión artificial

Diego Santiago Morales Arcos

Ingeniería en Sistemas

Trabajo de integración curricular presentado como requisito
para la obtención del título de
Ingeniero en Sistemas

Quito, 06 de enero de 2020

UNIVERSIDAD SAN FRANCISCO DE QUITO - USFQ

COLEGIO DE CIENCIAS E INGENIERÍA

HOJA DE CALIFICACIÓN
DE TRABAJO DE INTEGRACIÓN CURRICULAR

Prototipo de un controlador para un brazo robótico KUKA utilizando visión artificial

Diego Santiago Morales Arcos

Calificación:

Nombre del profesor, Título académico Daniel Riofrío PhD.

Firma del profesor: _____

Quito, 06 de enero de 2020

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante: _____

Nombres y apellidos: Diego Santiago Morales Arcos

Código: 00143821

Cédula de Identidad: 2100614392

Lugar y fecha: Quito, 06 de enero de 2020

Dedicatoria

A mi madre Juanita y a mi padre Henry.

Por haberme apoyado, aconsejado y por todo ese cariño indispensable durante mi primera y segunda carrera. Por ser mis ejemplos a seguir y por creer en mí más que nadie. Este nuevo logro es para ellos.

A mi hermana Kimberly.

Por ser mi constante tras todo este tiempo y por ser la persona que más quiero en este mundo.

A mi tutor Daniel.

Gracias a su guía pude encaminar y finalizar mi trabajo de titulación.

A mis mejores amigos Miguel, Raúl, Verito, Dani Pantoja y Dani León por ser parte siempre de mí, y por estar en las buenas y en las malas conmigo.

RESUMEN

El presente estudio se enfoca en la generación de una aplicación en lenguaje Python cuyas funciones son seleccionar puntos de forma manual y generar trayectorias de forma efectiva y óptima para un brazo robótico KUKA KR20 por medio de la implementación de visión artificial. Empleando cualquier tipo de dispositivo con cámara, se realiza el proceso de calibración y corrección de distorsión para mejorar la precisión del método. Empleando visión estereofónica se realiza la reconstrucción 3D con el uso de dos imágenes perfectamente alineadas. Por medio de la reconstrucción, se calcula la profundidad de cualquier punto capturado, para luego triangular y re proyectar el punto obteniendo sus coordenadas 3D. Estas coordenadas calculadas se utilizan como entradas para el programa que genera el código KRL. El código generado tiene la información completa de la acción a realizar, como el tipo de trayectoria, las coordenadas de cada punto en el espacio y las velocidades con las que el brazo llegará a esos puntos.

Palabras clave: Visión artificial, Python, KUKA, Robótica, OpenCV.

ABSTRACT

This research focuses on the development of an application using Python whose functions are to select points manually and generate effective and optimal trajectories for a KUKA KR20 robotic arm through the implementation of artificial vision. This research uses any type of camera device, the calibration and distortion correction processes are performed to improve the accuracy of the method. Using stereo vision, 3D reconstruction is performed with the use of two perfectly aligned images. Through reconstruction, the depth of any point captured is calculated, to then triangulate and re project the point obtaining its 3D coordinates. These calculated coordinates are used as inputs to the program that generates the KRL code. The code generated has the complete information of the action to perform, such as the type of the trajectory, the coordinates of each point in the space and the velocities with which the arm reaches those points.

Key words: Artificial Vision, Python, KUKA, Robotics, OpenCV.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

INTRODUCCIÓN

El campo de la Inteligencia Artificial ha introducido nuevas formas de simplificar el esfuerzo humano tanto a nivel personal como empresarial. Años atrás, la idea de fabricar robots inteligentes, capaces de tomar sus propias decisiones, se lo consideraba una fantasía que se veía solo en películas. Sin embargo, ahora podemos ver en las noticias: computadoras combinadas con inteligencia artificial que pueden llevar a cabo conversaciones simples o inclusive jugar al ajedrez como es el caso de la supercomputadora Deep Blue que fue capaz de vencer al campeón de 1997 Gary Kaspárov (Gibbs, 2019).

La Visión Artificial nació como una rama de la Inteligencia Artificial encargada del reconocimiento y procesamiento de imágenes con las cuales se puede adquirir, procesar, analizar y comprender las imágenes del entorno en tiempo real o aproximado con el fin de producir información numérica que las computadoras puedan entender sea en una imagen o en varias y actuar según la situación que se presente.

Actualmente, uno de los objetivos planteados en la industria es la combinación de la inteligencia artificial con la robótica en forma de automatización para reducir el esfuerzo humano, aumentar la calidad en los procesos y de esa forma mejorar los servicios/productos que se ofrecen a los clientes. Los brazos robóticos en la actualidad reducen el trabajo y aumentan la precisión de los procesos, pero están ligados a la presencia de un operador o a una programación previa de la acción que debe realizar. Esta programación la realiza un técnico del robot y la acción quedará de forma fija hasta una nueva programación.

El verano del 2018 se entregó bajo préstamo con propósitos de investigación un brazo robótico modelo Kuka al Laboratorio de Materiales para Ingeniería Mecánica. El brazo robótico, tal como se muestra en la Figura 1, posee 6 ejes puede realizar acciones tales como soldadura, ensamblaje, movimiento de piezas e inclusive puede incorporarse una extrusor y funcionar como una impresora 3D. El brazo robótico tiene incorporado un sistema que permite

su control por código donde se especifica varios puntos en el espacio formando una trayectoria para el movimiento del mismo. Sin embargo, cuando se quería realizar una acción larga o de precisión este proceso resulta largo al tener que ir tomando punto por punto y grabando dentro de la trayectoria.

Figura 1. Brazo robótico KUKA KR20 (KUKA Roboter GmbH, 2015).

El propósito del proyecto presentado en este documento es incorporar visión artificial para seleccionar y generar trayectorias para el brazo robótico KUKA en cualquier entorno de trabajo. Con el uso de cualquier cámara se espera implementar visión estereofónica para la reconstrucción 3D del espacio de trabajo disminuyendo así el tiempo para generar trayectorias, el esfuerzo humano y el tiempo que se invierte en la preparación del robot para realizar cualquier trabajo sin influir altos costos para la empresa o para el usuario.

En este proyecto, se espera realizar una aplicación capaz de generar código para el robot por medio de la selección manual de los puntos en una computadora, evitando el movimiento hacia el punto con el robot y eliminando así el método actual de generación de trayectorias.

MATERIALES Y MÉTODOS

1. Procesamiento de imágenes

Calibración de cámara

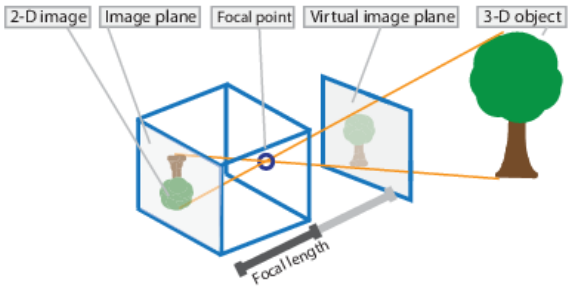

Figura 2. Proyección de una imagen a un punto focal.

Imagen extraída de: Mathworks

En la Figura 2, observamos como es el funcionamiento de una cámara estenopeica donde se utiliza una caja oscura con una pequeña apertura o agujero conocido como punto focal para concentrar toda la luz, en este sistema todos los puntos del entorno pasan por el punto focal plasmando esta información en un sensor o un plano 2D (Zhang, 2000); en otras palabras, nos dice que al capturar una imagen todos los puntos 2D (X, Y) son en realidad transformaciones de los puntos del espacio de coordenadas 3D (X, Y, Z). Entonces, si se realiza esta transformación 3D a 2D deberíamos poder hacer lo contrario y asi encontrar la ubicación del punto real en el mundo 3D solo con la información que una imagen nos da.

Sin embargo, una imagen no es perfecta y puede contener fallas conocidas como distorsiones que se explican en la siguiente sección, para esto se introduce el preprocesamiento de las imágenes o el proceso de calibración y corrección de la cámara, donde el objetivo de este proceso es encontrar información significativa sobre el sistema de visión y así poder corregir estas fallas. La información o los parámetros que se necesitan recuperar incluyen a las distancias focales, el centro de la cámara, la matriz de traslación y matriz de rotación.

Con estos parámetros podemos por ende formular una ecuación para encontrar este punto p en nuestro plano:

$$[p] = M * [P]$$

Ecuación 1. Transformación de puntos 3D a puntos 2D

Donde M es una matriz de proyección que convierte el punto del Mundo (X, Y, Z) en el punto 2D del plano (u, v).

En una visión más compleja, la calibración de la cámara nos proporciona una matriz de parámetros intrínsecos, parámetros extrínsecos y los coeficientes de distorsión. En particular, el modelo de cámara estenopeica (descrito anteriormente) es un modelo ideal que no considera los problemas ópticos que sufren las cámaras comerciales lo que provoca estas distorsiones (tal como se muestra en la Figura 3). En esta figura podemos observar cómo trazos largos y rectos se proyectan como curvas en la imagen.

Figura 3. Distorsión óptica

Estos problemas ópticos son omitidos generalmente en la fotografía profesional ya que pueden ser atractivas visualmente; sin embargo, para aplicaciones de precisión y en visión artificial se deben corregir, debido a que, un pixel ubicado de forma errónea formando una curva puede significar la pérdida o el aumento de unidades en cálculo de distancias.

Parámetros intrínsecos y extrínsecos

Los parámetros extrínsecos forman parte de los vectores de rotación y traslación que son los encargados después de hacer la transformación del punto en 3D P al punto en 2D p. Por otro lado, los parámetros intrínsecos forman parte de la matriz de la cámara y contienen la información relacionada a la misma (distancias focales y centro de la cámara) (Kaehler & Bradski, 2019).

La matriz de la cámara se define a continuación:

$$\begin{bmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{bmatrix}$$

Donde α y β son las distancias focales (respectivamente a x, y); γ es el sesgo de pixeles; y (uc, vc) corresponden al centro de la cámara.

Generalmente el sesgo de pixeles es un valor cercano a cero, y en muchas aplicaciones de visión estereofónica no se lo toma en cuenta. Para nuestra aplicación tampoco haremos uso de este valor por lo que la matriz de la cámara se define solo con las distancias focales y los centros de la cámara.

Tipos de distorsiones

La distorsión, como se menciona anteriormente, es el ruido que se puede encontrar en una imagen, pero estas tienen su origen en la simetría de una lente fotográfica, como menciona Cao en su publicación, las cámaras económicas para reducir costos y dimensiones utilizan lentes que causan distorsiones geométricas relativamente altas (Cao, Park, Shin, Lee & Cho, 2019), y como mencionamos anteriormente el propósito de este proyecto es utilizar cualquier cámara para poder realizar la generación de trayectorias.

Como menciona el modelo de distorsión de Brown en su libro (1996), las distorsiones se clasifican en distorsiones de barril y distorsiones de cojín.

En la distorsión de barril (Figura 4b), la ampliación de la imagen disminuye con la distancia desde el centro de la imagen o eje óptico. El efecto que se puede apreciar es el de una imagen en forma de esfera o barril. Generalmente este tipo de distorsión aparece en fotos con zoom.

En la distorsión de cojín (Figura 4c), la ampliación de la imagen aumenta con la distancia desde el centro de la imagen o eje óptico. El efecto que se puede apreciar es que aquellas líneas que no pasan por el centro de la imagen tienden a arquearse hacia el mismo.

Por último, una imagen sin distorsiones es presentada en la Figura 4a, esta nos muestra una imagen sin distorsiones, con líneas totalmente rectas y paralelas entre sí; sin embargo, las Figuras b y c pueden ser procesadas para que luzcan como la Figura a, que sería ideal para ser procesada para visión estereofónica.

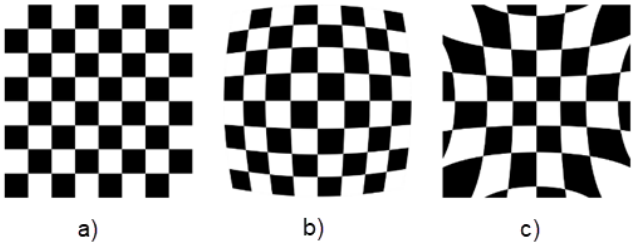

Figura 4. Tipos de distorsión

Homografía

Una homografía es una transformación en perspectiva de un plano, es decir, una nueva proyección de un plano de una cámara a una vista de cámara diferente, en la cual pueden existir nuevos cambios como la traslación (posición) o la rotación (orientación) de la cámara. (Kaehler & Bradski, 2019)

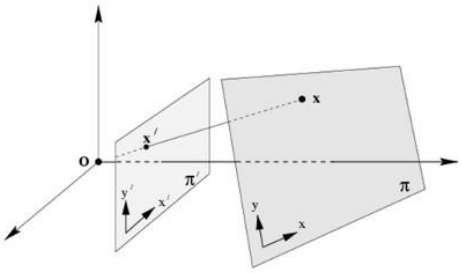

Figura 5. Homografía de dos vistas con el mismo punto x

Las transformaciones en perspectiva mapean puntos tridimensionales en planos de imágenes bidimensionales (Figura 5) utilizando la matriz de transformación que incorpora las características de la cámara: distancia focal, centro óptico y los parámetros extrínsecos (rotación y traslación).

Algoritmo de Zhang

En este documento todo el código utilizado usa librerías de OpenCV, es una biblioteca de código libre que tienen algoritmos de inteligencia artificial y visión artificial. Todos estos algoritmos han sido optimizados para poder trabajar de forma óptima e inclusive a ejecutarse en tiempo real.

A continuación, describimos el algoritmo utilizado por Zhang en su publicación (1998) como método para estimar los parámetros de la calibración de cámara.

Para estimar las matrices de parámetros intrínsecos y extrínsecos en el cual se requiere de imágenes con un patrón geométrico conocido y fijo. A estas imágenes se las toma desde múltiples vistas o puntos de referencia. Suponiendo que se tienen M fotos o vistas, dadas las vistas M, cada vista se compone de un conjunto de puntos para los que se establecen imágenes y coordenadas del mundo. Se consideran N puntos por vista.

En el algoritmo de Zhang, o incluso en cualquier tipo de calibración de la cámara en general, se tiene como fin la obtención de una forma de transformar el mundo real en 3D a coordenadas 2D de imagen en plano. Para obtener este método se utiliza generalmente una imagen de ajedrez dado que el patrón de cuadrícula formado en un tablero de ajedrez es un

patrón lineal muy simple. A su vez su comprensión de la estructura y la forma de los puntos es sencilla.

En la librería de calibración de cámara y reconstrucción de OpenCV se tienen métodos donde se aplica el reconocimiento de estos N puntos para cada vista, para nuestro caso se utiliza la función findchessboardcorners de la cual devuelve los N puntos que corresponden a las esquinas de un tablero de ajedrez, esto se puede observar en la Figura 6 a continuación.

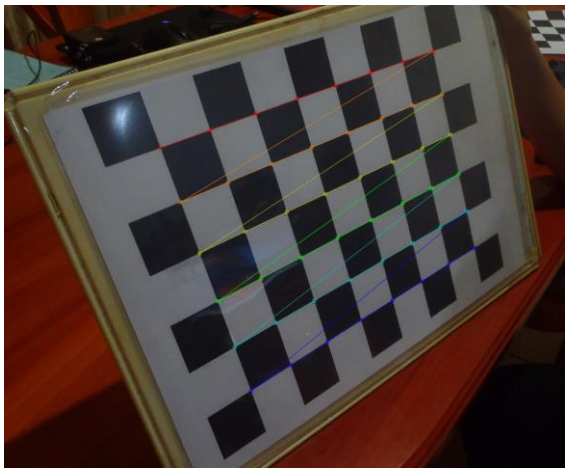

Figura 6. Resultado de la función findchessboardcorners

El método de Zhang consiste en:

- Para las M vistas, se tienen las imágenes M de I_0 a I_{M-1}
- Para cada imagen I_i donde $i = (0 \dots M-1)$: se calculan los N puntos de correspondencia
- Se tienen como U los puntos observados y como X los puntos del modelo. Para la imagen con sus puntos observados (U) extraídos de las vistas M, cada punto se lo denota como $U_{i,j}$, donde i es la vista; y j representa el punto extraído del tablero de ajedrez.
- Por lo tanto, de lo anterior tenemos $U_{i,j} = (u, v)$. Al mismo tiempo, representamos X de forma similar $X_{i,j} = (X, Y, Z)$

- A partir de cada correspondencia entre los puntos del modelo y los puntos observados se puede calcular una homografía, se calcula para cada vista una homografía.
- Del conjunto de homografías estimadas se calculan los parámetros intrínsecos $\alpha, \gamma, u_c, \beta, v_c$.
- Se optimizan los parámetros utilizando un optimizador LM.
- Una vez calculados los parámetros intrínsecos, se estiman los vectores de rotación y traslación (parámetros extrínsecos).
- Usando los parámetros intrínsecos y extrínsecos como suposición inicial para el Optimizador LM, se refinan todos los parámetros.
- Mientras más imágenes o vistas M se tengan mejor será la calibración.

2. Identificación de puntos en el espacio

Cálculo de la profundidad

Al momento de realizar la transformación del espacio 3D a un plano 2D, se pierde información importante del entorno, su profundidad.

Este aspecto hace perder la noción de que tan lejano esta cada punto o pixel de la cámara. Para recuperar esta información se necesita más imágenes o más cámaras lo que hace referencia a la visión estéreo.

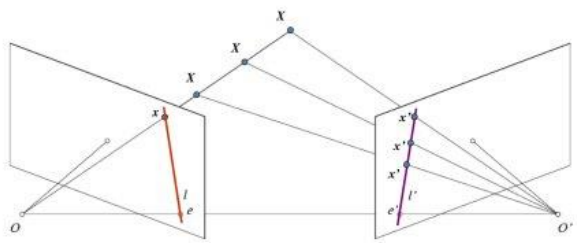

Figura 7. Dos cámaras apuntando al mismo punto

Como se presenta en la Figura 7, al tomar la imagen solo con la cámara izquierda, perdemos información de los puntos X, ya que todos los puntos se proyectan solo en uno. Esto

se soluciona con la imagen tomada por la cámara derecha que corrige la perdida de información obtenida en la primera imagen.

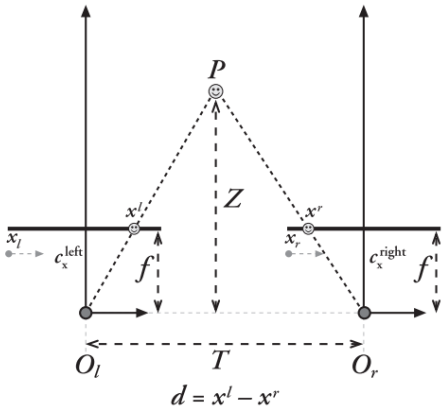

Figura 8. Triangulación de puntos

Asumimos para la Figura 8 que tenemos imágenes sin distorsión y perfectamente alineadas, como podemos observar que las cámaras tienen sus distancias focales definidas y sus centros de cámara son los mismos. Podemos en ambas imágenes observar el punto P como un punto xl y un punto xr tomados en un plano xy respectivamente (con coordenadas en pixeles), esto es conocido como disparidad y es la diferencia de las distancias xl y xr. Con estos datos podemos calcular Z o la profundidad del punto P por medio de una similitud de triángulos.

$$\frac{\boldsymbol{T-d}}{\boldsymbol{Z-f}} = \frac{\boldsymbol{T}}{\boldsymbol{Z}} => \boldsymbol{Z} = \frac{\boldsymbol{fT}}{\boldsymbol{xl-xr}}$$

Ecuación 2. Cálculo de la profundidad con dos imágenes

Como observamos en la ecuación derivada anteriormente, la profundidad es inversamente proporcional a la disparidad, lo que significa que mientras más alejados estemos del objeto la disparidad puede tender a cero y la profundidad tiende a infinito, esto se simplifica en la Figura 9. Por último, este problema de la disparidad tendiendo a cero nos dice que hay que realizar un análisis de sensibilidad de las distancias efectivas para nuestro modelo.

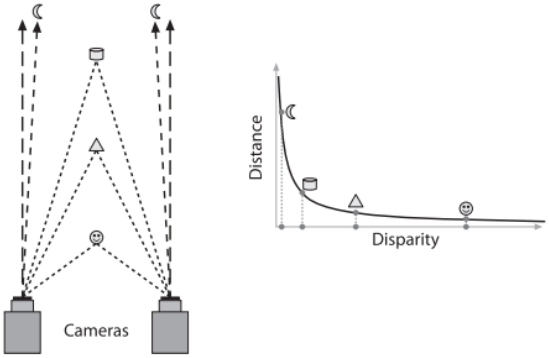

Figura 9. Relación disparidad vs profundidad

Reconstrucción 3D

Si realizamos el cálculo de profundidad de forma iterativa para todos los puntos de las dos fotos, podemos realizar la reconstrucción del entorno completo tal como lo observamos en la Figura 10.

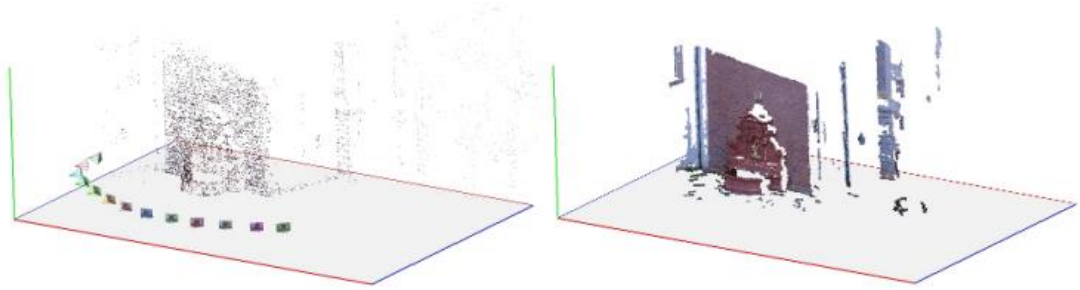

Figura 10. Reconstrucción 3D para varios puntos de dos imágenes

Re proyección

Una vez que obtenemos la profundidad del punto, se procede a realizar la triangulación del punto o la ubicación en las 3 dimensiones del punto desde el centro de la cámara, para esto necesitamos los parámetros de la cámara: distancias focales y el centro de la cámara, parámetros que ya encontramos tras la calibración de nuestra cámara.

La distancia focal de la cámara se puede relacionar directamente con la profundidad a la que está ubicado el punto y el cambio de pixeles por metro, centímetro o milímetro correspondientes al plano donde el punto este. Si derivamos la Ecuación 1 presentada

anteriormente podemos derivar ecuaciones para la triangulación de los puntos X y Y, la demostración se la muestra a continuación.

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \\ \boldsymbol{Z} \end{bmatrix}$$

Donde ya calculamos el valor de z en la sección anterior.

$$\boldsymbol{x'} = \frac{\boldsymbol{x}}{\boldsymbol{z}}$$

$$(\boldsymbol{u} - \boldsymbol{c_x}) = \boldsymbol{f_x} * \boldsymbol{x'}$$

$$\boldsymbol{x} = \frac{\boldsymbol{z}}{\boldsymbol{f_x}} * (\boldsymbol{u} - \boldsymbol{c_x})$$

Ecuación 3. Cálculo de coordenada x

$$\boldsymbol{y'} = \frac{\boldsymbol{y}}{\boldsymbol{z}}$$

$$\left(\boldsymbol{v} - \boldsymbol{c_y}\right) = \boldsymbol{f_y} * \boldsymbol{y'}$$

$$\mathbf{y} = \frac{\boldsymbol{z}}{\boldsymbol{f_y}} * (\boldsymbol{v} - \boldsymbol{c_y})$$

Ecuación 4. Cálculo de coordenada y

Donde u, v son las coordenadas en pixeles del punto de interés en la cámara, y c_x, c_y son los centros de la cámara, todo esto se presenta en la Figura 11 a continuación.

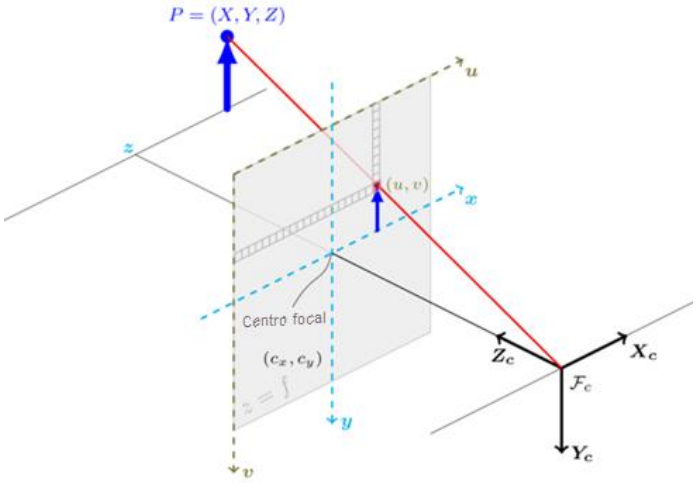

Figura 11. Triangulación de puntos

Imagen obtenida de https://docs.opencv.org

Por último, debemos ubicar el centro de la cámara en un entorno real (en nuestro caso desde el centro del robot), para así poder trasladar estas coordenadas a nuestro sistema de referencia.

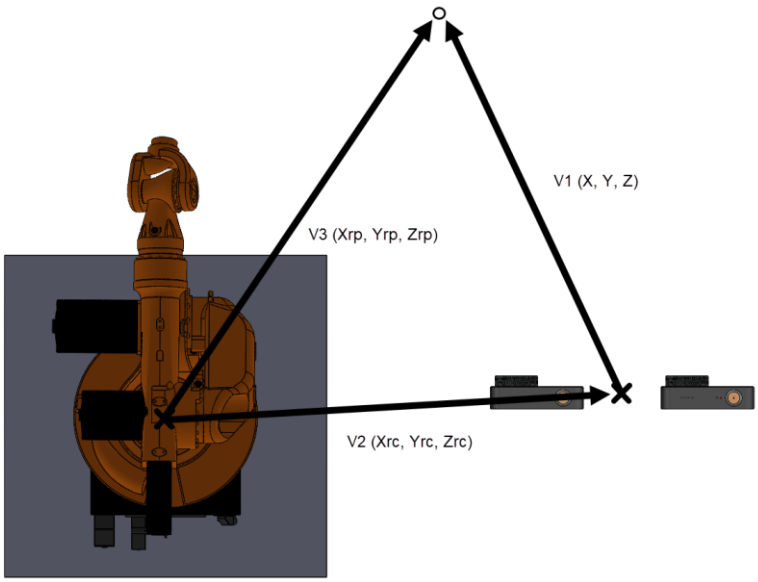

Figura 12. Triangulación desde el centro del robot

Lo que puede ser resulto por suma de vectores de la Figura 12 como lo presenta la Ecuación 5, donde el vector $\overline{V_3}$ es nuestro vector de interés.

$$\overrightarrow{V_3} = \overrightarrow{V_1} + \overrightarrow{V_2}$$

Ecuación 5. Triangulación desde el origen del robot

En resumen, para poder hacer la reconstrucción estéreo de un entorno 3D y la triangulación de puntos el proceso se lo reduce a 4 pasos:

1. No distorsión: Eliminar la distorsión de las imágenes o calibración de cámara.
2. Rectificación: Ajustar ángulos y distancias entre las cámaras.
3. Correspondencia: Encontrar el mapa de disparidad.
4. Triangulación: Encontrar las coordenadas 3D de todos los puntos de interés.

3. Trayectorias y generación de código KUKA

Para este trabajo de investigación se utilizó un brazo robótico KUKA KR20, un brazo robótico de 6 ejes que emplea un modelo de Matrices Homogéneas que representan los movimientos complejos del robot, estos movimientos en el espacio son representados por rotaciones y traslaciones que son simplificadas en este modelo. (Archila & Dutra, 2008). Las limitaciones de este brazo robótico dependen del área de trabajo donde está el robot, la carga máxima que el brazo puede levantar (20 kg) y las limitaciones físicas del robot en cuanto a su longitud (las limitaciones se las presenta a continuación en la Figura 13).

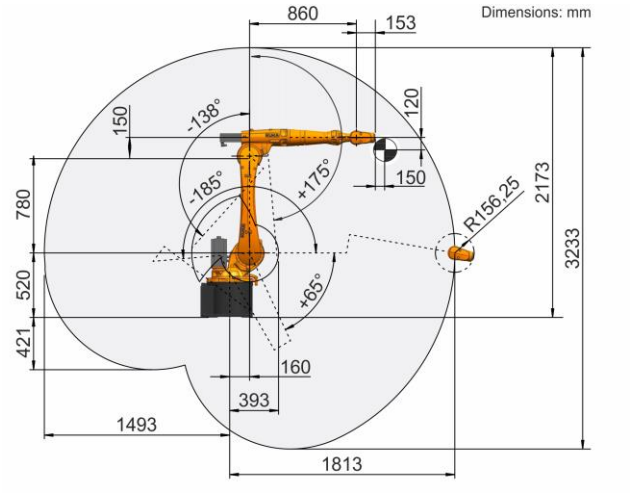

Figura 13. Limitaciones físicas del robot KUKA KR20

Imagen obtenida de: www.kuka-robotics.com

Para poder utilizar el brazo robótico se debe realizar previo a su uso la calibración de la herramienta de trabajo.

Calibración de herramienta de trabajo

Para la calibración de la herramienta de trabajo o conocida como TCP se utiliza un método conocido como Método XYZ 4 que consiste en mover al robot con la herramienta hacia un punto de referencia de 4 formas diferentes (KUKA Roboter GmbH, 2018). La figura a continuación simplifica el proceso:

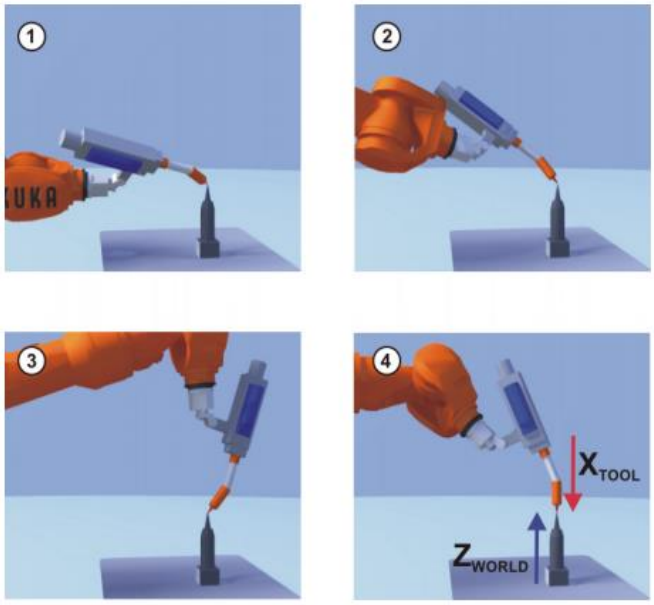

Figura 14. Método XYZ 4 para calibración de herramienta

Imagen obtenida del manual de operación del brazo robótico KUKA KR20

Sintaxis del código KRL

El lenguaje KRL tiene su similitud a la sintaxis del lenguaje Pascal; pero a su vez, para la programación en KRL, el brazo robótico necesita dos archivos: uno .dat y uno .src, toda la información se la almacena en estos dos archivos que contienen información específica de la trayectoria que va a recorrer el robot tales como las velocidades, las posiciones, referencias al tipo de herramienta que se está utilizando o al sistema de referencia o espacio de trabajo.

En el archivo .dat como se muestra a continuación, se especifican datos por default para el robot como los mostrados desde la línea 1 a la línea 4, se define el nombre del archivo o la función en la línea 5 y otros datos de los cuales el robot hace uso interno y no se tiene documentación de los mismos. Para la declaración de un punto, se necesitan de 3 líneas de código, la declaración del punto en coordenadas (línea 17), la definición de la herramienta y el espacio de trabajo (línea 18) y por último las velocidades y aceleraciones del brazo (línea 19). Estas tres definiciones se las debe realizar para cada punto de la trayectoria que el brazo realizará.

```
1.  &ACCESS RV
2.  &REL 1
3.  &PARAM EDITMASK = *
4.  &PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
5.  DEFDAT  diego2
6.  ;FOLD EXTERNAL DECLARATIONS;%{PE}%MKUKATPBASIS,%CEXT,%VCOMMON,%P
7.  ;FOLD BASISTECH EXT;%{PE}%MKUKATPBASIS,%CEXT,%VEXT,%P
8.  EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
9.  DECL INT SUCCESS
10. ;ENDFOLD (BASISTECH EXT)
11. ;FOLD USER EXT;%{E}%MKUKATPUSER,%CEXT,%VEXT,%P
12. ;Make your modifications here
13.
14. ;ENDFOLD (USER EXT)
15. ;ENDFOLD (EXTERNAL DECLARATIONS)
16. DECL BASIS_SUGG_T LAST_BASIS={POINT1[] "P2                    ",POINT2[] "P2
                  ",CP_PARAMS[] "CPDAT2                ",PTP_PARAMS[] "PDAT0
              ",CONT[] "                        ",CP_VEL[] "1
    ",PTP_VEL[] "100                    ",SYNC_PARAMS[] "SYNCDAT                ",SPL
    _NAME[] "S0                       ",A_PARAMS[] "ADAT0                    "}
17. DECL E6POS XP1={X -349.052399,Y -1833.10522,Z 793.852966,A -
    86.1348267,B 39.5483360,C 106.055023,S 6,T 27,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6
     0.0}
18. DECL FDAT FP1={TOOL_NO 10,BASE_NO 14,IPO_FRAME #BASE,POINT2[] " ",TQ_STATE FALSE}
19. DECL LDAT LCPDAT1={VEL 2.00000,ACC 100.000,APO_DIST 100.000,APO_FAC 50.0000,AXIS_VE
    L 100.000,AXIS_ACC 100.000,ORI_TYP #VAR,CIRC_TYP #BASE,JERK_FAC 50.0000,GEAR_JERK 5
    0.0000,EXAX_IGN 0}
20. DECL E6POS XP2={X -688.650391,Y -1546.12341,Z 780.054443,A -
    116.765717,B 40.8322296,C 106.624649,S 6,T 27,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6
     0.0}
21. DECL FDAT FP2={TOOL_NO 10,BASE_NO 14,IPO_FRAME #BASE,POINT2[] " ",TQ_STATE FALSE}
22. DECL LDAT LCPDAT2={VEL 2.00000,ACC 100.000,APO_DIST 100.000,APO_FAC 50.0000,AXIS_VE
    L 100.000,AXIS_ACC 100.000,ORI_TYP #VAR,CIRC_TYP #BASE,JERK_FAC 50.0000,GEAR_JERK 5
    0.0000,EXAX_IGN 0}
23. ENDDAT
```

Figura 15. Ejemplo KRL archivo .dat

Para el archivo .src tenemos definiciones similares a las del archivo .dat, y es necesario su definición debido a que el controlador del robot accede directamente a este archivo y llama al .dat para los detalles de las coordenadas, en otras palabras, el archivo .src es el controlador de la trayectoria del robot y el .dat contiene las especificaciones de cada punto de la trayectoria. De la misma forma aquí observamos que para cada punto, se hace un bloque específico donde se detalla información que el robot utiliza y que en este proyecto a excepción de la velocidad no modificaremos.

```
1.  &ACCESS RV
2.  &REL 1
3.  &PARAM EDITMASK = *
4.  &PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
5.  DEF diego2( )
6.  ;FOLD INI;%{PE}
7.    ;FOLD BASISTECH INI
8.      GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
9.      INTERRUPT ON 3
10.     BAS (#INITMOV,0 )
11.   ;ENDFOLD (BASISTECH INI)
12.   ;FOLD USER INI
13.     ;Make your modifications here
14.
15.   ;ENDFOLD (USER INI)
16. ;ENDFOLD (INI)
17.
18. ;FOLD LIN P1 Vel=1 m/s CPDAT1 Tool[10]:b1 Base[14]:b1;%{PE}%R 8.3.44,%MKUKATPBASIS,
    %CMOVE,%VLIN,%P 1:LIN, 2:P1, 3:, 5:1, 7:CPDAT1
19. $BWDSTART=FALSE
20. LDAT_ACT=LCPDAT1
21. FDAT_ACT=FP1
22. BAS(#CP_PARAMS,1)
23. LIN XP1
24. ;ENDFOLD
25.
26. ;FOLD LIN P2 Vel=1 m/s CPDAT2 Tool[10]:b1 Base[14]:b1;%{PE}%R 8.3.44,%MKUKATPBASIS,
    %CMOVE,%VLIN,%P 1:LIN, 2:P2, 3:, 5:1, 7:CPDAT2
27. $BWDSTART=FALSE
28. LDAT_ACT=LCPDAT2
29. FDAT_ACT=FP2
30. BAS(#CP_PARAMS,1)
31. LIN XP2
32. ;ENDFOLD
33.
34. END
```

Figura 16. Ejemplo KRL archivo .src

4. Prototipo de aplicación para control del brazo robot

Una vez que tenemos noción de las aplicaciones y herramientas a utilizar, implementamos todas en una aplicación compacta que es el fin de este proyecto, es decir se realizará una aplicación de escritorio donde cualquier usuario pueda, mediante el uso de una cámara y la toma de fotos, controlar y especificar un plan de trabajo para el brazo robótico KUKA KR20. La aplicación se compone de 3 herramientas principales y 1 herramienta para configuración de parámetros del robot:

Calibración de cámara

Por medio de varias fotos de tableros de calibración, el usuario puede encontrar los parámetros intrínsecos de la cámara que está utilizando para la captura de las imágenes.

Mapa de disparidades

El usuario puede generar el mapa de disparidades y la reconstrucción del entorno 3D por medio de dos imágenes las cuales pueden ser seleccionadas por el usuario.

Generación de código KRL

El usuario por medio de este menú puede seleccionar dos imágenes para luego hacer la selección de puntos por las cuales el programa calcula y encuentra la ubicación de los puntos desde el centro del robot hacia los puntos marcados para generar la trayectoria en archivos de código KRL para poder ser usados dentro del robot.

Por último, en la herramienta de configuración se puede configurar velocidades para el movimiento del robot y el nombre, número y las dimensiones de la herramienta.

RESULTADOS

1. Calibración de cámara

Tras utilizar el código adjunto en el Anexo B, y tras varias iteraciones donde se eliminan las imágenes de tableros repetidas o que adjuntan a la calibración con un error mayor se concluyó que a medida que se disminuye la resolución de la imagen el error tiende a disminuir hasta cierto valor de resolución donde el error sube nuevamente, este cálculo de la resolución optima mejora los tiempos de cómputo y disminuye el error en el cálculo de los parámetros intrínsecos de la cámara. Los resultados se los adjunta en la tabla a continuación (se encontró que el mejor set para calibración de cámara contiene 15 fotos de tableros).

Tabla 1. Resultados calibración de cámara

Resolución	**Error [%]**
1920x1440	5.98
1866x1400	5.13
1600x1200	4.95
1440x1080	4.50
1333x999	4.39
1066x800	3.57
960x720	3.22
800x600	28.91
640x479	20.00
533x400	27.66
266x200	60.09

A continuación, en la Figura 17 se muestra un ejemplo de la corrección de la distorsión en una imagen tras el proceso de calibración de la cámara. Como se puede observar el cambio en las líneas de las teclas en la parte posterior del teclado.

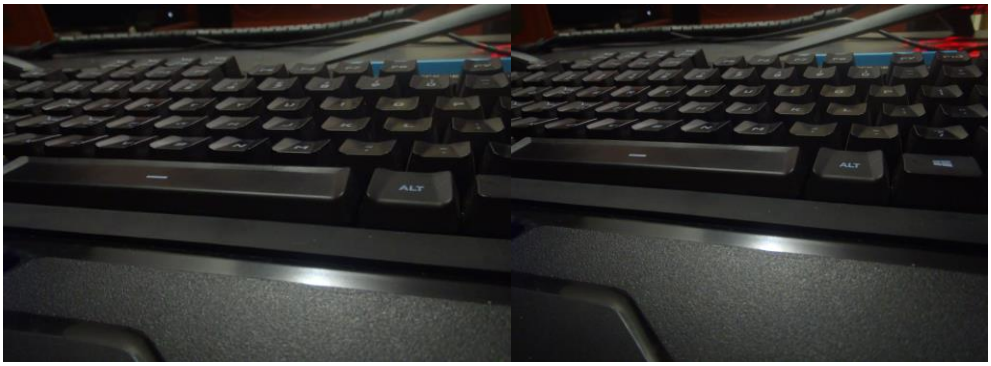

Figura 17. Imagen con distorsión vs Imagen con distorsión corregida

2. Identificación de puntos en el espacio

Cálculo de profundidad

Como mencionamos, por medio de la toma de dos imágenes separadas una distancia d, sin distorsión y perfectamente alineadas podemos obtener más información de la imagen como la profundidad de la misma.

Una vez conocido los valores de las distancias focales de la cámara, la separación de las dos imágenes y el valor en pixeles de los puntos a encontrar podemos comenzar a predecir profundidad de puntos u objetos en el plano 2D.

Figura 18. Par de imágenes separadas 40 cm para cálculo de profundidad

El cálculo de la profundidad no nos devuelve una distancia vectorial al punto si no nos devuelve la distancia desde el plano donde está la cámara hacia el punto, es decir la distancia en el eje X manejando nuestro eje de coordenadas como lo muestra la Figura 18.

Analizando varios puntos podemos encontrar la estimación de las profundidades para varios objetos como muestra la Figura 21.

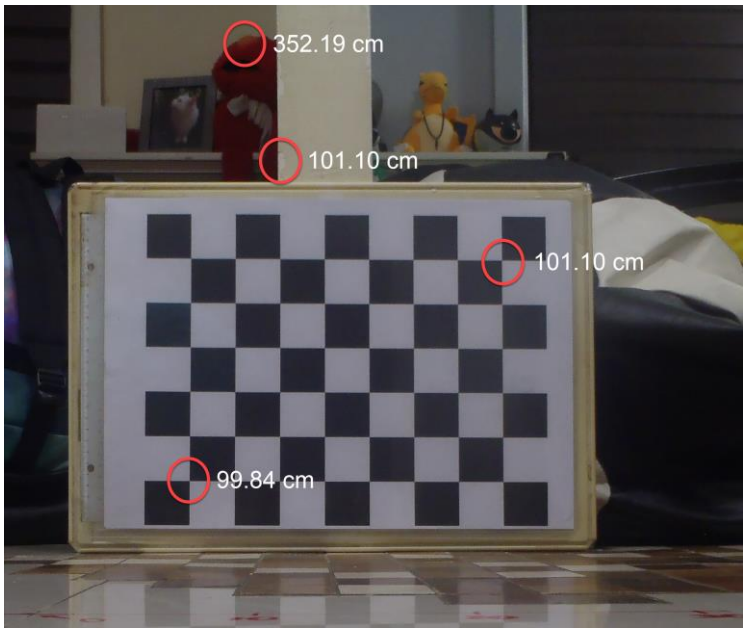

Figura 19. Imagen con valores de profundidad de varios puntos

Para el cálculo de errores nos enfocamos en los puntos del ajedrez al momento de encontrar la profundidad del tablero.

Probamos distancias de 50, 100 y 150cm desde la cámara hacia el tablero y probamos separaciones entre imágenes de 5, 10, 20, 30 y 40cm; sin embargo, para una distancia de 50cm desde la cámara al tablero, las separaciones de distancias ya no mostraban al tablero por lo que se redujo solo a separación entre imágenes de 5 y 10cm.

Tabla 2. Errores para mediciones a 150cm desde la cámara hacia el tablero

Separación entre imágenes [cm]	Profundidad [cm]	Error [%]
40	137.1786429	8.55%
30	178.5555136	19.04%
30	127.4342624	15.04%
20	136.4186478	9.05%
20	172.2241606	14.82%
20	155.1558647	3.44%
10	148.7245127	0.85%
10	131.7734926	12.15%
10	152.2445135	1.50%
10	144.6658322	3.56%
5	144.3830764	3.74%
5	142.9664402	4.69%
5	142.5119436	4.99%
5	150.4639289	0.31%
5	147.6049218	1.60%

Tabla 3. Errores para mediciones a 100cm desde la cámara hacia el tablero

Separación entre imágenes [cm]	Profundidad [cm]	Error [%]
40	94.69159798	5.31%
30	103.4648183	3.46%
30	98.13965243	1.86%
20	105.2882635	5.29%
20	100.3295919	0.33%
20	98.88399439	1.12%
10	100.7319061	0.73%
10	101.5883541	1.59%
10	102.7491361	2.75%
10	98.63462902	1.37%
5	102.2062335	2.21%
5	101.1652718	1.17%
5	100.2179334	0.22%
5	101.7305221	1.73%
5	100.2402453	0.24%

Tabla 4. Errores para mediciones a 50cm desde la cámara hacia el tablero

Separación entre imágenes [cm]	Profundidad [cm]	Error [%]
10	53.97231014	7.94%
5	55.94383166	11.89%
5	52.13501545	4.27%

Podemos observar que a medida que disminuimos la separación entre imágenes o la disparidad, el error disminuye, por lo que los mejores resultados se obtienen cuando la distancia entre imágenes es de 5cm y que la distancia óptima entre la cámara y el objeto debe estar entre los 100cm para obtener mejor noción de la profundidad con menor error.

Reconstrucción 3D

Una vez que se consideran los 100cm entre la cámara y el objeto o este caso el área de trabajo se procede a realizar el cálculo de la profundidad no solo para un punto si no para la mayor cantidad de puntos de forma iterativa hasta realizar un mapa de disparidad donde podemos hacer la reconstrucción 3D del entorno.

A continuación, se muestran los mapas de disparidad y la reconstrucción 3D para distintos sets de imágenes.

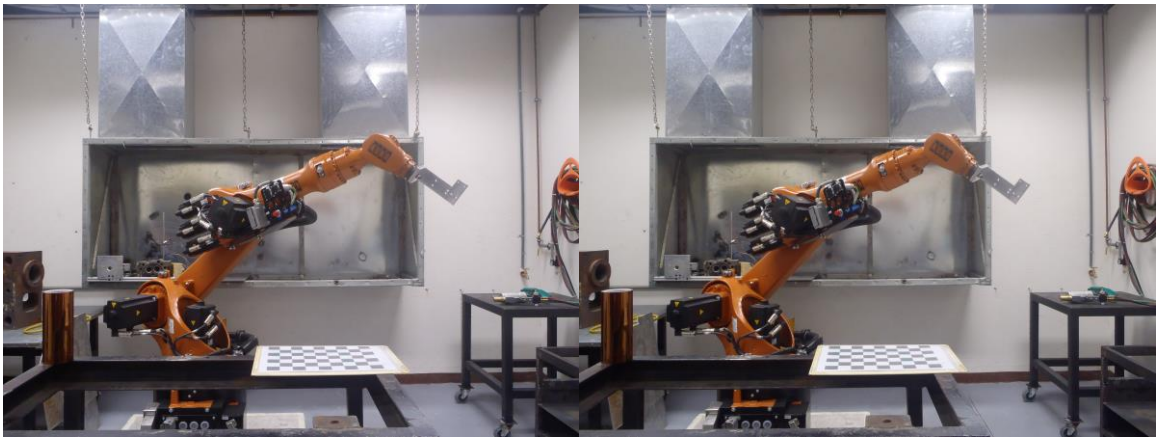

Figura 20. Set 1 de imágenes para reconstrucción 3D

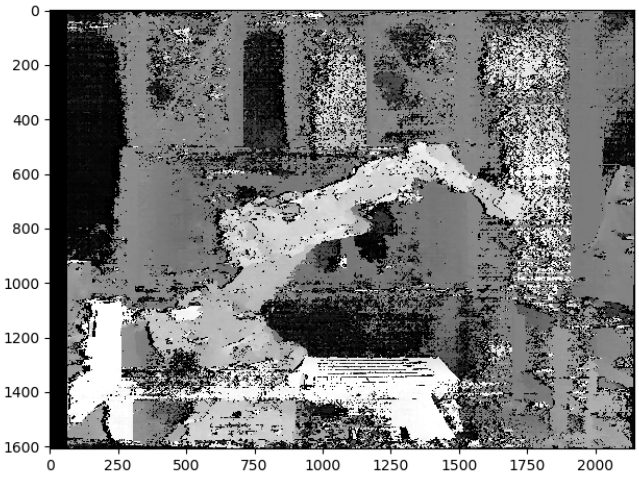

Figura 21. Mapa de disparidad para Set 1 de imágenes

Figura 22. Reconstrucción 3D para Set 1 de imágenes

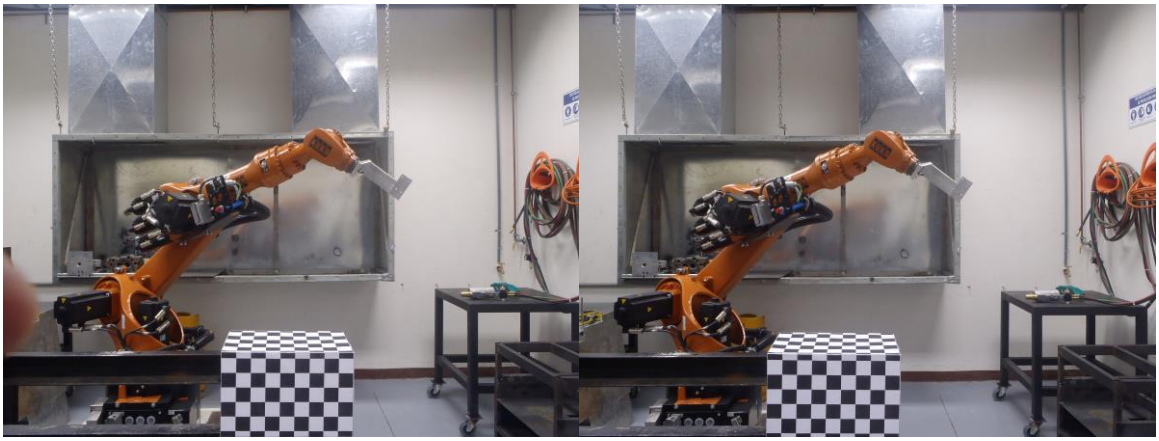

Figura 23. Set 2 de imágenes para reconstrucción 3D

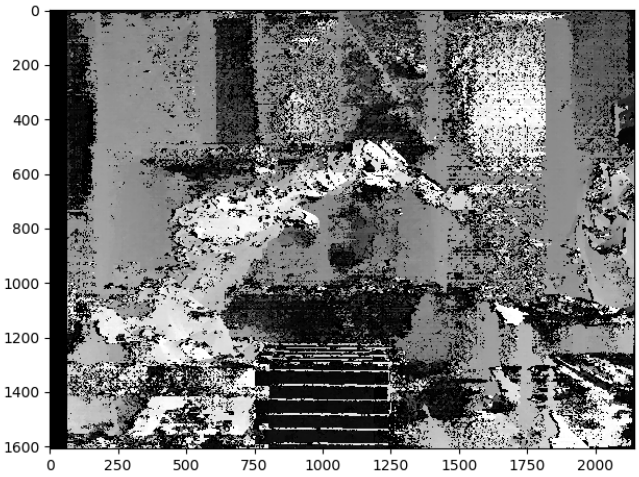

Figura 24. Mapa de disparidad para Set 2 de imágenes

Figura 25. Reconstrucción 3D para Set 2 de imágenes

Triangulación de puntos

Para el sistema que utilizamos y la herramienta utilizada, se obtuvieron los siguientes parámetros a utilizar en la aplicación:

Tabla 5. Distancia del robot al centro de la cámara

Coordenada	Distancia [mm]
X	-226.659
Y	-841.32
Z	926.16

Tabla 6. Tamaño de la herramienta a utilizar

Coordenada	Tamaño [mm]
X	39.62
Y	-24.19
Z	-104.31

Una vez que se obtiene la profundidad, se procede a calcular las coordenadas reales de los puntos y mover al robot hacia los puntos para encontrar los errores en milímetros de los puntos encontrados con la ubicación real del robot, esto se resume en la Tabla 7.

Tabla 7. Errores para triangulación de puntos

Coordenada	Error en coordenada [mm]
X	39.97
Y	22.42
Z	4.85

3. Prototipo de aplicación para control del brazo robot

A continuación, se muestra la interface de la aplicación y se explica su funcionamiento. El código se lo adjunta en el Anexo A al final del documento.

Interface principal

En la ejecución del archivo principal main.py, se ejecuta la interface gráfica y aparece el menú principal que muestra los dos menús: el de configuración y el de aplicaciones.

Figura 26. Interfaz principal aplicativo

Menú de configuración

En este menú, se pueden definir los ángulos de trabajo del robot, el tamaño de la herramienta a utilizar, la velocidad a la que operara el robot y el nombre y número de la herramienta que se utilizará, estos valores están seteados y no necesitan ser modificados cada vez que se ejecute la aplicación.

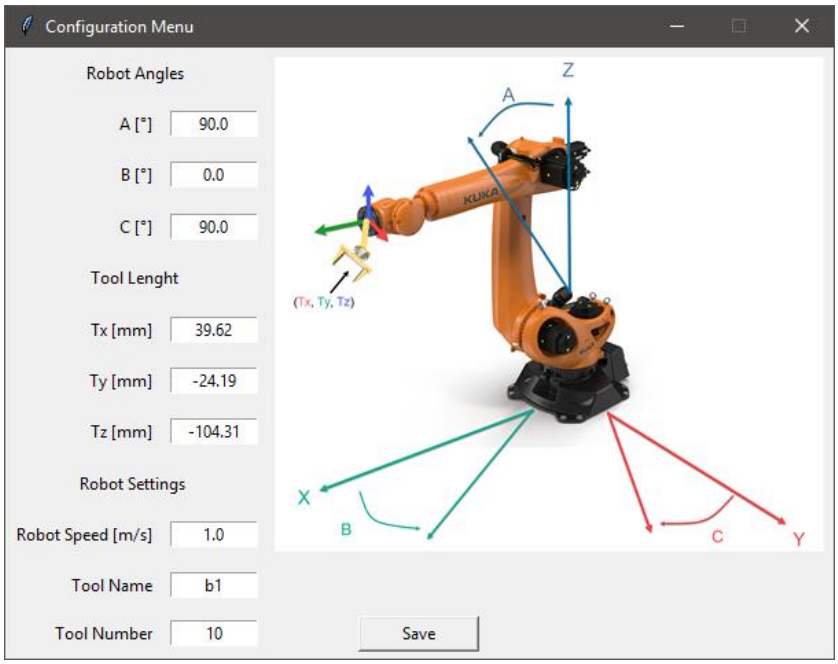

Figura 27. Interfaz de parámetros de configuración

Menú de calibración de cámara

En este menú, se definen los tamaños del tablero de ajedrez, el número de iteraciones que el algoritmo realiza hasta encontrar los puntos del tablero, el error máximo aceptable para la detección del punto en el ajedrez y la ruta del directorio donde se encuentra el set de

imágenes de calibración. De igual forma, estos valores están seteados y no necesitan ser modificados cada vez que se ejecute la aplicación.

Una vez que el algoritmo termina de ejecutarse, aparece un cuadro de dialogo mostrando el porcentaje de error del proceso de calibración.

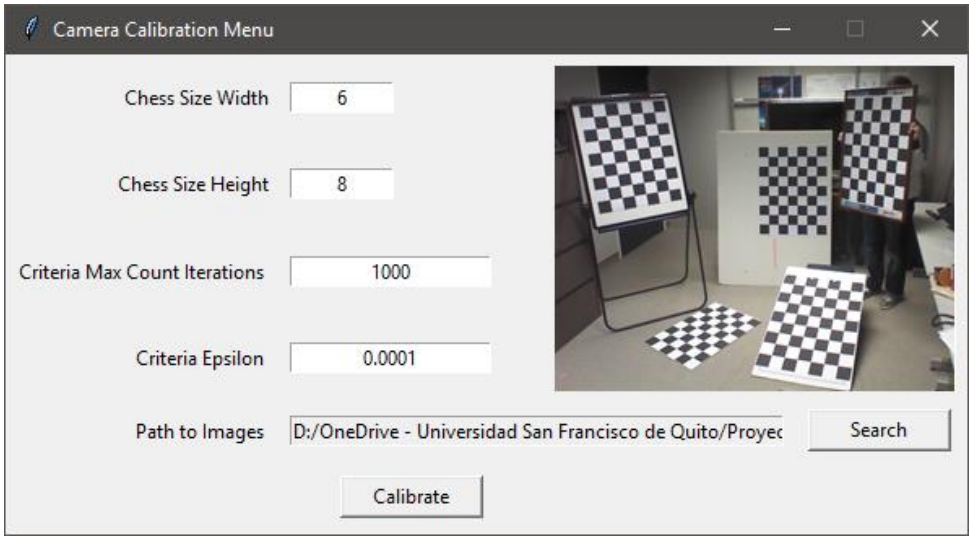

Figura 28. Interfaz de calibración de cámara

Menú de reconstrucción 3D

En este menú, se definen parámetros para la generación del mapa de disparidades y la reconstrucción 3D. Los parámetros utilizados en esta interfaz se definen a continuación:

- Window Size, Speckle Window Size y Speckle Range son parámetros con los que el algoritmo toma los pixeles adyacentes a la disparidad para realizar el clustering de pixeles.
- Disp12 Max Diff define la máxima diferencia permitida entre una disparidad y otra.
- Min disparity define el valor mínimo de las disparidades en el entorno.
- Num disparities define el número de disparidades del entorno.
- Uniqueness Ratio define el valor mejor para considerar una coincidencia correcta al clasificar un pixel entre clusters.

- Downsize Stereo define la cantidad de veces que se disminuye el tamaño de la imagen para generar la reconstrucción 3D.

Este menú tiene dos acciones: al ejecutar la generación del mapa de disparidad se abre una nueva ventana mostrando la imagen generada y al ejecutar la reconstrucción 3D se genera un archivo ply que contiene la reconstrucción 3D del modelo. Este archivo puede ser ejecutado y visualizado en el programa MeshLab.

Figura 29. Interfaz de reconstrucción 3D

Menú de selección de puntos y generación de código KRL

En este menú, se permite al usuario escoger los puntos para generar trayectorias, para esto el usuario puede seleccionar manualmente los puntos similares de las dos imágenes, se utilizan los parámetros generados en la calibración de la cámara. En la primera pantalla, se seleccionan las dos imágenes que se utilizaran (imagen izquierda e imagen derecha).

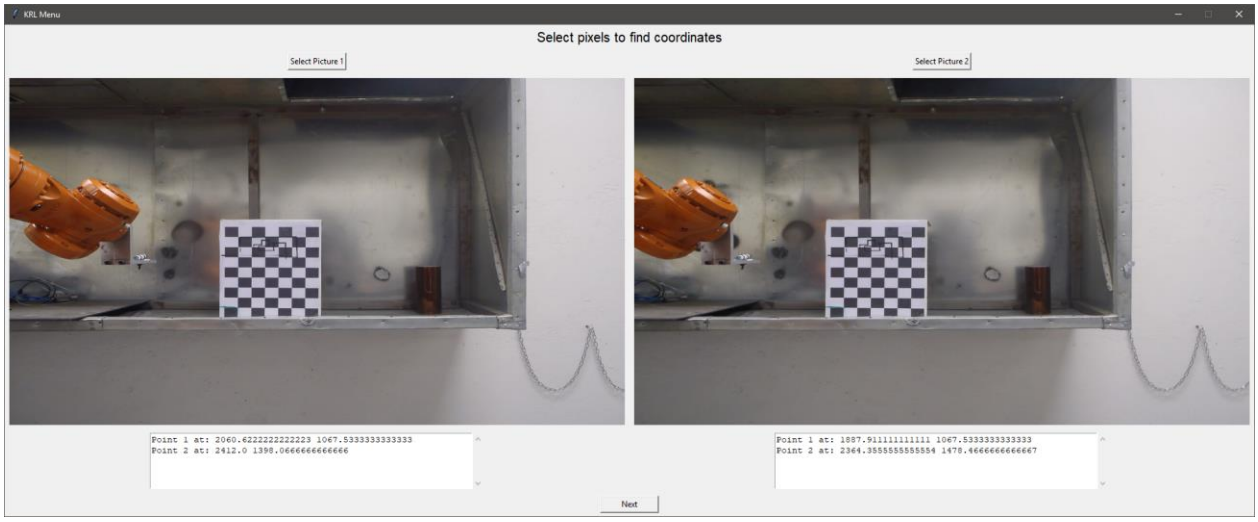

Figura 30. Interfaz 1 de selección de puntos

Una vez seleccionados los puntos, se puede continuar a la interfaz 2 donde el usuario debe ingresar las coordenadas de la cámara desde el centro del robot, la distancia T entre las dos imágenes y el tipo de trayectoria que se quiere generar.

Los tipos de trayectorias son: trayectoria punto a punto, donde el robot pasa por todos los puntos seleccionados y trayectoria polinomial, donde el robot pasa por el primer punto y el ultimo e intenta acercarse lo más posible por los puntos intermedios.

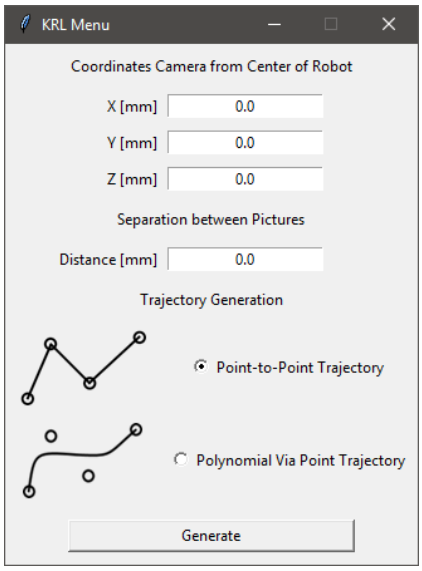

Figura 31. Interfaz 2 de generación de puntos

Por último, al iniciar la generación de código, el programa pide la ubicación para guardar los archivos generados y el nombre que se les dará a los mismos. Una vez finalizado,

el usuario puede manipular estos archivos dentro del robot y ejecutarlos para que el robot siga la trayectoria seleccionada.

DISCUSIONES Y RECOMENDACIONES

Uno de los primeros problemas que se encontró a medida que se desarrolló este proyecto, fue la dificultad para capturar dos imágenes perfectamente alineadas. Como se observa a continuación en la Figura 32, el tablero ha sido desplazado en el eje vertical y no solo en el eje horizontal, esto deriva problemas en el cálculo de la profundidad ya que en el cálculo no se considera la diferencia de pixeles en el eje vertical lo que termina en un error en la triangulación de los puntos.

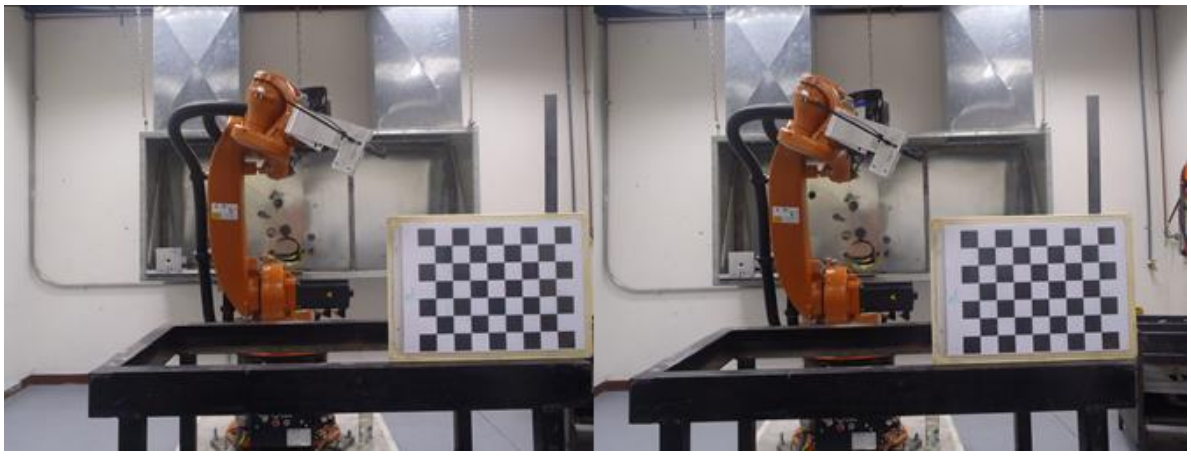

Figura 32. Comparación de la captura estereofónica

El primer acercamiento a la corrección de este error fue diseñar y construir una base estable para el trípode de la cámara en la cual la cámara se desplaza sin mover todo el trípode en sí.

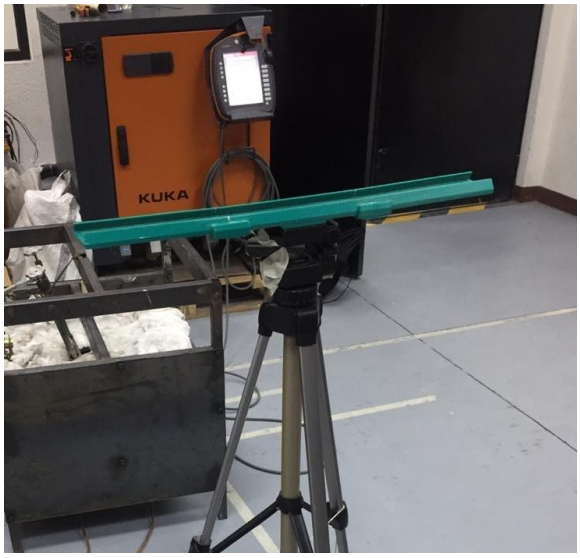

Figura 33. Base estable para trípode

Otro de los factores importantes en el cálculo de la profundidad, es mejorar el error en la calibración de la cámara, a pesar de utilizar un mayor set de imágenes para calibración, el error de calibración nunca llega a disminuir el 5%, esto puede representar un error en la distancia focal y por ende en el cálculo de la profundidad.

Para esto, una de las opciones es cambiarse a una cámara full frame que elimina las distorsiones al no tener lentes en su configuración (Max, 2017). Sin embargo, esto representa un alto costo al tener que adquirir estas cámaras que superan los $1500 en precio.

Otra opción, es agregar un sistema de detección de profundidad a un punto seleccionado, sea por medio de infrarrojo o por sonar si se tiene la profundidad de un punto, se puede realizar la triangulación del punto con las imágenes utilizando los mismos cálculos presentados en este documento.

Otro de los factores a considerar, es el de la selección de los pixeles en la aplicación, al tener imágenes con millones de pixeles un leve movimiento en la selección puede no ser el pixel que se desee. La solución sería dividir la interface en 3 pantallas donde primero se selecciona los pixeles de la imagen izquierda y luego los pixeles de la imagen derecha, esto

afecta un poco a la interfaz amigable con el usuario, pero se puede recuperar y disminuir el error en la selección de los pixeles.

Para futuras investigaciones, se recomienda buscar métodos para mejorar la captura de las imágenes tanto para la calibración de la cámara, como para la captura de imágenes para la selección de trayectorias. Se puede hacer uso de cámaras estereofónicas o incluyendo otra cámara para mantener siempre fijas la captura de la imagen izquierda y derecha.

Se recomienda mejorar la aplicación implementando múltiples hilos para poder ejecutar más de un proceso a la vez, es decir iniciar una calibración de cámara mientras se prueban nuevos parámetros para la generación del mapa de disparidades.

Para la aplicación se puede también agregar cuadros de diálogos y manejo de excepciones a la aplicación para manejar los campos vacíos o valores que no son números dentro de los parámetros.

CONCLUSIONES

El principio del proyecto era recuperar la información de la profundidad de una imagen 2D, lo cual se lo realizó por medio del análisis de una imagen extra perfectamente alineada separada una distancia conocida, la información de la profundidad nos ayudó a re proyectar el punto de una coordenada 2D a una coordenada 3D obteniendo asi la ubicación real del punto desde el centro de la cámara. En conclusión, la triangulación de cualquier punto en el espacio se la realiza conociendo primeramente la profundidad del punto, si se tiene errores en este cálculo, la triangulación tambien reflejara errores.

Se simplificó y se optimizó la generación del código para el robot, al programa presentado en este documento se lo alimenta por medio de la selección de pixeles en dos imágenes, lo cual resulta en la disminución del tiempo invertido en la planeación de un trabajo para el robot.

Por último, se concluye que los errores presentados en el proyecto pueden ser corregidos si se mejora la técnica de la captura de las imágenes y se disminuye el error en la calibración de la cámara, dos factores que pueden ser investigados en el futuro para hacer uso de esta herramienta y eliminar asi el proceso manual del movimiento del robot a los puntos.

REFERENCIAS BIBLIOGRÁFICAS

[1] Gibbs, S. (2019). AlphaZero AI beats champion chess program after teaching itself in four hours. Obtenido el 26 de noviembre del 2019, de https://www.theguardian.com/technology/2017/dec/07/alphazero-google-deepmind-ai-beats-champion-program-teaching-itself-to-play-four-hours

[2] KUKA Roboter GmbH. (2015). KR20-3 Assembly Instructions. Obtenido el 30 de noviembre de 2019, de: http://www.wtech.com.tw/public/download/manual/kuka/KUKA%20KR%2020-3.pdf

[3] Zhang Z., (2000). A Flexible New Technique for Camera Calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 22, No. 11, p. 1330-1334.

[4] Mathworks. (2019). What Is Camera Calibration? MATLAB & Simulink. Obtenido el 30 de noviembre de 2019, de: https://www.mathworks.com/help/vision/ug/camera-calibration.

[5] Kaehler, A., & Bradski, G. (2019). Learning OpenCV.

[6] KUKA Roboter GmbH. (2018). KUKA System Software 8.3. Operating and Programming Instructions for System Integrators. Zugspitzstraße 140 D-86165. Augsburg-Germany

[7] Cao, V., Park, Y., Shin, J., Lee, J., & Cho, H. (2019). A Simple Method for Correcting Lens Distortion in Low-Cost Camera Using Geometric Invariability.

[8] Brown, D.C. (1996). Decentering distortion of lenses. Photogrammetric Engineering and Remote Sensing 24, 555–566

[9] Zhang, Z. (1998). A flexible new technique for camera calibration. Technical Report MSR-TR-98-71.

[10] Max, T. (2017). Technical Aspects of a Digital SLR. Retrieved 1 December 2019, from https://www.learnopencv.com/technical-aspects-of-a-digital-slr/

ANEXO A: CÓDIGO PYTHON CONTROLADOR KUKA

Archivo main.py

```
1.  import mod_disparity_map
2.  import fnmatch
3.  import os
4.  import mod_generate_kukacode
5.  from tkinter import *
6.  from tkinter import messagebox
7.  from tkinter.filedialog import askdirectory, asksaveasfilename
8.  from tkinter.filedialog import askopenfilename
9.  from tkinter.scrolledtext import ScrolledText
10.
11. import numpy as np
12. from PIL import Image
13.
14. import mod_camera_calibration
15.
16.
17. class App:
18.     def __init__(self, master):
19.         menubar = Menu(master)
20.         master.config(menu=menubar)
21.
22.         self.choose_one = False
23.         self.one_width = 0
24.         self.one_height = 0
25.         self.choose_two = False
26.         self.two_width = 0
27.         self.two_height = 0
28.         self.counter_one = 1
29.         self.counter_two = 1
30.         self.pixels_one = []
31.         self.pixels_two = []
32.
33.         filemenu = Menu(menubar, tearoff=0)
34.         filemenu.add_command(label="Robot Options", command=self.configuration_menu
    )
35.         filemenu.add_separator()
36.         filemenu.add_command(label="Quit", command=root.quit)
37.
38.         editmenu = Menu(menubar, tearoff=0)
39.         editmenu.add_command(label="Calibrate Camera", command=self.calibrate_menu)

40.         editmenu.add_command(label="Disparity Map", command=self.disparity_map_menu
    )
41.         editmenu.add_command(label="Kuka Code Generation", command=self.kuka_genera
    tion_menu)
42.
43.         menubar.add_cascade(label="Configuration", menu=filemenu)
44.         menubar.add_cascade(label="Applications", menu=editmenu)
45.
46.         if getattr(sys, 'frozen', False):
47.             self.current_path = sys._MEIPASS
48.         else:
49.             self.current_path = os.path.dirname(__file__)
50.
51.         self.frame = Frame(master)
52.         self.frame.grid()
53.
```

```
54.         photo = PhotoImage(file=self.current_path + "//Binaries//logo.png")
55.         w = Label(self.frame, image=photo)
56.         w.photo = photo
57.         w.grid(row=0, column=4, columnspan=5, rowspan=3, padx=5, pady=5)
58.
59.         self.chess_size = [0, 0]
60.         self.criteria_max_count = 10
61.         self.criteria_epsilon = 0.01
62.         self.path_images = ''
63.
64.     def configuration_menu(self):
65.         toolnameS = StringVar()
66.         toolnameS.set(np.load('./Binaries/tool_name.npy'))
67.         toolnumberS = StringVar()
68.         toolnumberS.set(np.load('./Binaries/tool_number.npy'))
69.         speedrobotS = StringVar()
70.         speedrobotS.set(np.load('./Binaries/speed_robot.npy'))
71.         angleaS = StringVar()
72.         angleaS.set(np.load('./Binaries/angle_a.npy'))
73.         anglebS = StringVar()
74.         anglebS .set(np.load('./Binaries/angle_b.npy'))
75.         anglecS = StringVar()
76.         anglecS.set(np.load('./Binaries/angle_c.npy'))
77.         tool_x = StringVar()
78.         tool_x.set(np.load('./Binaries/toolx.npy'))
79.         tool_y = StringVar()
80.         tool_y.set(np.load('./Binaries/tooly.npy'))
81.         tool_z = StringVar()
82.         tool_z.set(np.load('./Binaries/toolz.npy'))
83.
84.         def save_configuration():
85.             np.save('./Binaries/tool_name', toolnameS.get())
86.             np.save('./Binaries/angle_a', angleaS.get())
87.             np.save('./Binaries/angle_b', anglebS.get())
88.             np.save('./Binaries/angle_c', anglecS.get())
89.             np.save('./Binaries/toolx', tool_x.get())
90.             np.save('./Binaries/tooly', tool_y.get())
91.             np.save('./Binaries/toolz', tool_z.get())
92.             np.save('./Binaries/tool_number', toolnumberS.get())
93.             np.save('./Binaries/speed_robot', speedrobotS.get())
94.             root2.destroy()
95.
96.         root2 = Toplevel()
97.         root2.title('Configuration Menu')
98.         root2.resizable(0, 0)
99.         frame = Frame(root2)
100.                frame.grid()
101.
102.                Label(frame, pady=8, text="Robot Angles").grid(row=0, column=0, colu
    mnspan=2, padx=5)
103.                Label(frame, pady=8, text="A [°]").grid(row=1, column=0, sticky=E, p
    adx=5)
104.                Label(frame, pady=8, text="B [°]").grid(row=2, column=0, sticky=E, p
    adx=5)
105.                Label(frame, pady=8, text="C [°]").grid(row=3, column=0, sticky=E, p
    adx=5)
106.                Label(frame, pady=8, text="Tool Lenght").grid(row=4, column=0, colum
    nspan=2, padx=5)
107.                Label(frame, pady=8, text="Tx [mm]").grid(row=5, column=0, sticky=E,
     padx=5)
108.                Label(frame, pady=8, text="Ty [mm]").grid(row=6, column=0, sticky=E,
     padx=5)
109.                Label(frame, pady=8, text="Tz [mm]").grid(row=7, column=0, sticky=E,
     padx=5)
110.                Label(frame, pady=8, text="Robot Settings ").grid(row=8, column=0, c
    olumnspan=2, padx=5)
```

```
111.                Label(frame, pady=8, text="Robot Speed [m/s]").grid(row=9, column=0,
     sticky=E, padx=5)
112.                Label(frame, pady=8, text="Tool Name").grid(row=10, column=0, sticky
    =E, padx=5)
113.                Label(frame, pady=8, text="Tool Number").grid(row=11, column=0, stic
    ky=E, padx=5)
114.
115.                a_angle = Entry(frame, width=10, justify='center', textvariable=angl
    eaS)
116.                a_angle.grid(row=1, column=1, sticky=W, padx=5)
117.
118.                b_angle = Entry(frame, width=10, justify='center', textvariable=angl
    ebS)
119.                b_angle.grid(row=2, column=1, sticky=W, padx=5)
120.
121.                c_angle = Entry(frame, width=10, justify='center', textvariable=angl
    ecS)
122.                c_angle.grid(row=3, column=1, sticky=W, padx=5)
123.
124.                toolx = Entry(frame, width=10, justify='center', textvariable=tool_x
    )
125.                toolx.grid(row=5, column=1, sticky=W, padx=5)
126.
127.                tooly = Entry(frame, width=10, justify='center', textvariable=tool_y
    )
128.                tooly.grid(row=6, column=1, sticky=W, padx=5)
129.
130.                toolz = Entry(frame, width=10, justify='center', textvariable=tool_z
    )
131.                toolz.grid(row=7, column=1, sticky=W, padx=5)
132.
133.                robot_speed = Entry(frame, width=10, justify='center', textvariable=
    speedrobotS)
134.                robot_speed.grid(row=9, column=1, sticky=W, padx=5)
135.
136.                tool_name = Entry(frame, width=10, justify='center', textvariable=to
    olnameS)
137.                tool_name.grid(row=10, column=1, sticky=W, padx=5)
138.
139.                tool_number = Entry(frame, width=10, justify='center', textvariable=
    toolnumberS)
140.                tool_number.grid(row=11, column=1, sticky=W, padx=5)
141.
142.                photo = PhotoImage(file=self.current_path + "//Binaries//angles.png"
    )
143.                w2 = Label(frame, image=photo)
144.                w2.photo = photo
145.                w2.grid(row=0, column=3, rowspan=10, padx=5, pady=5)
146.
147.                Button(frame, width=10, padx=5, text="Save", command=save_configurat
    ion).grid(row=11, column=0, columnspan=4,
148.
              pady=5)
149.                root2.mainloop()
150.
151.            def disparity_map_menu(self):
152.                min_disparity_s = StringVar()
153.                min_disparity_s.set(np.load('./Binaries/min_disparity.npy'))
154.                num_disparities_s = StringVar()
155.                num_disparities_s.set(np.load('./Binaries/num_disparities.npy'))
156.                block_size_s = StringVar()
157.                block_size_s.set(np.load('./Binaries/block_size.npy'))
158.                disp12_s = StringVar()
159.                disp12_s.set(np.load('./Binaries/disp12.npy'))
160.                uniqueness_ratio_s = StringVar()
161.                uniqueness_ratio_s.set(np.load('./Binaries/uniqueness_ratio.npy'))
```

```
162.                window_s = StringVar()
163.                window_s.set(np.load('./Binaries/window_size.npy'))
164.                speckle_window_s = StringVar()
165.                speckle_window_s.set(np.load('./Binaries/speckle_window_size.npy'))

166.                speckle_range_s = StringVar()
167.                speckle_range_s.set(np.load('./Binaries/speckle_range.npy'))
168.                downsize_stereo_s = StringVar()
169.                downsize_stereo_s.set(np.load('./Binaries/downsize_stereo.npy'))
170.
171.                K = np.load('./Camera Parameters/K.npy')
172.                dist = np.load('./Camera Parameters/dist.npy')
173.                focal_length_mm = np.load('./Camera Parameters/FocalLength.npy')
174.
175.
176.                def generate_disparity():
177.                    np.save('./Binaries/min_disparity', min_disparity_s.get())
178.                    np.save('./Binaries/num_disparities', num_disparities_s.get())
179.                    np.save('./Binaries/block_size', block_size_s.get())
180.                    np.save('./Binaries/disp12', disp12_s.get())
181.                    np.save('./Binaries/uniqueness_ratio', uniqueness_ratio_s.get())

182.                    np.save('./Binaries/speckle_window_size', speckle_window_s.get()
    )
183.                    np.save('./Binaries/speckle_range', speckle_range_s.get())
184.                    np.save('./Binaries/downsize_stereo', downsize_stereo_s.get())
185.                    np.save('./Binaries/window_size', window_s.get())
186.                    disparity_button.set("Working...")
187.                    root3.update_idletasks()
188.                    mod_disparity_map.disparity_map(K, dist, dir_img_path_1.get(), d
    ir_img_path_2.get(),
189.                                                    int(downsize_stereo_s.get()), in
    t(window_s.get()),
190.                                                    int(min_disparity_s.get()), int(
    num_disparities_s.get()),
191.                                                    int(block_size_s.get()), int(uni
    queness_ratio_s.get()),
192.                                                    int(speckle_window_s.get()), int
    (speckle_range_s.get()),
193.                                                    int(disp12_s.get()))
194.                    disparity_button.set("Generate Disparity Map")
195.
196.
197.                def generate_stereo():
198.                    np.save('./Binaries/min_disparity', min_disparity_s.get())
199.                    np.save('./Binaries/num_disparities', num_disparities_s.get())
200.                    np.save('./Binaries/block_size', block_size_s.get())
201.                    np.save('./Binaries/disp12', disp12_s.get())
202.                    np.save('./Binaries/uniqueness_ratio', uniqueness_ratio_s.get())

203.                    np.save('./Binaries/speckle_window_size', speckle_window_s.get()
    )
204.                    np.save('./Binaries/speckle_range', speckle_range_s.get())
205.                    np.save('./Binaries/downsize_stereo', downsize_stereo_s.get())
206.                    np.save('./Binaries/window_size', window_s.get())
207.                    d_reconstruction.set("Working...")
208.                    root3.update_idletasks()
209.                    mod_disparity_map.stereo_map(K, dist, dir_img_path_1.get(), dir_
    img_path_2.get(),
210.                                                    int(downsize_stereo_s.get()), in
    t(window_s.get()),
211.                                                    int(min_disparity_s.get()), int(
    num_disparities_s.get()),
212.                                                    int(block_size_s.get()), int(uni
    queness_ratio_s.get()),
```

```
213.                                                    int(speckle_window_s.get()), int
    (speckle_range_s.get()), int(disp12_s.get())
214.                                                    , float(focal_length_mm))
215.                     d_reconstruction.set("Generate Disparity Map")
216.                     import webbrowser, os
217.                     path = ".//"
218.                     webbrowser.open(os.path.realpath(path))
219.
220.                 def select_picture_one():
221.                     path = askopenfilename(filetype=[("Jpg file", "*.jpg"), ("Png fi
    le", "*.png")])
222.                     dir_img_path_1.set(path)
223.
224.                 def select_picture_two():
225.                     path = askopenfilename(filetype=[("Jpg file", "*.jpg"), ("Png fi
    le", "*.png")])
226.                     dir_img_path_2.set(path)
227.
228.
229.                 root3 = Toplevel()
230.                 root3.title('Disparity Map Menu')
231.                 root3.resizable(0, 0)
232.                 frame = Frame(root3)
233.                 frame.grid()
234.
235.                 Label(frame, pady=8, text="Select Pictures").grid(row=0, column=1, c
    olumnspan=2)
236.
237.                 Label(frame, pady=8, text="Picture 1 (Left)").grid(row=1, column=0,
    sticky=E)
238.                 dir_img_path_1 = StringVar()
239.                 e = Entry(frame, width=50, textvariable=dir_img_path_1)
240.                 e.config(state='readonly')
241.                 e.grid(row=1, column=1, columnspan=5, padx=5)
242.                 Button(frame, width=10, padx=5, text="Search", command=select_pictur
    e_one).grid(row=1, column=6, columnspan=1,
243.
                padx=5)
244.
245.                 Label(frame, pady=8, text="Picture 2 (Right)").grid(row=2, column=0,
     sticky=E, padx=5)
246.                 dir_img_path_2 = StringVar()
247.                 e = Entry(frame, width=50, textvariable=dir_img_path_2)
248.                 e.config(state='readonly')
249.                 e.grid(row=2, column=1, columnspan=5, padx=5)
250.                 Button(frame, width=10, padx=5, text="Search", command=select_pictur
    e_two).grid(row=2, column=6, columnspan=1)
251.
252.                 Label(frame, pady=8, text="Disparity Map").grid(row=3, column=1, col
    umnspan=2)
253.
254.                 Label(frame, pady=8, text="Window Size").grid(row=4, column=1, stick
    y=E)
255.                 win_size = Entry(frame, width=20, justify='center', textvariable=win
    dow_s)
256.                 win_size.grid(row=4, column=2, sticky=W, padx=5)
257.
258.                 Label(frame, pady=8, text="Min disparity").grid(row=5, column=1, sti
    cky=E)
259.                 min_disparity = Entry(frame, width=20, justify='center', textvariabl
    e=min_disparity_s)
260.                 min_disparity.grid(row=5, column=2, sticky=W, padx=5)
261.
262.                 Label(frame, pady=8, text="Num disparities (Divisible by 16)").grid(
    row=6, column=1, sticky=E)
```

```
263.                num_disparities = Entry(frame, width=20, justify='center', textvaria
    ble=num_disparities_s)
264.                num_disparities.grid(row=6, column=2, sticky=W, padx=5)
265.
266.                Label(frame, pady=8, text="Block Size").grid(row=7, column=1, sticky
    =E)
267.                block_size = Entry(frame, width=20, justify='center', textvariable=b
    lock_size_s)
268.                block_size.grid(row=7, column=2, sticky=W, padx=5)
269.
270.                Label(frame, pady=8, text="Disp12 Max Diff").grid(row=8, column=1, s
    ticky=E)
271.                disp12_max = Entry(frame, width=20, justify='center', textvariable=d
    isp12_s)
272.                disp12_max.grid(row=8, column=2, sticky=W, padx=5)
273.
274.                Label(frame, pady=8, text="Uniqueness Ratio").grid(row=9, column=1,
    sticky=E)
275.                uniqueness_ratio = Entry(frame, width=20, justify='center', textvari
    able=uniqueness_ratio_s)
276.                uniqueness_ratio.grid(row=9, column=2, sticky=W, padx=5)
277.
278.                Label(frame, pady=8, text="Speckle Window Size").grid(row=10, column
    =1, sticky=E)
279.                speckle_window = Entry(frame, width=20, justify='center', textvariab
    le=speckle_window_s)
280.                speckle_window.grid(row=10, column=2, sticky=W, padx=5)
281.
282.                Label(frame, pady=8, text="Speckle Range").grid(row=11, column=1, st
    icky=E)
283.                speckle_range = Entry(frame, width=20, justify='center', textvariabl
    e=speckle_range_s)
284.                speckle_range.grid(row=11, column=2, sticky=W, padx=5)
285.
286.                disparity_button = StringVar()
287.                disparity_button.set("Generate Disparity Map")
288.                Button(frame, width=30, padx=5, text="Generate Disparity Map", comma
    nd=generate_disparity,
289.                       textvariable = disparity_button).grid(row=12, column=1, colum
    nspan=4)
290.
291.                Label(frame, pady=8, text="3D Reconstruction").grid(row=13, column=1
    , columnspan=2)
292.                Label(frame, pady=8, text="Downsize Stereo").grid(row=14, column=1,
    sticky=E)
293.                downsize_stereo = Entry(frame, width=20, justify='center', textvaria
    ble=downsize_stereo_s)
294.                downsize_stereo.grid(row=14, column=2, sticky=W, padx=5)
295.
296.                d_reconstruction = StringVar()
297.                d_reconstruction.set("Generate 3D Reconstruction")
298.                Button(frame, width=30, padx=5, text="Generate 3D Reconstruction", c
    ommand=generate_stereo,
299.                       textvariable=d_reconstruction).grid(row=15, column=1, columns
    pan=4, pady=10)
300.                root3.mainloop()
301.
302.            def kuka_generation_menu(self):
303.                self.choose_one = False
304.                self.one_width = 0
305.                self.one_height = 0
306.                self.choose_two = False
307.                self.two_width = 0
308.                self.two_height = 0
309.                self.counter_one = 1
310.                self.counter_two = 1
```

```
311.
312.                 self.pixels_one = []
313.                 self.pixels_two = []
314.                 self.trayectory = IntVar()
315.                 self.trayectory.set(1)
316.                 self.x_camera = DoubleVar()
317.                 self.y_camera = DoubleVar()
318.                 self.z_camera = DoubleVar()
319.                 self.distance_pic = DoubleVar()
320.
321.                 def next_action():
322.                     def generate_kuka_code():
323.                         focal_length_y = np.load('./Camera Parameters/K.npy')[0][0]
324.                         focal_length_z = np.load('./Camera Parameters/K.npy')[1][1]
325.                         width = self.one_width + self.two_width
326.                         width = width / 2
327.                         height = self.one_height + self.two_height
328.                         height = height / 2
329.                         tool_x = float(np.load('./Binaries/toolx.npy'))
330.                         tool_y = float(np.load('./Binaries/tooly.npy'))
331.                         tool_z = float(np.load('./Binaries/toolz.npy'))
332.                         coordenates = mod_generate_kukacode.generate_coordinates(sel
    f.x_camera.get(), self.y_camera.get(),
333.                                                                                  sel
    f.z_camera.get(), focal_length_y,
334.                                                                                  foc
    al_length_z
335.                                                                                  , w
    idth, height, self.distance_pic.get(),
336.                                                                                  sel
    f.pixels_one, self.pixels_two, tool_x,
337.                                                                                  too
    l_y, tool_z)
338.
339.                         print(coordenates)
340.
341.                         toolnameS = str(np.load('./Binaries/tool_name.npy'))
342.                         toolnumberS = int(np.load('./Binaries/tool_number.npy'))
343.                         speedrobotS = float(np.load('./Binaries/speed_robot.npy'))
344.                         angleaS = float(np.load('./Binaries/angle_a.npy'))
345.                         anglebS = float(np.load('./Binaries/angle_b.npy'))
346.                         anglecS = float(np.load('./Binaries/angle_c.npy'))
347.
348.                         filename = asksaveasfilename(initialdir="/", title="Select f
    ile")
349.                         name = os.path.basename(filename)
350.                         directory = os.path.dirname(filename)
351.                         mod_generate_kukacode.main(coordenates, [angleaS, anglebS, a
    nglecS], toolnumberS, toolnameS,
352.                                                                speedrobotS,
353.                                                                self.trayectory.get(), directory,
     name)
354.
355.                         root4.destroy()
356.                         root5.destroy()
357.
358.                     if self.choose_one and self.choose_two:
359.                         if len(self.pixels_one) == len(self.pixels_two) and len(self
    .pixels_one) > 0:
360.                             def on_closing_root5():
361.                                 root4.deiconify()
362.                                 root5.destroy()
363.
364.                             root5 = Toplevel()
```

```
365.                            root5.title('KRL Menu')
366.                            root5.protocol("WM_DELETE_WINDOW", on_closing_root5)
367.                            root5.resizable(0, 0)
368.                            frame2 = Frame(root5)
369.                            frame2.grid()
370.
371.                            Label(frame2, pady=8, text="Coordinates Camera from Cent
    er of Robot").grid(row=0, column=0, columnspan=2, padx=5)
372.                            Label(frame2, pady=5, text="X [mm]").grid(row=1, column=
    0, sticky=E)
373.                            center_x = Entry(frame2, width=20, justify='center', tex
    tvariable=self.x_camera)
374.                            center_x.grid(row=1, column=1, sticky=W, padx=5)
375.
376.                            Label(frame2, pady=5, text="Y [mm]").grid(row=2, column=
    0, sticky=E)
377.                            center_y = Entry(frame2, width=20, justify='center', tex
    tvariable=self.y_camera)
378.                            center_y.grid(row=2, column=1, sticky=W, padx=5)
379.
380.                            Label(frame2, pady=5, text="Z [mm]").grid(row=3, column=
    0, sticky=E)
381.                            center_z = Entry(frame2, width=20, justify='center', tex
    tvariable=self.z_camera)
382.                            center_z.grid(row=3, column=1, sticky=W, padx=5)
383.
384.                            Label(frame2, pady=8, text="Separation between Pictures"
    ).grid(row=4, column=0, columnspan=2,
385.
          padx=5)
386.                            Label(frame2, pady=5, text="Distance [mm]").grid(row=5,
    column=0, sticky=E)
387.                            distance = Entry(frame2, width=20, justify='center', tex
    tvariable=self.distance_pic)
388.                            distance.grid(row=5, column=1, sticky=W, padx=5, pady=5)

389.
390.                            Label(frame2, pady=8, text="Trajectory Generation").grid
    (row=6, column=0, columnspan=2, padx=5)
391.
392.                            R1 = Radiobutton(frame2, text="Point-to-
    Point Trajectory", variable=self.trayectory, value=1)
393.                            R1.grid(row=7, column=1, padx=5, pady=5)
394.
395.                            photo = PhotoImage(file=self.current_path + "//Binaries/
    /punto a punto.png")
396.                            w = Label(frame2, width=100, height=60, image=photo)
397.                            w.photo = photo
398.                            w.grid(row=7, column=0, padx=10, pady=5)
399.
400.                            R2 = Radiobutton(frame2, text="Polynomial Via Point Traj
    ectory", variable=self.trayectory, value=2)
401.                            R2.grid(row=8, column=1, padx=5, pady=5)
402.
403.                            photo = PhotoImage(file=self.current_path + "//Binaries/
    /contorno.png")
404.                            w = Label(frame2, width=96, height=60, image=photo)
405.                            w.photo = photo
406.                            w.grid(row=8, column=0, padx=10, pady=5)
407.
408.                            Button(frame2, width=30, padx=5, text="Generate", comman
    d=generate_kuka_code).grid(row=9, column=0,
409.
                               columnspan=2,
410.
                               pady=10)
```

```
411.                             root4.withdraw()
412.                             root5.mainloop()
413.                         elif len(self.pixels_one) == 0:
414.                             print("pixels are cero select at least one")
415.                         else:
416.                             print("are not equal")
417.                     else:
418.                         print("an image was not selected")
419.
420.                 def select_image_1():
421.                     path = askopenfilename(filetype=[("Jpg file", "*.jpg"), ("Png fi
    le", "*.png")])
422.                     pil_image = Image.open(path)
423.                     self.one_width = pil_image.width
424.                     self.one_height = pil_image.height
425.                     self.choose_one = True
426.                     imageresized = pil_image.resize((924, 520), Image.ANTIALIAS)
427.                     file_out = 'Temp//temp1.png'
428.                     imageresized.save(file_out)
429.                     photo = PhotoImage(file=file_out)
430.                     w1.config(image=photo)
431.                     w1.photo_ref = photo
432.                     self.pixels_one.clear()
433.                     self.counter_one = 1
434.                     T1.configure(state='normal')
435.                     T1.delete('1.0', END)
436.                     T1.configure(state='disabled')
437.                     frame.update_idletasks()
438.
439.                 def select_image_2():
440.                     path = askopenfilename(filetype=[("Jpg file", "*.jpg"), ("Png fi
    le", "*.png")])
441.                     pil_image = Image.open(path)
442.                     self.two_width = pil_image.width
443.                     self.two_height = pil_image.height
444.                     self.choose_two = True
445.                     imageresized = pil_image.resize((924, 520), Image.ANTIALIAS)
446.                     file_out = 'Temp//temp2.png'
447.                     imageresized.save(file_out)
448.                     photo = PhotoImage(file=file_out)
449.                     w2.config(image=photo)
450.                     w2.photo_ref = photo
451.                     self.pixels_two.clear()
452.                     self.counter_two = 1
453.                     T2.configure(state='normal')
454.                     T2.delete('1.0', END)
455.                     T2.configure(state='disabled')
456.                     frame.update_idletasks()
457.
458.                 def callback1(event):
459.                     if self.choose_one:
460.                         frame.focus_set()
461.                         myx = self.one_width * event.x / 924
462.                         myy = self.one_height * event.y / 520
463.                         T1.configure(state='normal')
464.                         T1.insert('end', 'Point '+str(self.counter_one)+" at: "+str(
    myx)+" "+str(myy)+"\n")
465.                         T1.configure(state='disabled')
466.                         self.counter_one = self.counter_one + 1
467.                         self.pixels_one.append([myx, myy])
468.
469.                 def callback2(event):
470.                     if self.choose_two:
471.                         frame.focus_set()
472.                         myx = self.two_width * event.x / 924
473.                         myy = self.two_height * event.y / 520
```

```
474.                         T2.configure(state='normal')
475.                         T2.insert('end', 'Point ' + str(self.counter_two) + " at: "
    + str(myx) + " " + str(myy) + "\n")
476.                         T2.configure(state='disabled')
477.                         self.counter_two = self.counter_two + 1
478.                         self.pixels_two.append([myx, myy])
479.
480.                 root4 = Toplevel()
481.                 root4.title('KRL Menu')
482.                 root4.resizable(0, 0)
483.                 frame = Frame(root4)
484.                 frame.grid()
485.
486.                 Label(frame, pady=5, text="Select pixels to find coordinates", font=
    (None, 15)).grid(row=0, column=0,
487.
                      columnspan=2)
488.                 Button(frame, width=10, padx=5, text="Select Picture 1", command=sel
    ect_image_1).grid(row=1, column=0,
489.
                       padx=5, pady=5)
490.                 Button(frame, width=10, padx=5, text="Select Picture 2", command=sel
    ect_image_2).grid(row=1, column=1,
491.
                       padx=5, pady=5)
492.                 if getattr(sys, 'frozen', False):
493.                     self.current_path = sys._MEIPASS
494.                 else:
495.                     self.current_path = os.path.dirname(__file__)
496.
497.                 photo1 = PhotoImage(file=self.current_path + "//Binaries//background
    .png")
498.                 w1 = Label(frame, image=photo1)
499.                 w1.bind("<Button-1>", callback1)
500.                 w1.photo = photo1
501.                 w1.grid(row=2, column=0, rowspan=3, padx=5, pady=5)
502.
503.                 photo2 = PhotoImage(file=self.current_path + "//Binaries//background
    .png")
504.                 w2 = Label(frame, image=photo2)
505.                 w2.bind("<Button-1>", callback2)
506.                 w2.photo = photo2
507.                 w2.grid(row=2, column=1, rowspan=3, padx=5, pady=5)
508.
509.                 T1 = ScrolledText(frame, height=5, width=60)
510.                 T1.config(state='disabled')
511.                 T1.grid(row=5, column=0, rowspan=5, padx=5, pady=5)
512.
513.                 T2 = ScrolledText(frame, height=5, width=60)
514.                 T2.config(state='disabled')
515.                 T2.grid(row=5, column=1, rowspan=5, padx=5, pady=5)
516.
517.                 Button(frame, width=10, padx=5, text="Next", command=next_action).gr
    id(row=10, column=0, padx=5, pady=5,
518.
       columnspan=2)
519.                 root4.mainloop()
520.
521.             def calibrate_menu(self):
522.                 chess_size_width = StringVar()
523.                 chess_size_width.set(np.load('./Binaries/chess_size_width.npy'))
524.                 chess_size_height = StringVar()
525.                 chess_size_height.set(np.load('./Binaries/chess_size_height.npy'))
526.                 max_iterations = StringVar()
527.                 max_iterations.set(np.load('./Binaries/max_iterations.npy'))
528.                 criteria_epsilon = StringVar()
```

```
529.                criteria_epsilon.set(np.load('./Binaries/criteria.npy'))
530.
531.                def calibrate_camera():
532.                    try:
533.                        self.chess_size[0] = int(chess_size_width.get())
534.                    except:
535.                        self.chess_size[0] = 0
536.                    try:
537.                        self.chess_size[1] = int(chess_size_height.get())
538.                    except:
539.                        self.chess_size[1] = 0
540.                    try:
541.                        self.criteria_max_count = int(max_iterations.get())
542.                    except:
543.                        self.criteria_max_count = 0
544.                    try:
545.                        self.criteria_epsilon = float(criteria_epsilon.get())
546.                    except:
547.                        self.criteria_epsilon = 0
548.
549.                    if int(self.chess_size[0]) <= 0 or int(self.chess_size[1]) <= 0:

550.                        messagebox.showerror("Error", "Unvalid values for chess size
    ")
551.                    elif self.criteria_max_count <= 0:
552.                        messagebox.showerror("Error", "Unvalid value for max number
    of iterations")
553.                    elif self.criteria_epsilon <= 0:
554.                        messagebox.showerror("Error", "Unvalid value for error in it
    eration")
555.                    elif self.path_images == '':
556.                        messagebox.showerror("Error", "Unvalid directory with calibr
    ation images")
557.                    else:
558.                        np.save('./Binaries/chess_size_width', chess_size_width.get(
    ))
559.                        np.save('./Binaries/chess_size_height', chess_size_height.ge
    t())
560.                        np.save('./Binaries/max_iterations', max_iterations.get())
561.                        np.save('./Binaries/criteria', criteria_epsilon.get())
562.                        btn_text.set("Working...")
563.                        root2.update_idletasks()
564.                        error = mod_camera_calibration.calibratecamera(self.chess_si
    ze[0], self.chess_size[1], self.path_images,
565.                                                                       self.criteria
    _max_count, self.criteria_epsilon)
566.                        messagebox.showinfo("Calibration done", "Error: " + str(erro
    r) + "%")
567.                        root2.destroy()
568.
569.                def open_folder_images():
570.                    pathString = askdirectory()
571.                    if pathString != "":
572.                        lista_imagenes = fnmatch.filter(os.listdir(pathString), '*.j
    pg')
573.                        if len(lista_imagenes) != 0:
574.                            dir_img_path.set(pathString)
575.                            self.path_images = pathString
576.                        else:
577.                            messagebox.showerror("Error", "Folder has no images")
578.                            dir_img_path.set('')
579.                            self.path_images = ''
580.
581.                root2 = Toplevel()
582.                root2.title('Camera Calibration Menu')
583.                root2.resizable(0, 0)
```

```
584.                frame = Frame(root2)
585.                frame.grid()
586.
587.                Label(frame, pady=8, text="Chess Size Width").grid(row=0, column=0,
    sticky=E, padx=5)
588.                Label(frame, pady=8, text="Chess Size Height").grid(row=1, column=0,
     sticky=E, padx=5)
589.                Label(frame, pady=8, text="Criteria Max Count Iterations ").grid(row
    =2, column=0, sticky=E, padx=5)
590.                Label(frame, pady=8, text="Criteria Epsilon ").grid(row=3, column=0,
     sticky=E, padx=5)
591.                Label(frame, pady=8, text="Path to Images ").grid(row=4, column=0, s
    ticky=E, padx=5)
592.
593.                chess_size_1 = Entry(frame, width=10, justify='center', textvariable
    =chess_size_width)
594.                chess_size_1.grid(row=0, column=1, sticky=W, padx=5)
595.
596.                chess_size_2 = Entry(frame, width=10, justify='center', textvariable
    =chess_size_height)
597.                chess_size_2.grid(row=1, column=1, sticky=W, padx=5)
598.
599.                photo = PhotoImage(file=self.current_path + "//Binaries//camera_cali
    bration.png")
600.                w2 = Label(frame, image=photo)
601.                w2.photo = photo
602.                w2.grid(row=0, column=3, rowspan=4, columnspan=4, padx=5, pady=5)
603.
604.                max_count = Entry(frame, justify='center', textvariable=max_iteratio
    ns)
605.                max_count.grid(row=2, column=1, sticky=W, columnspan=2, padx=5)
606.
607.                epsilon = Entry(frame, justify='center', textvariable=criteria_epsil
    on)
608.                epsilon.grid(row=3, column=1, sticky=W, columnspan=2, padx=5)
609.
610.                dir_img_path = StringVar()
611.
612.                e = Entry(frame, width=50, textvariable=dir_img_path)
613.                e.config(state='readonly')
614.                e.grid(row=4, column=1, columnspan=5, padx=5)
615.
616.                Button(frame, width=10, padx=5, text="Search", command=open_folder_i
    mages).grid(row=4, column=6, padx=10)
617.                btn_text = StringVar()
618.                btn_text.set("Calibrate")
619.                Button(frame, width=10, padx=5, text="Calibrate", command=calibrate_
    camera, textvariable=btn_text) \
620.                    .grid(row=5, column=1, columnspan=2, pady=10)
621.
622.                root2.mainloop()
623.
624.
625.        root = Tk()
626.        root.title('Kuka Controller Prototype')
627.        root.resizable(0, 0)
628.        app = App(root)
629.        root.mainloop()
```

Archivo mod_generate_kukacode.py

```
def main(points, angles, tool, tool_name, speed, type_trayectory, path, name):
    create_files(path, name)
    if type_trayectory == 1:
        fdat.write("&ACCESS RV\n"
                   "&REL 1\n"
                   "&PARAM EDITMASK = *\n"
                   "&PARAM TEMPLATE = C:\\KRC\\Roboter\\Template\\vorgabe\n"
                   "DEFDAT  " + timestr + "\n"
                                          ";FOLD EXTERNAL DECLARATIONS;%{PE}%MKUKATPBASIS,%CEXT,%VCOMMON,%P\n"
                                          ";FOLD BASISTECH EXT;%{PE}%MKUKATPBASIS,%CEXT,%VEXT,%P\n"
                                          "EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )\n"
                                          "DECL INT SUCCESS\n"
                                          ";ENDFOLD (BASISTECH EXT)\n"
                                          ";FOLD USER EXT;%{E}%MKUKATPUSER,%CEXT,%VEXT,%P\n"
                                          ";Make your modifications here\n\n"
                                          ";ENDFOLD (USER EXT)\n"
                                          ";ENDFOLD (EXTERNAL DECLARATIONS)\n"

                                          "DECL BASIS_SUGG_T LAST_BASIS={POINT1[] \"P" + str(len(points)) +
                   "                      \",POINT2[] \"P" + str(len(points)) +
                   "                      \",CP_PARAMS[] \"CPDAT" + str(len(points)) + "                  "

    "\",PTP_PARAMS[] \"PDAT0                  \",CONT[] \"
\",CP_VEL[] \"1.3       "

    "              \",PTP_VEL[] \"100                    \",SYNC_PARAMS[] \"SYNCDAT                \","

    "SPL_NAME[] \"S0                     \",A_PARAMS[] \"ADAT0                   \"} \n\n"
                   )

        fsrc.write("&ACCESS RV\n"
                   "&REL 1\n"
                   "&PARAM EDITMASK = *\n"
                   "&PARAM TEMPLATE = C:\\KRC\\Roboter\\Template\\vorgabe\n"
                   "DEF " + timestr + "()\n"
                                      ";FOLD INI;%{PE}\n"
                                      "  ;FOLD BASISTECH INI\n"
                                      "    GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )\n"
                                      "    INTERRUPT ON 3 \n"
                                      "    BAS (#INITMOV,0 )\n"
                                      "  ;ENDFOLD (BASISTECH INI)\n"
                                      "  ;FOLD USER INI\n"
                                      "    ;Make your modifications here\n"
                                      "\n"
                                      "  ;ENDFOLD (USER INI)\n"
                                      ";ENDFOLD (INI)\n"
                                      "\n")

        for i in range(len(points)):
            if i == 0:
                fdat.write("DECL E6POS XP" + str(i + 1) +
                           "={"
                           "X " + str(points[i][0]) +
```

```
                                ","
                                "Y " + str(points[i][1]) +
                                ","
                                "Z " + str(points[i][2]) +
                                ","
                                "A " + str(angles[0]) +
                                ","
                                "B " + str(angles[1]) +
                                ","
                                "C " + str(angles[2]) +
                                ",S 2,T 10,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6 0.0}\n"

                                "DECL FDAT FP" + str(i + 1) +
                                "={"
                                "TOOL_NO " + str(tool) +
                                ","
                                "BASE_NO 0,IPO_FRAME #BASE,POINT2[] \" \",TQ_STATE FALSE }\n"
                                "DECL PDAT PPDAT" + str(i + 1) +
                                "={VEL 100.000,ACC 100.000,APO_DIST 100.000,APO_MODE #CDIS,GEAR_JERK 50.0000,EXAX_IGN 0}\n")

                    fsrc.write(";FOLD PTP P" + str(i + 1) +
                                " Vel=100 % PDAT" + str(i + 1) +
                                " Tool[" + str(tool) + "]:" + tool_name + " Base[0]"
                                                                          ";%{PE}%R 8.3.44,%MKUKATPBASIS,%CMOVE,%VPTP,"
                                                                          "%P 1:PTP, 2:P1, 3:, 5:100, 7:PDAT" + str(i + 1) +
                                "\n$BWDSTART=FALSE"
                                "\nPDAT_ACT=PPDAT" + str(i + 1) +
                                "\nFDAT_ACT=FP" + str(i + 1) +
                                "\nBAS(#PTP_PARAMS,100)"
                                "\nPTP XP" + str(i + 1) +
                                "\n;ENDFOLD\n")

                else:
                    fdat.write("DECL E6POS XP" + str(i + 1) +
                                "={"
                                "X " + str(points[i][0]) +
                                ","
                                "Y " + str(points[i][1]) +
                                ","
                                "Z " + str(points[i][2]) +
                                ","
                                "A " + str(angles[0]) +
                                ","
                                "B " + str(angles[1]) +
                                ","
                                "C " + str(angles[2]) +
                                ",S 2,T 10,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6 0.0}\n"

                                "DECL FDAT FP" + str(i + 1) +
                                "={"
                                       "TOOL_NO " + str(tool) +
                                       ","
                                       "BASE_NO 0,IPO_FRAME #BASE,POINT2[] \" \",TQ_STATE FALSE}\n"
                                       "DECL LDAT LCPDAT" + str(i + 1) +
                                       "={VEL 2.00000,ACC 100.000,APO_DIST 100.000,APO_FAC 50.0000,AXIS_VEL 100.000,"
                                       "AXIS_ACC 100.000,ORI_TYP #VAR,CIRC_TYP #BASE,JERK_FAC 50.0000,GEAR_JERK 50.0000,EXAX_IGN 0}"
                                       "\n")
```

```
                    fsrc.write(";FOLD LIN P" + str(i + 1) +
                               " Vel=" + str(speed) + " m/s CPDAT" + str(i + 1) +
                               " Tool[" + str(tool) + "]:" + tool_name + " Base[0];%{PE}%R 8.3.44,"
                                                                                  "%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, "
                                                                                  "2:"
                                                                                  "P" + str(i + 1) +
                               ", 3:, "
                               "5:1.3, 7:CPDAT" + str(i + 1) +
                               "\n$BWDSTART=FALSE"
                               "\nLDAT_ACT=LCPDAT" + str(i + 1) +
                               "\nFDAT_ACT=FP" + str(i + 1) +
                               "\nBAS(#CP_PARAMS,1.3)" +
                               "\nLIN XP" + str(i + 1) +
                               "\n;ENDFOLD\n")
                if (i + 1) != len(points):
                    fdat.write("\n")

            fsrc.write("END")
            fdat.write("ENDDAT")

        elif type_trayectory == 2:
            fdat.write("&ACCESS RV\n"
                       "&REL 1\n"
                       "&PARAM EDITMASK = *\n"
                       "&PARAM TEMPLATE = C:\\KRC\\Roboter\\Template\\vorgabe\n"

                       "DEFDAT  " + timestr + "\n"
                                              ";FOLD EXTERNAL DECLARATIONS;%{PE}%MKUKATPBASIS,%CEXT,%VCOMMON,%P\n"
                                              ";FOLD BASISTECH EXT;%{PE}%MKUKATPBASIS,%CEXT,%VEXT,%P\n"
                                              "EXT  BAS (BAS_COMMAND  :IN,REAL :IN )\n"
                                              "DECL INT SUCCESS\n"
                                              ";ENDFOLD (BASISTECH EXT)\n"
                                              ";FOLD USER EXT;%{E}%MKUKATPUSER,%CEXT,%VEXT,%P\n"
                                              ";Make your modifications here\n\n"
                                              ";ENDFOLD (USER EXT)\n"
                                              ";ENDFOLD (EXTERNAL DECLARATIONS)\n"

                                              "DECL BASIS_SUGG_T LAST_BASIS={POINT1[] \"P" + str(len(points)) +
                       "                      \",POINT2[] \"P" + str(len(points)) +
                       "                      \",CP_PARAMS[] \"CPDAT" + str(len(points)) + "                  "

            "\",PTP_PARAMS[] \"PDAT0                  \",CONT[] \"
        \",CP_VEL[] \"1.3       "

            "              \",PTP_VEL[] \"100                    \",SYNC_PARAMS[] \"SYNCDAT                \","

            "SPL_NAME[] \"S0                      \",A_PARAMS[] \"ADAT0
        \"} \n\n"
                       )

            fsrc.write("&ACCESS RV\n"
                       "&REL 1\n"
```

```
                            "&PARAM EDITMASK = *\n"
                            "&PARAM TEMPLATE = C:\\KRC\\Roboter\\Template\\vorgabe\n"

                            "DEF " + timestr + "()\n"
                                                ";FOLD INI;%{PE}\n"
                                                "  ;FOLD BASISTECH INI\n"
                                                "    GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )\n"
                                                "    INTERRUPT ON 3 \n"
                                                "    BAS (#INITMOV,0 )\n"
                                                "  ;ENDFOLD (BASISTECH INI)\n"
                                                "  ;FOLD USER INI\n"
                                                "    ;Make your modifications here\n"

                                                "\n"
                                                "  ;ENDFOLD (USER INI)\n"
                                                ";ENDFOLD (INI)\n"
                                                "\n")

                if len(points) > 2:
                    for i in range(len(points)):
                        if i == 0:
                            fdat.write("DECL E6POS XP" + str(i + 1) +
                                       "={"
                                       "X " + str(points[i][0]) +
                                       ","
                                       "Y " + str(points[i][1]) +
                                       ","
                                       "Z " + str(points[i][2]) +
                                       ","
                                       "A " + str(angles[0]) +
                                       ","
                                       "B " + str(angles[1]) +
                                       ","
                                       "C " + str(angles[2]) +
                                       ",S 2,T 10,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6 0.0}\n"

                                       "DECL FDAT FP" + str(i + 1) +
                                       "={"
                                       "TOOL_NO " + str(tool) +
                                       ","
                                       "BASE_NO 0,IPO_FRAME #BASE,POINT2[] \" \",TQ_STATE FALSE}\n"
                                       "DECL PDAT PPDAT" + str(i + 1) +
                                       "={VEL 100.000,ACC 100.000,APO_DIST 100.000,APO_MODE #CDIS,GEAR_JERK 50.0000,EXAX_IGN 0}\n")

                            fsrc.write(";FOLD PTP P" + str(i + 1) +
                                       " Vel=100 % PDAT" + str(i + 1) +
                                       " Tool[" + str(tool) + "]:" + tool_name + " Base[0]"
                                                                                 ";%{PE}%R 8.3.44,%MKUKATPBASIS,%CMOVE,%VPTP,"
                                                                                 "%P 1:PTP, 2:P1, 3:, 5:100, 7:PDAT" + str(
                                   i + 1) +
                                       "\n$BWDSTART=FALSE"
                                       "\nPDAT_ACT=PPDAT" + str(i + 1) +
                                       "\nFDAT_ACT=FP" + str(i + 1) +
                                       "\nBAS(#PTP_PARAMS,100)"
                                       "\nPTP XP" + str(i + 1) +
                                       "\n;ENDFOLD\n")

                        elif (i + 1) == len(points):
                            fsrc.write(";FOLD LIN P" + str(i + 1) +
```

```
                                        " Vel=" + str(speed) + " m/s CPDAT" + str(i + 1) +
                                        " Tool[" + str(tool) + "]:" + tool_name + " "
    "Base[0]"
    ";%{PE}%R 8.3.44,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, "
    "2:"
    "P" + str(i + 1) +
                                        ", 3:, "
                                        "5:1.3, 7:CPDAT" + str(i + 1) +
                                        "\n$BWDSTART=FALSE"
                                        "\nLDAT_ACT=LCPDAT" + str(i + 1) +
                                        "\nFDAT_ACT=FP" + str(i + 1) +
                                        "\nBAS(#CP_PARAMS,1.3)" +
                                        "\nLIN XP" + str(i + 1) +
                                        "\n;ENDFOLD\n")
                                fdat.write("DECL E6POS XP" + str(i + 1) +
                                        "={"
                                        "X " + str(points[i][0]) +
                                        ","
                                        "Y " + str(points[i][1]) +
                                        ","
                                        "Z " + str(points[i][2]) +
                                        ","
                                        "A " + str(angles[0]) +
                                        ","
                                        "B " + str(angles[1]) +
                                        ","
                                        "C " + str(angles[2]) +
                                        ",S 2,T 10,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6 0.0}\n"

                                        "DECL FDAT FP" + str(i + 1) +
                                        "={"
                                        "TOOL_NO " + str(tool) +
                                        ","
                                        "BASE_NO 0,IPO_FRAME #BASE,POINT2[] \" \",TQ_STATE FALSE}\n"
                                        "DECL LDAT LCPDAT" + str(i + 1) +
                                        "={VEL 2.00000,ACC 100.000,APO_DIST 100.000,APO_FAC 50.0000,AXIS_VEL 100.000,"
                                        "AXIS_ACC 100.000,ORI_TYP #VAR,CIRC_TYP #BASE,JERK_FAC 50.0000,GEAR_JERK 50.0000,EXAX_IGN 0}"
                                        "\n")
                        else:
                            fsrc.write(";FOLD LIN P" + str(i + 1) +
                                        " CONT Vel=" + str(speed) + " m/s CPDAT" + str(i + 1) +
                                        " Tool[" + str(tool) + "]:" + tool_name + " "
    "Base[0]"
    ";%{PE}%R 8.3.44,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, "
    "2:"
    "P" + str(i + 1) +
                                        ", 3:, "
                                        "5:1.3, 7:CPDAT" + str(i + 1) +
                                        "\n$BWDSTART=FALSE"
                                        "\nLDAT_ACT=LCPDAT" + str(i + 1) +
```

```
260.                                         "\nFDAT_ACT=FP" + str(i + 1) +
261.                                         "\nBAS(#CP_PARAMS,1.3)" +
262.                                         "\nLIN XP" + str(i + 1) + " C_DIS C_DIS"
                                                                                 
263.                                                                             "\n;ENDFOLD\n")
                                                                                 
264.                                 fdat.write("DECL E6POS XP" + str(i + 1) +
265.                                            "={"
266.                                            "X " + str(points[i][0]) +
267.                                            ","
268.                                            "Y " + str(points[i][1]) +
269.                                            ","
270.                                            "Z " + str(points[i][2]) +
271.                                            ","
272.                                            "A " + str(angles[0]) +
273.                                            ","
274.                                            "B " + str(angles[1]) +
275.                                            ","
276.                                            "C " + str(angles[2]) +
277.                                            ",S 2,T 10,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0
    ,E6 0.0}\n"
278.                                            
279.                                            "DECL FDAT FP" + str(i + 1) +
280.                                            "={"
281.                                            "TOOL_NO " + str(tool) +
282.                                            ","
283.                                            "BASE_NO 0,IPO_FRAME #BASE,POINT2[] \" \",TQ_
    STATE FALSE}\n"
284.                                            "DECL LDAT LCPDAT" + str(i + 1) +
285.                                            "={VEL 2.00000,ACC 100.000,APO_DIST 100.000,A
    PO_FAC 50.0000,AXIS_VEL 100.000,"
286.                                            "AXIS_ACC 100.000,ORI_TYP #VAR,CIRC_TYP #BASE
    ,JERK_FAC 50.0000,GEAR_JERK 50.0000,EXAX_IGN 0}"
287.                                            "\n")
288.                                            
289.                                     if (i + 1) != len(points):
290.                                         fdat.write("\n")
291.                                         
292.                                 fsrc.write("END")
293.                                 fdat.write("ENDDAT")
294.                             else:
295.                                 for i in range(len(points)):
296.                                     fdat.write("DECL E6POS XP" + str(i + 1) +
297.                                                "={"
298.                                                "X " + str(points[i][0]) +
299.                                                ","
300.                                                "Y " + str(points[i][1]) +
301.                                                ","
302.                                                "Z " + str(points[i][2]) +
303.                                                ","
304.                                                "A " + str(angles[0]) +
305.                                                ","
306.                                                "B " + str(angles[1]) +
307.                                                ","
308.                                                "C " + str(angles[2]) +
309.                                                ",S 2,T 10,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6
    0.0}\n"
310.                                                
311.                                                "DECL FDAT FP" + str(i + 1) +
312.                                                "={"
313.                                                "TOOL_NO " + str(tool) +
314.                                                ","
315.                                                "BASE_NO 0,IPO_FRAME #BASE,POINT2[] \" \",TQ_STAT
    E FALSE}\n"
316.                                                "DECL PDAT PPDAT" + str(i + 1) +
```

```
317.                                    "={VEL 100.000,ACC 100.000,APO_DIST 100.000,APO_M
    ODE #CDIS,GEAR_JERK 50.0000,EXAX_IGN 0}\n")
318.
319.                            fsrc.write(";FOLD PTP P" + str(i + 1) +
320.                                       " Vel=100 % PDAT" + str(i + 1) +
321.                                       " Tool[" + str(tool) + "]:" + tool_name + " Base[
    0]"
322.                                                                                    ";%{PE}
    %R 8.3.44,%MKUKATPBASIS,%CMOVE,%VPTP,"
323.                                                                                    "%P 1:P
    TP, 2:P1, 3:, 5:100, 7:PDAT" + str(
324.                                    i + 1) +
325.                                       "\n$BWDSTART=FALSE"
326.                                       "\nPDAT_ACT=PPDAT" + str(i + 1) +
327.                                       "\nFDAT_ACT=FP" + str(i + 1) +
328.                                       "\nBAS(#PTP_PARAMS,100)"
329.                                       "\nPTP XP" + str(i + 1) +
330.                                       "\n;ENDFOLD\n")
331.
332.                        fsrc.write("END")
333.                        fdat.write("ENDDAT")
334.
335.                fdat.close()
336.                fsrc.close()
337.
338.
339.            def generate_coordinates(x_camera, y_camera, z_camera, focal_length_y, focal
    _length_z, width_y, height_z,
340.                                     distance_pictures, pixels_1, pixels_2, x_tool, y_to
    ol, z_tool):
341.                coordenates = []
342.
343.                for i in range(len(pixels_1)):
344.                    y1 = pixels_1[i][0]
345.                    z1 = pixels_1[i][1]
346.                    y2 = pixels_2[i][0]
347.                    z2 = pixels_2[i][1]
348.
349.                    focal_length = (focal_length_y + focal_length_z) / 2
350.                    x_from_center = absolute_value(focal_length * distance_pictures / (y
    1 - y2))
351.                    delta_y_pix_1 = width_y / 2 - y1
352.                    delta_y_pix_2 = width_y / 2 - y2
353.                    delta_z_pix_1 = height_z / 2 - z1
354.                    delta_z_pix_2 = height_z / 2 - z2
355.                    delta_y_pix_mm = x_from_center / focal_length_y
356.                    delta_z_pix_mm = x_from_center / focal_length_z
357.                    y_from_center_1 = delta_y_pix_mm * delta_y_pix_1
358.                    y_from_center_2 = delta_y_pix_mm * delta_y_pix_2
359.                    z_from_center_1 = delta_z_pix_mm * delta_z_pix_1
360.                    z_from_center_2 = delta_z_pix_mm * delta_z_pix_2
361.                    y_from_robot = (y_from_center_1 + y_from_center_2) / 2 + y_camera +
    y_tool
362.                    z_from_robot = (z_from_center_1 + z_from_center_2) / 2 + z_camera +
    z_tool
363.                    x_from_robot = x_from_center + x_camera + x_tool
364.
365.                    coordenates.append([x_from_robot, y_from_robot, z_from_robot])
366.
367.                return coordenates
368.
369.
370.            def absolute_value(a):
371.                if a < 0:
372.                    a = -a
373.                return a
```

```
374.
375.
376.        def create_files(path, name):
377.            global fsrc, fdat, timestr
378.
379.            timestr = name
380.            fsrc = open(path + "\\" + timestr + ".src", "w+")
381.            fdat = open(path + "\\" + timestr + ".dat", "w+")
382.
383.
384.        if __name__ == '__main__':
385.            my_points = [[720.0, 450.0, 1300.0]]
386.            my_angles = ["75.0", "50.0", "100.0"]
387.            my_tool_number = "10"
388.            my_tool_name = "b1"
389.            my_speed = "1.0"
390.            generate_coordinates(324.36755, -
    794.04309, 1239.6069, 3583.76038, 3575.42648, 4230, 3128, 25.0,
391.                                 [[1344, 1810], [1344, 1810]], [[1287, 1822], [1287,
     1822]])
392.            main(my_points, my_angles, my_tool_number, my_tool_name, my_speed, 1, "D
    :\\", "hola")
```

Archivo mod_disparity_map.py

```
1.  import cv2
2.  import numpy as np
3.  from matplotlib import pyplot as plt
4.
5.
6.  def create_output(vertices, colors, filename):
7.      colors = colors.reshape(-1, 3)
8.      vertices = np.hstack([vertices.reshape(-1, 3), colors])
9.      ply_header = '''ply
10.         format ascii 1.0
11.         element vertex %(vert_num)d
12.         property float x
13.         property float y
14.         property float z
15.         property uchar red
16.         property uchar green
17.         property uchar blue
18.         end_header
19.         '''
20.     with open(filename, 'w') as f:
21.         f.write(ply_header % dict(vert_num=len(vertices)))
22.         np.savetxt(f, vertices, '%f %f %f %d %d %d')
23.
24.
25. def downsample_image(image, reduce_factor):
26.     for i in range(0, reduce_factor):
27.         if len(image.shape) > 2:
28.             row, col = image.shape[:2]
29.         else:
30.             row, col = image.shape
31.         image = cv2.pyrDown(image, dstsize=(col // 2, row // 2))
32.     return image
33.
34.
35. def disparity_map(K, dist, image_1, image_2, downsize, win_size, min_disp, num_disp
    , block_size, uniqueness_ratio,
36.                     speckle_window_size, speckle_range, disp12):
37.     # Load pictures
38.     img_1 = cv2.imread(image_1)
39.     img_2 = cv2.imread(image_2)
40.
41.     # Get height and width.
42.     h, w = img_2.shape[:2]
43.
44.     # Get optimal camera matrix for better undistortion
45.     new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K, dist, (w, h), 1, (w,
    h))
46.
47.     # Undistort images
48.     img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
49.     img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
50.
51.     # Downsample each image
52.     img_1_downsampled = downsample_image(img_1_undistorted, downsize)
53.     img_2_downsampled = downsample_image(img_2_undistorted, downsize)
54.
55.     # Create Block matching object.
56.     stereo = cv2.StereoSGBM_create(minDisparity=min_disp,
57.                                    numDisparities=num_disp,
58.                                    blockSize=block_size,
59.                                    uniquenessRatio=uniqueness_ratio,
60.                                    speckleWindowSize=speckle_window_size,
```

```
61.                                     speckleRange=speckle_range,
62.                                     disp12MaxDiff=disp12,
63.                                     P1=8 * 3 * win_size ** 2,
64.                                     P2=32 * 3 * win_size ** 2)
65.
66.     # Compute disparity map
67.     print("\nComputing the disparity  map...")
68.     disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
69.
70.     # Show disparity map before generating 3D cloud to verify that point cloud will
     be usable.
71.     plt.imshow(disparity_map, 'gray')
72.     plt.savefig('disparity.png')
73.     img = cv2.imread('disparity.png')
74.     cv2.namedWindow('Disparity Map', cv2.WINDOW_NORMAL)
75.     cv2.resizeWindow('Disparity Map', 1280, 720)
76.     cv2.imshow('Disparity Map', img)
77.
78.
79. def stereo_map(K, dist, image_1, image_2, downsize, win_size, min_disp, num_disp, b
    lock_size, uniqueness_ratio,
80.                     speckle_window_size, speckle_range, disp12, focal_length_mm):
81.
82.     print("\nGenerating the 3D map...")
83.     # Load pictures
84.     img_1 = cv2.imread(image_1)
85.     img_2 = cv2.imread(image_2)
86.
87.     # Get height and width.
88.     h, w = img_2.shape[:2]
89.
90.     # Get optimal camera matrix for better undistortion
91.     new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K, dist, (w, h), 1, (w,
    h))
92.
93.     # Undistort images
94.     img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
95.     img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
96.
97.     # Downsample each image
98.     img_1_downsampled = downsample_image(img_1_undistorted, downsize)
99.     img_2_downsampled = downsample_image(img_2_undistorted, downsize)
100.
101.          stereo = cv2.StereoSGBM_create(minDisparity=min_disp,
102.                                         numDisparities=num_disp,
103.                                         blockSize=block_size,
104.                                         uniquenessRatio=uniqueness_ratio,
105.                                         speckleWindowSize=speckle_window_size,
106.                                         speckleRange=speckle_range,
107.                                         disp12MaxDiff=disp12,
108.                                         P1=8 * 3 * win_size ** 2,
109.                                         P2=32 * 3 * win_size ** 2)
110.
111.          disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
112.
113.          # Get new downsampled width and height
114.          h, w = img_2_downsampled.shape[:2]
115.
116.
117.          # This transformation matrix is derived from Prof. Didier Stricker's pow
    er point presentation on computer vision.
118.          # Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-
    15/3DCV_lec01_camera.pdf
119.          Q2 = np.float32([[1, 0, 0, 0],
120.                           [0, -1, 0, 0],
121.                           [0, 0, focal_length_mm * 0.05, 0],
```

```
                              [0, 0, 0, 1]])

        # Reproject points into 3D
        points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
        # Get color points
        colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)

        # Get rid of points with value 0 (i.e no depth)
        mask_map = disparity_map > disparity_map.min()

        # Mask colors and points.
        output_points = points_3D[mask_map]
        output_colors = colors[mask_map]

        # Define name for output file
        output_file = 'Reconstruction.ply'

        # Generate point cloud
        print("\n Creating the output file... \n")
        create_output(output_points, output_colors, output_file)
```

Archivo mod_camera_calibration.py

```
1.  import cv2
2.  import numpy as np
3.  import glob
4.  from tqdm import tqdm
5.  import PIL.ExifTags
6.  import PIL.Image
7.
8.
9.  def calibratecamera(chess_size1, chess_size2, images_path, max_count_iteration, eps
    ilon):
10.     chessboard_size = (chess_size1, chess_size2)
11.
12.     obj_points = []  # 3D points in real world space
13.     img_points = []  # 3D points in image plane
14.
15.     objp = np.zeros((np.prod(chessboard_size), 3), dtype=np.float32)
16.     objp[:, :2] = np.mgrid[0:chessboard_size[0], 0:chessboard_size[1]].T.reshape(-
    1, 2)
17.
18.     calibration_paths = glob.glob(images_path + '/*.jpg')
19.
20.     for image_path in tqdm(calibration_paths):
21.         image = cv2.imread(image_path)
22.         gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
23.         print("Image loaded, Analizying...")
24.         ret, corners = cv2.findChessboardCorners(gray_image, chessboard_size, None)

25.         if ret:
26.             print("Chessboard detected!")
27.             criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, max_cou
    nt_iteration, epsilon)
28.             cv2.cornerSubPix(gray_image, corners, (17, 13), (-1, -1), criteria)
29.             obj_points.append(objp)
30.             img_points.append(corners)
31.         else:
32.             print("Chessboard not detected! " + image_path)
33.
34.     # Calibrate camera
35.     ret, K, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points, gray_i
    mage.shape[::-1], None, None)
36.
37.     # Save parameters into numpy file
38.     np.save("./Camera Parameters/ret", ret)
39.     np.save("./Camera Parameters/K", K)
40.     np.save("./Camera Parameters/dist", dist)
41.     np.save("./Camera Parameters/rvecs", rvecs)
42.     np.save("./Camera Parameters/tvecs", tvecs)
43.
44.     # Get exif data in order to get focal length.
45.     exif_img = PIL.Image.open(calibration_paths[0])
46.
47.     exif_data = {
48.         PIL.ExifTags.TAGS[k]: v
49.         for k, v in exif_img._getexif().items()
50.         if k in PIL.ExifTags.TAGS}
51.
52.     # Get focal length in tuple form
53.     focal_length_exif = exif_data['FocalLength']
54.
55.     # Get focal length in decimal form
56.     focal_length = focal_length_exif[0] / focal_length_exif[1]
57.
```

```
58.
59.
60.     # Save focal length
61.     np.save("./Camera Parameters/FocalLength", focal_length)
62.
63.     # Calculate projection error.
64.     mean_error = 0
65.     for i in range(len(obj_points)):
66.         img_points2, _ = cv2.projectPoints(obj_points[i], rvecs[i], tvecs[i], K, di
    st)
67.         error = cv2.norm(img_points[i], img_points2, cv2.NORM_L2) / len(img_points2
    )
68.         mean_error += error
69.
70.     total_error = mean_error / len(obj_points)
71.     print(total_error * 100)
72.     return total_error*100
73.
74.
75. if __name__ == '__main__':
76.     chess_size_width = 6
77.     chess_size_heigh = 8
78.     folder_path = 'D:/OneDrive - Universidad San Francisco de Quito/Proyectos Progr
    amacion/Python/Tesis/Camera Calibration - Test 2'
79.     calibratecamera(chess_size_heigh, chess_size_width, folder_path, 100, 0.0001)
```