Escuela Politécnica Superior de Linares

Grado en Ingeniería Mecánica

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DESARROLLO DE APLICACIONES ROBÓTICAS

Alumno: Jorge Carreras López

Tutor: Prof. D. Luis Felipe Sesé
Depto.: Ingeniería Mecánica y Minera

Junio, 2017

ÍNDICE

ÍNDICE DE FIGURAS

1 RESUMEN

Este trabajo consiste en la monitorización de parámetros cinemáticos y el desarrollo de aplicaciones industriales basados en el brazo robótico IRB120 de la marca ABB, el cual se encuentra en el laboratorio de mecánica.

Se realiza un estudio cinemático de cada eslabón del brazo robótico, ya que esos datos no son proporcionados por el fabricante se ha estimado oportuno obtener las medidas. Para ello se ha creado una interfaz gráfica, la cual muestra las gráficas de la velocidad angular, la velocidad y la aceleración de cada uno de los eslabones.

El estudio cinemático se realizará a través de unos sensores denominados acelerómetros, y la monitorización entre estos y Matlab se realizará a través de Arduino.

Por otro lado, se han investigado la utilización de las entradas y salidas digitales del equipo. Como una aplicación de estos conocimientos se ha diseñado una serie de programas de simulaciones de soldadura, estas permiten aprender la gestión de esas entradas y salidas digitales que nos ofrece el sistema de control incorporado al robot.

ABSTRACT

The purpose of this work is to monitor the kinematic parameters and the development of industrial applications based on the ABB's IRB120 robot arm located in the mechanical laboratory.

Therefore, since such data are not provided by the company, it has been considered appropriate to obtain measurements by performing a kinematic study of each robotic arm link. In addition, to carry these measurements out, we have designed a graphical interphase that shows the graphs of the angular velocity, velocity and acceleration of each of the links.

The kinematic study will be carried out through sensors called accelerometers, and the monitoring between these and Matlab will be done through Arduino. On the other hand, the use of the digital inputs and outputs of the equipment has been investigated.

To end, by applying this knowledge, a series of programs of welding simulations have been designed, which allow learning the management of those digital inputs and outputs that the control system in the robot offers us.

2 OBJETIVO Y MOTIVACIÓN

Se presenta el objetivo de realizar dicho trabajo, la motivación y los objetivos que se quieren cumplir.

2.1 Objeto

La titulación del Grado en Ingeniería Mecánica, al igual que el resto de titulaciones impartidas en la Escuela Politécnica Superior de Linares, debe concluir según la normativa del plan vigente con la elaboración y la defensa por parte del estudiante de un Trabajo Fin de Grado. Este trabajo debe realizarse en la fase final del plan de estudios y debe estar orientado a la evaluación de competencias asociadas al título.

2.2 Motivación

Programar en diferentes sectores será de vital importancia. Esto combinado con las facilidades que aportan los robots al mundo de la industria hacen de la realización de este proyecto, un paso a delante hacia el futuro de un ingeniero.

La realización de mi compañero Axel José Córdoba López de la puesta a punto del brazo robótico me hizo querer aventurarme más en este mundo de la robótica, permitiendo que el alumnado de la Escuela Politécnica Superior de Linares adquiera conocimientos del brazo robótico, conocer más a fondo todas las posibilidades que el mundo de la robótica puede aportar.

Además conocer y profundizar en el campo de la sensorización para la determinación de parámetros que son interesantes desde el punto de vista cinemático pero que los sistemas comerciales no muestran.

2.3 Objetivos

Por lo anteriormente comentado, se estima oportuno realizar este Trabajo Final de Grado en el que se pretende alcanzar los siguientes objetivos:

- Realizar la sensorización del brazo robótico que permita conocer los parámetros cinemáticos más importantes como son la velocidad y aceleración de cada uno de sus miembros.
- Para ello recurrimos a Arduino que recoge la señal de los diferentes sensores dispuesto.
- La información de los sensores será procesada y gestionada por una aplicación gráfica en lenguaje MATLAB.
- Profundizar en el control de señales digitales por parte del equipo robótico; como resultado se pretende realizar una serie de aplicaciones de carácter didáctico que simule acciones industriales.
- Añadir opciones y equipamientos a la célula robótica.

3 INTRODUCCIÓN

Los robots tal como se conciben hoy en día no necesariamente tratan de asemejarse al ser humano, sino de cubrir la necesidad de éste de la forma más óptima posible. La robótica industrial nace de la automatización, encaminada a la reducción de costes y a su vez realizar trabajos en lo que el individuo podría correr algún riesgo o no pudiera ejercerlas.

En este trabajo se tiene el concepto de robot como un manipulador industrial multifuncional y reprogramable, centrándose el autor en el aspecto de la parametrización cinemática y de programación. Este tipo de robot industrial es el más utilizado hasta la fecha, destinado a la fabricación flexible en líneas de producción. Para la realización de este trabajo se ha continuado con la labor realizada por el un anterior estudiante quien en un anterior trabajo fin de grado realizo un estudio cinemático en términos generales del robot y una serie de prácticas para la asignatura “Mecánica de Robots” optativa de cuarto curso de Ingeniería Mecánica, realizando una puesta en marcha del brazo robótico.

Este estudio pretende solventarla la poca información que el sistema robótico comercial ofrece sobre los parámetros cinemáticos. Así se podrá determinar parámetros de velocidad y aceleración de cada uno de los eslabones y proporcionar una información valiosa para futuros estudios cinemáticos. Adicionalmente se pretende conocer mejor las posibilidades de gestión de señales digitales de entrada y salida que nos puede proporcionar el equipo robótico, ya que podrían ser muy útiles para realizar muchas de las labores que necesita la industria hoy en día..

Por lo tanto, nos centramos en la cinemática del robot, la cinemática es un estudio muy importante ya que nos posibilita conocer y predecir en qué lugar se encontrará un cuerpo, que velocidad tendrá al cabo de cierto tiempo, o bien que lapso llegará a su destino. Hacer la descripción del movimiento de un cuerpo significa precisar, a cada instante, su posición en el espacio. Para ello se necesitan herramientas que nos ayuden a tomar medidas, como en este caso la sensorización del sistema.

La cinemática es de vital importancia para no solo entender el movimiento de un cuerpo, sino cuantificarlo y hacerlo preciso.

4 FUNDAMENTOS

En este apartado se expondrá los fundamentos de la robótica, su morfología y los conocimientos cinemático necesarios.

Nos centraremos en el estudio cinemático, este consiste en referenciar las velocidades y las posiciones del robot respecto a un sistema de coordenadas. Buscando siempre la matriz jacobiana, ya que esa es una matriz diferencial que relaciona el vector de velocidades articulares con otro vector de velocidades expresado en un espacio distinto.

A su vez se hablará de la programación, de sus requisitos y de la sensorización tan importantes para este trabajo. Se comentarán técnicas de filtrado ya que se usarán para facilitar la interpretación de los datos experimentales. Por último, se hablará de Arduino, sus protocolos de comunicación y se describirá el entorno de trabajo.

4.1 Fundamentos de Robótica

Se entiende por robot industrial a un dispositivo mecánico que sea capaz de efectuar operaciones diferentes sin necesidad de llevar a cabo cambios importantes en el mismo. La definición más precisa la obtenemos de la Asociación Francesa de Normalización (AFNOR), aprobada en agosto de 1983. AFNOR define en primer lugar el manipulador (mecanismo multifuncional con varios grados de libertad, mandado por un operador humano o un sistema automático) y a continuación el robot industrial, entendido como un manipulador dotado de servosistemas de posición, reprogramable y polivalente. [1]

Figura 1. Cadena de montaje. [2]

A continuación se definirán conceptos básicos para la mejor compresión del documento. [1]

Manipulador

Mecanismo compuesto generalmente de elementos en serie, articulados o deslizantes entre sí, cuyo objetivo es el agarre y el desplazamiento de objetos siguiendo diversos grados de libertad. Es multifuncional y puede ser mandado directamente por un operador humano o por cualquier sistema lógico (levas, lógica neumática, lógica eléctrica cableada, o bien programado).

Robot industrial

Manipulador automático son servosistemas de posición, reprogramable, polivalente, capaz de posicionar y orientar materiales, piezas útiles o dispositivos especiales a lo largo de movimientos variables y programables para la ejecución de tareas variadas.

Se presenta a menudo bajo la forma de uno o varios brazos terminados en una muñeca. Su unidad de mando utiliza, esencialmente, un dispositivo de memoria y, eventualmente, de percepción y de adaptación al entorno y a las circunstancias.

Estas máquinas polivalentes son generalmente concebidas para efectuar la misma función de manera cíclica y pueden ser adaptadas a otras funciones sin modificación permanente del material.

A partir de estas definiciones permiten distinguirlos de otros equipos usados en la industria. En particular cabe diferenciarlos de las máquinas automáticas preparadas para la realización de un conjunto de operaciones previamente establecidas y que son difícilmente reprogramables para la realización de otros procesos u operaciones, requiriendo con ello ser objeto de modificaciones más o menos importantes.

Un robot está formado por varios eslabones los cuales se unen mediante articulaciones con distintos grados de libertad. Según la disposición de los eslabones y del tipo de articulación se obtienen una configuración para cada robot.

La constitución física de la mayor parte de los robots industriales guarda cierta similitud con la autonomía de las extremidades superiores del cuerpo humano, por lo que se podría hacer referencia a los distintos elementos que componen el robot, se usan términos como cintura, hombro, brazo, codo, muñeca, etc. [3]

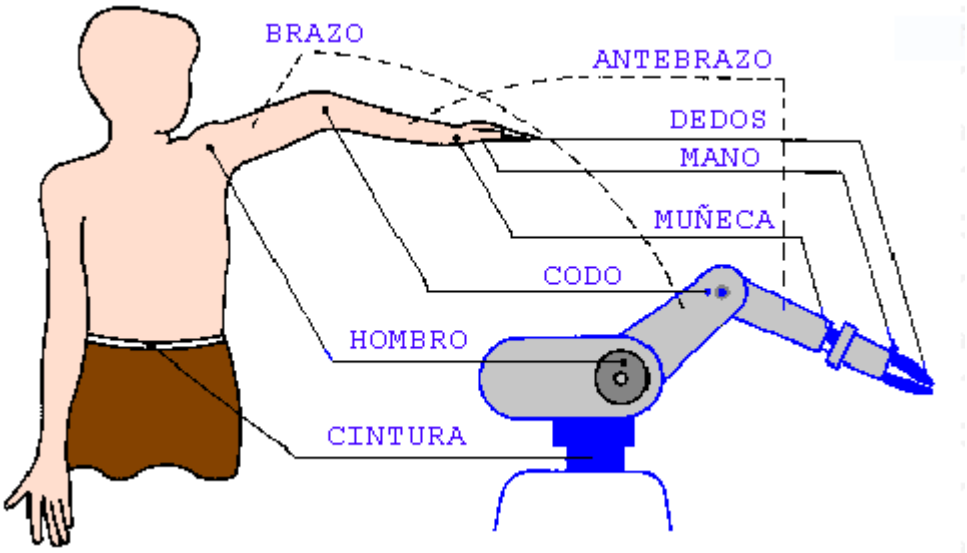


Figura 2. Morfología de brazo robótico, con la similitud de un cuerpo humano.[4]

Los elementos que forman parte de la totalidad del robot son:

- Manipulador: Es el componente principal. Está formado por una serie de elementos estructurales sólidos o eslabones unidos mediante articulaciones que permiten un movimiento relativo entre cada dos eslabones consecutivos. Las partes que conforman el manipulador reciben el nombre de: cuerpo, brazo, muñeca, y elemento terminal.
- Controlador: Es el encargado de regular cada uno de los movimientos del manipulador, las acciones, cálculos y procesado de la información. El controlador recibe y envía señales a otras máquinas-herramientas (por medio de señales de entradas/salidas) y almacena programas.
- Dispositivos de entradas y salidas: La señal de entrada y salida se obtienen mediante el armario que se incluye en el robot las cuales le permiten tener comunicación con otras máquinas-herramientas.
- Ordenador (de manera adicional)

4.1.1 Cinemática Linear

La cinemática relaciona los ángulos de la articulación con la posición y la orientación del final del robot. Esto significa que, para un robot en serie, podemos pensar en la cinemática como un mapeo hacia delante desde el espacio articular hasta el espacio de los movimientos rígidos del cuerpo, es decir, podemos expresar el movimiento del robot como el conjunto de sus movimientos. [5]

La imagen de este mapeo es el espacio de trabajo del robot. En general, el espacio de trabajo será sólo un subespacio de todos los movimientos rígidos del cuerpo, es decir, todas las posiciones y orientaciones alcanzables por el final del robot.

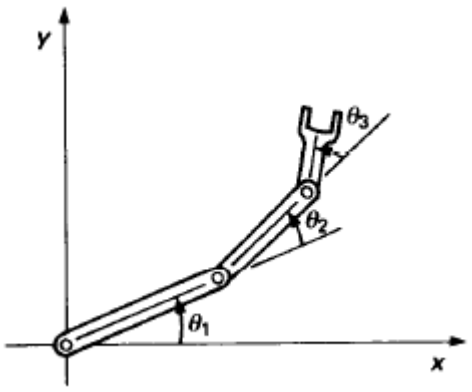


Figura 3. Ejemplo posición de un manipulador en el plano. [5]

Consideremos un punto x_c', y_c', z_c', que es el conjunto de los ángulos $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$:

$$\boldsymbol{K}: (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) \rightarrow (x_c', y_c', z_c') \qquad (1)$$

El mapa se da explícitamente en términos de las matrices A:

$$\begin{pmatrix} x_c' \\ y_c' \\ z_c' \end{pmatrix} = \boldsymbol{A}_1(\theta_1), \boldsymbol{A}_2(\theta_2), \boldsymbol{A_3}(\theta_3), \boldsymbol{A_4}(\theta_4), \boldsymbol{A_5}(\theta_5), \boldsymbol{A_6}(\theta_6) \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \qquad (2)$$

x_c, y_c, z_c son las coordenadas del punto en el origen. Por lo tanto, tenemos tres funciones:

$$x_c' = k_1(\theta_1, \dots, \theta_6) \qquad (3)$$
$$y_c' = k_2(\theta_1, \dots, \theta_6)$$
$$z_c' = k_3(\theta_1, \dots, \theta_6)$$

Estas funciones son de los ángulos de rotación. Sin embargo, si sólo estamos interesados en algún punto, es posible linealizar el mapa. Es decir, encontramos una aproximación lineal al mapa original. Así que hacemos pequeños cambios en los ángulos de rotación y obtenemos:

$$\delta x = \frac{\partial k_1}{\partial \theta_1}\partial\theta_1 + \frac{\partial k_1}{\partial \theta_2}\partial\theta_2 + \cdots + \frac{\partial k_1}{\partial \theta_6}\partial\theta_6 \qquad (4)$$
$$\delta y = \frac{\partial k_2}{\partial \theta_1}\partial\theta_1 + \frac{\partial k_2}{\partial \theta_2}\partial\theta_2 + \cdots + \frac{\partial k_2}{\partial \theta_6}\partial\theta_6$$
$$\delta z = \frac{\partial k_3}{\partial \theta_1}\partial\theta_1 + \frac{\partial k_3}{\partial \theta_2}\partial\theta_2 + \cdots + \frac{\partial k_3}{\partial \theta_6}\partial\theta_6$$

Sí escribimos $(\delta x_c', \delta y_c', \delta z_c')^T = \boldsymbol{\Delta x}$ y $(\delta\theta_1, \dots, \delta\theta_6)^T = \boldsymbol{\Delta\theta}$, podemos resumir las ecuaciones en:

$$\boldsymbol{\Delta x} = \mathbf{J}\ \boldsymbol{\Delta\theta} \qquad (5)$$

La matriz **J** es l **jacobiana** del mapa, permite aproximar linealmente la función en un punto. Esta es la matriz de las derivadas parciales.

$$\mathbf{J} = \begin{pmatrix} \frac{\partial k_1}{\partial \theta_1} & \frac{\partial k_1}{\partial \theta_2} & \cdots & \frac{\partial k_1}{\partial \theta_6} \\ \frac{\partial k_2}{\partial \theta_1} & \frac{\partial k_2}{\partial \theta_2} & \cdots & \frac{\partial k_2}{\partial \theta_6} \\ \frac{\partial k_3}{\partial \theta_1} & \frac{\partial k_3}{\partial \theta_2} & \cdots & \frac{\partial k_3}{\partial \theta_6} \end{pmatrix} \quad (6)$$

La matriz jacobiana se parece mucho a la primera derivada de una función de una variable. Para una función de varias variables tenemos una versión del teorema de Taylor:

$$\boldsymbol{x} + \boldsymbol{\Delta x} \approx \boldsymbol{k}(\boldsymbol{\theta}) + \boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{\Delta\theta} \quad (7)$$

Para una pequeña variación de $\boldsymbol{\theta}$ el mapa se aproxima por su valor en $\boldsymbol{\theta}$ más $\boldsymbol{J}(\boldsymbol{\theta})$ veces la variación de $\boldsymbol{\Delta\theta}$

4.1.2 Velocidades lineales

Un uso importante de la jacobiana es la relación de las rotaciones con la velocidad.

Como vimos anteriormente, $\boldsymbol{\Delta x} \approx \boldsymbol{J\Delta\theta}$**.** Dividiendo por $\boldsymbol{\Delta t}$ y obteniendo el límite, obtenemos la relación exacta [5]:

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\,\dot{\boldsymbol{\theta}} \quad (8)$$

Los puntos, como de costumbre, denotan diferenciación con respecto al tiempo. Es bastante general, pero usualmente nos interesa la velocidad lineal de algún punto, o la velocidad angular. Los movimientos que pueden realizar los robots son muy generales. Sin embargo, para cualquier movimiento de un cuerpo rígido hay siempre un centro de rotación. Del mismo modo para una superficie esférica siempre hay un eje de rotación. Para movimientos rígidos en tres dimensiones siempre hay una línea fija (el eje del tornillo). Si un cuerpo rígido experimenta algún movimiento complicado, entonces en cualquier momento del movimiento habrá un centro instantáneo de rotación. Del mismo modo obtenemos los ejes instantáneos de rotación. Como veremos más adelante estos conceptos están estrechamente relacionados con las velocidades que nos interesan.

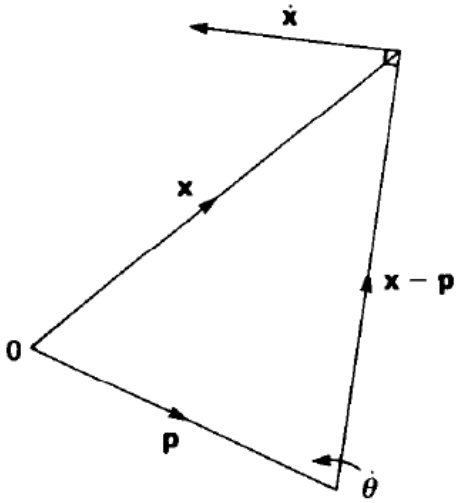


Figura 4.Velocidad lineal dada por un centro instantáneo de rotación. [5]

En dos dimensiones tenemos una relación simple entre la velocidad de un punto y el centro instantáneo de rotación, Figura 4. Se puede mostrar usando las matrices 3x3 que representan los movimientos rígidos. Sea $\vec{\boldsymbol{p}}$ el centro de rotación y supongamos que deseamos conocer la velocidad del punto $\boldsymbol{x}$. Ahora la posición de x está dada por:

$$\begin{pmatrix} x(t) \\ 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{R}(\theta) & \big(\boldsymbol{I} - \boldsymbol{R}(\theta)\big)\mathrm{p} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x(0) \\ 1 \end{pmatrix} \quad (9)$$

Siendo $\boldsymbol{R}$ *la matriz de rotación.* Ahora asumamos que $\theta = 0$ cuando $t = 0$.

$$\begin{pmatrix} \dot{x}(0) \\ 0 \end{pmatrix} = \begin{pmatrix} \dot{\boldsymbol{R}}(0)\dot{\theta} & -\dot{\boldsymbol{R}}(0)p\dot{\theta} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x(0) \\ 1 \end{pmatrix} \quad (10)$$

Sabemos que la matriz de rotación $\boldsymbol{R} = \begin{pmatrix} \cos\theta & -sen\,\theta \\ sen\,\theta & \cos\theta \end{pmatrix}$, como $\theta = 0$:

$$\dot{\boldsymbol{R}}(0) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad (11)$$

Podemos comenzar a medir el tiempo en cualquier momento, no solo para $t = 0$. Podemos quitar la dependencia del tiempo y escribir:

$$\dot{x} = (p_x - y)\dot{\theta} \quad (12)$$
$$\dot{y} = (x - p_x)\dot{\theta}$$

Este resultado se puede utilizar calcular el jacobiano del manipulador plano. Para uno en tres ejes tenemos:

$$\begin{pmatrix} x(t) \\ 1 \end{pmatrix} = \boldsymbol{A}_1(\theta_1)\boldsymbol{A}_2(\theta_2)\boldsymbol{A_3}(\theta_3) \begin{pmatrix} x(0) \\ 1 \end{pmatrix} \quad (13)$$

De nuevo podemos organizar las cosas para nuestro punto de interés $\theta_1 = \theta_2 = \theta_3 = 0$. Entonces, dado que $\boldsymbol{A}_i(0)$ es la matriz identidad, cuando diferenciamos y fijamos los ángulos de unión a cero, obtenemos:

$$\begin{pmatrix} \dot{x}(0) \\ 0 \end{pmatrix} = \dot{\boldsymbol{A}}_1(0) \begin{pmatrix} x(0) \\ 0 \end{pmatrix} \dot{\theta}_1 + \dot{\boldsymbol{A}}_2(0) \begin{pmatrix} x(0) \\ 0 \end{pmatrix} \dot{\theta}_2 + \dot{\boldsymbol{A_3}}(0) \begin{pmatrix} x(0) \\ 0 \end{pmatrix} \dot{\theta}_3 \quad (14)$$

Para las matrices **A**, el centro de rotación es simplemente la posición actual de la junta. Asique si denominamos a la posición de la junta i por j_i, tenemos:

$$\dot{x} = (j_{1y} - y)\dot{\theta_1} + (j_{2y} - y)\dot{\theta_2} + (j_{3y} - y)\dot{\theta_3} \quad (15)$$
$$\dot{y} = (x - j_{1y})\dot{\theta_1} + (x - j_{2y})\dot{\theta_2} + (x - j_{3y})\dot{\theta_3}$$

En forma matricial:

$$\begin{pmatrix}\dot{x}\\ \dot{y}\end{pmatrix} = \begin{pmatrix}(j_{1y} - y) & (j_{2y} - y) & (j_{3y} - y)\\ (x - j_{1y}) & (x - j_{2y}) & (x - j_{3y})\end{pmatrix}\begin{pmatrix}\dot{\theta_1}\\ \dot{\theta_2}\\ \dot{\theta_3}\end{pmatrix} \quad (16)$$

Por lo tanto, la jacobiana es:

$$\boldsymbol{J}(\theta_1, \theta_2, \theta_3) = \begin{pmatrix}(j_{1y} - y) & (j_{2y} - y) & (j_{3y} - y)\\ (x - j_{1y}) & (x - j_{2y}) & (x - j_{3y})\end{pmatrix} \quad (17)$$

4.1.3 Velocidades angulares

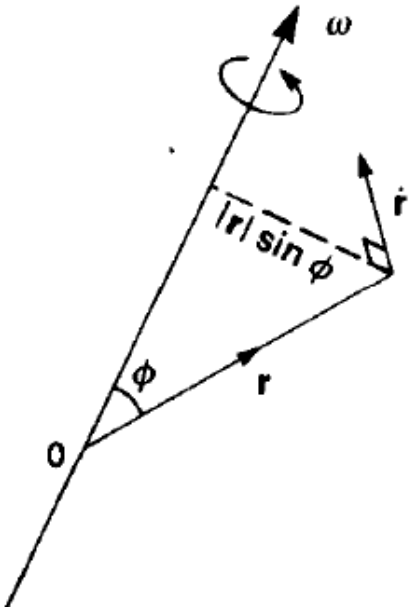


Figura 5. Velocidad angular en relación a la velocidad lineal. [5]

La velocidad angular de un cuerpo rígido es un vector. Está alineado a lo largo del eje de rotación del cuerpo y su magnitud es la velocidad angular alrededor del eje [5]. Consideremos un punto definido por el vector r unido a un cuerpo que gira con velocidad angular $\boldsymbol{\omega}$, véase la figura 5. La velocidad lineal del punto está dada por $\dot{r} = \omega \wedge r$. Si representamos las rotaciones por matrices 3x3 tenemos que:

$$\boldsymbol{r}(t) = \boldsymbol{R}(t)\boldsymbol{r}(0) \quad (18)$$

Y para $t = 0$.

$$\dot{\boldsymbol{r}}(t) = \dot{\boldsymbol{R}}(t)\boldsymbol{r}(0) \quad (19)$$

Para cualquier vector **a** tenemos:

$$\dot{\boldsymbol{R}}(0)a = \omega \wedge a \qquad (20)$$

Esto se puede utilizar para encontrar la velocidad del último eslabón. Para tres articulaciones, la transformación global **K** es el producto de tres rotaciones, $\boldsymbol{K} = \boldsymbol{R}_1(\theta_1)\boldsymbol{R}_2(\theta_2)\boldsymbol{R}_3(\theta_3)$. La derivada cuanto todos los ángulos de la articulación son cero:

$$\dot{\boldsymbol{K}} = \dot{\boldsymbol{R}}_1 + \dot{\boldsymbol{R}}_2 + \dot{\boldsymbol{R}}_3 \qquad (21)$$

Y por lo tanto la velocidad angular del eslabón final es justamente la suma de las velocidades angulares de la articulación. Ahora, debido a que cada articulación sólo gira alrededor de su eje, podemos escribir las velocidades angulares de la articulación como $\omega_i = \hat{v}_i\dot{\theta}_i$; Donde $\hat{v}_i$ es el vector unitario a lo largo de la articulación i. La velocidad angular del último eslabón puede expresarse mediante la siguiente ecuación matricial:

$$\omega = \begin{pmatrix} \hat{v}_{1x} & \hat{v}_{2x} & \hat{v}_{3x} \\ \hat{v}_{1y} & \hat{v}_{2y} & \hat{v}_{3y} \\ \hat{v}_{1z} & \hat{v}_{2z} & \hat{v}_{3z} \end{pmatrix} \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{pmatrix} \qquad (22)$$

Observe que las columnas del jacobiano aquí son sólo los vectores a lo largo de los ejes comunes.

4.1.4 Combinación de velocidad lineal y angular

Para un cuerpo rígido moviéndose en tres dimensiones que queremos conocer la velocidad angular y lineal en un punto. Tenemos que considerarlo como un movimiento de tornillo general.

$$\begin{pmatrix} \boldsymbol{R} & t \\ 0 & 1 \end{pmatrix}$$

La velocidad del punto x viene dada por:

$$\begin{pmatrix} \dot{x} \\ 0 \end{pmatrix} = \begin{pmatrix} \dot{\boldsymbol{R}} & \dot{t} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} \qquad (23)$$

El vector de translación viene dado por $t = \frac{\theta p}{2\pi}\hat{v} + (\boldsymbol{I} - \boldsymbol{R})\boldsymbol{u}$, donde p es el paso del tornillo, v el vector a lo largo del eje, u es un punto del eje.

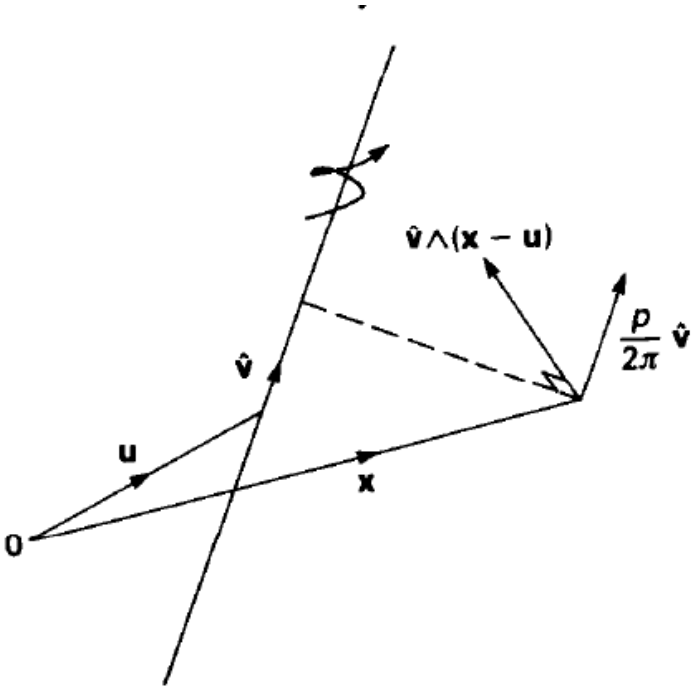


Figura 6. Movimiento de un tornillo. [5]

Las derivadas de las matrices de rotación:

$$\dot{x} = \omega \wedge x + s \qquad (24)$$

La velocidad lineal $s = \dot{t}$, es una velocidad característica del movimiento. Físicamente es la velocidad lineal del punto en una línea a través del origen, paralelo al eje de rotación. Podemos encontrar s diferenciando t:

$$s = \dot{t} = \theta \frac{p}{2\pi} \hat{v} - \dot{\boldsymbol{R}} u = \theta \frac{p}{2\pi} \hat{v} - \omega \wedge u \qquad (25)$$

Podemos combinar la velocidad angular y lineal en seis componentes de los vectores $\binom{\omega}{s}$, como la velocidad angular de $\omega = \hat{v}\dot{\theta}$, la velocidad de un punto atada a la articulación será dada por:

$$x = (\hat{v} \wedge (x - u) + (\frac{p}{2\pi} \hat{v})\dot{\theta} \qquad (26)$$

Conectando las 6 articulaciones, en un robot en serie, la velocidad lineal y angular podrían escribirse:

$$\binom{\omega}{s} = \begin{pmatrix} \hat{v}_1 \\ u_1 \wedge \hat{v}_1 + \frac{p_1}{2\pi} \hat{v}_1 \end{pmatrix} \dot{\theta}_1 + \begin{pmatrix} \hat{v}_2 \\ u_2 \wedge \hat{v}_2 + \frac{p_2}{2\pi} \hat{v}_2 \end{pmatrix} \dot{\theta}_2 + \cdots + \begin{pmatrix} \hat{v}_6 \\ u_6 \wedge \hat{v}_6 + \frac{p_6}{2\pi} \hat{v}_6 \end{pmatrix} \dot{\theta}_6 \qquad (27)$$

Expresado en forma matricial:

$$\begin{pmatrix}\omega\\ s\end{pmatrix} = \boldsymbol{J}\begin{pmatrix}\dot{\theta}_1\\ \dot{\theta}_2\\ \dot{\theta}_3\\ \dot{\theta}_4\\ \dot{\theta}_5\\ \dot{\theta}_6\end{pmatrix} \quad (28)$$

4.1.5 Cinemática individual

Para la realización de este trabajo, es necesario introducir unos fundamentos sobre el movimiento circular uniformemente acelerado. Ya que al separar el conjunto total del robot y centrarnos de manera individual en cada eslabón, los movimientos que realizan estos son circulares. [6]

4.1.5.1 Posición angular, θ.

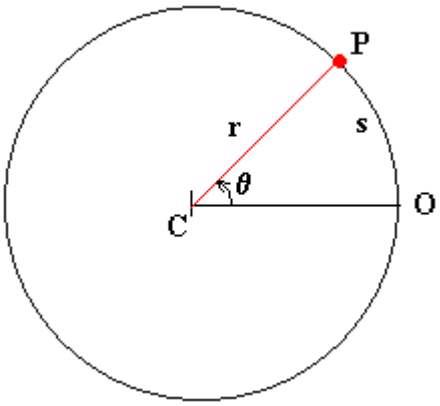


Figura 7. Circunferencia posición angular

En el instante t el móvil se encuentra en el punto P. Su posición angular viene dada por el ángulo θ, que hace en punto P, el centro de la circunferencia C y el origen de ángulos O.

El ángulo θ, es el cociente entre la longitud de arco s y el radio de la circunferencia r, $\theta = \frac{s}{r}$. La posición angular es el cociente entre dos longitudes y, por lo tanto, no tiene dimensiones.

4.1.5.2 Velocidad angular, ω.

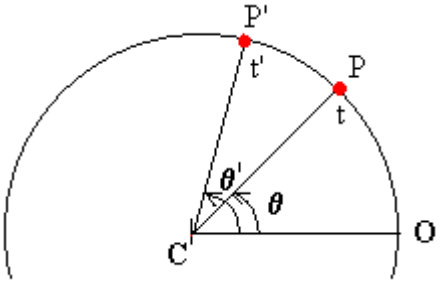


Figura 8. C. Velocidad angular.

En el instante t' el móvil se encontrará en la posición P' dada por el ángulo θ'. El móvil se habrá desplazado $\Delta\theta = \theta' - \theta$ en el intervalo de tiempo $\Delta t = t' - t$ comprendido entre t y t'

Se denomina velocidad angular media al cociente entre el desplazamiento y el tiempo.

$$\vec{\omega} = \omega\vec{k} = \dot{\theta}\vec{k} \quad (29)$$

4.1.5.3 Aceleración.

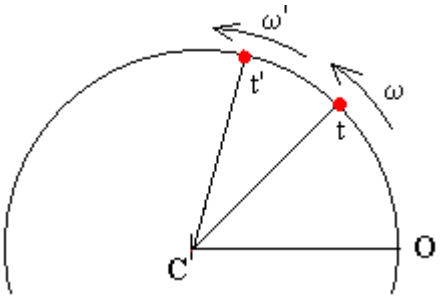


Figura 9. C. Aceleración angular

Si en el instante *t* la velocidad angular del móvil es ω y en el instante *t'* la velocidad angular del móvil es ω'. La velocidad angular del móvil ha cambiado $\Delta\omega = \omega' - \omega$ en el intervalo de tiempo $\Delta t = t' - t$ comprendido entre *t* y *t'*.

Se denomina aceleración angular media al cociente entre el cambio de velocidad angular y el intervalo de tiempo que tarda en efectuar dicho cambio.

$$\vec{\alpha} = \frac{\mathrm{d}\omega}{\mathrm{d}t} = \alpha\vec{k} = \omega\vec{k} = \ddot{\theta}\vec{k} \qquad (30)$$

La aceleración es la suma de dos vectores:

Aceleración tangencial y la aceleración normal,

$$\vec{a} = \vec{\alpha} \times \vec{r} + \vec{\omega} \times\vec{\omega}\times\vec{r} \qquad (31)$$

4.1.5.4 Movimiento circular uniformemente acelerado.

Un movimiento circular uniformemente acelerado es aquél cuya aceleración α es constante. Dada la aceleración angular podemos obtener el cambio de velocidad angular $\omega - \omega_0$ entre los instantes t_0y *t,* mediante integración.

$$\omega - \omega_0 = \alpha\,(t - t_0) \qquad (32)$$

Dada la velocidad angular ω en función del tiempo, obtenemos el desplazamiento $\theta - \theta_0$ del móvil entre los instantes t_o, t integrando obtenemos:

$$\theta - \theta_0 = \omega_0(t - t_0) + \frac{1}{2}\alpha(t - t_o)^2 \qquad (33)$$

Para un instante inicial t_0 se toma como cero:

$$\begin{gathered} \alpha = cte \\ \omega = \omega_0 + \alpha t \\ \theta = \theta_0 + \omega_0 t + \frac{1}{2}\alpha^2 \end{gathered} \qquad (34)$$

Relacionando la velocidad ω con el desplazamiento $\theta - \theta_0$:

$$\omega^2 = \omega_0^2 + 2\alpha(\theta - \theta_0) \qquad (35)$$

4.1.6 Programación de Robots

Programar un robot consiste en indicar paso por paso las diferentes acciones que éste deberá realizar durante su funcionamiento automático. La flexibilidad en la aplicación del robot y, por tanto, su utilidad dependerá en gran medida de las características de su sistema de programación. [1]

A pesar de no unificar los procedimientos de programación de los robots y que cada fabricante ha desarrollado su método particular, válido únicamente para sus propios robots, existen algunos sistemas de programación que han servido de modelo para el desarrollo

de otros. Tal es el caso del lenguaje AL(Finkek-74) desarrollado por la Universidad de Stanford en los años setenta y que ha servido de referencia para muchos de los sistemas comerciales existentes. [1]

4.1.6.1 Requerimientos de un sistema de programación de robots

Pesa a no haber normalización entre los métodos de programación de robots existentes, las necesidades comunes han originado un cierto paralelismo y afinidad entre casi todos ellos. Estas circunstancias permiten establecer una serie de características generales que se manifiestan en los elementos de programación que contienen:

- ***Entorno de programación:*** Programar con el robot es complicado en el sentido que es un proceso continuo de prueba y error. Ésta es la principal causa que lleva a que la mayoría de los sistemas de programación de robot sean de tipo interpretado, pudiéndose realizar un seguimiento pasa a paso. En coincidencia con otras tecnologías industriales, se puede observar la tendencia a utilizar con entorno sistemas operativos de amplia difusión, con la facilidad que le supone al usuario al encontrarse familiarizado con el sistema. Esto ha llevado al establecimiento de la norma UNE EN ISO 15187-2003 [UNE-03], en la que se normaliza las interfaces gráficas que el usuario puede utilizar para la programación de robots. La norma no especifica el lenguaje, sino las funciones más importantes.
- ***Modelado del entorno:*** Es la representación que tiene el robot de los objetos con los que interacciona. Normalmente, este modelo se limita a características geométrica: posición y orientación de los objetos. Para definir la posición y orientación de los objetos del modelo, lo más frecuente es asignar a cada objeto de manera solidaria un sistema de referencia, de manera que la posición y la orientación de este sistema referidos a un sistema base.
- ***Tipo de datos:*** Un sistema de programación de robots cuenta con los tipos de datos convencionales (enteros, reales, booleanos, etc) con otros específicamente destinados a definir las operaciones de interacción con el entorno. Ejemplo: Ángulos de Euler, Cuaternios, Matrices.
- ***Manejo de entradas y salidas:*** La comunicación del robot con otras máquinas o procesos que cooperan con él, es fundamentar para conseguir su integración y sincronización en los procesos de fabricación. Para el manejo de salidas el robot posee instrucciones de activación o desactivación de las mismas. En cuanto a las entradas, el robot tiene capacidad de leerlas y controlar el flujo del programa en función de su valor. Otra aplicación

importante de las entradas-salidas del robot, ya sean digitales o analógicas, es la integración de sensores, incorporando la información de estos al desarrollo de la tarea. La información proporcionada por los sensores puede utilizarse en la programación de robots de muy diversas formas, como, modificar trayectorias, obtener la identidad y posición de objetos, cumplir con restricciones externas.

- ***Comunicaciones:*** La integración de los elementos que participan en el mismo con el resto de los sistemas de control y gestión de la producción se ha convertido en el presente en una necesidad. La mayor parte de robots industriales de nueva generación han dispuesto vías de comunicación basadas en estándares.
- ***Control del movimiento del robot:*** un método de programación debe incluir la posibilidad de especificar el movimiento del robot. Incluso en ocasiones puede ser necesario indicar si el movimiento debe realizarse en cualquier caso o debe estar condicionada por algún sensor.

4.1.6.2 Programación en RAPID.

RAPID es un lenguaje de programación de alto nivel, que la marca ABB emplea para programar sus robots. La programación consiste en indicar las diferentes acciones que tiene que realizar el robot durante su funcionamiento. [7]

Una aplicación RAPID está compuesta por diferentes programas y módulos de sistemas. Un programa está formado por las siguientes partes:

- Main: Rutina principal que ejecuta el robot, siguiendo paso a paso las funciones que la componen.
- Subrutinas: Dentro de la rutina principal, se puede hacer referencia a subrutinas. Esto sirva para disminuir el tamaño de la rutina y programar de una forma organizada. Además, permite la utilización de subrutinas en diferentes rutinas. Para hacer funcionar estas subrutinas solo se debe escribir el nombre en la rutina principal Main.
- Datos de programa: Estos datos son necesarios en todos los programas. Sirven para definir las posiciones, sistemas de coordenadas, herramientas, etc. Que se necesitan para la ejecución. Pueden estar definidos tanto en la Rutina principal, como en las subrutinas, pero siempre deben estar definidos todos los datos antes de las líneas de funciones.

  Estos datos pueden ser definidos según su variabilidad:
  - Constantes (CONS): Valor fijo.
  - Variables (VAR): Pueden variar durante la ejecución del programa.

- Persistentes (PERS): Al cambiar su valor también se cambia su inicialización. En el momento inicial es necesario asignarle un valor.

También pueden clasificarse según el tipo de datos:

- Bool: Valores lógicos.
- Clock: Medición de tiempo.
- Confdata: Datos de configuración.
- Jointtarget: Datos de posición de los ejes.
- Loaddata : Datos de carga.
- Num: Valores numéricos.
- Orient : Orientación.
- Robtarget: Datos de posición.
- Speeddata : Datos de velocidad.
- String: Cadena de Caracteres.
- Tooldata: Datos de herramienta.
- Wobjdata: Datos del objeto de trabajo.
- Zonedata: Datos de la zona.

A continuación, se muestran los distintos comandos para programar en RAPID:

- Instrucciones de movimiento
  - **MoveJ** Punto, Velocidad, Zona, Herramienta:

El movimiento se realiza de punto a punto sin necesidad de seguir una trayectoria.

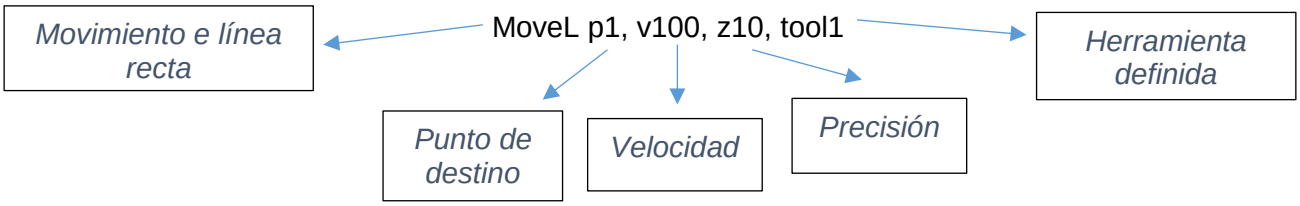
  - **MoveL** Punto, Velocidad, Zona, Herramienta:

El robot mueve su extremo de forma lineal.

  - **MoveC** Punto_Intermedio, Punto_Destino, Velocidad, Zona, Herramienta: El robot genera un arco de circunferencia pasando por un punto intermedio hasta el punto de destino, para ello el ángulo del arco debe ser ≤180 º.
    - **Offs** (Punto_inicial,Desp_x,Desp_y,Desp_z):

Sirve para añadir un desplazamiento a una posición predefinida.

- Control de flujo:
  - IF / ELSE: Crea una condición, si cumple ejecuta esa parte del programa.
  - WHILE: Para crear bucles dentro del programa.
- Un ejemplo de una instrucción para el brazo robótico sería:



- Podemos ver cómo se pueden especificar tanto el punto al que queremos hacer llegar el extremo del robot como velocidad del movimiento, la precisión y la herramienta que está utilizando el robot.
- La precisión '*z*', es un dato tipo '**ZoneData**' que especifica con qué precisión se debe alcanzar la posición solicitada antes de permitir que el robot se mueva hasta el punto siguiente.

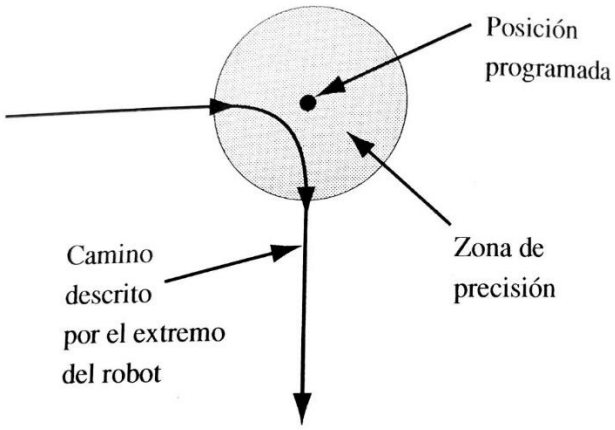


Figura 10.Funcionamiento de un punto de paso. [1]

4.1.6.3 Movimiento del robot mediante FlexPendant

Para la ayuda al movimiento del robot, el FlexPendant consta de una ventana llamada "Ventana de movimientos" en la que se puede visualizar y modificar el tipo de movimiento, el sistema de coordenadas de referencia y herramienta usada. También muestra la posición final de la herramienta, y los ángulos girados por los ejes. [8]

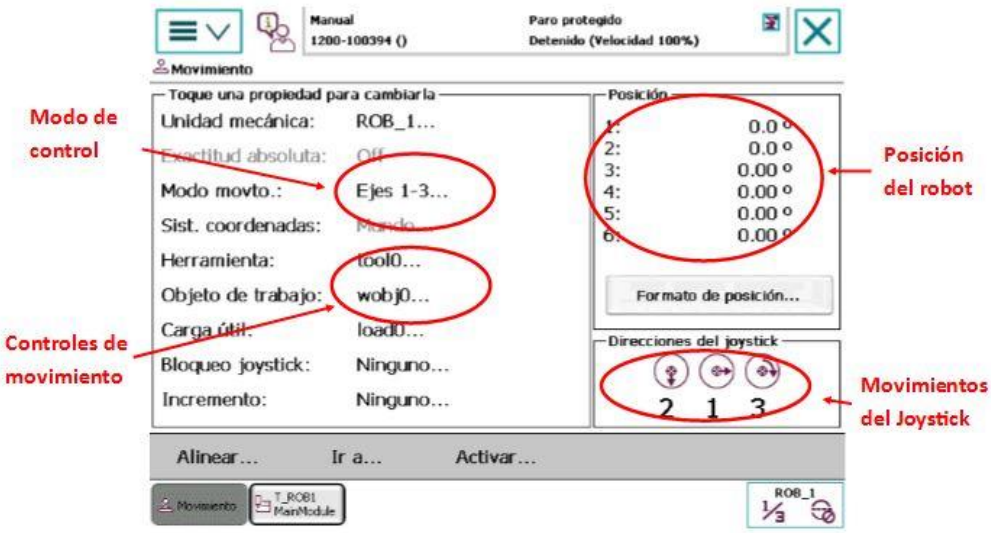


Figura 11. Ventana de movimiento en FlexPendant [8].

4.2 Fundamentos de Sensorización

Se denomina sensor a todos los elementos que permiten captar estados o eventos físicos de modo que puedan ser transformados en variaciones de tensión o impulso eléctricos así poder ser interpretados. Existen multitud de sensores por ejemplo para medir temperaturas, posiciones, radiación, luz, vibración. Todos tienen en común que de algún modo varían la intensidad o tensión en alguna de las ramas del circuito eléctrico en el que se hayan conectado.[9]

Los sensores en Arduino los podemos encontrar en varios formatos:

- Independientes: Se tratan de componentes sin soporte alguno, como suelen venir las resistencias, con el elemento y sus terminales.
- Montados en placas: Muchos sensores basan sus medidas en divisiones de tensión. Normalmente se componen de tres terminales, siendo uno de ellos de tensión, otro el de señal y por último otro con tierra. Otros incluyen lecturas analógicas o digitales. O que incorporen led.

Figura 12. Sensor de ultrasonido.[10]

- Montados en shields: Son placas destinadas exclusivamente a Arduino. Incorporando uno o varios sensores y todos los elementos necesarios para su funcionamiento. Al montarlo con la placa los pines coinciden.

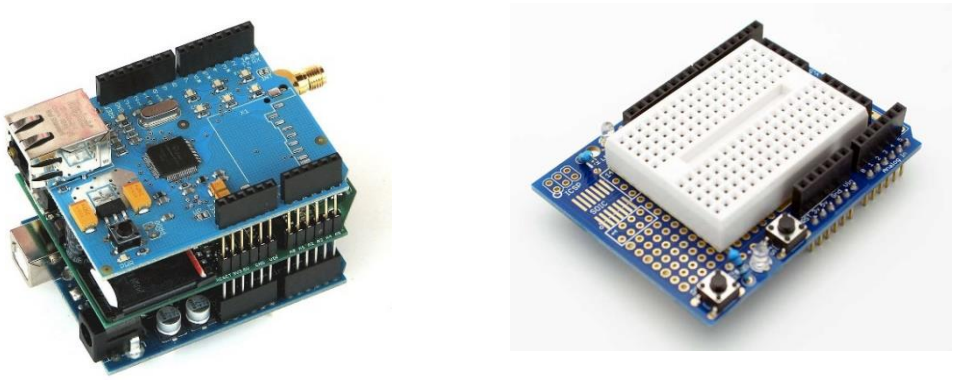


Figura 13. Shield Arduino.[10]

Existen muchos tipos de sensores, por lo que la posible clasificación depende del tipo, aporten o función que tenga [11]:

Según su aporte de energía, estos pueden ser **Modulares** que precisan de una fuente externa de alimentación o **Generadores** que toman únicamente la energía del medio donde miden. Según la señal de salida estos, pueden ser **Analógicos** donde la salida varía de forma continua. Cuando la información está en la frecuencia se denominan "cuasi-digitales". **Digitales** salida varía en pasos discretos. Según el modo de funcionamiento, pueden ser por **Deflexión** cuya magnitud medida genera un efecto físico o por **Comparación** que intenta mantener nula la deflexión mediante la aplicación de un efecto opuesto al generado. Según relación **entrada-salida**, Orden cero, primer orden… o según la **magnitud medida** ya sea la temperatura, presión, aceleración, pH…

Para la realización de este trabajo es necesario la incorporación de acelerómetros. Ya que estos nos permiten conocer la inclinación del eslabón donde este aplicado el sensor. Se procede a explicar estos sensores de forma detallada.

4.2.1 Acelerómetro.

El acelerómetro es un dispositivo capaz de medir aceleraciones, es decir, variaciones de la velocidad por unidad de tiempo. Teniendo en cuenta la segunda ley de Newton ($F = m \cdot a$) y suponiendo una masa (m) constante, resulta que la aceleración media (a) será proporcional a la fuerza (F) que se le aplique al acelerómetro.

Estas fuerzas pueden ser estáticas o dinámicas, las estáticas incluyen la gravedad mientras que dinámicas pueden incluir vibraciones y/o movimientos, con el empleo de filtros pasa bajo o pasa alto se podrá discriminar la componente no deseada.

Generalmente, los acelerómetros contienen placas capacitivas internamente, algunas de estas placas son fijas, mientras que otras están unidas a resortes minúsculos que se mueven internamente conforme las fuerzas de aceleración que actúan sobre el sensor. Como las placas se mueven, la capacitancia entre ellas cambia. A partir de esos cambios le determina la aceleración.

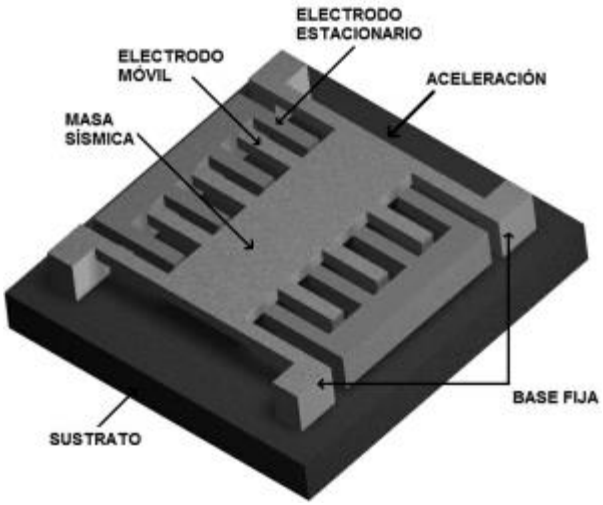


Figura 14. Sistema aceleración en un eje. [12]

4.2.2 Giróscopo

Los giroscopios MEMS (componentes electrónicos y mecánicos miniaturizados) miden la velocidad angular (ω) usando el efecto Coriolis. Cuando se hace girar el giroscopio, una pequeña masa se desplaza con los cambios de velocidad angular. Un sensor capacitivo mide esa velocidad angular que es proporcional al recorrido de dicha masa.

A partir de la variación del ángulo de giro es posible determinar el grado de inclinación del dispositivo en el que se encuentra montado.

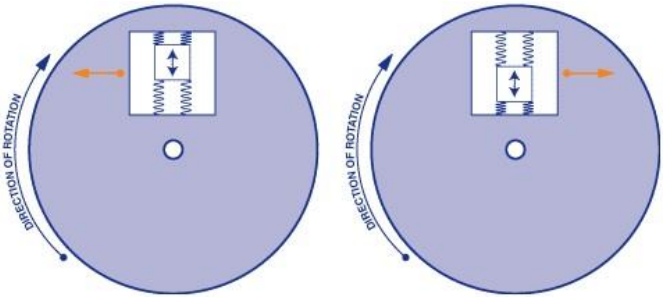


Figura 15. Funcionamiento de un sensor giroscópico MEMS. [12[

4.2.3 Cálculo de la inclinación del sensor

El acelerómetro mide cambios de velocidad, pero cuando se encuentra estático la única aceleración que detecta es la gravedad, por lo tanto, en el eje Z tendremos 1 g sentido hacia abajo. [13]

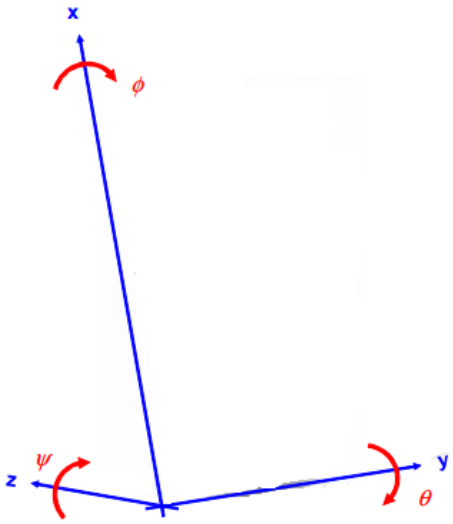


Figura 16. Sistema de Coordenadas y rotación.

Pero una vez que inclinemos el sensor, la fuerza de la gravedad genera componentes vectoriales en los ejes $G_x, G_y\ y\ G_z$. Estas componentes son las que utilizamos para estimar el ángulo de inclinación que tiene el sensor respecto a la fuerza de la gravedad (g).

$$G = \begin{pmatrix} G_x \\ G_y \\ G_z \end{pmatrix} = R \cdot g = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (36)$$

Llamamos R a la matriz de rotación tridimensional que representa la transformación de coordenadas desde el sistema fijo al sistema móvil:

$$[\boldsymbol{Rx}(\varphi)] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \operatorname{sen}\theta \\ 0 & -\operatorname{sen}\varphi & \cos\theta \end{bmatrix}$$

$$[\boldsymbol{Ry}(\theta)] = \begin{bmatrix} \cos\theta & 0 & -\operatorname{sen}\theta \\ 0 & 1 & 0 \\ \operatorname{sen}\theta & 0 & \cos\theta \end{bmatrix} \quad (37)$$

$$[\boldsymbol{Rz}(\gamma)] = \begin{bmatrix} \cos\gamma & \operatorname{sen}\gamma & 0 \\ -\operatorname{sen}\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Hay seis configuraciones diferentes de estas tres matrices de rotación, la matriz de rotación no conmuta, lo que significa que la matriz de rotación R depende del orden en el que se aplique.

$$R_{xyz}\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = R_x(\varphi)R_y(\theta)R_z(\gamma)\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta\cos\gamma & \cos\theta\, sen\,\gamma & -sen \\ \cos\gamma\, sen\,\theta\ sen\,\varphi - \cos\varphi\ sen\,\gamma & \cos\varphi\cos\gamma + sen\,\theta\, sen\,\varphi\, sen\,\gamma & \cos\theta\, sen\ \varphi \\ \cos\varphi\cos\gamma\, sen\theta + sen\,\varphi\, sen\,\gamma & \cos\varphi\, sen\,\theta\, sen\,\gamma - \cos\gamma\, sen\ \varphi & \cos\theta\cos\varphi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} -\,sen\,\theta \\ \cos\theta\, sen\ \varphi \\ \cos\theta\ \cos\varphi \end{pmatrix}$$

$$\frac{G}{\|G\|} = \begin{pmatrix} -\,sen\,\theta \\ \cos\theta\, sen\ \varphi \\ \cos\theta\ \cos\varphi \end{pmatrix}$$

$$\frac{1}{\sqrt{G_x^2 + G_y^2 + G_z^2}} \begin{pmatrix} G_x \\ G_y \\ G_z \end{pmatrix} = \begin{pmatrix} -\,sen\,\theta \\ \cos\theta\, sen\ \varphi \\ \cos\theta\ \cos\varphi \end{pmatrix}$$

$$\boldsymbol{\tan\varphi = \left(\frac{G_y}{G_z}\right)}$$

$$\boldsymbol{\tan\theta = \left(\frac{-G_x}{G_y\, sen\,\varphi + G_z\cos\theta}\right) = -\frac{G_x}{\sqrt{G_y^2 + G_z^2}}} \qquad (38)$$

$$\boldsymbol{R}_{xyz} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = R_x(\varphi)R_y(\theta)R_z(\gamma) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \cos\gamma\cos\theta - sen\,\theta\, sen\,\varphi\, sen\,\gamma & sen\,\gamma\cos\theta + sen\,\theta\, sen\,\varphi\cos\gamma & -sen\,\theta\cos\varphi \\ -\cos\varphi\, sen\ \gamma & \cos\varphi\cos\gamma & sen\ \varphi \\ \cos\theta\, sen\,\varphi\, sen\,\gamma + sen\,\theta\cos\gamma & -\cos\gamma\cos\theta\, sen\,\varphi + sen\,\gamma\, sen\,\theta & \cos\theta\cos\varphi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} -\,sen\,\theta\cos\gamma \\ sen\,\gamma \\ \cos\theta\ \cos\varphi \end{pmatrix}$$

$$\frac{G}{\|G\|} = \begin{pmatrix} -\,sen\,\theta\cos\gamma \\ sen\,\gamma \\ \cos\theta\ \cos\varphi \end{pmatrix}$$

$$\frac{1}{\sqrt{G_x^2 + G_y^2 + G_z^2}} \begin{pmatrix} G_x \\ G_y \\ G_z \end{pmatrix} = \begin{pmatrix} -\,sen\,\theta\cos\gamma \\ sen\,\gamma \\ \cos\theta\ \cos\varphi \end{pmatrix}$$

$$\boldsymbol{\tan\varphi = \frac{G_y}{\sqrt{G_x^2 + G_z^2}}}$$

$$\boldsymbol{\tan\theta = \left(\frac{-G_x}{G_z}\right)} \qquad (39)$$

Como ya conocemos las ecuaciones para calcular el ángulo, lo implementamos. Para poder realizar una potencia en Arduino, utilizamos el comando **pow(**base, exponente**)** y para realizar raíces cuadradas utilizamos el comando **sqrt().**

4.3 Técnicas de filtrado

Para eliminar el mayor ruido posible y conseguir que el acelerómetro mantenga unos parámetros estables, hay diferentes algoritmos que filtran la información. El mejor es el *Filtro Kalman*, desarrollado por Rudolf E. Kalman en el 1960. Es considerado uno de los grandes descubrimientos del siglo XX por la implicación en el filtrado de sensores. [12]

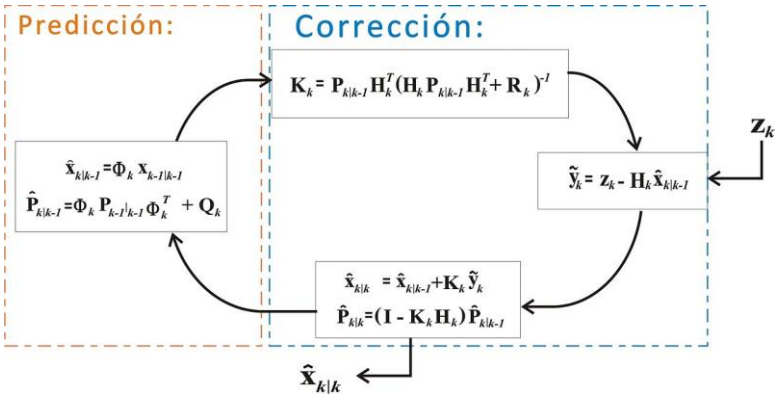


Figura 17. Algoritmo recursivo de Filtro de Kalman.[12]

El filtro Kalman es capaz de calcular el error de cada medida a partir de las medias anteriores, eliminar y dar valor real del ángulo. El problema de este algoritmo es que requiere un gran consumo para el microprocesador ya que necesita de mucho cálculo.

El uso del **filtro complementario** requiere un menor coste de procesamiento ya el algoritmo utilizado es una simplificación de Kalman y con se adquiere buena precisión. El funcionamiento de este filtro consiste en la unión de dos filtros: un filtro pasa altos (FPA) cuya respuesta en frecuencia se atenúa las componentes de baja frecuencia, pero no las de alta, lo aplicamos al giroscopio y un filtro pasa bajos (FPB) que funciona de forma inversa que el pasa altos, lo aplicamos para el acelerómetro.

4.4 Arduino

Para recoger, procesar y controlar la señal de los sensores, utilizaremos Arduino.

Arduino es una plataforma electrónica de código abierto basado en hardware y software fácil de usar. Las placas son capaces de leer entradas (luz, un botón, un motor,..) y convertirlo en una salida. El microcontrolador de la placa se programa usando Arduino pgramemming language (basado en Wiring) y el Arduino developmen environment (basado en Processing). [14]

En cuanto a la parte hardware, contiene un microcontrolador de la familia Atmel llamado ATMEGA. Existen varias versiones de este microcontrolador, siendo un microcontrolador de 8 o 32 bits. Por norma general, las placas de Arduino, excepto algunas versiones, están fabricadas con microcontroladores de 8 bits.

Para la realización de este trabajo se ha utilizado una placa de Arduino Uno R3 está basado en ATmega328P. Tiene 14 pins digitales input/output, 6 analógicos inputs, conexión USB, ICSP y botón de reset.

Podemos descomponer el R3 en diferentes partes:

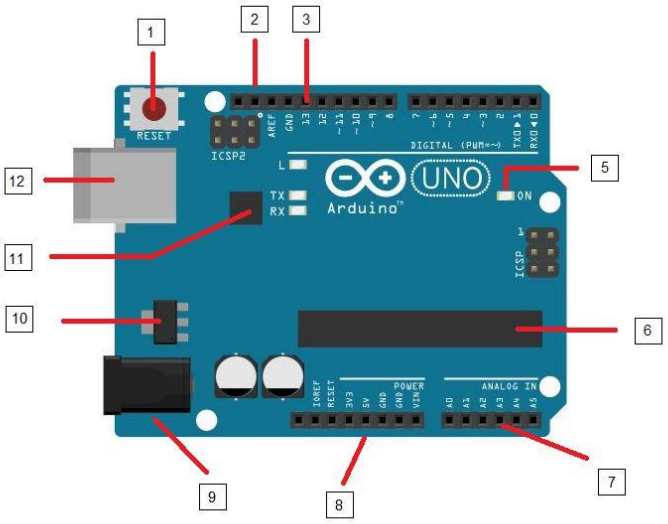


Figura 18. Partes de Arduino Uno.

1. Pulsador de Reset: Sirve para inicializar de nuevo el programa cargado en microcontrolador.

2. Puerto de entrada y salida digital: Son los zócalos donde conectar los sensores y actuadores que necesiten señal digital.

3. Led pin 13: Es un led integrado en la placa para poder ser utilizado en montajes. Corresponde al pin 13.

4. Reloj oscilador: En el caso de Arduino Uno, la frecuencia de funcionamiento es de 16 Mhz.

5. Led de encendido: Se ilumina cuando la placa está correctamente alimentada.

6. Microcontrolador: El cerebro de la placa. Actualmente se monta con un ATmega328 de Atmel, pero se puede encontrar con microprocesadores ATmega8 o ATmega168, en tal caso las conexiones son idénticas.

7. Entradas analógicas: Zócalo con distintos pines de entrada analógica que permiten leer entradas distintas de los 0 y 1 digitales.

8. Zócalo de tensión: Aquí encontramos pines con los que alimentar nuestro circuito y entradas de regencia voltaica.

9. Puerto de corriente continua: Se utiliza para alimentar la placa si no se usa alimentación vía USB.

10. Regulador de tensión: Sirve para controlar la tensión a enviar a los terminales de alimentación. La placa puede ser alimentad tanto por USB como por alimentación externa.

11. Chip de interface USB: Es el encargado de controlar la comunicación con el puerto USB.

12. Puerto USB: Se utiliza tanto para conectar con el ordenador y trasferir los programas al microcontrolador como para dar electricidad a la placa. También se puede usar como puerto de transferencia serie hacia la placa, tanto para transmisión como para recepción.

La placa Uno puede ser encendida a través de la conexión USB o con una fuente externa. El encendido es seleccionado automáticamente.

La alimentación externa (no USB) puede venir desde un adaptador AC-DC (wall-wart) o desde una batería. Los cables de las baterías pueden conectarse en los punes Gnd y Vin.

La placa puede operar con un suministro externo de 6 a 20 voltios. Si es suministrada con menos de 7V, la placa podría ser inestable ya que el pin de 5V podría dar menos. Si usamos más de 12V, el regulador de tensión puede sobrecalentarse y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son:

- **Vin.** La entra de tensión a la placa cuando está usando una fuente de alimentación externa (al contrario de los 5V de la conexión USB).
- **5V.** La salida de este pin regula 5V. La placa puede ser alimentada con el DC (7-12V), conexión USB (5V), Vin (7-12V)
- **3V3.** 3.3V generados por el regulador de la placa. La corriente máxima es de 50 mA.
- **GND.** Pins de tierra.
- **IOREF.** Este pin de la placa Arduino proporciona la referencia de tensión con la que opera el microcontrolador. Un escudo correctamente configurado puede leer el voltaje de la clavija IOREF y seleccionar la fuente de alimentación apropiada.

Cada uno de los pines digitales de Uno pueden ser usados como entrada o salida, usando funciones *pinMode(), digitalWrite(), digitalRead()*. Operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40mA y tiene una resistencia interna «pull-up» (desconecta por defecto) de 20-50K Ohms.

Algunos pines tienen funciones especiales:

- **Serial: 0 (Rx) y 1 (Tx).** Usados para recibir (Rx) y transmitir (Tx) datos TTL. Estos pines están conectados a los pines correspondientes del ATmega8U2 USB-to-TTL.
- **Interruptores externos: 2 y 3.** Estos pines pueden ser configurados para disparar un interruptor en un valor bajo, un margen creciente o decreciente, o un cambio de valor. (*attachInterrupt()*).
- **PWM: 3, 5, 6, 9, 10 y 11.** Proporcionan salida PWM de 8 bits con la función *analogWrite()*.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13(SCK).** Estos pines soportan comunicación SPI usando dicha biblioteca.
- **LED: 13.** Hay un LED empotrado conectado al pin digital 13. Cuando el pin está a valor HIGH, el LED este encendido, cuando el pin está LOW, está apagado.
- **TWI: A4 o SDA y A5 o SCL.** Soporta comunicación I^2C usando la biblioteca Wire. (Para este proyecto estos serán los pines utilizados).
- **AREF:** Voltaje de referencia para las entradas analógicas. Usando *analogReference()*.

La placa Uno tiene 6 entradas analógicas, cada una de las cuales proporcionan 10 bits de resolución (por ejemplo 1024 valores diferentes). Por defecto miden 5 voltios desde tierra, aunque es posible cambiar el valor más alto de su rango usando el pin ARF y algún código de bajo nivel.

4.4.1 Comunicación Arduino.

Arduino Uno tiene una serie de instalaciones para comunicarse con un ordenador, otra placa de Arduino y otros microcontroladores. El ATmega328 proporcionan una comunicación serie UART TTL (5V), que está disponible en los pines digitales 0 (Rx) y 1 (Tx). El ATmega 16U2 en la placa canaliza la comunicación en serie al USB proporcionan un puerto de comunicación virtual al software del ordenador. El 16U2 utiliza los controladores USB COM estándar y no necesita ningún controlador externo. Sin embargo, en Windows, se requiere un archivo .inf.

Una biblioteca *SoftwareSerial* permiete la comunicación serie en cualquiera de los pines digitales.

El ATmega328 suporta comunicación I2C (TWI) y SPI. Arduino incluye una biblioteca Wire para simplificar el uso del bus I2C.

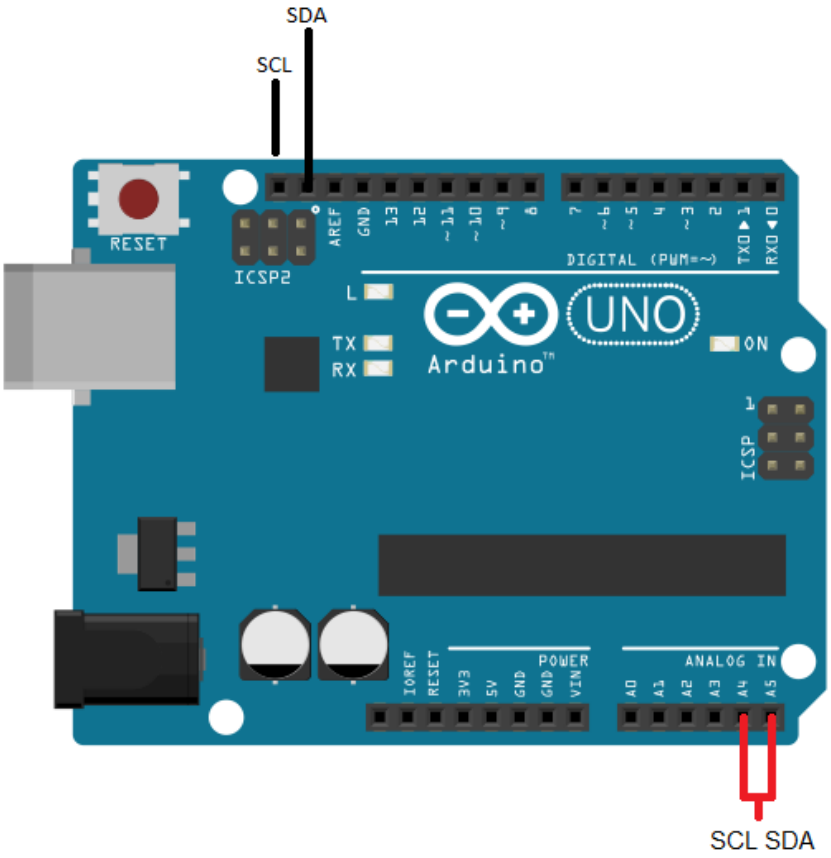


Figura 19. Conexiones SCL y SDA

Esta biblioteca permite comunicarse con los dispositivos I2C (TWI). En la placa Arduino Uno R3, SDA (data line) y SCL (clock line), pero también se pueden utilizar los pines A4 (SDA) y A5 (SCL) para establecer dicha comunicación.

Las funciones son:

- **begin():** Wire.begin(). Inicia la biblioteca de Wire y se uno a la comunicación I2C como maestro o esclavo.
- **requestFrom():** Wire.requestFrom(). Utilizado por el maestro para solicitar bytes desde un dispositivo esclavo.
- **beginTransmission():** Wire.beginTransmission(address). Comenzará una transmisión al I2C esclavo con la dirección dada.
- **endTramission():** Wire.endTransmission(). Termina una transmisión a un dispositivo esclavo que se inició por beginTransmission () y transmite los bytes que se han puesto en cola de write().
- **write():** Wire.write(value). Escribe los datos en un dispositivo esclavo en respuesta a una petición de un maestro, o colas de bytes para la transmisión de un maestro al dispositivo esclavo

- **available():** Wire.available(). Devuelve el número de bytes disponibles para su recuperación con read().
- **read():** Wire.read(). Lee un byte que se ha transmitido desde un dispositivo esclavo de un maestro después de llamar a requestFrom () o se transmite de un maestro a un esclavo.
- **onRecive():** Wire.onReceive(handler). Registra una función que se llamará cuando un dispositivo esclavo recibe una transmisión de un maestro.
- **onRequest():** Wire.onRequest(handler). Registrar una función que se llamará cuando se solicite datos maestros de este dispositivo esclavo.

4.4.2 Entorno de Arduino.

Se procederá a describir los diferentes aspectos que presenta la interfaz de Arduino.[15]

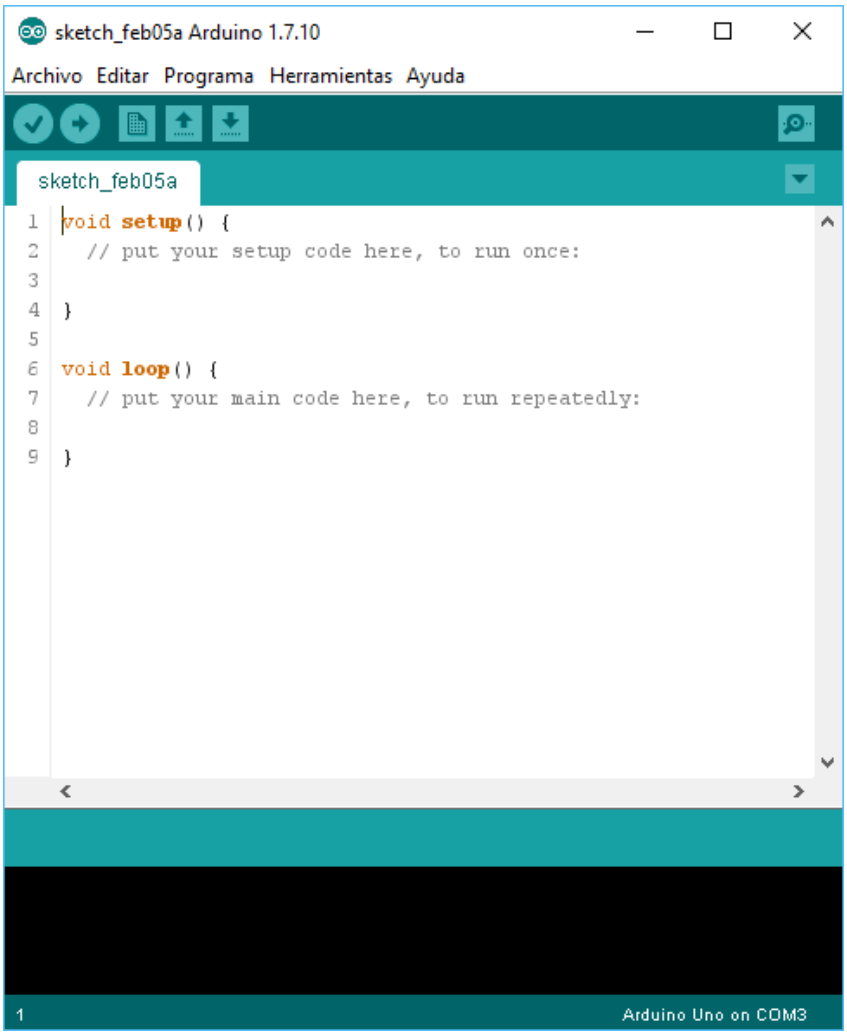


Figura 20. Interfaz Arduino IDE.

En la Figura 20, se pueden distinguir varias secciones. En primer lugar, la barra de opciones, donde se podrá crear un nuevo *sketch*, se podrá seleccionar la tarjeta que se va a programar o acceder al menú de ayuda para consultas o dudas que nos surjan durante

la programación. En la sección intermedia se encuentra la zona en donde se escribirá el programa que se va a introducir a la placa de Arduino y, justo debajo, una sección en negro donde se obtendrá información acerca de la compilación del programa, los errores que puedan surgir, o la memoria que ocupará el programa en la memoria del microcontrolador. A continuación, se describen cada una de estas secciones.

En la primera sección, el menú de opciones, se encontrarán todas las herramientas necesarias para poder comenzar con la programación de nuestro *sketch*.

Desde la utilización de ejemplos hasta la selección de placas que podemos programar con la herramienta.

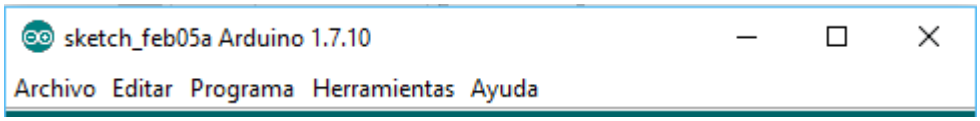


Figura 21. Menú de opciones de la herramienta.

Dentro de **Archivo**, encontraremos las siguientes opciones:

- Nuevo: Nos permitirá generar un nuevo *sketch* en blanco para programar.
- Abrir: Abrirá un explorador para que se pueda seleccionar algún *sketch* que ya se haya generado previamente.
- Abrir reciente: Se despliega una pequeña ventana donde aparecerán los programas que se han utilizado últimamente.
- Proyectos: Se despliega una pestaña en donde aparecerán los últimos proyectos utilizados.
- Ejemplos: Despliega una pestaña en la que se encuentran todos los ejemplos que Arduino proporciona para facilitar la programación de determinados componentes o para saber cómo se realizan la programación de determinados sensores.
- Cerrar: Cerrará la ventana del *sketch*, preguntándonos primero si queremos guardar los cambios realizados en caso de haber hecho alguno.
- Salvar: Guardará el *sketch* con el nombre que le hemos dado en el primer guardado que hayamos hecho del mismo. En caso de no haberlo guardado nunca, nos abrirá el explorador de archivos preguntándonos dónde lo queremos guardar y con qué nombre.
- Guardar como: Abrirá directamente el explorador de archivos para que le ubiquemos dónde queremos guardar el *sketch* y el nombre que le queremos dar.

- Configurar pantalla: Como su propio nombre indica, sirve para configurar el tamaño de la página en donde estamos escribiendo el programa.
- Imprimir: Imprime el código que hemos generado.
- Preferencias: Nos guía a la pantalla de preferencias donde configurar el comportamiento del editor.
- Salir: Cierra el entorno de programación.

En la pestaña **Editar** tenemos la siguiente información:

- Deshacer: La escritura del código retrocede un paso.
- Rehacer: La escritura del código avanza un paso. Es decir, si hemos retrocedido 2 o 3 pasos en la escritura, solo podremos rehacer el mismo número de pasos.
- Cortar, Copiar, Pegar, Seleccionar todo, …: Son los comandos que podemos encontrar en cualquier editor de textos.
- Copiar al foro: Copia el programa que se está escribiendo en el portafolios en un formato preparado para que pueda ser pegado en los comentarios de un hilo del foro.
- Copiar como HTML: Del mismo modo que copia el código en el formato adecuado para el foro, esta opción lo hace en formato HTML.
- Ir a línea: Nos lleva a la línea que le indiquemos.
- Comentar / Descomentar: Cuando se está construyendo un programa en cualquier lenguaje de programación, se pueden escribir notas aclarativas de tal forma que se pueda entender mejor el código y no sea un problema a la hora de realizar la compilación del mismo. Esta opción permite, como su propio nombre indica, comentar o descomentar determinadas líneas del código para que no dé fallo en su compilación, o por si queremos eliminar alguna de las sentencias que forman parte del programa porque en ese momento no necesitamos que se ejecute su funcionalidad.

En la pestaña **Programa** se tienen las siguientes opciones:

- Verificar / Compilar: Esta opción hará que el entorno compruebe que la sintaxis del código es la correcta y que no haya fallos en la estructura del mismo. En caso de encontrar algún error, será notificado en la tercera sección del entorno (el rectángulo negro), indicando la sentencia que está fallando y la línea en la que se encuentra. Si todo ha ido bien nos dará un mensaje de “compilado”.
- Mostrar Carpeta de Programa: Abre el explorador de archivos en la ubicación donde se encuentra el *sketch* que estamos preparando.

- Incluir librería: Permite generar una línea en nuestro código que empezará por la palabra *#include* y vendrá seguida de la librería que se va a utilizar.
- Añadir fichero: Permite añadir un fichero que no forme parte el proyecto al programa.

En la pestaña **Herramienta** se despliegan las siguientes opciones:

- Auto Formato: Reestructura el programa aplicando la sangría correspondiente en cada una de las líneas del código en función de su ubicación en el mismo.
- Monitor serie: Este monitor es una herramienta muy importante sobre todo cuando se trata de depurar el programa. El monitor muestra los datos enviados por el puerto serie o USB desde Arduino y también permite el envío de datos hacia la placa.
- Placa: Despliega un listado de las placas disponibles en la herramienta. Además de las que vienen incluidas, se pueden añadir nuevas placas, ya sean desarrolladas por la comunidad o por el propio fabricante.
- Puerto: Indica el puerto al que nos estamos conectando con la herramienta. Aquí se debe seleccionar el puerto al que está conectada la placa.
- Quemar bootloader: Permite cargar el *bootloader* a la placa. Este es un pequeño programa que ha sido cargado previamente en el microcontrolador de la placa y que nos permite cargar código sin necesidad de hardware adicional. El *bootloader* solo está activo unos segundos cuando se resetea el Arduino y después comienza el *sketch* que está cargado en la *flash* de Arduino.
- Ayuda: Podemos encontrar varias entradas con referencias a ayudas y más información sobre el entorno de programación o el mundo de Arduino.

La configuración del entorno se realiza mediante el menú **Archivo>Preferencias**, que nos dirige a la ventada de preferencias. En esta ventana se puede configurar valores que afectan al aspecto del entorno de desarrollo como el tamaño de letra a utilizar o el idioma y otros valores que afectan más de modo funcional, como la asociación de la extensión *.ino* este programa o el directorio sobre el fichero *preferences .txt* dentro del directorio de instalación del entorno y pudiendo accederse a él mediante el enlace proporcionado en la parte inferior de la ventana de preferencias. En caso de modificar manualmente este fichero es muy recomendable hacerlo cuando el entorno de programación no se esté ejecutando y hacer previamente una copia de seguridad.

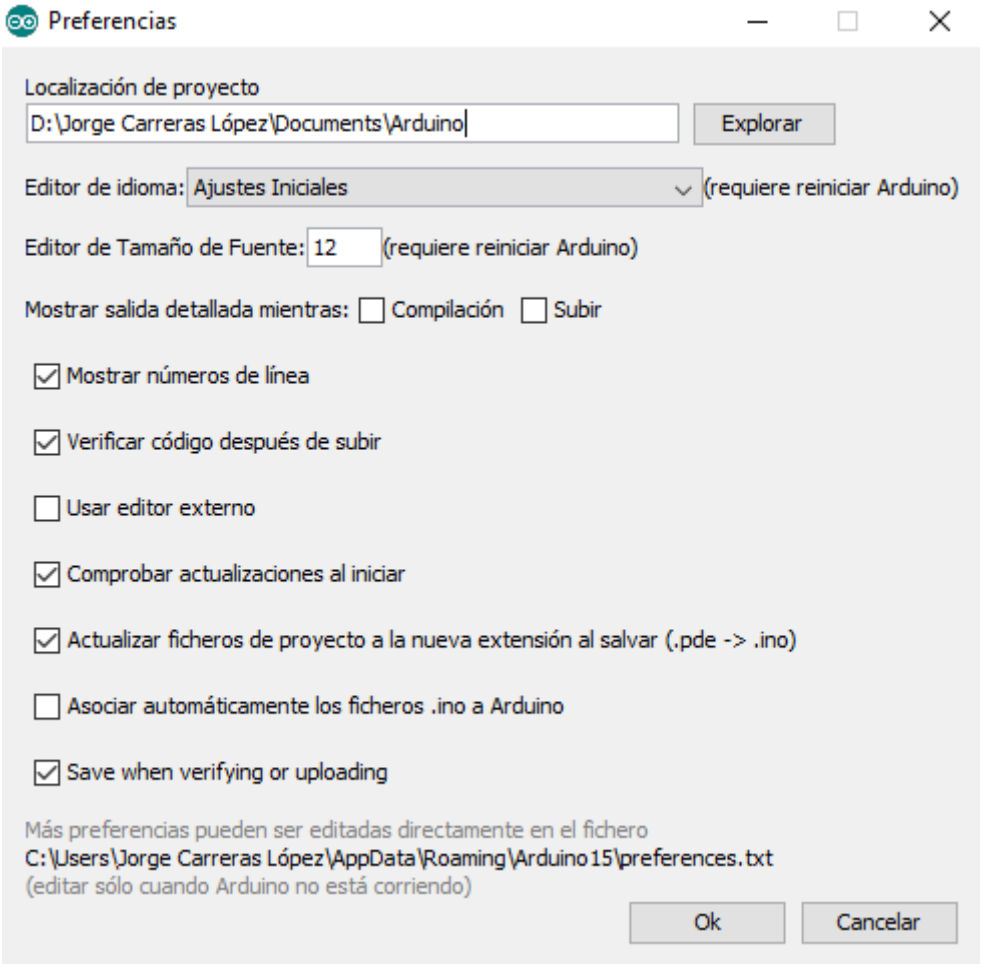


Figura 22. Ventana de preferencias.

En la parte central es donde se realizará la codificación del programa. El entorno Arduino no ofrece tantas ayudas como otros entornos de programación, como puede ser funciones de autocompletado, por ejemplo, pero sí ofrece la posibilidad de visualizar la sintaxis por colores, pudiendo hacer un mejor seguimiento de la estructura con la que estamos programando nuestro *sketch*.

Por último, se definirán los botones que se encuentran justo debajo de la barra de herramientas. Son botones a los que se puede acceder desde dicha barra, pero se sitúan más visibles ya que su uso es más frecuente.

Figura 23. Botonera de acceso rápido.

De izquierda a derecha tenemos lo siguiente:

- Verificar: Realiza la compilación del programa.
- Subir: Carga el programa en la placa Arduino.
- Nuevo: Genera una nueva ventana en blanco para crear un nuevo *sketch*

- Abrir: Abre un fichero previamente creado.
- Salvar: Guarda el progreso que se haya realizado en la programación del *sketch.*

5 MATERIALES Y MÉTODOS

En este apartado realizará una descripción de los equipos y herramientas informáticas utilizadas para la realización de este Trabajo Fin de Grado, donde se describirá cual es la función de cada uno.

5.1 Equipos utilizados.

En este subapartado nos centramos en especificar las características de los equipos utilizados.

5.1.1 Brazo robótico IRB-120.

Se trata de un Robot IRB 120 de la compañía ABB, con 6 ejes de movimiento, con una carga máxima de 3 kg y un alcance máximo de 580mm. [16]

Figura 24. Brazo robótico ABB IRB 120. [17]

Entre sus características principales encontramos:

- Suministro de señal integrado: 10 señales en la muñeca.
- Suministro de aire integrado: 4 de aire en la muñeca (5 bar).
- Repetibilidad de posición: 0.01mm.
- Grado de protección: IP30.
- Controlador: IRC5 Compact.

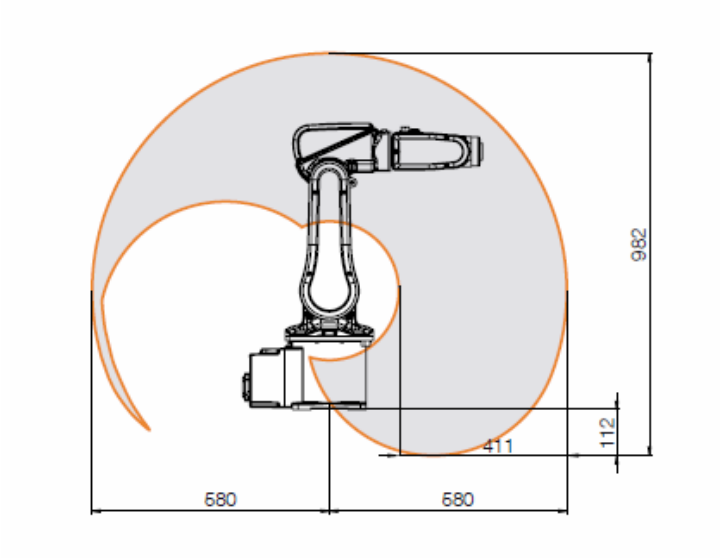


Figura 25. Rango de movimiento del IRB120. [16]

Los movimientos que puede realizar el brazo robótico son:

- Eje 1 Rotación: +165º a -165º, velocidad máxima 250 º/s
- Eje 2 Brazo: +110º a -110º, velocidad máxima 250 º/s
- Eje 3 Brazo: +70º a -110º, velocidad máxima 250 º/s
- Eje 4 Muñeca: +160º a -160º, velocidad máxima 320 º/s
- Eje 5 Recodo: +120º a -120, velocidad máxima 320 º/s
- Eje 6 Giro: +400º a -400º, velocidad máxima 420 º/s

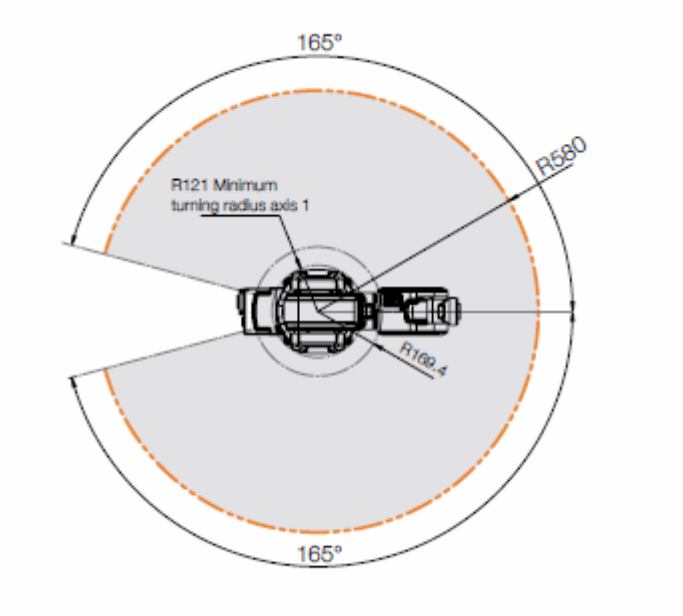


Figura 26. Rango del IRB120. [16]

Conexión eléctrica:

- Suministro de voltaje: 200-600 V, 50/60 Hz
- Potencia nominal: 3 kVA
- Consumo: 0.25kW

Dimensiones:

- Base: 180 x 180 mm
- Altura: 700 mm
- Peso: 25 kg

5.1.2 *Controlador IRC5 Compact.*

El controlador IRC5 es el sistema que controla y gestiona el robot.

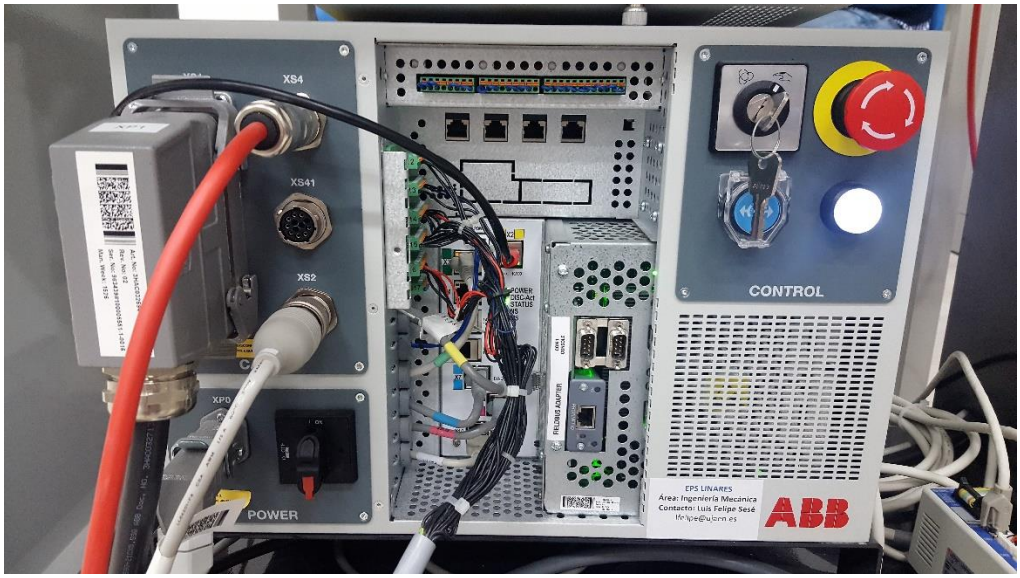


Figura 27. Armario compacto IRC5.

El módulo controlador IRC5 permite la gestión del mismo robot y contiene todos los elementos electrónicos de control como el ordenador principal, las tarjetas de E/S y la unidad de almacenamiento. El módulo de accionamiento contiene todos los elementos

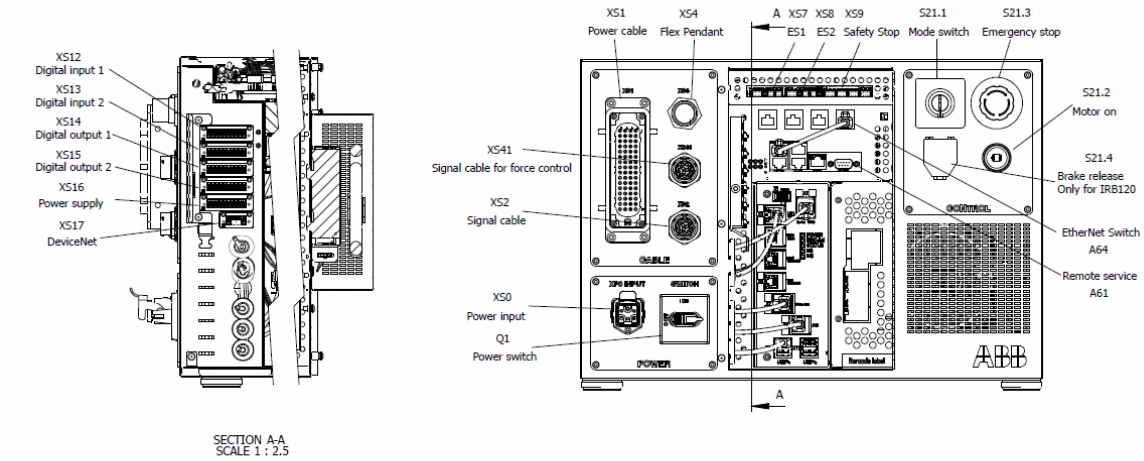


Figura 28. Esquema eléctrico IRC5. [7]

electrónicos de alimentación que proporcionan la alimentación a los motores del robot. Un módulo de accionamiento IRC5 puede contener los accionamientos de los seis ejes del robot y además dos o tres accionamientos para los ejes externos en función del modelo de robot. En el controlador IRC5 Compacto, el módulo de control y el de accionamientos están integrados en un solo armario.

5.1.3 FlexPedant.

O también llamado unidad de programación, es un dispositivo que maneja muchas funciones del uso del sistema de robot, como ejecutar programas, mover el manipulador, crear y editar programas, etc.

Se compone de elementos de hardware, botones, joysticks y de software. Se conecta al módulo de controlador a través de un cable con conector integrado.

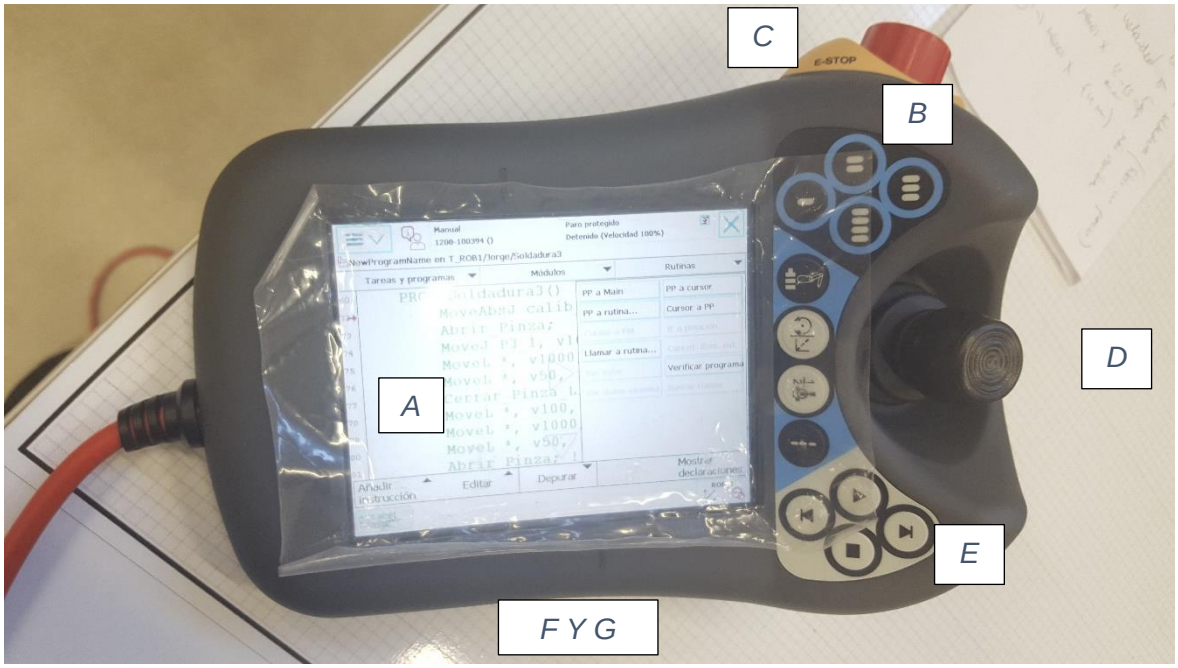


Figura 29. FlexPedant.

Tabla 1. Descripción de FlexPedant.

Pos	Descripción
A	Pantalla
B	Teclas programables
C	Botón de paro de emergencia
D	Joystick
E	Teclas de ejecución de programas
F	Conexión de memoria USB
G	Alojamiento de puntero

Características:

- Pantalla táctil: Muestra los textos e información gráfica. Es posible tener abiertas varias ventanas a la vez.
- Teclas de ejecución de programas: Teclas para iniciar y detener programas y ejecutarlos paso a paso hacia delante o hacía atrás.
- Teclas programables: Cuatro teclas definidas por el usuario que pueden configurarse.
- Dispositivo de habilitación: Durante el modo manual, será necesario presionar hasta la mitad de su recorrido para poner el sistema como los motores encendidos.
- Joystick: Joystick tridimensional que se utiliza para mover el robot manualmente.
- Boto de paro de emergencia.

5.1.4 Pinza eléctrica SMC.

La herramienta terminal que dispone el brazo robótico es una pinza eléctrica de la marca SMS modelo LEHZ25K2-14-R16P1. [18]

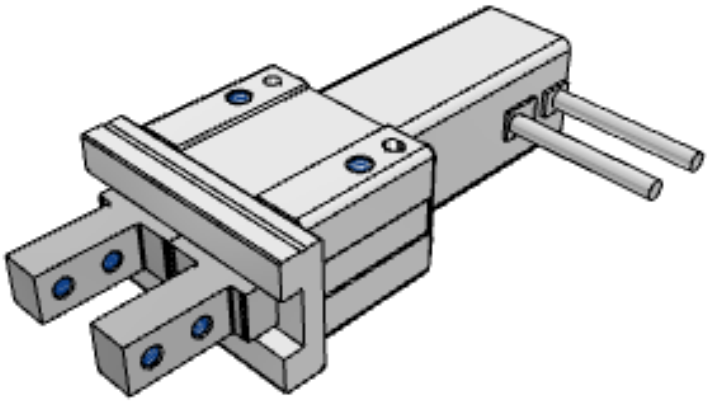


Figura 30. Modelo 3D Pinza eléctrica. [18]

Las especificaciones del producto son las siguientes:

Tabla 2. Especificaciones del modelo LEHZ25K2-14-R16P1.

Tamaño del cuerpo	25 mm
Tamaño Motor	Estandar
Carrera	14 mm
Entrada del cable del motor	Básico, entrada en lado izquierdo
Tipo de cable del motor	R (Flexible)
Tipo de controlador	6P
Longitud del cable E/S	1.5m
Montaje del controlador	Montaggio con viti

5.1.5 MPU-6050

El MPU6050 es una IMU (dispositivo capaz de mesurar la fuerza y la velocidad) que dispone de seis grados de libertad, tres grados de libertad del acelerómetro y tres grados de libertad para el giróscopo. [19]

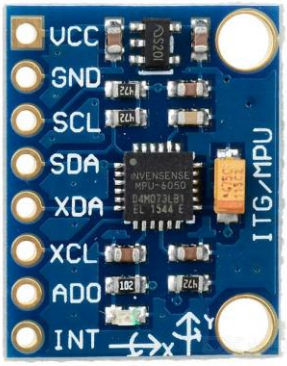


Figura 31. MPU6050.

Características del sensor:

Alimentación:

- Tensión de alimentación mínima de 2.5 V
- Tensión de alimentación máxima de 3.6 V

Giróscopo:

- Rango de escala programable para sensibilidades de ± 250, ± 500, ± 1000, ± 2000 º/s.
- Sensibilidad al giro de 131, 65.5, 32.8, 16.4 º/s.
- Calibración por software.
- Conversor analógico-digital.
- Corriente de funcionamiento de 3,6 mA.

Acelerómetro:

- Rango de escala programable para sensibilidades de ± 2, ±4, ±8 y ± 16 G.
- Sensibilidad a la aceleración de 16284, 8192, 4096, 2048 LSB/g.

Otras:

- Filtros digitales programables por el usuario para giroscopios, acelerómetros y sensor de temperatura.
- Sensor de temperatura digital.
- MotionFusion de 9 ejes mediante el procesador de movimiento digital (DPM) en chip
- Bus I2C maestro auxiliar para lecturas de datos de sensores externos.
- Sensibilidad transversal mínima entre los ejes del acelerómetro y el giroscopio.
- Estructura MEMS sellada herméticamente y adherida a nivel de la oblea.
- Resistencia a impactos de 10.000g
- I2C de modo rápido de 400kHz para comunicarse con todos los registros.

5.1.6 ADXL-345

El ADXL345 es un acelerómetro digital de consumo bajo, este mide la aceleración dinámica resultante de un movimiento o choque. El sensor incorpora un bloque de memoria FIFO (first-in/first-out)[1] en un chip que almacena hasta 32 conjuntos de datos X, Y, Z. [20]

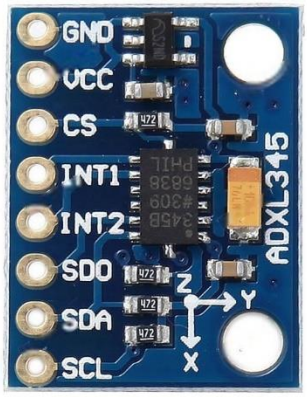


Figura 32. ADXL345.

Características del sensor:

- Rango de escalas programable ± 2, ±4, ±8 y ± 16 G.
- Tecnología FIFO integrada que minimiza la carga del procesador del host.

[1]«Primero en entrar, primero en salir» (en inglés first in, first out o FIFO) es un concepto utilizado en estructuras de datos, contabilidad de costes y teoría de colas. Guarda analogía con las personas que esperan en una cola y van siendo atendidas en el orden en que llegaron, es decir, que "la primera persona que entra es la primera persona que sale".

- Monitoreo de actividad / inactividad.
- Detección de caída libre.
- Tensión de alimentación mínima 2 V.
- Tensión de alimentación máxima 3.6 V.

5.1.7 Arduino UNO R3.

El Arduino UNO va a ser el encargado de registrar los datos proporcionados por los sensores y “traducirlos” realizando una serie de cálculos para al final dar los datos que leerá MatLab.

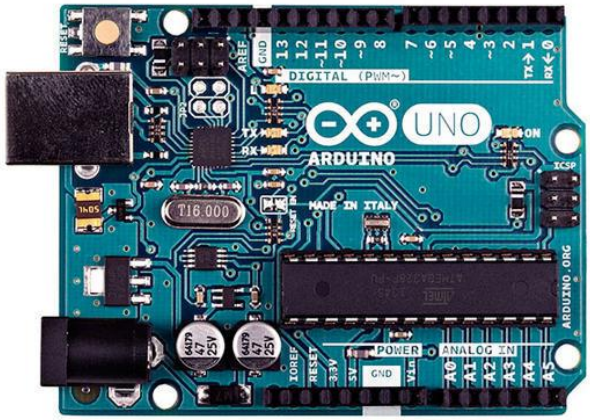


Figura 33.Arduino UNO R3.

Microcontrolador	ATmega328
Voltaje de operación	5V
Voltaje Entrada (Recomendada)	7-12V
Voltaje Entrada (Límite)	6-20V
Pins Digitales I/O	14 (6 PWM output)
PWM Pins I/O	6
Pins Analógicos entrada	6
DC I/O Pin	20 mA
DC 3.3V Pin	50 mA
Memoria Flash	32 KB
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Velocidad del reloj	16 MHz
LED_BUILTIN	13

Longitud	68.6 mm
Ancho	53.4 mm
Peso	25 g

Tabla 3. Datos técnicos Placa Arduino UNO R3 [7]

5.1.8 Otros

Con el objetivo de profundizar en la gestión de entradas y salidas del sistema de control se decidió simular un proceso de soldadura robotizada. Para ello se empleó un láser activado según programación. El controlador subministra una salida a 24V, mientras que el láser utilizado funciona a 3,3V, por lo que era necesario un regulador de tensión. El regulador utilizado es el mostrado en la Figura 34.

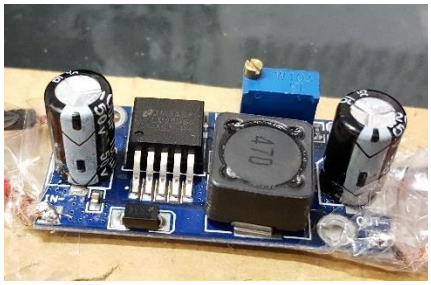


Figura 34. Regulador de tensión 24 a 3,3 V.

El láser utilizado es el modelo RYS1230, se trata de un láser de color rojo, con una longitud de onda de 650nm.

Figura 35. Láser.

5.2 Adquisición de datos.

La adquisición de datos se compone de dos fases:

1. Toma de datos de los sensores a través de Arduino.
2. Lectura de los datos a través de Matlab.

Para la obtención de los datos de los sensores el primer paso a seguir es de la colocación de estos en sus respectivas posiciones ya que la GUI diseñada para una

colocación predeterminada. A continuación, se mostrará la posición de cada sensor en figuras. Los sensores están colocados acorde a las direcciones de sus ejes, para que siempre el efecto de la gravedad quede en el eje Z. Esto es muy importante ya que los programas de calibración de cada sensor están pensado con ese objetivo.

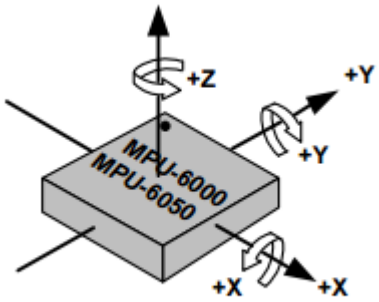


Figura 36.Orientación de los ejes.

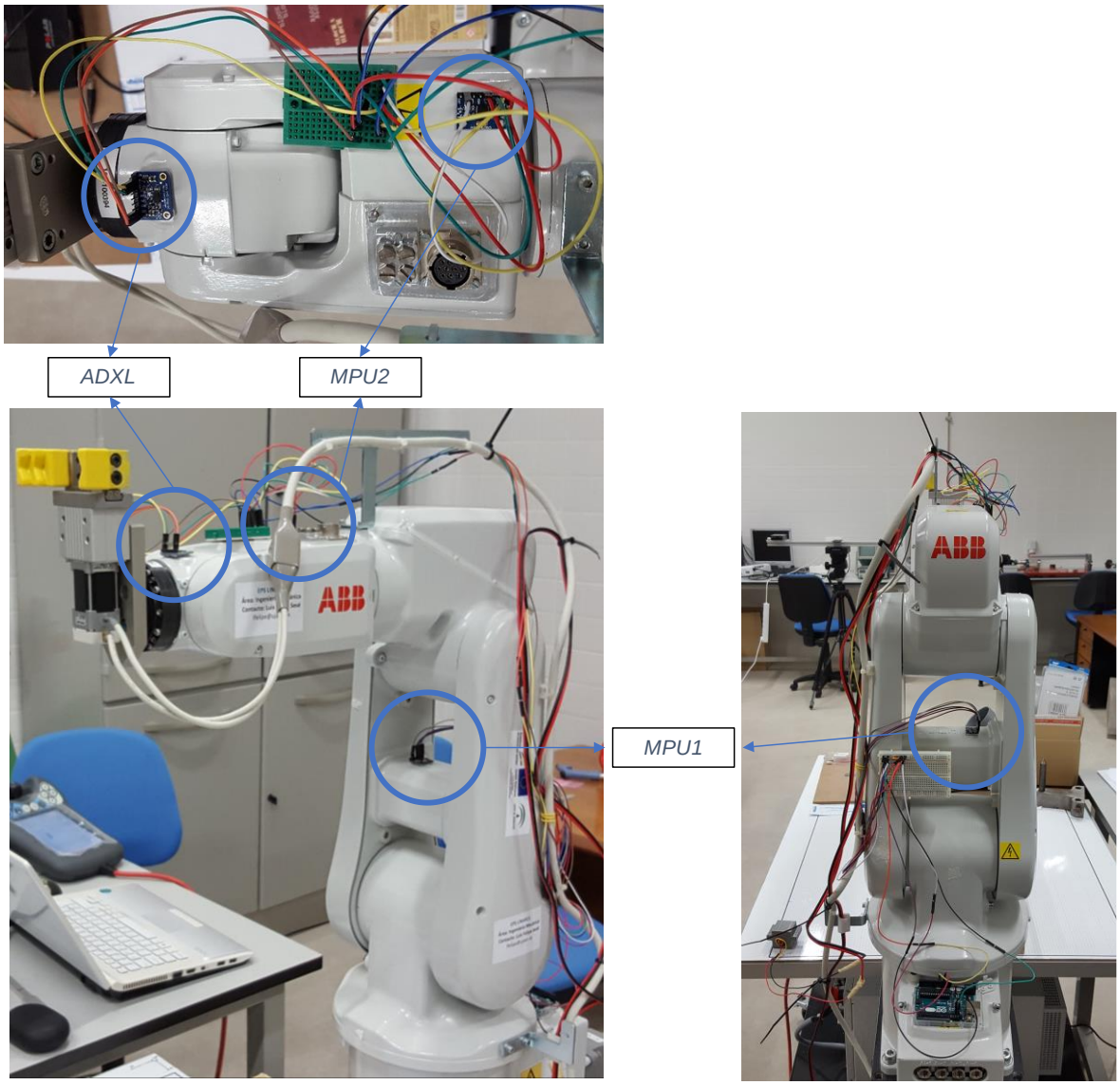


Figura 37. Colocación de los sensores ADXL y MPU.

Observe que tenemos dos sensores MPU6050 y otro ADXL345, para que Arduino reconozca los dos MPU6050 tendremos que diferenciarlos, creando dos conexiones I2C, para ello utilizaremos la conexión ADO (Figura 38). Para el caso del MPU1 la conexión ADO debe ir a tierra (Ground) y el sensor tomará el valor de 0x68 y para el MPU2 irá a fuente de alimentación (Vcc) y el sensor parará a 0x69. Como solo tenemos un único sensor ADXL345, no será necesario utilizar la conexión ADO.

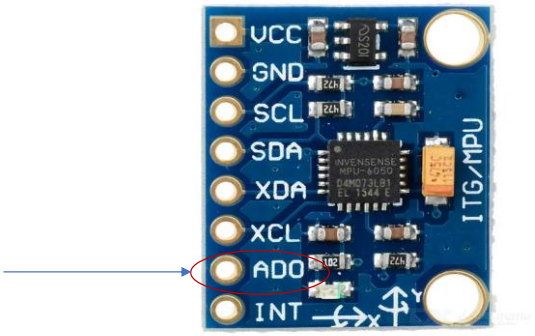


Figura 38. Conexión ADO

En las tablas podremos ver de forma detallada que conexión es necesaria para cada sensor.

Para el caso del MPU-6050 los pines utilizados son:

MPU-6050	**Arduino UNO**
Vcc	Vcc
GND	GND
SDA	SDA / Pin Análogo 4
SCL	SCL / Pin Análogo 5
ADO (0x68)	Ground
ADO (0x69)	Vcc

Para el ADXL 345 los pines utilizados son:

ADXL 345	**Arduino UNO**
Vcc	Vcc
GND	GND
SDA	SDA / Pin Análogo 4
SCL	SCL / Pin Análogo 5

De forma que en el código de Arduino quedaría implementado de esta forma:

```
MPU6050 sensor1 (0x68);
MPU6050 sensor2 (0x69);
ADXL345 accel;
```

En la siguiente figura podemos ver las conexiones que hay que realizar de una forma más gráfica. En la Figura 39 diferenciamos los dos MPU6050 de color azul, el ADX345 de rojo. Para realizar las conexiones necesarias utilizamos una placa (Board). Aunque para la realización de este trabajo ha sido necesario utilizar dos ya que facilitaba las conexiones.

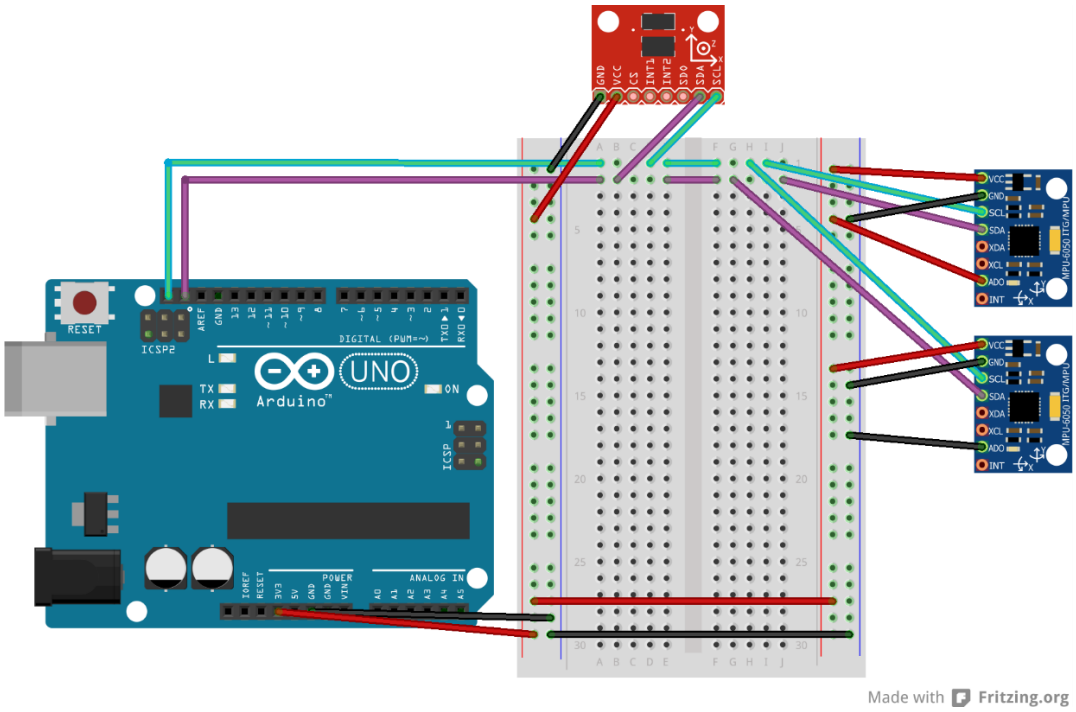


Figura 39. Conexión sensores con Arduino.

Una vez realizada las conexiones, podremos conectar el Arduino a nuestro ordenador a través del cable USB. Pero para que el programa funciones tendremos que tener instalado las librerías necesarias.

Las librerías son colecciones de código que facilitan la interconexión de sensores, pantallas, módulos electrónicos, etc. El entorno de Arduino ya incluye algunas librerías que facilitan, por ejemplo, mostrar texto en pantallas LCD.

Estas normalmente incluyen:

- Un archivo .cpp (código).
- Un archivo .h (contiene las propiedades y métodos o funciones de la librería)
- Un archivo Keyword.txt (palabra que resalta en el IDE).

- Un directorio con ejemplos que ayudara a entender la librería.

Las librerías utilizadas para la ejecución del programa son:

Código:

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
#include "ADXL345lib.h"
#include <ADXL345.h>
```

Estas librerías son proporcionadas por los mismos fabricantes de los sensores.

Los datos que puede mostrar los sensores son datos son valores “RAW” sin procesar, es decir un entero de 16 bits, por lo tanto es necesario modificar ese dato leído para transformarlo por uno de velocidad o aceleración.

Como se ha comentado anteriormente el rango de estos sensores puede ser 2/4/8G al igual que el giróscopo puede cambiar también. En nuestro caso interesa que los valores sean pequeños ya que el brazo robótico no va a adquirir valores muy altos.

La configuración elegida será 2G para el caso de los dos acelerómetros y 250º/s para el giróscopo.

Variable	Mínimo	Central	Máximo
Dato (sinprocesar)	-32768	0	+32767
Aceleración	-2g	0g	+2g
Velocidad angular	-250º/s	0º/s	+250º/s

Para obtener el ángulo, utilizaremos la matriz de rotación de los ángulos de Euler cuya solución son los ángulos de inclinación de un sólido rígido. Primero obtenemos los datos enteros utilizando el comando:

Código:

```
int ax1, ay1, az1;
int gx1, gy1, gz1;
int ax2, ay2, az2;
int gx2, gy2, gz2;
```

Una vez que obtenemos los datos tenemos que procesarlos para la obtención del ángulo. Para ello utilizamos la ecuación 38 pero como son datos sin procesar multiplicaremos por $\frac{180}{\pi}$.

Incorporamos al programa la ecuación 38, para los dos sensores. Código:

```
float accel_ang_y=atan(-ax1/sqrt(pow(ay1,2) + pow(az1,2)))*(180.0/3.14159265359);
float accel_ang_x  = atan2(ay1, az1) * (180.0/3.14159265359);
float accel_ang_x2  = atan2(ay2, az2) * (180.0/3.14159265359);
float accel_ang_y2= atan(-ax2/sqrt(pow(ay2,2) + pow(az2,2))*(180.0/3.14159265359);
```

Estas valdrían para los sensores MPU6050, para el caso del ADXL345 vendría implementado en la librería por lo que únicamente necesitaríamos llamar al comando necesario.

Código:

```
AccelRotation accelRot;
accelRot = accel.readPitchRoll();
```

Para el caso del MPU6050 disponemos también de un giroscopio.

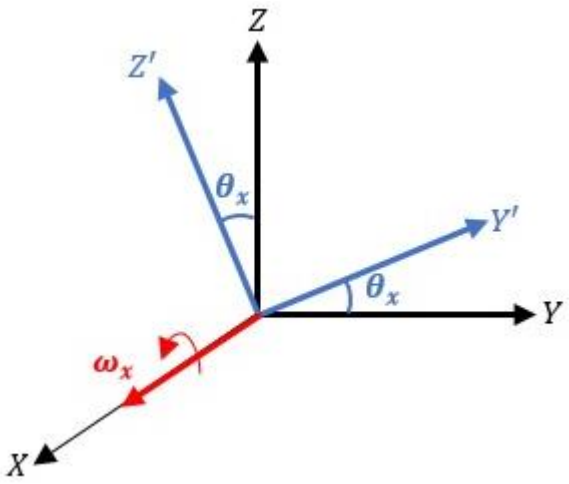


Figura 40. Ángulo de rotación giroscopio

Como se ha comentado anteriormente, un giróscopo mide la velocidad angular respecto a un eje de giro (Figura 40). No se ve influenciado por las vibraciones. Como el resultado que queremos que nos muestre es el ángulo, utilizaremos:

$$\theta_x = \theta_{xo} + \omega_x \Delta t \qquad (40)$$

$$\theta_y = \theta_{yo} + \omega_y \Delta t$$

$$\theta_z = \theta_{zo} + \omega_z \Delta t$$

Para la obtención de los valores de giro del estabón 1 utilizaremos esta ecuación, aunque se obtengan errores. En el apartado de conclusiones se incidirá más el porqué de estos errores.

Código:

```
giro = (gz1/131)*dt + giro_prev;
gz_prev = gz1;
giro_prev = giro;
```

Todos estos ángulos que obtenemos mediante las ecuaciones pueden presentar errores de ruidos, que pueden indicar valores falsos. Para obtener medidas que se ajusten a la realidad necesitamos un filtrado de los datos recogidos por el sensor ya que el sensor puede obtener valores erróneos, para nuestro caso utilizamos el filtro complementario. Este filtro incluye un filtro pasa bajo, el cual no deja pasar cambios rápidos de aceleración, por lo que hace a la medida más exacta. También incluye un filtro pasa altos, este funciona de forma contaría al pasa bajos, eliminando así la deriva del giroscopio.

$$Angulo = C \cdot (Angulo(t-1) + AnguloGyro \cdot dt) + (1 - C) \cdot AnguloAccel \qquad (41)$$

Donde:

- **Angulo:** Variable a calcular.
- **C:** Constante de los filtros
- **Angulo(t-1):** Ángulo calculado en el instante de tiempo anterior.
- **AnguloGyro:** Ángulo de giro calculado únicamente con el giroscopio
- **dt:** Tiempo transcurrido entre muestras
- **AnguloAccel:** Ángulo de giro calculado únicamente con el acelerómetro.

Este tipo de filtrado solo puede ser utilizado por los acelerómetros MPU6050 ya que el ADXL345 no tiene giroscopio.

Código:

```
ang_x = 0.96*(ang_x_prev+(gx1/131)*dt) + 0.04*accel_ang_x;
ang_y = 0.96*(ang_y_prev+(gy1/131)*dt) + 0.04*accel_ang_y;
ang_x_prev=ang_x;
ang_y_prev=ang_y;
ang_x2 = 0.96*(ang_x_prev2+(gx2/131)*dt) + 0.04*accel_ang_x2;
ang_y2 = 0.96*(ang_y_prev2+(gy2/131)*dt) + 0.04*accel_ang_y2;
ang_x_prev2=ang_x2;
ang_y_prev2=ang_y2;
```

5.3 Interfaz gráfica.

La interfaz gráfica o GUI de Matlab es la encargada de mostrar las gráficas y los datos de velocidad y aceleración. A continuación, se procederá a explicar el programa de forma detallada.

5.3.1 *Ventana inicial.*

En esta ventana principal de la GUI de MatLab, encontramos 11 apartados diferenciados, que se procederá a dar su descripción:

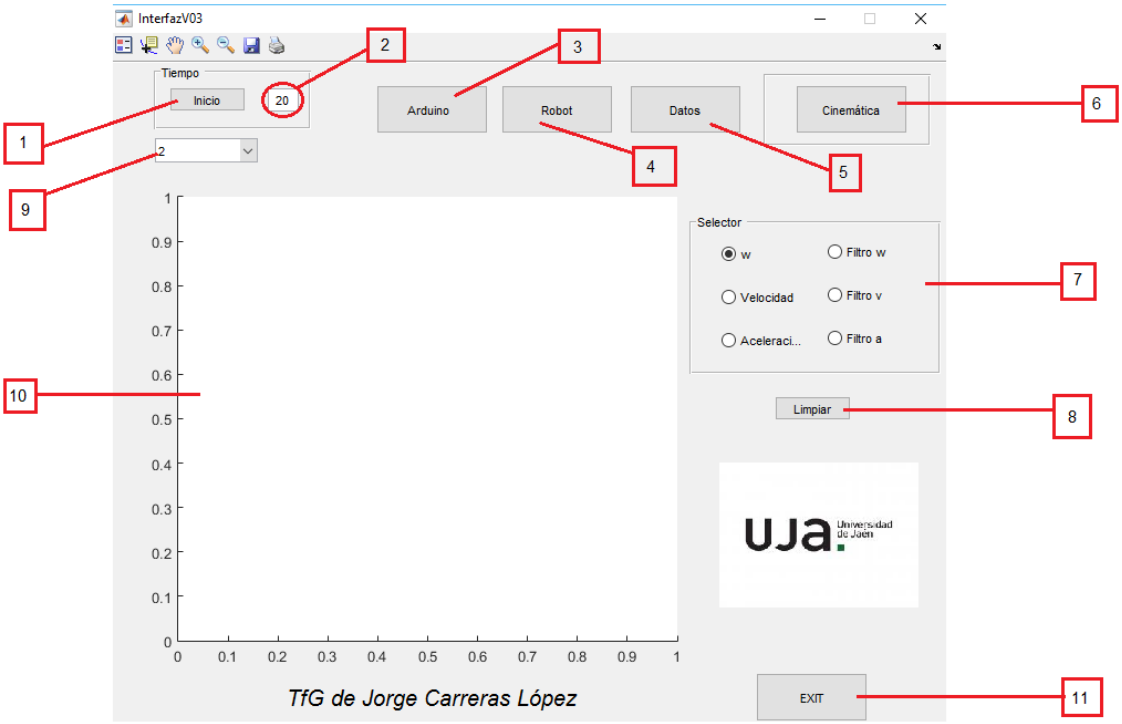


Figura 41. Ventana Inicial GUI.

1. Botón de Inicio, ejecuta el programa.
2. Se podrá seleccionar los segundos que el programa va estar ejecutándose.
3. Abre una nueva ventana, en la que muestra los programas de Arduino necesarios para su calibración y la comunicación del sensor con Matlab.

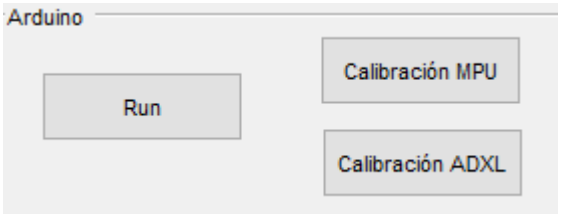


Figura 42. Botón Arduino de la interfaz.

4. Abre una nueva ventana en la que muestra una imagen del brazo donde se muestran los giros del robot.

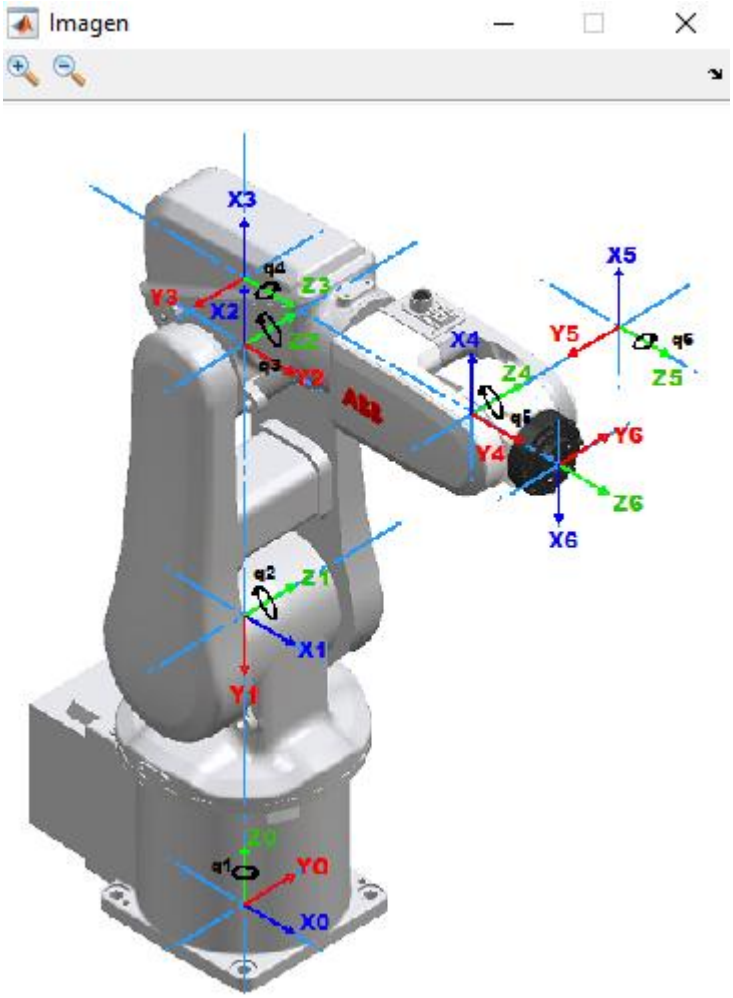


Figura 43.Botón Robot de la Interfaz.

5. El botón de Datos, muestra una tabla donde saldrá el resultado numérico de los datos obtenidos por los sensores. Podremos seleccionar la velocidad angular, la velocidad lineal y la aceleración. También se podrá guardar los datos.

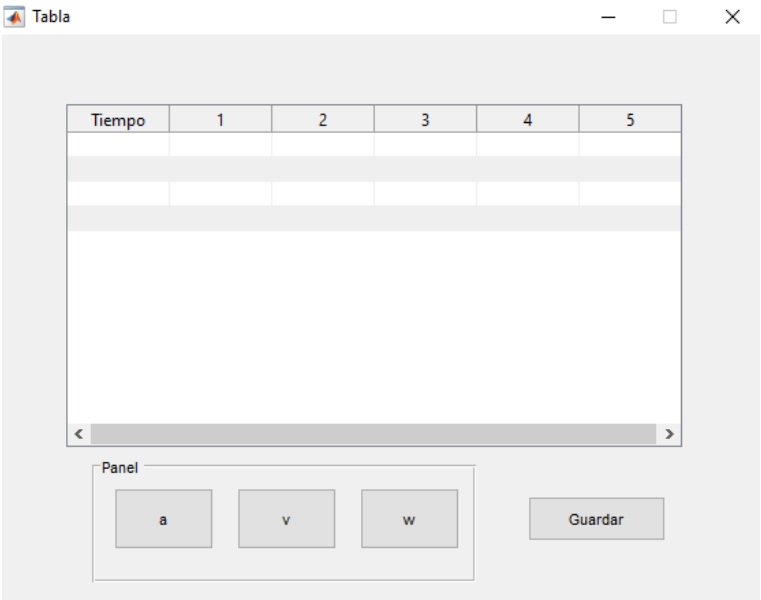


Figura 44. Botón de Datos de la interfaz.

6. En esta ventana muestra la posición del robot respecto a X, Y, Z y los cuaternios.

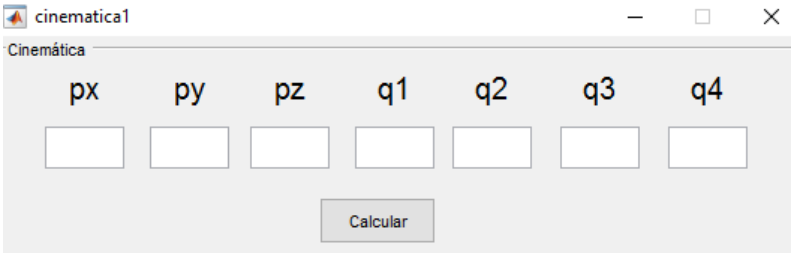


Figura 45. Botón Cinemática de la interfaz.

7. Selector de velocidad angular, velocidad lineal y aceleración, y su respectivo filtrado.
8. Limpia la gráfica.
9. Aquí podremos elegir el eslabón que queremos que nos grafique.
10. Aquí es donde se dibujarán las gráficas.
11. Botón de salida, que cierra el programa y el MatLab.

5.3.2 Conexión Matlab – Arduino.

El encargado de realizar y dibujar las gráficas es MatLab. Por ello hay que exportar los datos en tiempo real de los sensores. Para realizar dicha conexión es necesario en primer lugar eliminar cualquier dato que llegase por el puerto, para ello utilizamos el comando *delete*. A continuación, es necesario asignar el *BaudRate* (tasa de transmisión de símbolos) este debe coincidir con el de Arduino. Creamos una variable, en nuestro caso esta signado con la letra (s), donde incluimos el comando *serial* y el puerto donde esté

conectado Arduino. Para abrir o cerrar la conexión utilizaremos los comandos f*open* o *fclose*.. Código:

```
delete(instrfind({'port'},{'COM3'}));
baudrate = 57600;
s = serial('com3','BaudRate',baudrate);
fopen(s);
```

Los datos que nos proporcionan los sensores están llenos de ruido y de picos. Por lo que es necesario pasarlos por un filtro. El nuevo filtrado será ya con los comandos de MatLab.

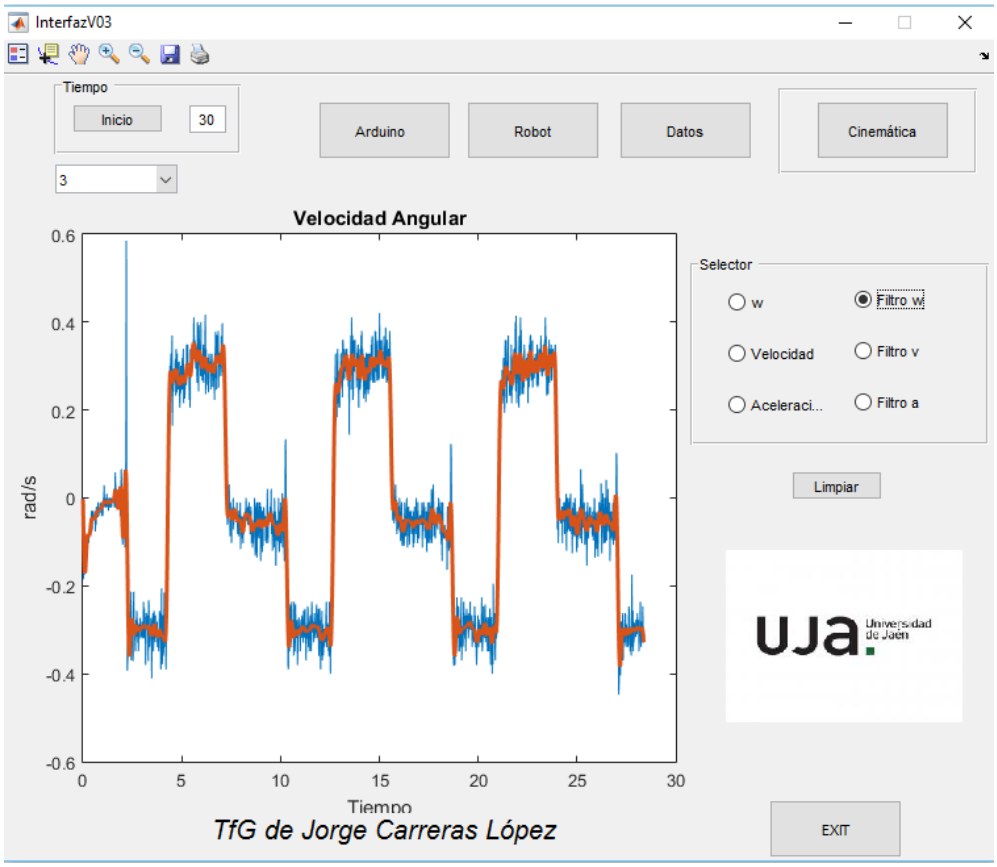


Figura 46. Datos filtrados (roja) y sin filtrar (azul).

Sintaxis:

y = filtfilt(b,a,x)

y = filtfilt(SOS,G,x)

y = filtfilt(d,x)

El filtro utilizado es el filtrado digital de fase cero (*filtfilt)* realiza el filtrado digital mediante los datos de entrada. Por lo tanto, hace dos filtrados. Primero filtra el vector x, y su respuesta la rota y le vuelve aplicar el mismo filtro.

6 RESULTADOS.

En este apartado se presentará de la GUI de MatLab cuya función es mostrar los datos de la velocidad angular, la velocidad línea y la aceleración, de cada uno de sus eslabones.

Se ha realizado un estudio con distintas velocidades programadas, para analizar los resultados de cada eslabón individualmente y comprobar la relación de las medidas reales y las comandadas. La velocidad del robot puede seleccionar con la función (*v)*. Para que el robot se mueva a la velocidad programada, tendremos que activar el modo automático. Para ello giraremos la llave de la Figura 47 en posición automática [A] y pulsaremos el botón de motores [B]. En modo manual, la velocidad máxima es de 250 mm/s.

Figura 47. Configuración modo automático.

El movimiento seleccionado para comparar los datos experimentales y comandados consiste:

- El eslabón girara de una posición de reposo a 0 grados hasta una posición de 50º tras la espera de 5 segundos.
- Seguidamente el eslabón pasará a -40 grados realizando una nueva detención de 5 segundo.
- Por último, el eslabón regresara a la posición de reposo a una velocidad de 100 mms^{-1}.

Aunque la interfaz muestre los datos de velocidad, aceleración y velocidad angular para todos los eslabones, por motivos de validación, este estudio se ha centrado en los eslabones 2 y 3, ya que son los de mayor envergadura.

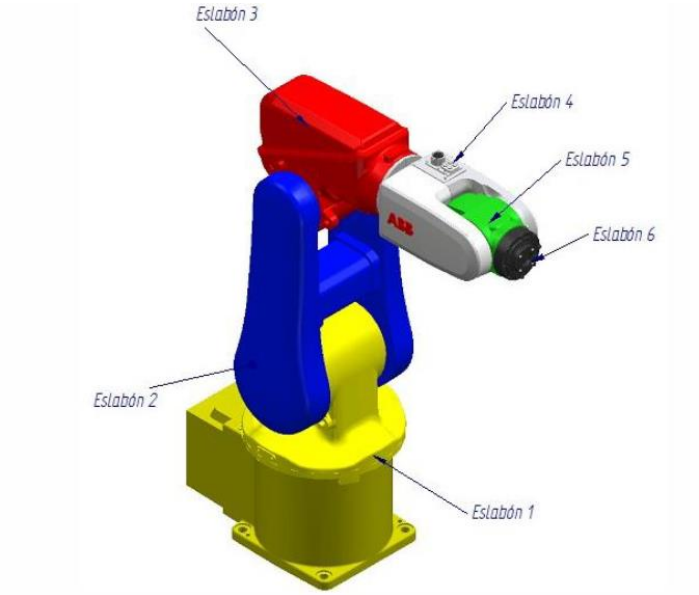


Figura 48. Eslabones del brazo robótico [8]

El sistema de programación del robot tan sólo permite comandar la velocidad del punto de la herramienta del mismo (TCP). De hecho, ese es uno de los motivos por lo que se realiza este estudio, conocer las velocidades de los eslabones para tener una velocidad TCP dada. Es por esto por lo que esta velocidad experimental habrá que calcularla en función de las velocidades de cada uno de los eslabones. En concreto se estudiarían las velocidades comandadas de 200, 500 y 1000 mm/s, exceptuando para la herramienta que solo se realizara el análisis para una velocidad de 1000 mm/s.

La primera comprobación que efectuamos es si realmente la herramienta adquiere la velocidad asignada por el programa. Ya que, si esta no fuera la misma, podría suceder que fuera un problema en la medición del sensor o que realmente la velocidad que se le asigna al robot no sea la misma con la que se mueve en el extremo. Por lo tanto se va a realizar un estudio de los parámetros cinemáticos en la herramienta.

En la Figura 49 obtenemos los valores para el movimiento descrito anteriormente, en el eje de ordenadas encontramos las unidades de velocidad, en metros por segundos (ms^{-1}). En el eje de abscisas está ubicado el tiempo en segundos (s). Los datos proporcionados son directamente la velocidad filtrada por MatLab. Una de las primeros aspectos que podemos observar en la figura es la diferenciación de secuencias de tres movimientos descritos anteriormente enumerados y recuadrados en un círculo rojo.

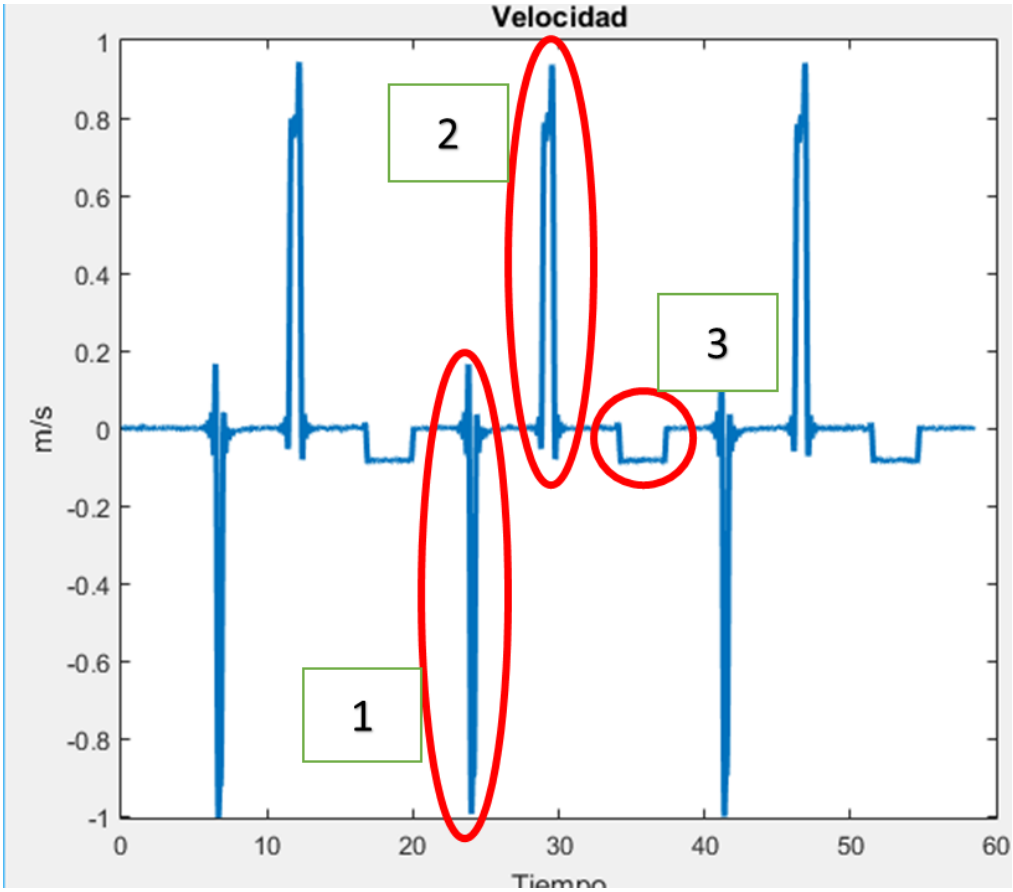


Figura 49. Velocidad lineal en la herramienta para una velocidad v1000.

El primer movimiento es el más corto ya que tan solo necesita recorrer 50º, el segundo sería el tramo más largo recorriendo en total 90º y por último el robot volvería a el estado de reposo (con todos los ángulos a cero) a una velocidad fija de 100 mms^{-1}.

Que las velocidades 1 y 3 aparezcan en negativo se debe a la orientación de los ejes del sensor.

Para el estudio de la herramienta, en los 60 segundos que ha durado la toma de datos el robot ha realizado 3 series de movimientos completos. También podemos apreciar que la repetibilidad de los datos es bastante alta, sin demasiados desbarajustes entre una secuencia y otra.

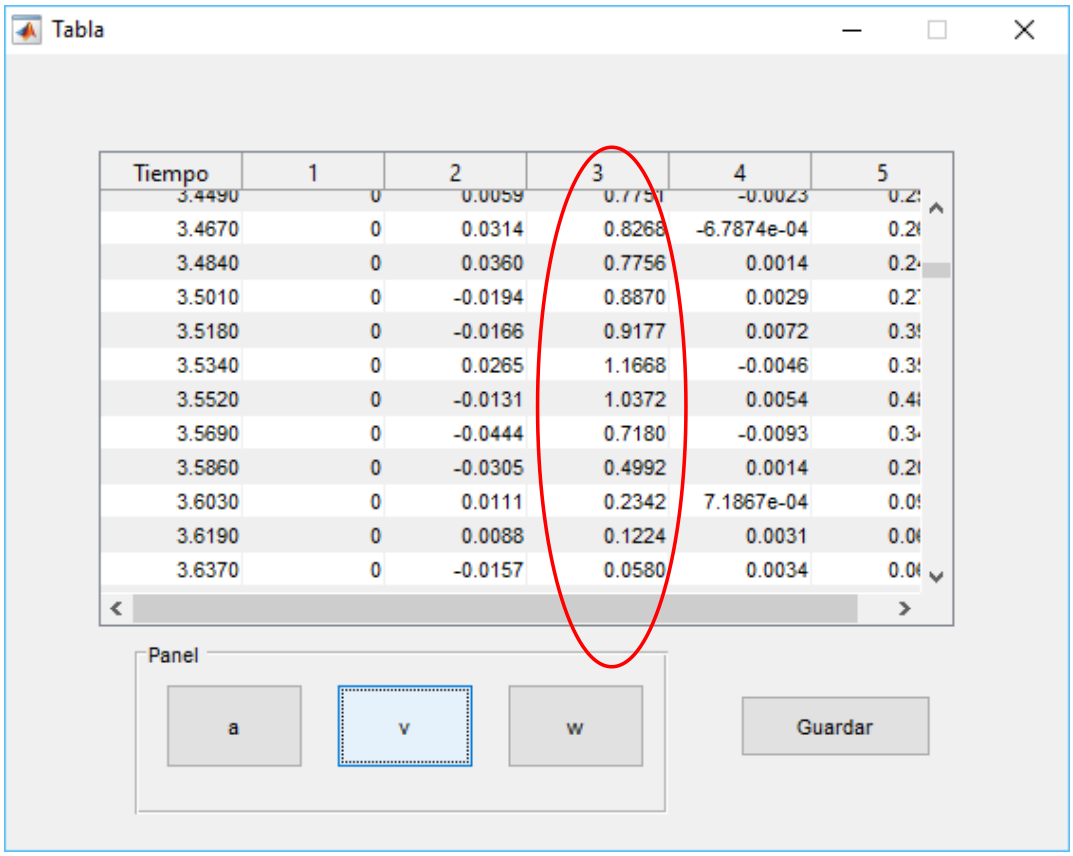


Figura 50. Datos numéricos para la herramienta.

Observando la Figura 50 los datos de forma numérica, vemos que la velocidad máxima alcanzada supera un poco los 1000 mms^{-2}, siendo el valor más alto 1166 mms^{-1}, lo que representa un error del 16,6% pero una vez filtrada la señal se reduce a un 2%.

Viendo esta gráfica y los damos por concluido el estudio de la herramienta afirmando que la velocidad adquiere el robot es aproximada a la programada (1000 mms^{-1}).

6.1 Estudio de los eslabones 2 y 3

Una vez estudiada la velocidad y aceleración en la herramienta, se procede a realizar un estudio de los eslabones 2 y 3 a diferentes velocidades. La GUI diseñada en MatLab ofrece el mismo estudio para todos los eslabones., pero se ha optado por estos dos ya que son de mayor longitud y donde se pueden obtener resultados más claros.

6.1.1 Eslabón 2

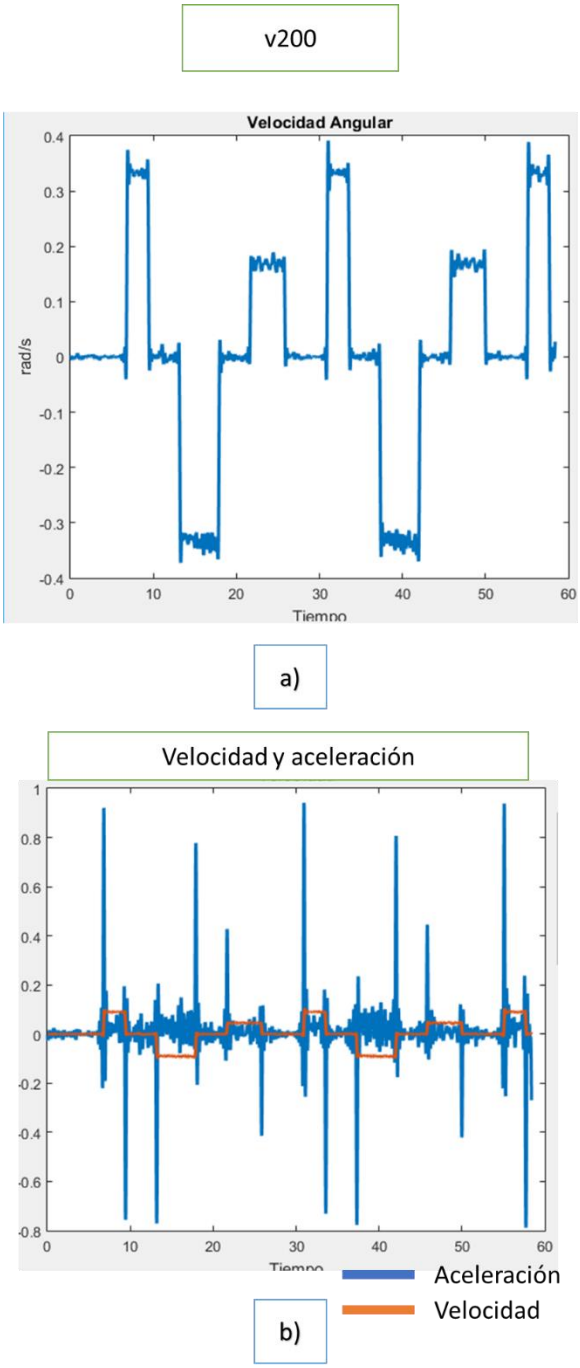


Figura 51. Velocidad y aceleración en el eslabón dos para una velocidad programada de 200mm/s.

En las siguientes Figura 51 se va a mostrar la velocidad angular, la velocidad lineal y la aceleración del eslabón dos para una velocidad programada de 200 mms^{-1}.

Podemos obvserbar dos series completas de tres movimientos, como es normal el segundo movimiento tarda más en ejecutarse ya que es el más largo, y el tercero corresponde al retorno al reposo que siempre se realiza auna velocidad de 100 mm/s. En las gráficas aparecen velocidades negativas esto se debe a los ejes del sensor y su orientación. Se ha dedicido superponer las gráficas de velocidad lineal y aceleración para realizar un análisis más detallado.

Para el apartado a) correspondiente a la velocidad angular de eslabón 2 para una velocidad programada de 200 mm/s, en el eje de ordenadas tenemos rad/s y en el eje e abcisas los segundos (s). La velocidad angular máxima alcanzada tanto por un movimiento u otro es de 0,37 rad/s en grados es de 21,2 º/s. Aunque se aprecien picos de velocidad, por lo general el movimiento se realiza de forma constante. En el apartado b) nos encontramos la gráfica de la velocidad de color rojo, este eje de ordenadaas esta representada en m/s, la aceleración esta representada en color azul y el eje de ordenadas esta representada en m/s^2. La velocidad máxima alcanzada es de 100 mm/s, pero rápidamente se reduce su velocidad a una media de 90 mm/s y se mantiene de forma constante. Para la aceleración hay que diferenciar entre cuando esta acelerando y cuando esta frenando. Cuando la velocidad es positiva, lás aceleraciónes aparecerán en positivo y las frenadas en negativo, pero cuando la velocidad es negativa estás aparecerán de forma contraria, es decir, las aceleraciones aparecerán en negativo y las frenadas en positivo. Al principio del movimiento primero el eslabón experimenta una aceleración alcanzando el valor de 900 mm/s^2 a continuación realiza una frenada de 200 mm/s^2, cuando la velocidad es contastante el valor de la aceleración es 0. Al final del recorrido el robot realiza una desalceleración de 0,75 mm/s^2. Para el segundo movimiento (recuerde que la velocidad está en negativa) el eslabón 2 alcanza una aceleración menor de 0,78 mm/s^2 y al final del recorrido desacelera a 0,75 mm/s^2. Estos valores se repiten a lo largo de la serie.

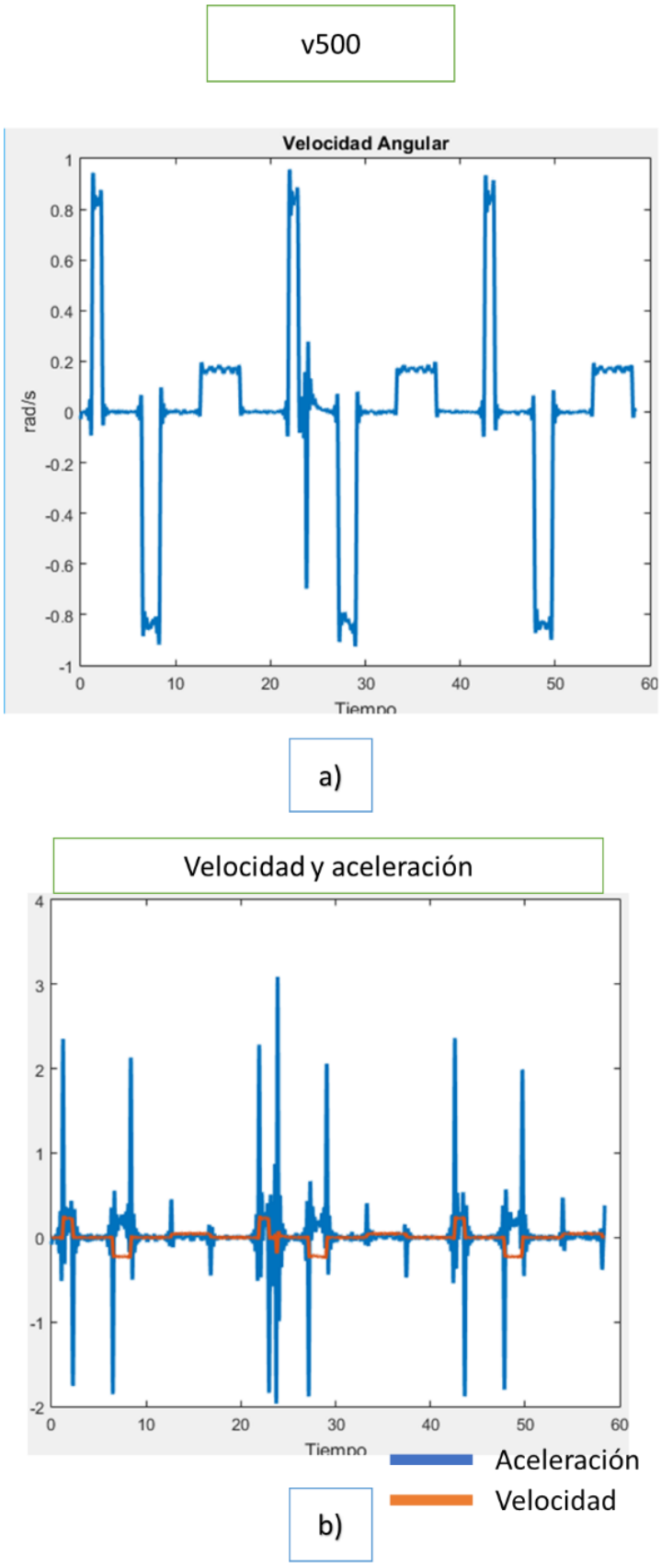


Figura 52. Velocidad y aceleración para una velocidad de 500 mm/s programada.

Procedemos a aumentar la velocidad a 500 mm/s, Por lo tanto en la siguiente Figura 52 se mostrar la velocidad angular, la velocidad lineal y la aceleración del eslabón dos a una velocidad programada de 500 mm/s.

Como en el caso anterior en la Figura 52 tenemos las gráficas de velocidad angular, y la velocidad junto a la acelerecación. Para esta gráfica se observa tres series de movimientos completos, pero para la segunda serie se aprecia datos que no concuerdan con el resto de series, por lo que que esta segunda serie, no valdría. Ya que salen datos erróneos.

La velocidad angular aumenta hasta alcanzar una velocidad constante de 0,85 rad/s es decir 49 º/s. En cuanto a la velocidad lineal, alcanza una velocidad pico de 250 mm/s y de media los 230 mm/s. Respecto a la aceleración, alcanza aceleraciones por encima de los 2000 mm/s^2 y desaceleraciones un poco por debajo de esos 2000 mm/s^2.

Se procederá aumentar una vez más la velocidad, para obtener resultados más y por si se vuelve a producir errores.

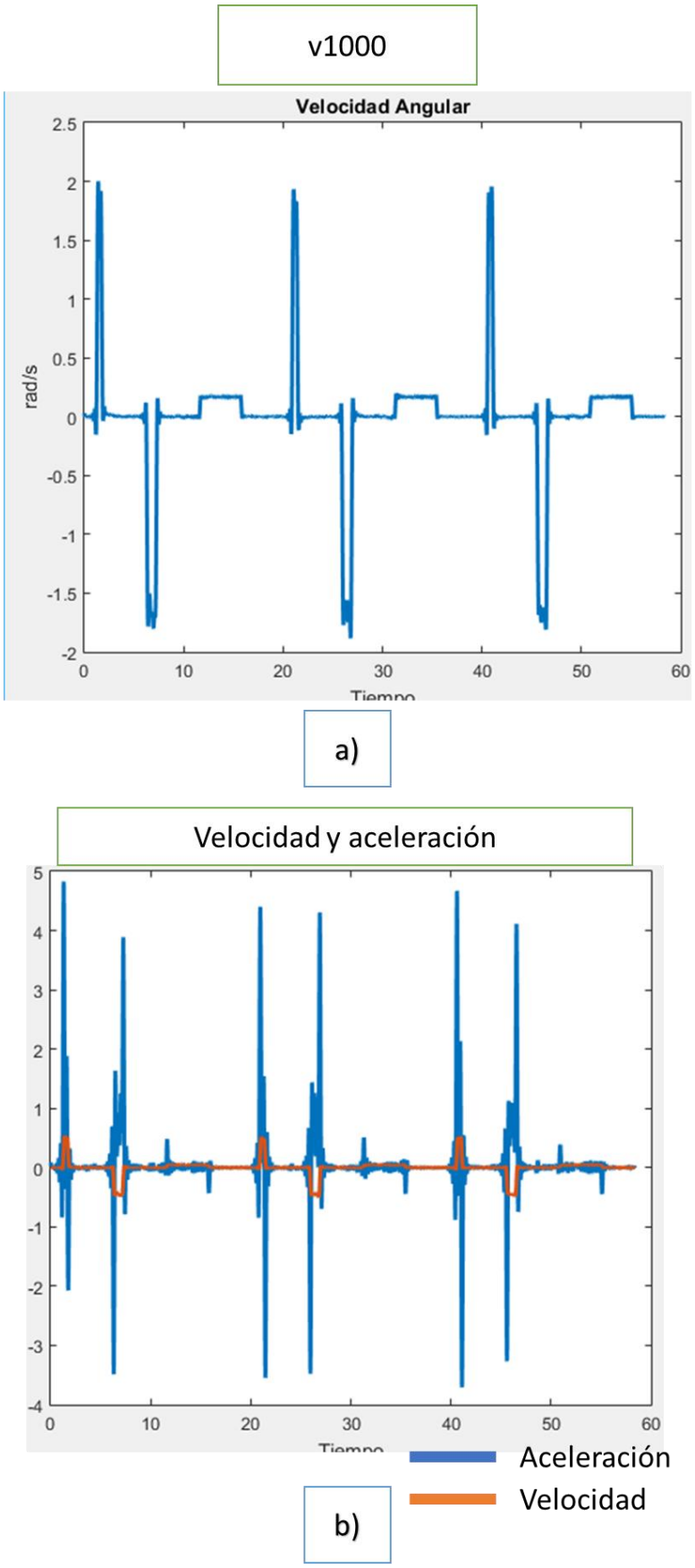


Figura 53. Velocidad y aceleración para una velocidad programada de 1000 mm/s en el eslabón 2..

Para esta última velocidad de 1000 mm/s, en la Figura 53, la velocidad angular en el tramo más corto alcanza una velocidad de 2 rad/s es decir, 114,6 º/s, todavía el robot podría aumentar más su velocidad ya que soporta hasta los 250 º/s. La velocidad alcanzada por el eslabón 2 a una velocidad programada de v1000 es de 550 mm/s – 500 mm/s. Para el apartado de la aceleración, para el primer movimiento, alcanza 4500 mm/s^2 de aceleración y produce una desaceleración de 3500 mm/s^2. Para el segundo movimiento la aceleración es algo menor, igual que se ha visto ocasiones anteriores siendo esta de 3500 mm/s^2 y produciendo una desaceleración de 4000 mm/s^2.

6.1.2 Eslabón 3

A continuación, se comparar la velocidad y la aceleración para el caso del eslabón 3, en la Figura 54 encontramos las gráficas de velocidad y aceleración superpuesta para las velocidades de 200, 500 y 1000 mm/s.

Para una velocidad programada de v200, el eslabón 3 alcanza unas velocidades de 140 mm/s y aceleraciones de 1200 mm/s^2. Cuando v500 la velocidad alcanza valores de 400 mm/s y aceleraciones de 3000 mm/s^2 y por último cuando v1000 el eslabón 3 alcanza valores de 800 mm/s y aceleraciones de hasta 6000 mm/s^2.

Los datos son mayores que los obtenidos en el eslabón 2. Claramente dependiendo del tipo de movimiento que realicemos con el robot, los eslabones se moverán con mayor o menor velocidad.

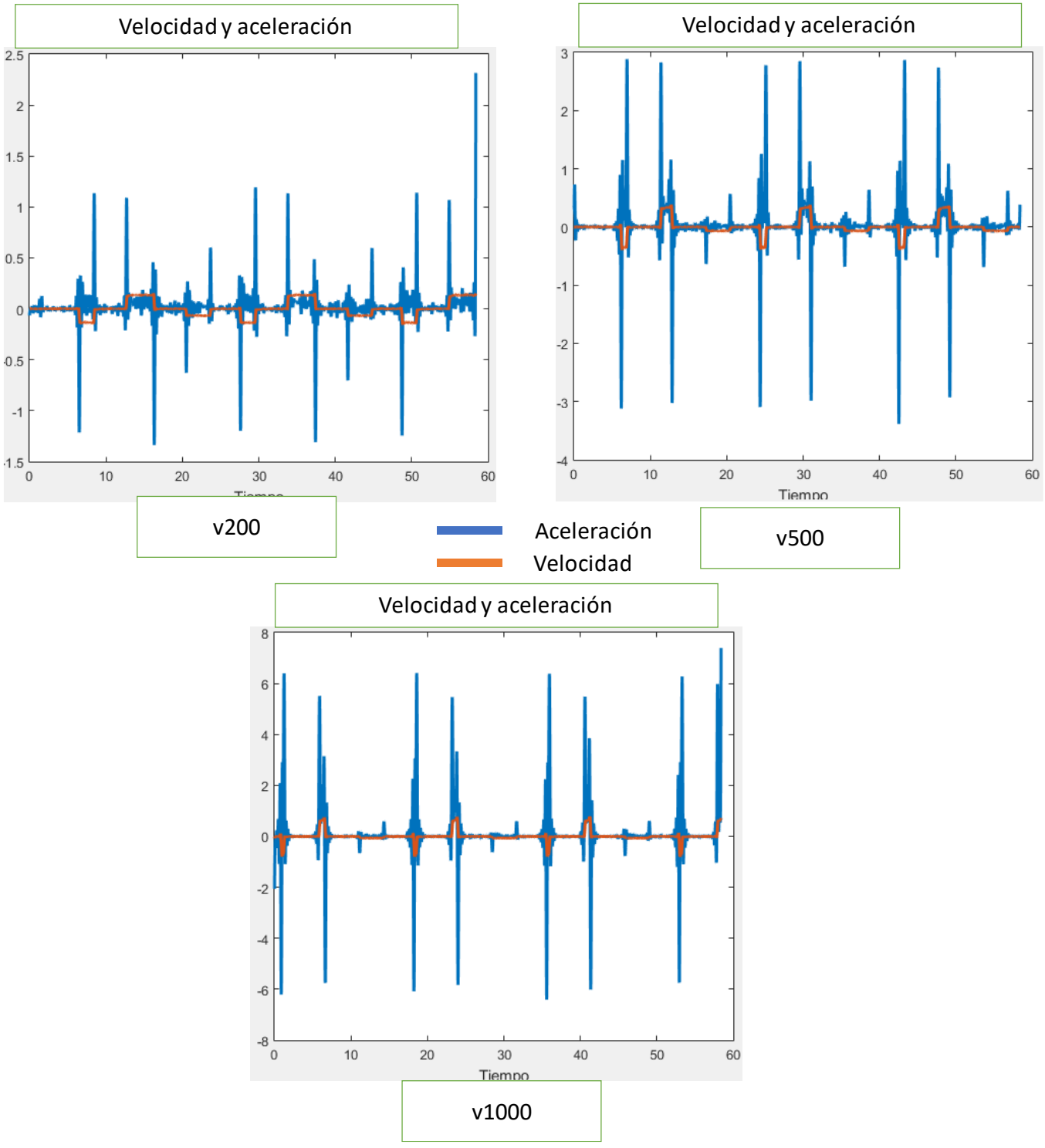


Figura 54. Velocidad y aceleración en el eslabón 3 para las digerentes velocidades programadas.

6.2 Comparativa de MPU605 y ADXL345

Como se ha comentado anteriormente, tenemos dos acelerómetros diferentes, siendo uno el MPU6050 que incluye giroscopio y otro el ADXL345. La idea de poner dos sensores diferentes ha sido para poder realizar una comparativa y poder comprobar cuál de los dos sensores de adapta mejor.

Se mostrará una comparativa de la velocidad angular y otra de la velocidad lineal junto con la aceleración en la Figura 55.

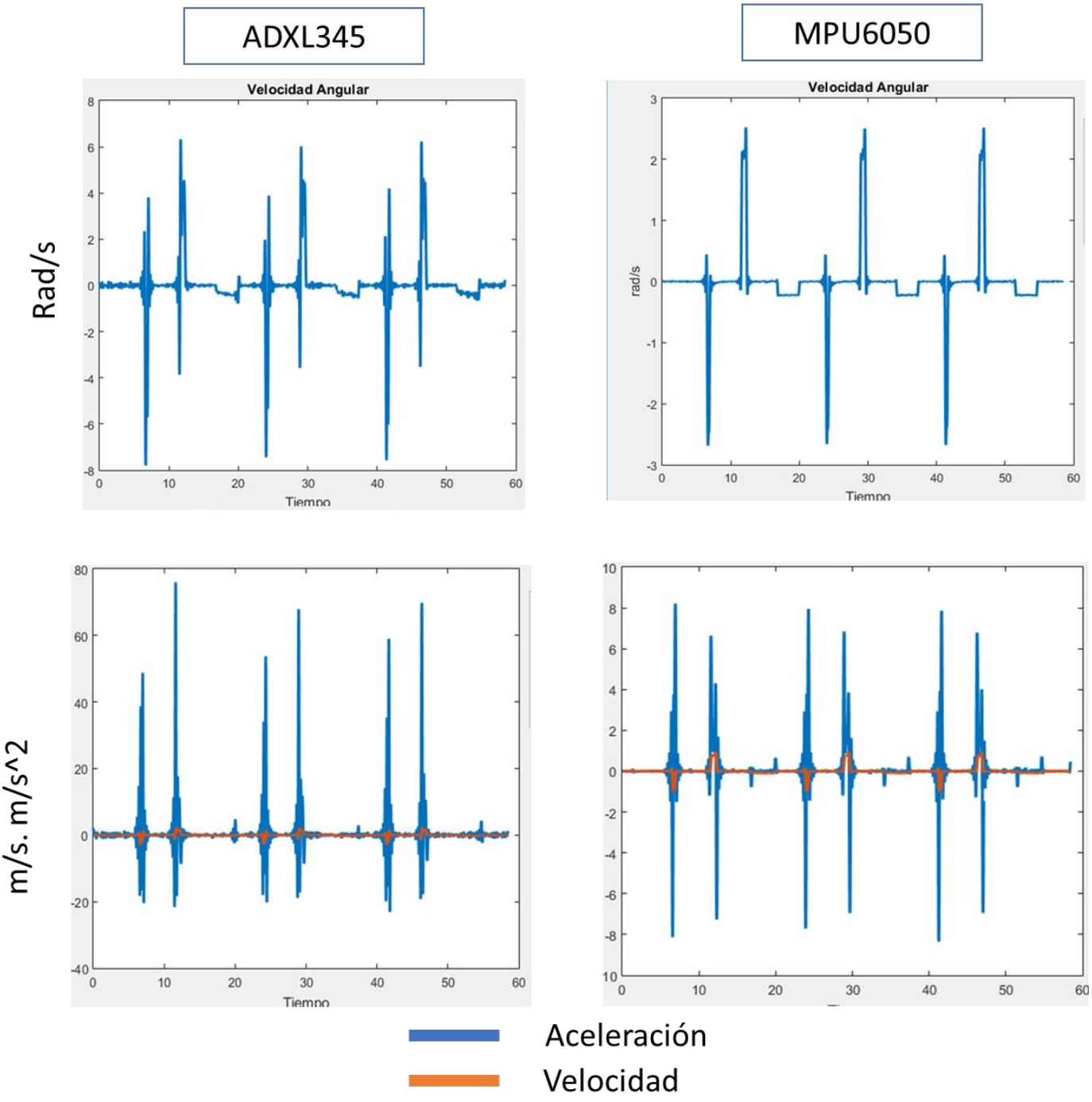


Figura 55. Comparativa MPU5060 con el ADXL345.

En el lado izquierdo de la figura tenemos el sensor ADXL y en el lado derecho el ADXL. Las gráficas superiores corresponden a los datos de velocidad angular y las inferiores a la velocidad y aceleración de manera superpuestas, siendo de color rojo la velocidad y de color azul la aceleración.

. Existen diferencias notables entre los datos tomados por un sensor y otro. Mientras que en el MPU existen una gran similitud entre una serie de movimientos y otra, para el

caso del ADXL se encuentran más distorsionados, produciendo velocidades angulares negativas que no existen. En el movimiento al reposo, el de menor velocidad, para el MPU aparece una velocidad constante mientras que en el acelerómetro ADXL aparece con una tendencia ascendente. Teniendo el MPU una tendencia más lineal que el ADXL.

Para las dos imágenes inferiores, las de velocidad e aceleración, se observan aun un mayor número de diferencias entre los sensores. El ADXL muestra aceleraciones desorbitadas muy lejos de la realidad, ya que sengún el ADXL la aceleración alcanza 80 m/s^2 a diferencia del MPU que la aceleración es de 8 m/s^2 algo acorde con la realidad.

Las diferencias que se pueden apreciar entre estos dos acelerómetros son debidas a que el ADXL no incorpora un giroscopio incorporado, por lo que no se le puede aplicar el filtro complementario y el error se va acumulando y no se consigue corregir la media a diferencia del MPU que si tiene un giróscopo.

6.3 Interfaz cinemática

Por último, la interfaz incorpora la posibilidad de obtener el problema cinemático, este relaciona las coordenadas del extremo del robot en función de las coordenadas de las articulaciones. El estudio de la cinemática se realizó anteriormente por el alumno Axel José Córdoba López. La idea es completar el programa que él realizo, con el fin de obtener las coordenadas del robot sin necesidad del panel FlexPendant.

En la siguiente figura se mostrar una comparativa entre lo calculado por la interfaz y los datos que aparece en la unidad de programación.

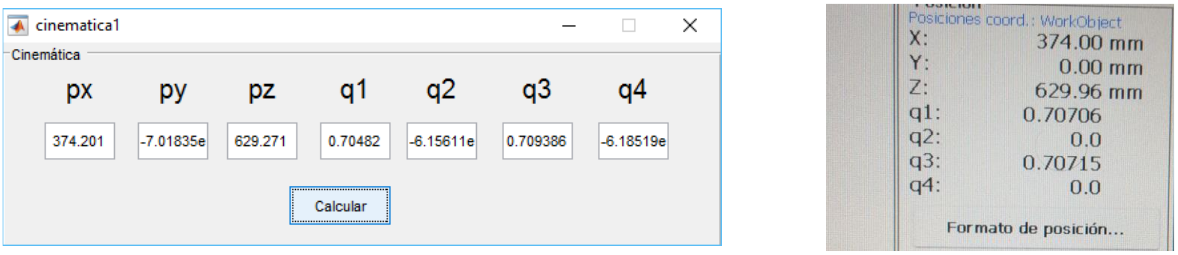


Figura 56. Datos de la posición en el espacio de trabajo cuando el robot se encuentra en reposo.

En la Figura 56, se puede apreciar que los resultados de la interfaz coinciden con el panel FlexPedant. Siendo el margen de error de <0.1%. Pero al realizar movimientos con el brazo robótico este margen de error va aumentando de forma considerable.

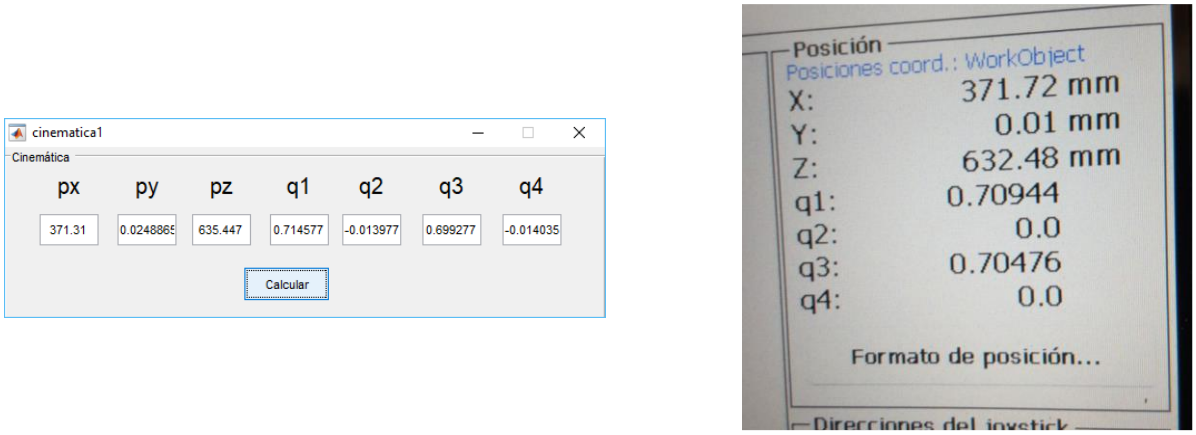


Figura 57. Datos del espacio de trabajo para una variación pequeña.

La variación del movimiento respecto a la del reposo ha sido de un grado. Y el margen de error ha aumentado del 0,1 a 0,5 %. Para solo hacer una pequeña modificación en la posición del robot el error es bastante amplio.

Cuando realizamos movimientos mayores a 10º para un eslabón lo resultados se muestran en laFigura 58.

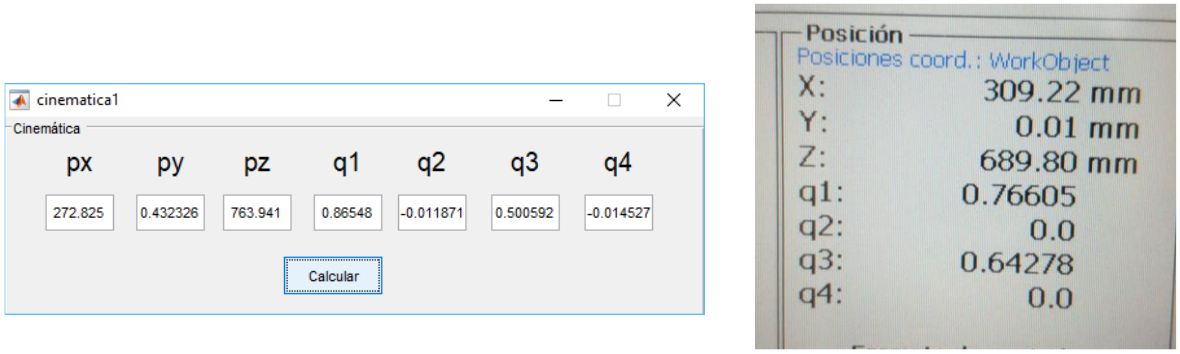


Figura 58. Datos del espacio de trabajo para una variación del ángulo de 10º.

Se puede apreciar que los resultados ya no coinciden con la realidad (datos del lado derecho) siendo el margen de error bastante amplio para una variación pequeña de un solo eslabón. El margen de error varía desde el 15% hasta incluso el 300%.

Se intento corregir esta variación que se produce entre la interfaz creada y la unidad de programación, se creó un bucle en el cálculo cinemático para que el valor que obtenga de la toma de sensores no sea el primer valor, si no un dato obtenido después de la toma de 300 datos, por si el cálculo erróneo fuera porque el sensor se intenta estabilizar, pero los resultados no cambian.

7 SIMULACION DE APLICACIÓN INDUSTRIAL CON MÓDULO DE E/S

Otro de los objetivos principales de este proyecto es mostrar una aplicación que simule una actividad industrial en la que se emplee el módulo de entradas y salidas. Cosa que no se ha realizado en anteriores Trabajos Fin de Grado de esta escuela.

En este caso se ha optado por simular una soldadura mediante un emisor laser. Para ello han sido necesarias dos tareas principales. En la primera se ha estudiado cómo acoplar la herramienta al brazo robótico de la manera más cómoda y versátil posible para que no entorpeciera con otras prácticas. Por otro lado, ha sido necesario profundizar en el conocimiento sobre el módulo de entradas y salidas del IRC5.

El modelo IRB 1600ID especializado para soldadura por arco (Figura 59), tiene todos los cables y mangueras colocados dentro del brazo superior, adaptando al robot a la aplicación de soldadura por arco. Todos los componentes necesarios para la soldadura tales como el hilo para soldar, el gas de protección y el aire a presión están integrados en el robot.

Figura 59. IRB 1600ID [17]

7.1 Estudio del acoplamiento físico

La herramienta con el fin de sujetar el láser ha sido creada con la idea de utilizar componentes que el robot ya tiene implementados. Por lo tanto, la herramienta no tiene la misma forma convencional de un robot soldador.

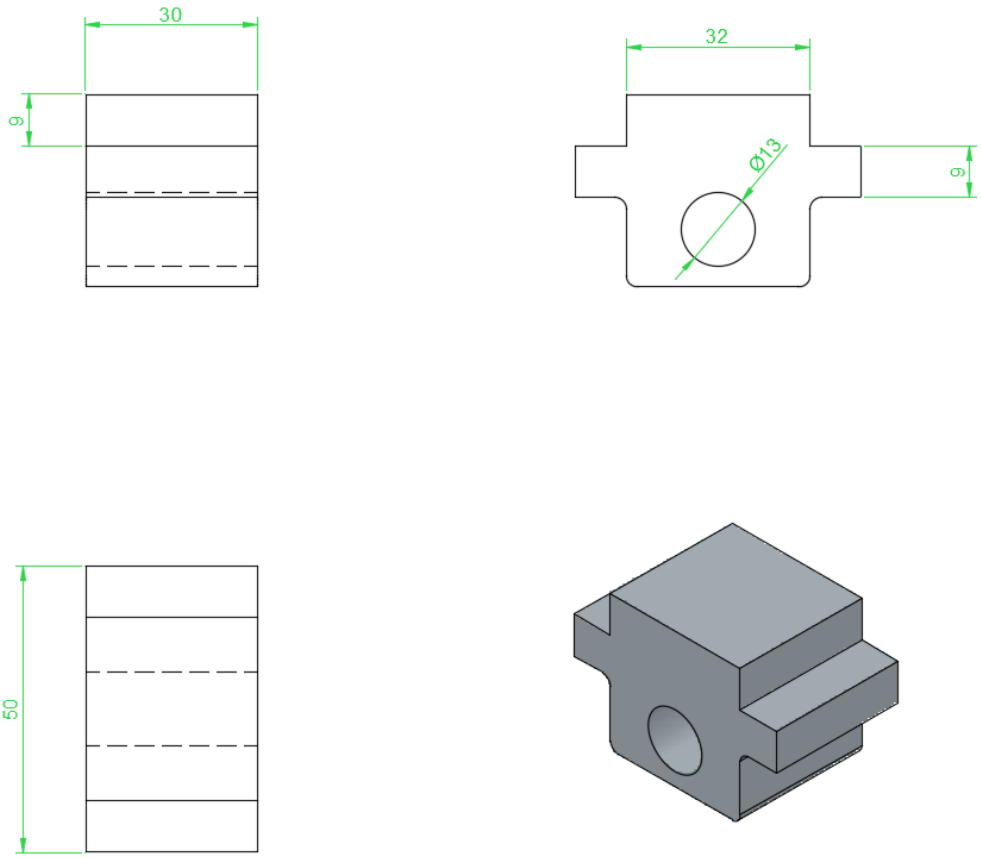


Figura 60. Planos de la herramienta para sostener el láser

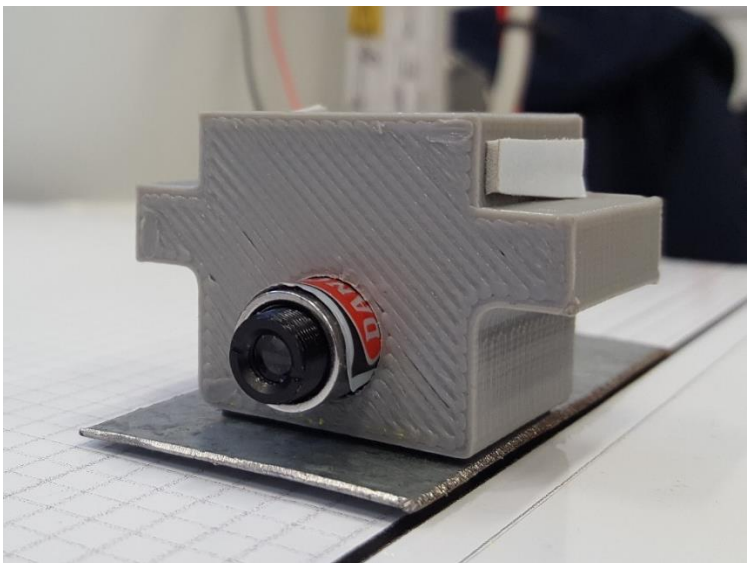


Figura 61. Herramienta para sostener láser.

La pinza del robot dispone de actuadores eléctricos que tienen como ventaja la posibilidad de regular el recorrido de los dedos. Estos dedos se encuentran en posición paralela y se mueven de forma síncrona para el agarre de la pieza. Dispone de 14 mm de desplazamiento entre los dedos y una fuerza de agarre variable de 16 a 40 N. La forma los dedos de la pinza complica el agarre de las piezas, por ello dispone de dos taladros pasantes M5 en la dirección de apretura/cierre para añadir una pieza de agarre para la pinza que adecue la superficie de contacto con la pieza a agarrar.

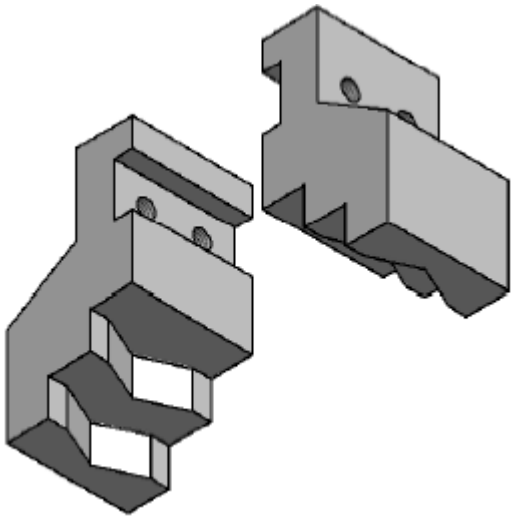


Figura 62. Mordaza de sujeción. [8]

Para este robot se dispone de unas mordazas de agarre genérico de superficie plana con dos aberturas triangulares como podemos apreciar en la Figura 62, que se añaden a los dedos mediante dos tornillos para aumentar el rango de las piezas a manipular. También tiene una capa de plástico que protege a la pinza de golpes y arañazos y la hace más rugosa para que las piezas no se deslicen al agarrarlas Figura 63.

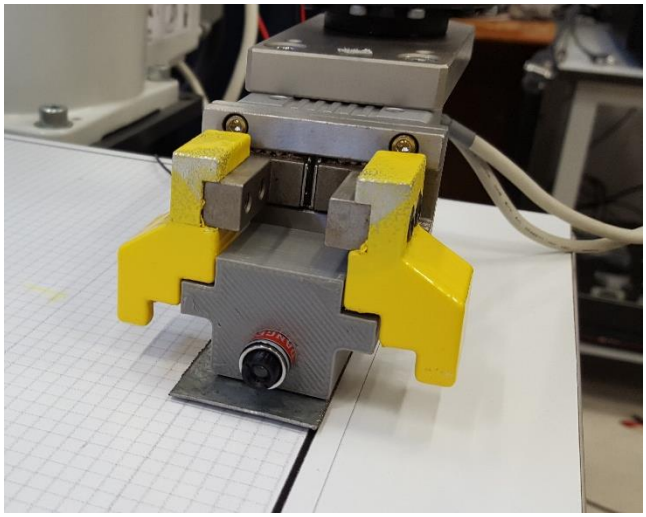


Figura 63. Conjunto de herramienta y mordaza.

El objetivo de la pieza ha sido adaptarlo a las mordazas de la pinza, con el fin de que el robot pueda manipular objetos y realizar la simulación de la soldadura.

La pieza da sido diseñada en AutoCAD, y ha sido exportada en formato .stl para ser impresa en 3D.

La herramienta descansa sobre el tablero apoyada sobre una plataforma. Para impedir que se mueva de la mesa se le ha colocado unos adhesivos imanes. Estos tienen suficiente fuerza como para que la pieza no se mueva (Figura 64, Figura 65) mientras que el robot realiza otra serie de movimientos y facilita el agarre cuando sea necesario.

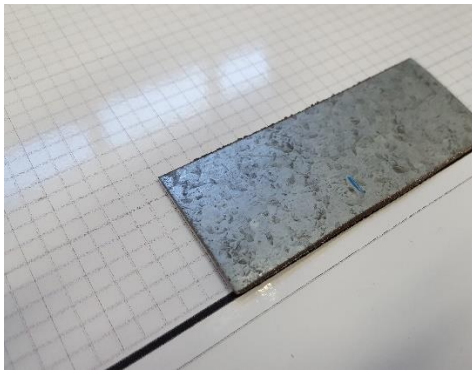


Figura 64. Placa soporte herramienta.

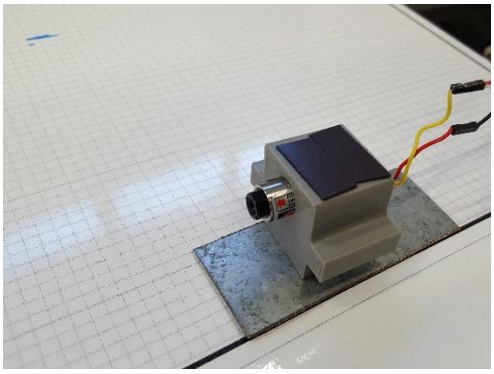


Figura 65. Imanes adhesivos soporte pieza.

7.2 Conexión Láser – IRC5.

La tarjeta de E/S utiliza DeviceNet. DevieceNet es una red digital, multipunto para conexión entre sensores, actuadores y sistemas de automatización industrial en general. Esta tecnología fue desarrollada para tener máxima flexibilidad entre diferentes fabricantes.

El controlador IRC5 cuenta con una tarjeta de entradas y salidas, en total el robot cuenta con 16 entradas digitales y 16 salidas digitales.

La salida de la tarjeta I/O DSQC652 es a 24V, por lo que es necesario reducir el voltaje para que sea compatible con el emisor laser (3,3V). Para ello se ha utilizado un reductor de voltaje regulable (Figura 34).

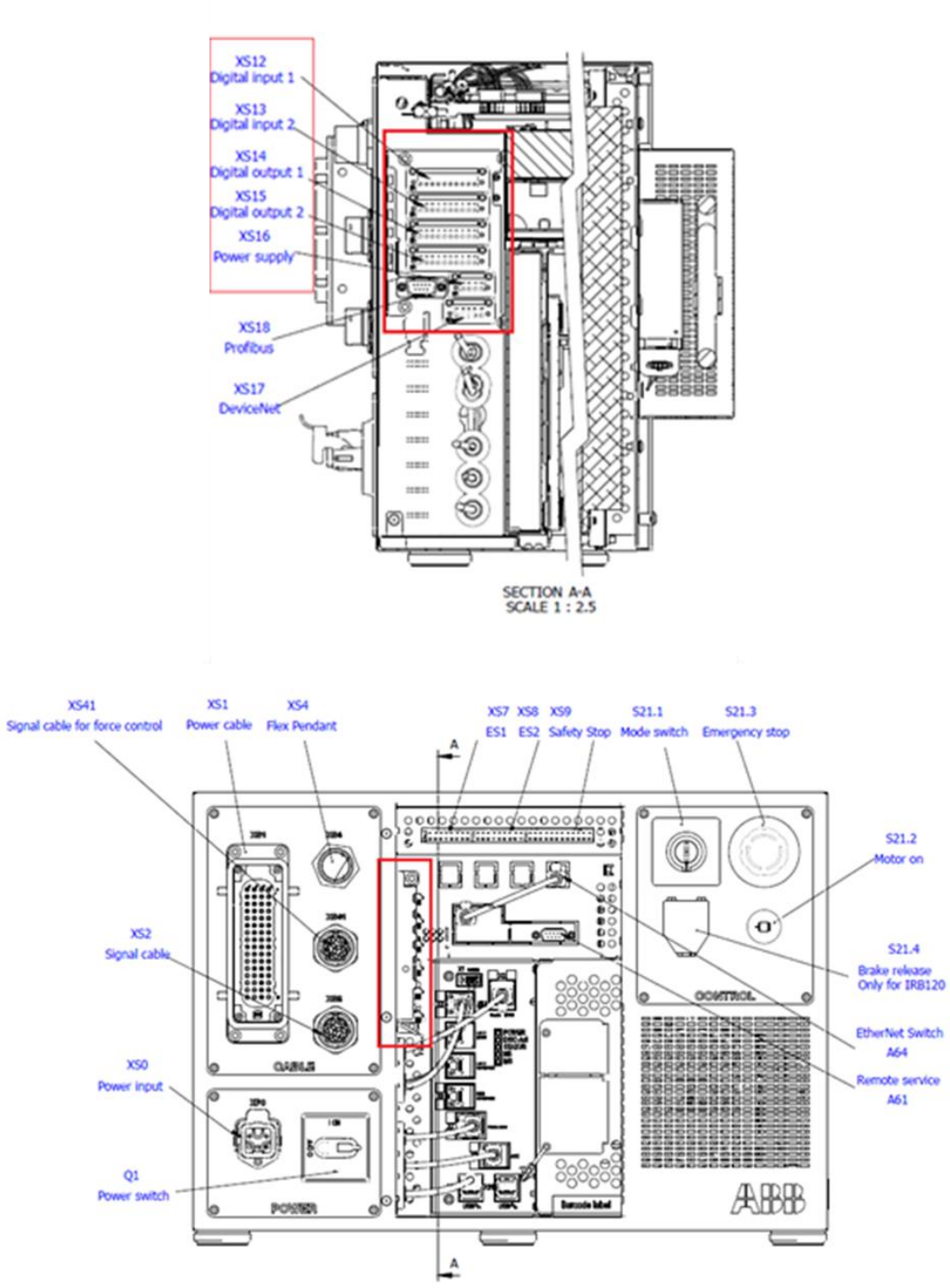
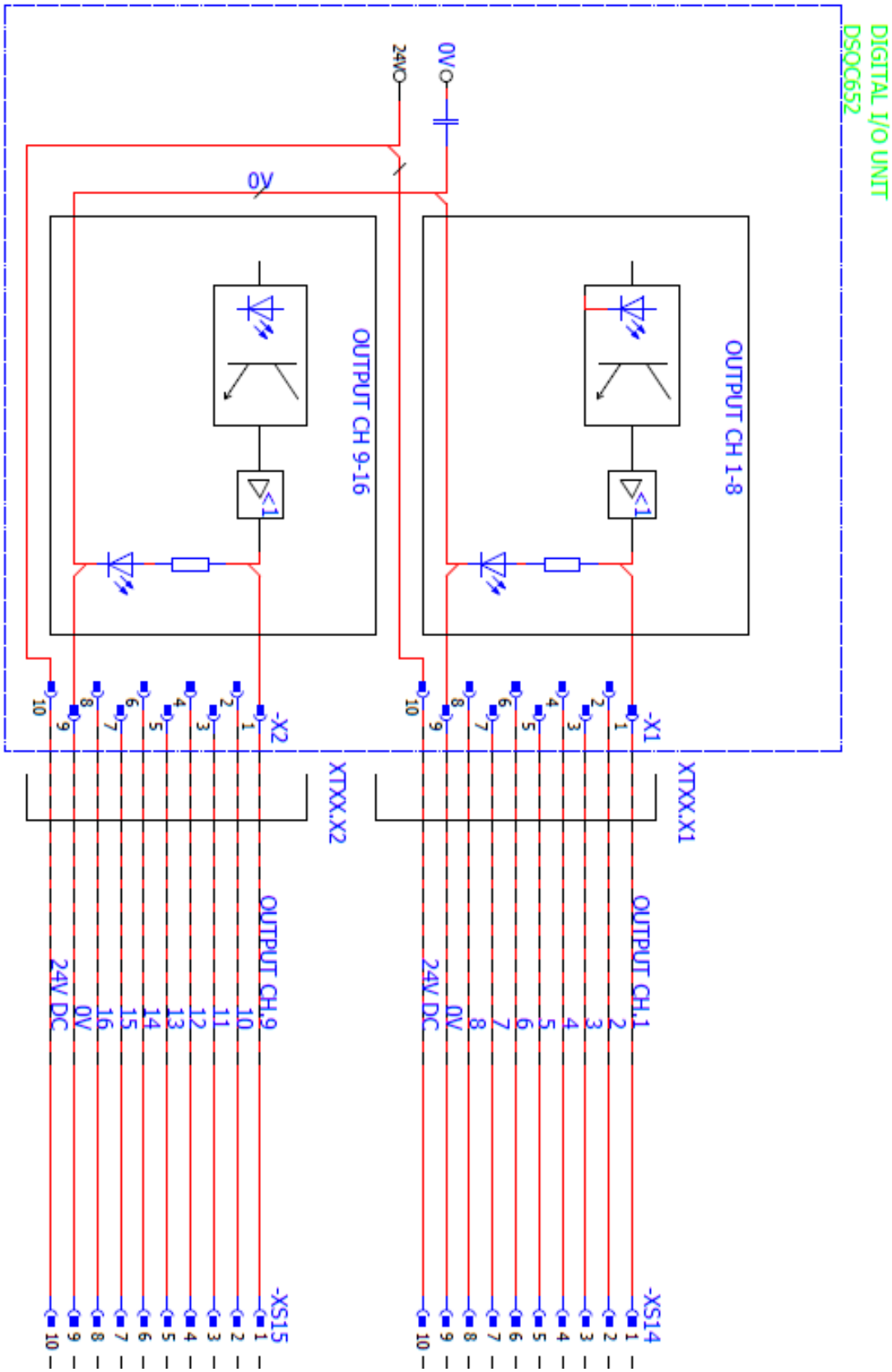


Figura 66. Tarjeta I/O DSQC652.

Al tratarse de una cabina de formación, las salidas las tiene cableadas en la parte exterior tal como se indica en la Figura 66. Se mostrará el diagrama de conexión:


DIGITAL I/O UNIT
DSOC652
24VO
0VO
0V
OUTPUT CH 1-8
OUTPUT CH 9-16
-X1
-X2
XTXX.X1
XTXX.X2
OUTPUT CH.1
OUTPUT CH.9
24V DC
0V
-XS14
-XS15

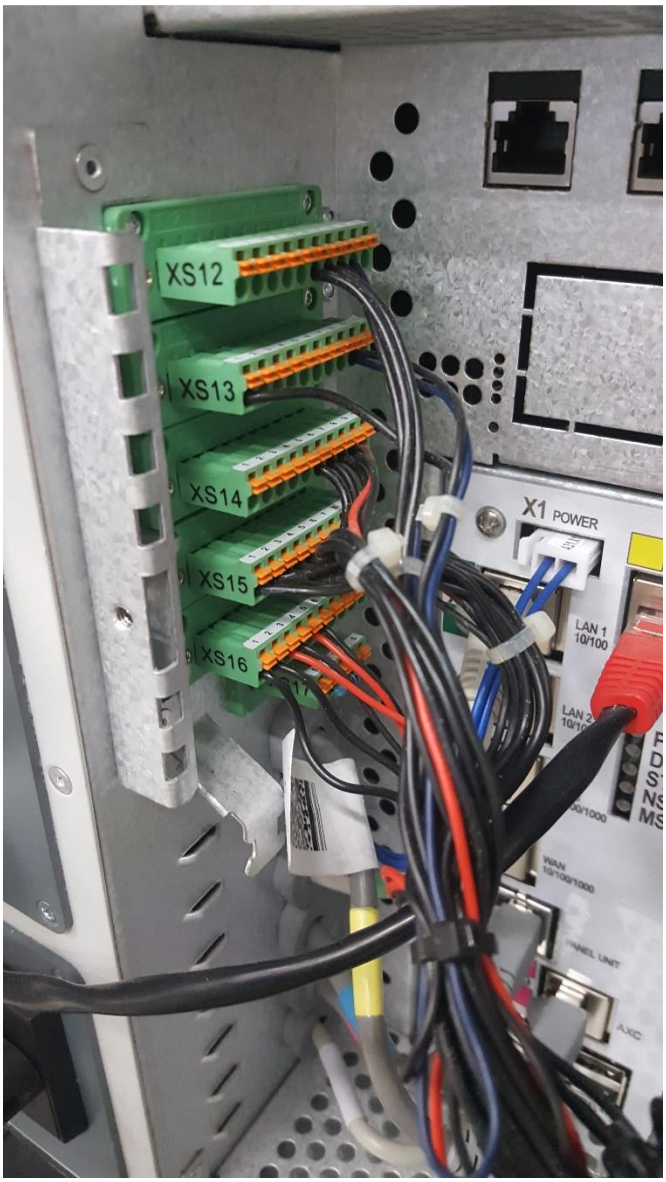


Figura 67. XS14 Conexión.

Por lo tanto para la salida de la señal, conectaríamos los cables en el Customer Do XS14, OUTPUT CH 1 y la toma de tierra en la línea 9.

Para realizar el cambio de valor de una salida digital en RAPID, utilizaremos el comando **SetDO** cuyos argumentos pueden ser:

SetDo [\SDelay][\Sync] Signal Valuer

- **[\SDelay]:** Retarda el cambio durante la duración especificada, en segundos (máximo 2s)
- **[\Sync]:** Sincronización, la ejecución del programa espera hasta que la señal cambie físicamente el valor especificado.
- **Signal:** El nombre de la salida que se desea cambiar.
- **Valuer:** El valor deseado para la señal, 0 ó 1.

7.3 Simulación de aplicación de soldadura 1

Las piezas a soldar en esta primera simulación son las de la Figura 68. Destacado en rojo queda marcado por donde se realizaría la soldadura.

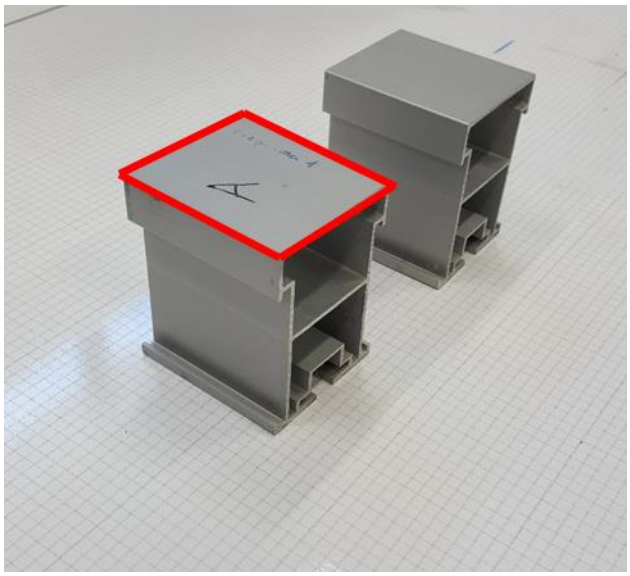


Figura 68. Piezas a soldar.

En esta simulación consiste en superponer una pieza sobre otra, y realizar una soldadura simulada alrededor de esta.

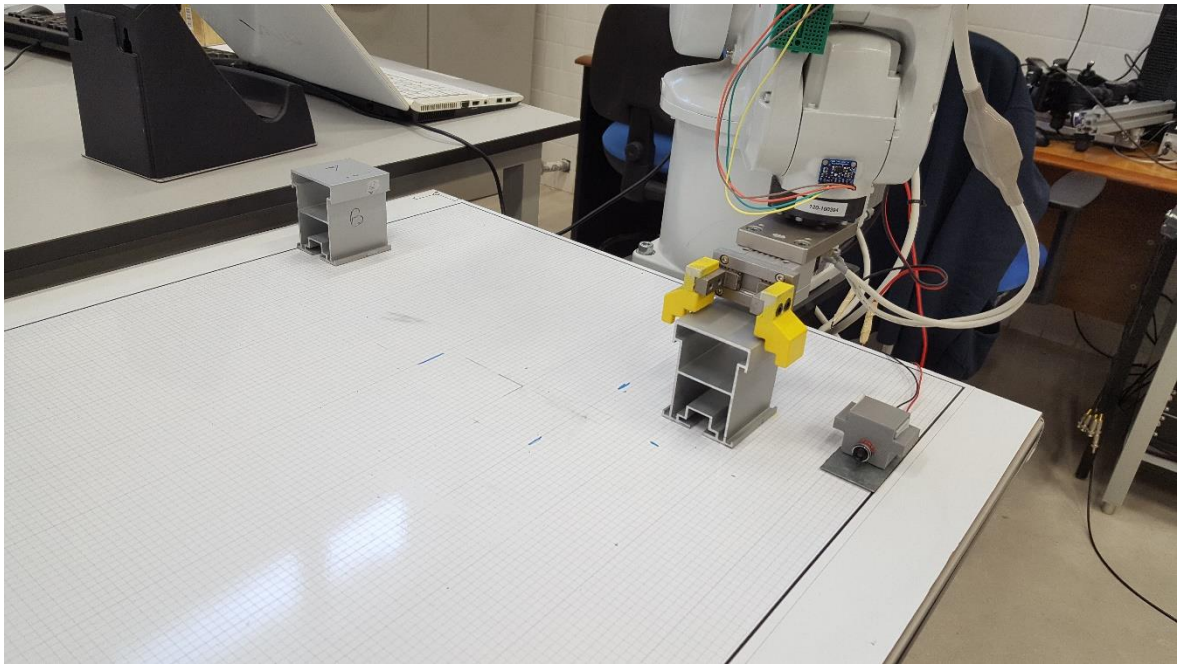


Figura 69. Soldadura 1. Posición inicial de las piezas.

El operario se encargará de colocar las piezas en la posición correcta (como se muestran en la Figura 69).

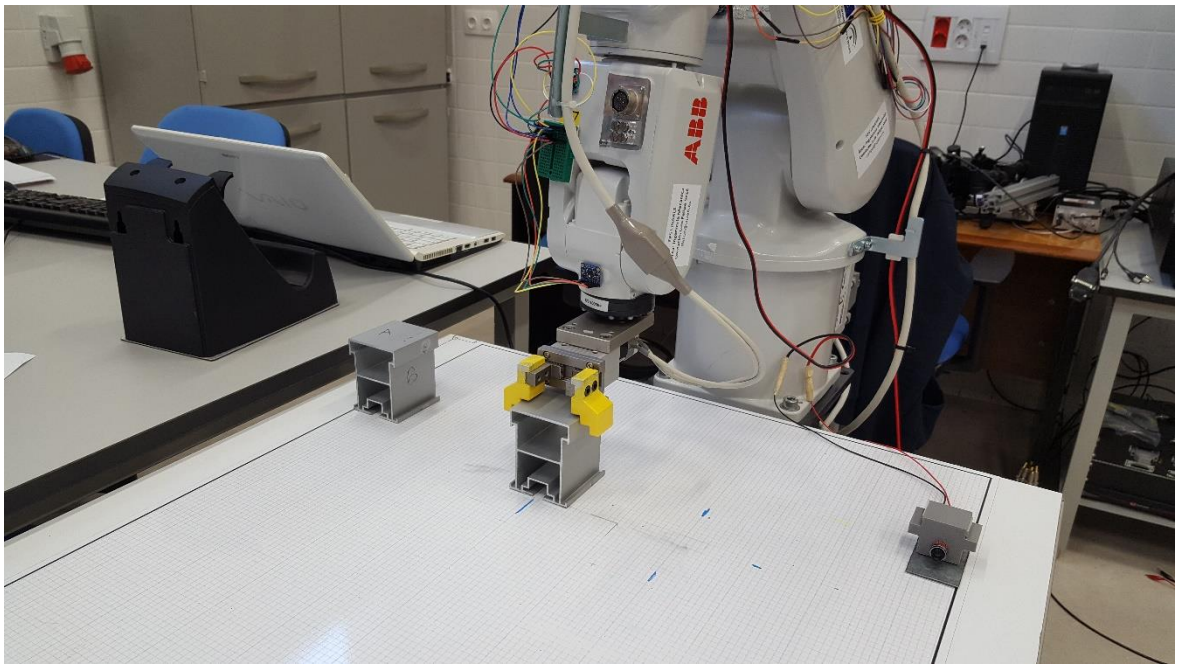


Figura 70. Soldadura 1. Inicio colocación de la primera pieza.

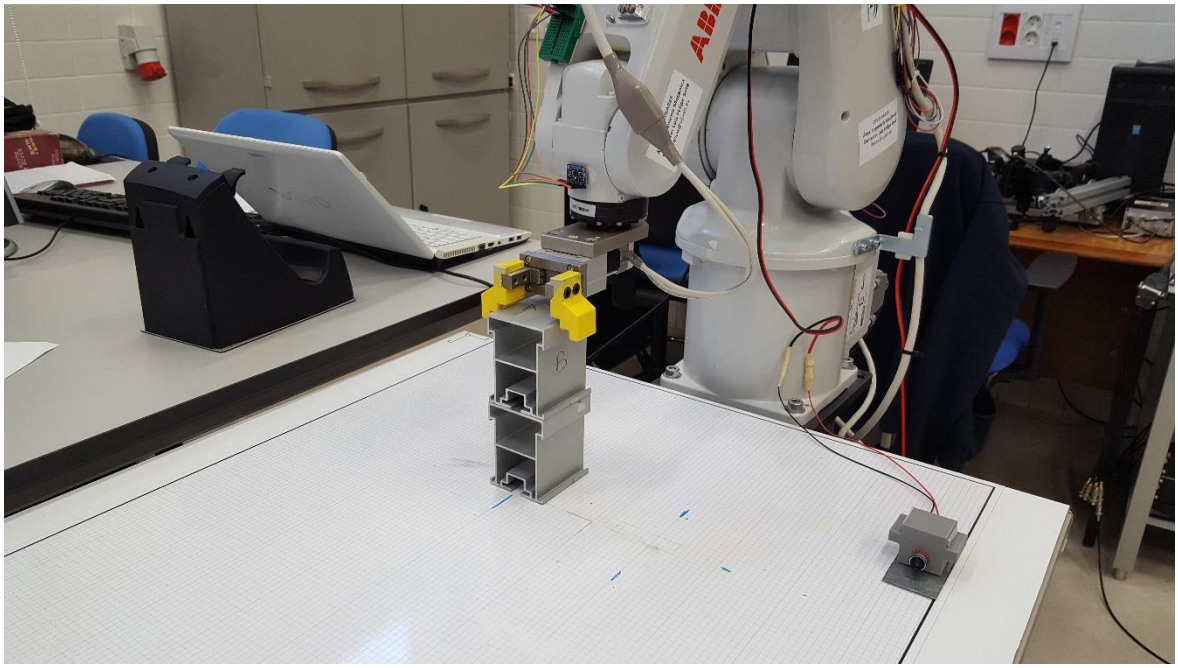


Figura 71. Soldadura 1. Colocación de la segunda pieza.

Las primeras instrucciones que recibe el brazo robótico son de apilar, es decir supero poner una pieza encima de la otra. A continuación (Figura 70-71), procedería a coger la herramienta laser creada para realizar la simulación de soldadura. Como se comentó anteriormente la pieza tiene unos imanes adhesivos para fijarla a la mesa consiguiendo así que la herramienta no se mueva, aunque el brazo robótico se aleje.

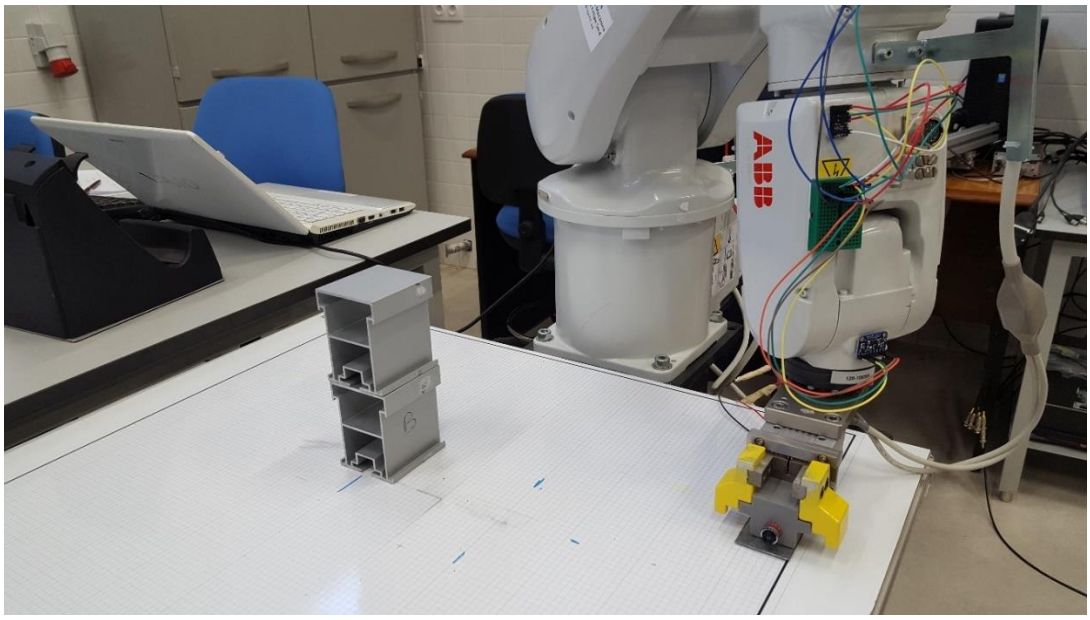


Figura 72. Soldadura 1. Brazo robótico con herramienta laser.

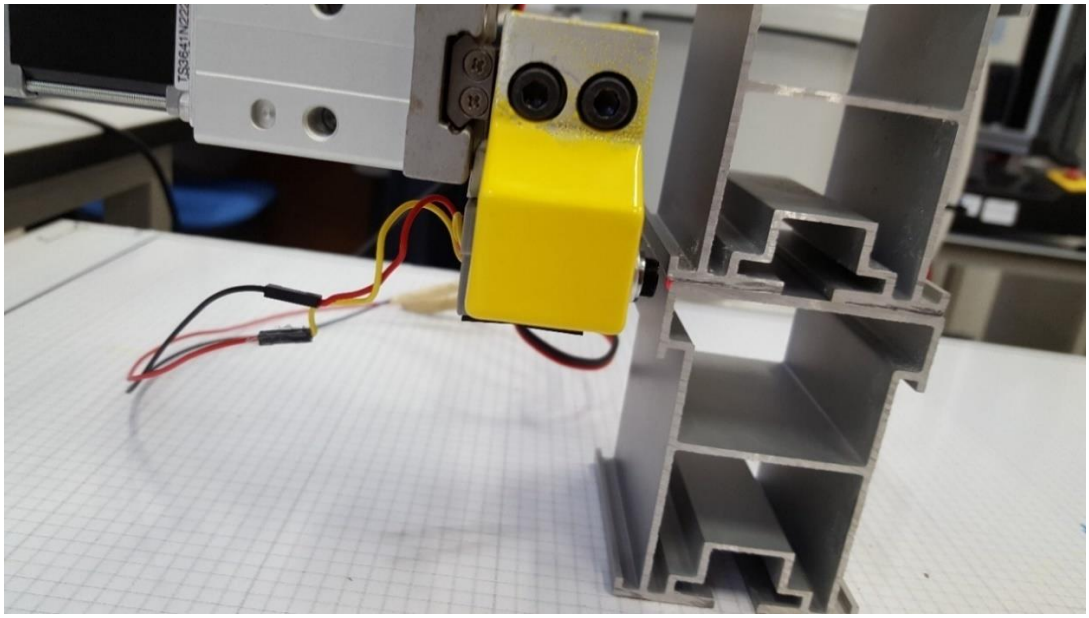


Figura 73. Soldadura 1. Inicio de soldadura.

Cabe destacar que como el robot tiene otro tipo de aplicaciones en la industria, sus limitaciones no alcanzan para realizar la totalidad de la pieza. Si se intentara realizar la soldadura de una tirada, el robot chocaría con sus propios eslabones impidiendo realizar el movimiento.

Por lo tanto, realiza una soldadura en L, levanta y realiza otra L para terminar de solar la pieza. Finalmente, el robot suelta la herramienta en su lugar de origen.

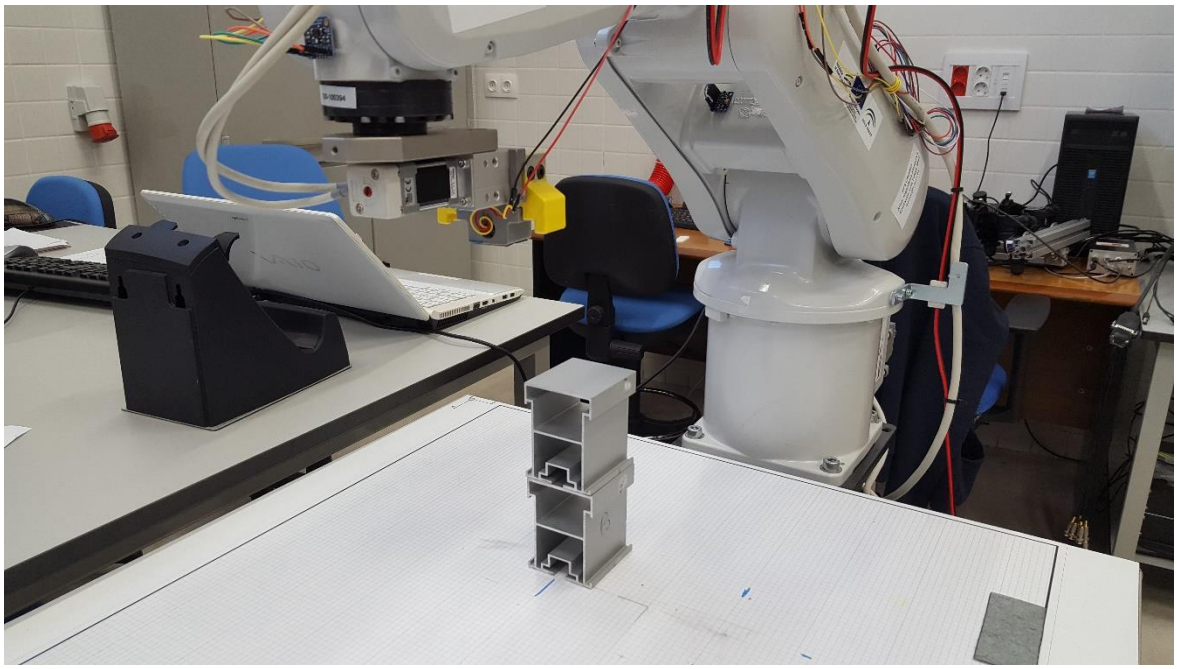


Figura 74. Soldadura 1. Nueva posición para terminar la soldadura.

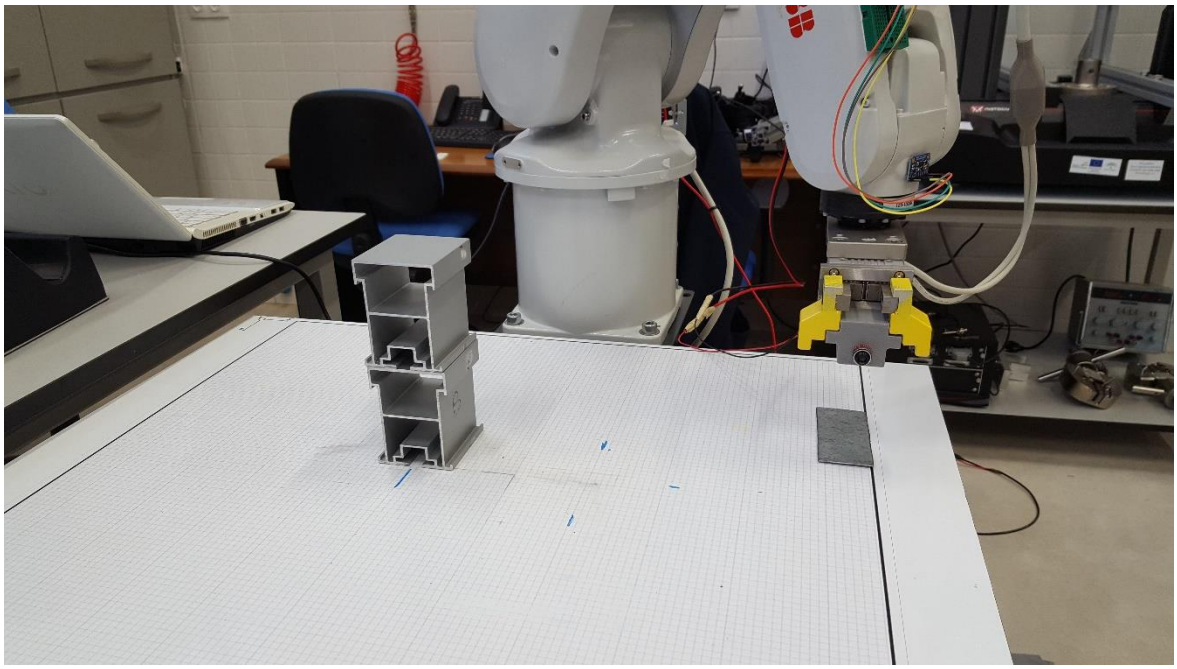


Figura 75. Soldadura 1, Colocación del láser en su posición original y fin de la soldadura.

En el anexo se podrá encontrar el código utilizado para realizar la soldadura, recibiendo el nombre de soldadura 1.

7.4 Simulación de soldadura 2

Para esta soldadura se realiza en un movimiento 3D, y no en un plano como en el anterior que la altura no variaba. El alumnado podrá aprender a programar con diferentes tipos de planos y resolviendo las limitaciones del propio brazo robótico.

Para ello, se va a realizar una soldadura alrededor de un codo a 45º, en este caso la tubería es de PVC, recuerdo al lector de este proyecto que nos encontramos ante una simulación. El recorrido que realizara la soldadura viene destacada en rojo en la Figura 76.

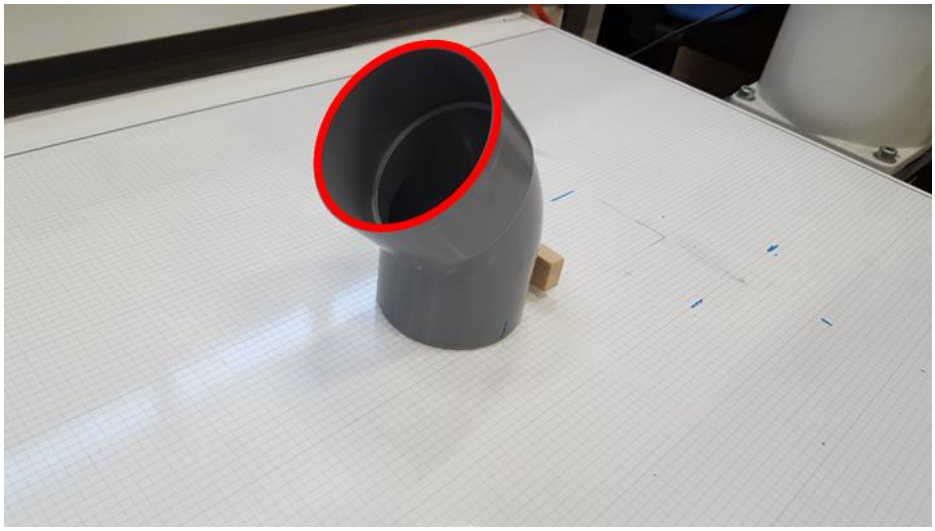


Figura 76.Soldadura 2. Tubería a soldar.

Figura 77.Soldadura 2, Tubería a soldar 2.

Para facilitar el agarre de la pinza que tenemos en el brazo robótico, se le ha adherido al codo una pieza de madera como se puede apreciar en la Figura 77, para que este pueda manipularlo con facilidad.

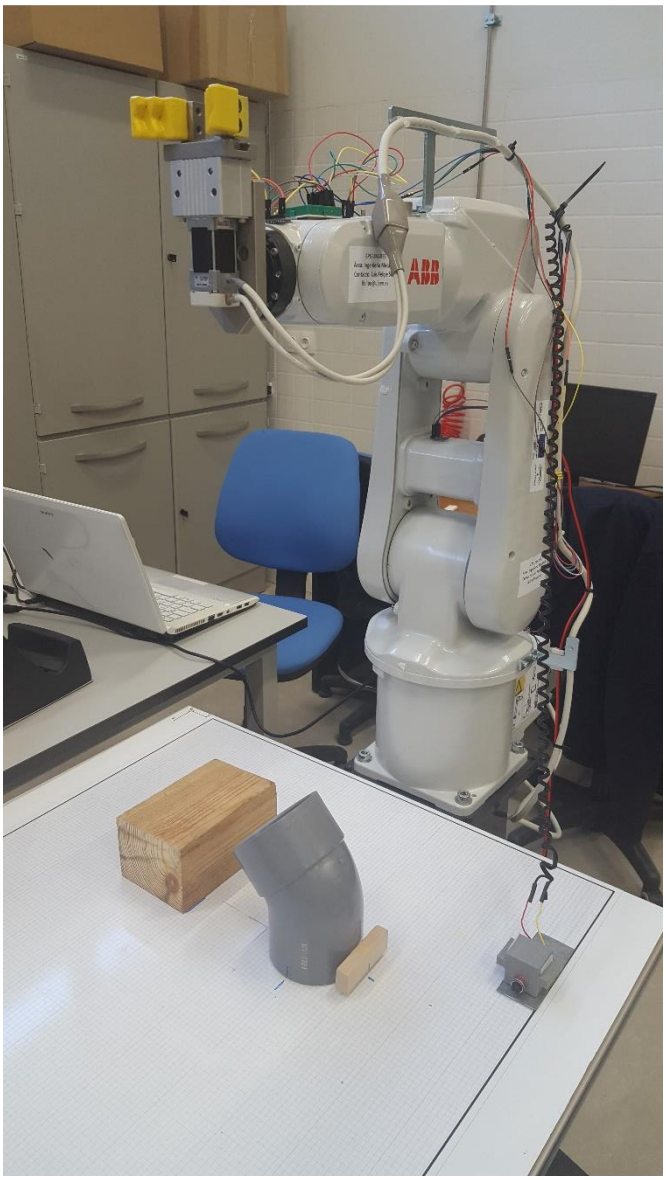


Figura 78. Soldadura 2. .

Para realizar esta soldadura, necesitamos elevar un poco el codo para evitar las limitaciones de nuestro brazo robótico, ya que la pinza podría chocar con la mesa de trabajo. En estas fotografías se podrá comprobar que el cable que une nuestra herramienta para realizar la soldadura ha cambiado, esto se debe a que se mejoró el diseño a un cable tipo teléfono, ya que quedaba más recogido y facilitaba el manejo del brazo robótico. Impidiendo así que los cables interfieran.

El operario se encargaría de colocar las piezas en sus respectivas ubicaciones y le colocaría la tapadera a la tubería como se muestra en la Figura 78.

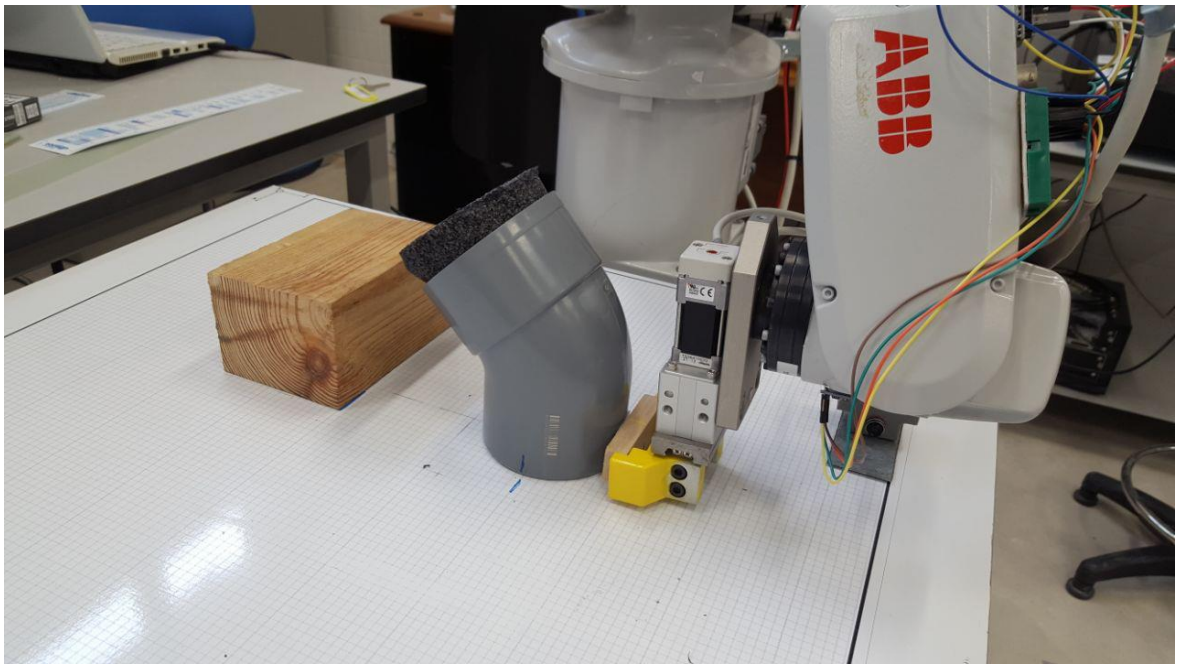


Figura 79. Soldadura 2. Brazo robótico iniciando la colocación de la pieza.

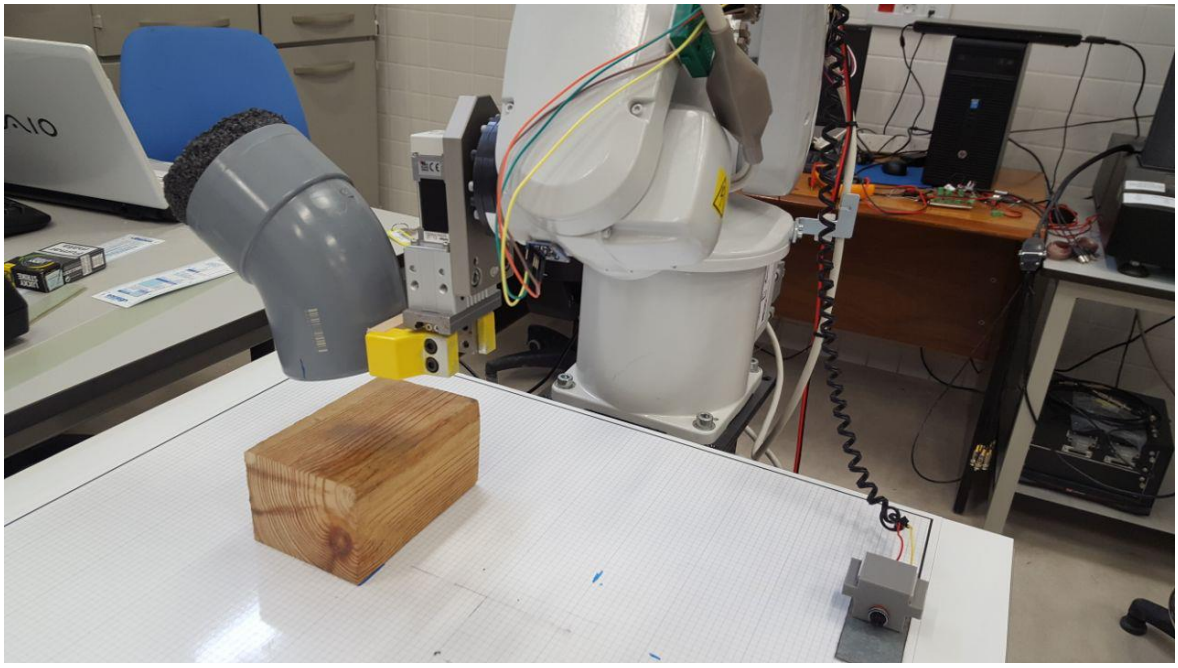


Figura 80. Soldadura 2. Colocación de la pieza en el altillo.

La primera instrucción que recibe el robot es la de elevar el codo hasta la plataforma, en el anexo se puede encontrar las instrucciones realizadas. Ya que de esta forma el robot puede realizar la soldadura sin que el robot choque contra la mesa. (Figura 79,Figura 80)

Una vez que la pieza está colocada, el robot procedería a coger la herramienta de simulación para proceder a soldar.

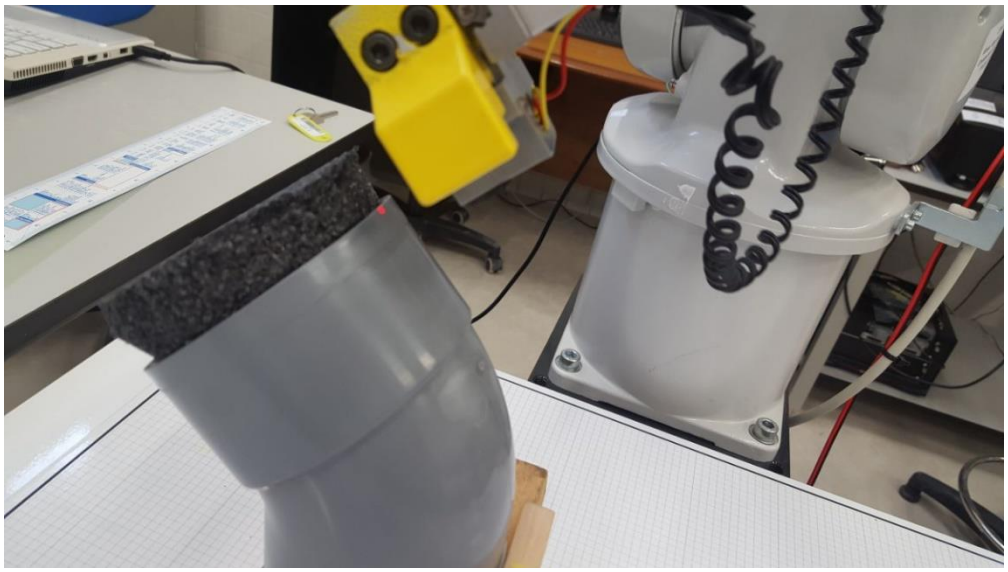


Figura 81. Soldadura 2. Inicio de la soldadura.

Figura 82. Soldadura 2. Medio recorrido de la soldadura.

Figura 83. Soldadura 2. Final del recorrido de la primera parte a soldar.

las Figura 81, Figura 82 y Figura 83 muestran el proceso de soldadura que solo abarca 180 grados de los 360 posibles, ya que no se puede realizar completamente por las propias limitaciones del robot. Por lo que será necesario rotar la pieza para que el robot termine. Antes de rotar la pieza, será necesario depositar la herramienta.

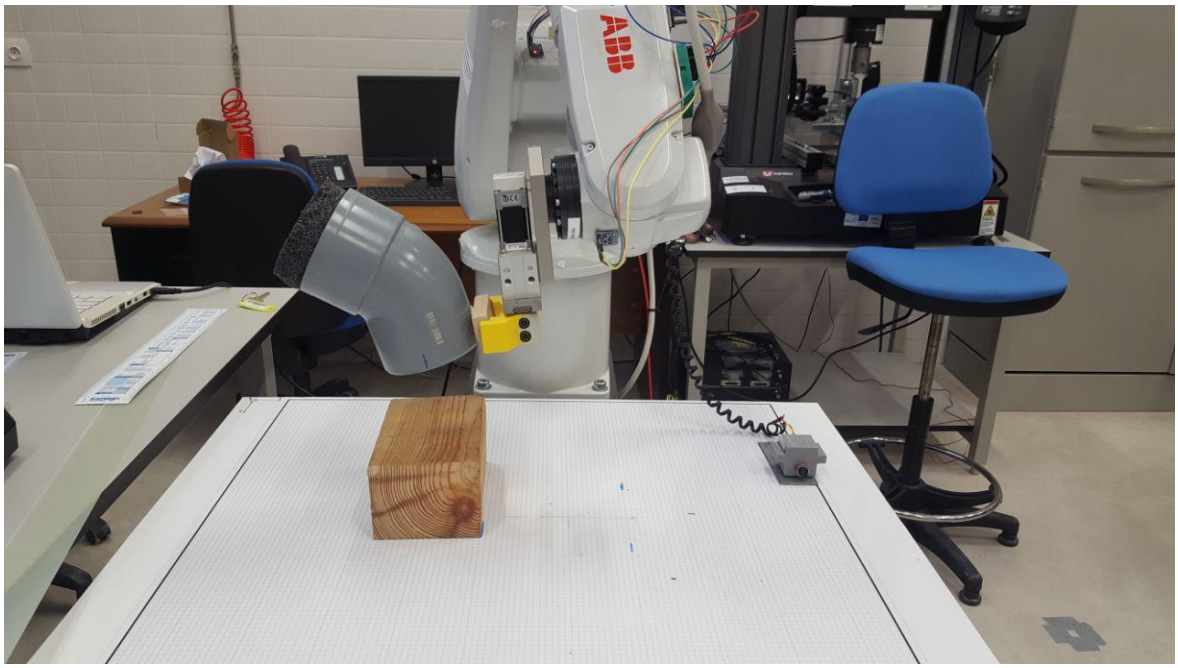


Figura 84. Soldadura 2. Levantamiento de pieza para darle la vuelta.

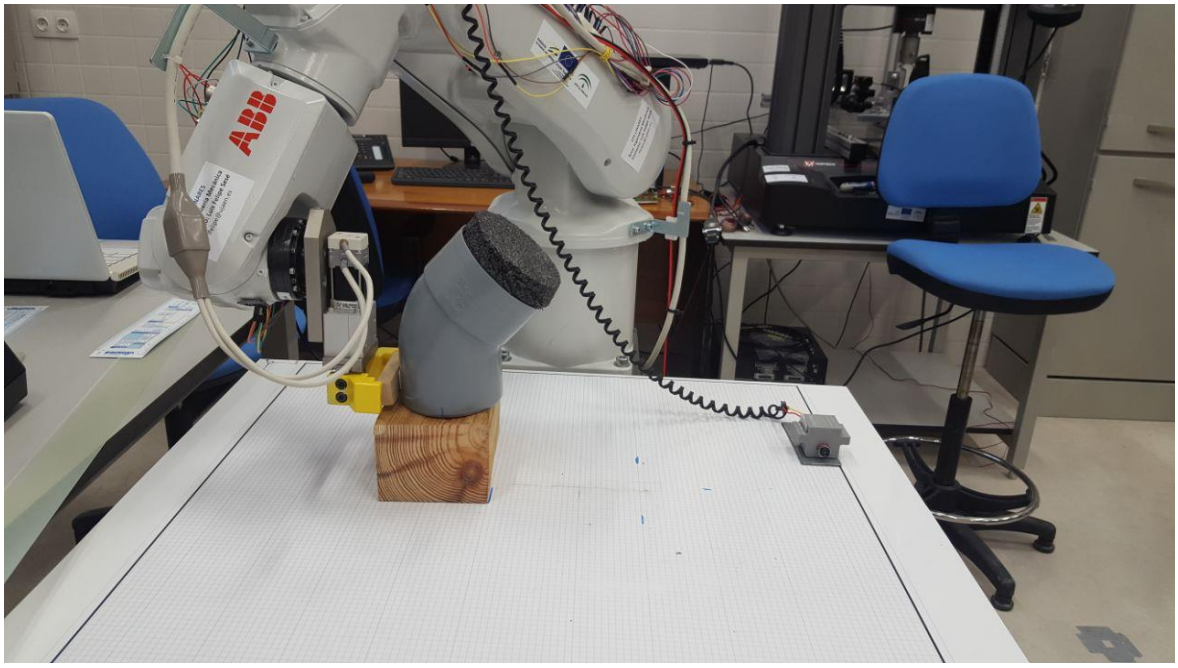


Figura 85. Soldadura 2. Rotación de la pieza 180º.

Una vez ubicada la pieza tal y como se muestra en la Figura 85, se puede proceder a terminar la soldadura.

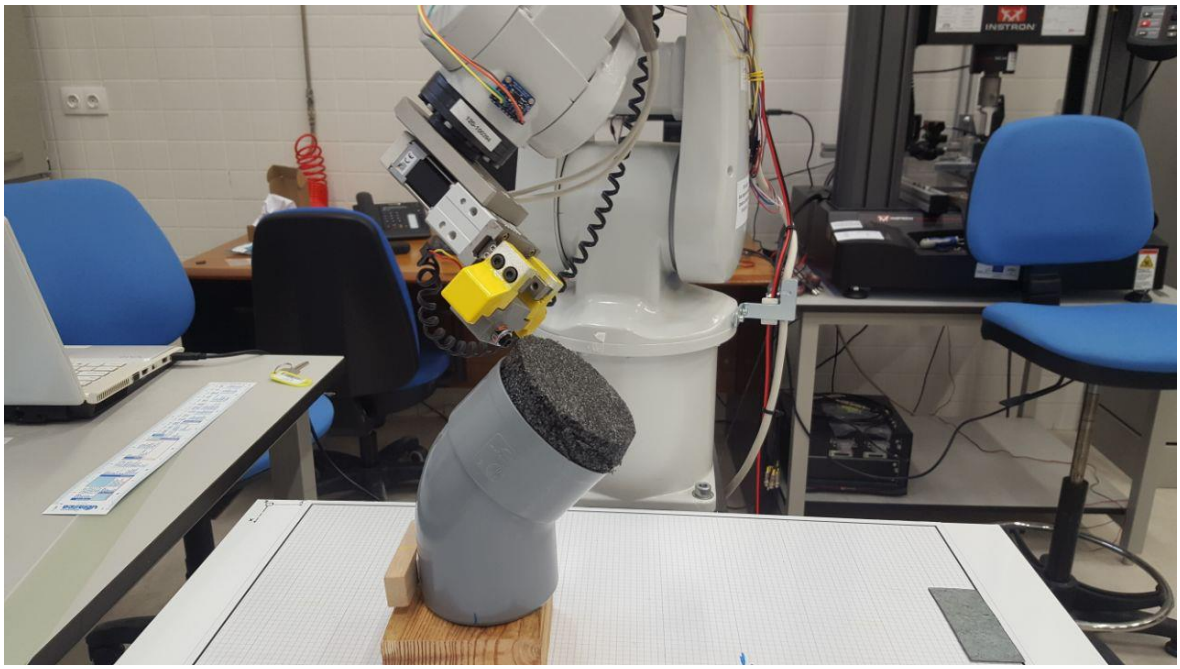


Figura 86. Soldadura 2. Inicio de la segunda vuelta de soldadura.

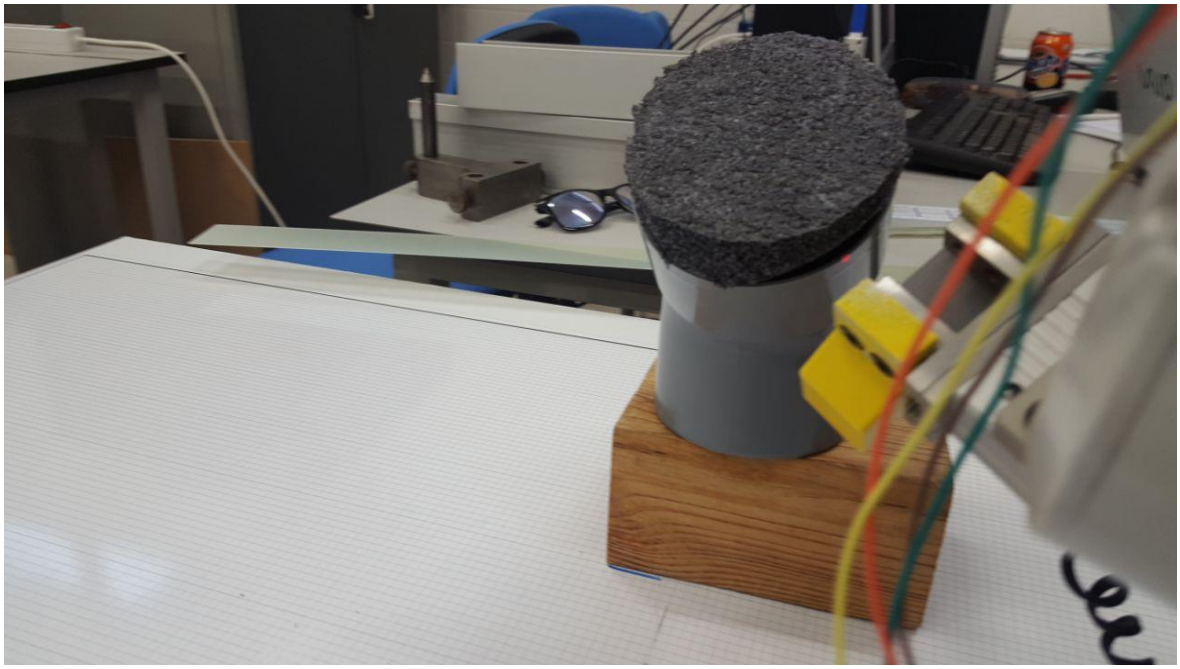


Figura 87. Soldadura 2. Láser incidiendo en la pieza completando el proceso de soldadura.

Como muestra la Figura 86 y Figura 87, el robot termina la soldadura describiendo el mismo movimiento anterior pero de forma inversa. Finalmente el robot colocaría la herramienta en su lugar de origen y sería el operario el encargado de reitrar la pieza ya terminada y colocar la nueva pieza en la ubicación inicial.

8 DISCUSIÓN Y CONCLUSIONES

Mediante este trabajo final de grado se ha desarrollado un sistema de sensores y una herramienta informática que permite la monitorización de las variables cinemáticas de velocidad angular, velocidad lineal y aceleración lineal de un brazo robótico. Todo ello es gestionado mediante una interfaz sencilla que permite grabar todos esos datos.

Los resultados de la interfaz han sido satisfactorios, ya que las velocidades comandadas por el programa coincidían con los de la herramienta. Como pudimos apreciar en la Figura 49 la herramienta alcanzaba los 1000 mm/s.

Al poder comparar las gráficas de velocidad y aceleración, se pudo realizar un análisis más detallado del funcionamiento del robot, pudiendo observar en que momento el robot realiza las acerelaciones y obteniendo los resultados esperados, aunque se ha podido apreciar que estas gráficas de aceleración son más ruidoas al aumentar la velocidad. Mencionando los errores que se han podido encontrar en la GUI podemos encontrar, que los datos obtenidos por el acelerómetro ADXL345 no han sido satisfactorios, ya que al no tener giróscopo no se le puede aplicar el filtro complementario aumentado a cada paso el error. En los trabajos futuros se aportarán propuestas para mejorar los resultados. La GUI también incorpora la posibilidad de calcular la posición y sus quaternios, pero se han tenido márgenes de error bastante altos de más del 300% para variaciones menores de 10º. Se ha intentado investigar las posibles causas de este desajuste tan grande pero no ha sido posible encontrarlo.

Adicionalmente, se han propuesto dos simulaciones de aplicaciones industriales. Estás simulaciones han tenido que realizarse solventando las limitaciones del propio brazo robótico ya que en más de una ocasión al realizar la programación el autor se ha encontrado con choques de eslabones o que la pieza se encuentre en los límites del espacio de trabajo. Por consiguiente, el proceso de soldadura abarca un tiempo demasiado amplio, ya que requiere realizar la soldadura en dos turnos, que se podría acortar bastante teniendo un robot especializado para la tarea. Pero estas simulaciones podrían acercar al alumnado de la Escuela Politécnica superior de Linares a entender conceptos de entradas y salidas y ver las posibilidades que estas ofrecen, también como lidiar con las limitaciones que se tienen y que al final realice el proceso.

También se ha pretendido acercar la plataforma Arduino a los Ingenieros Mecánicos ya que puede llegar a ser una herramienta muy útil pero que no se imparte en ninguna asignatura. El uso de esta herramienta ha ayudado a aumentar los conocimientos sobre la sensorización y como emplear los datos (programando Arduino) para pasar de un dato bruto a un valor real.

9 TRABAJOS FUTUROS

En los siguientes apartados se presentarán propuestas de trabajos futuros.

9.1 Aumentar el número de sensores

Unos de los problemas ha sido que el ADXL no está a la altura para la realización de este trabajo, por lo que se invita a dotar al robot de más sensores MPU6050 y se puedan obtener resultados más fiables. Para ello necesitara dotar al sensor de dos sensores más. Para solventar el problema que tiene el protocolo de comunicación I2C que no permite el acceso a más de dos sensores será necesario un chip multiplicador de entradas I2C como el de la Figura 88.

Figura 88. I2C multiplexor chip.

9.2 Utilización de las E/S

En este trabajo se ha utilizado una salida de las que ofrece el IRC5 para la simulación de una soldadura. Pero esto es solo el principio que pueden ofrecernos este tipo de conexiones.

Una posible incorporación para las entradas y salidas, podría ser de dotar al robot de una cinta de transporte. Donde el robot arrancara para mover las piezas y cogerlas. Dotando a la cinta de un final de carrera, cuando detecte un objeto, este le enviara la orden al robot de cogerlo y colocarlo. O dotar al robot de sensores de proximidad para que este se pare cuando detecte un objeto y no se choque. El uso de entradas y salidas abre un abanico de posibilidades muy amplio.

9.3 Dinámica de robot

Se puede realizar un estudio de la dinámica del robot, que el robot ya tiene implantado los acelerómetros, se puede realizar este tipo de estudios.

10 BIBLIOGRAFÍA

[1] **Barrientos, A.** *Fundamentos de robótica.* Madrid : McGraw, 2007.

[2] **Wordpress.** Wordpress. [En línea] https://singularidadpara7000millones.wordpress.com/tag/robots-industriales/. [Fecha de consulta: 15/03/2017]

[3] **Groover, M.** *Robótica Industrial.* Madrid : McGraw-Hill, 1989.

[4] **Platea.** Platea. [En línea] http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/sistema/morfologia.htm. [Fecha de consulta: 10/02/2017]

[5] **J.M., Seling.** *Introductory Robotics.* s.l. : Prentice Hall.

[6] **Carrillo, Antonio Díaz.** *Cinemática.*

[7] **Group, ABB.** *Curso Básico Robot IRC5.* 2015.

[8] **Córdoba López, Axel José.** *Puesta en marcha brazo robótico y desarrollo de aplicaciones.* Linares : s.n., 2016.

[9] **Crespo, Jose Enreique.** Aprendiendo Arduino. [En línea] https://aprendiendoarduino.wordpress.com/2015/03/23/shields-para-arduino/.[Fecha de consulta: 27/03/2017]

[10] **Aprendiendo Arduino.** Aprendiendo Arduino. [En línea] https://aprendiendoarduino.wordpress.com/2015/03/23/shields-para-arduino/.[Fecha de consulta: 04/05/2017]

[11] **White, R.M.** *A Sensor Classification Scheme.*

[12] **5hertz.** 5hertz. [En línea] http://5hertz.com/tutoriales/?p=228. [Fecha de consulta: 10/11/2016]

[13] **Pedley, Mark.** *Tilt Sensing Using a Three-Axis.* s.l. : NXP, 2013. AN3461.

[14] **Arduino.** Arduino. [En línea] https://www.arduino.cc/.[Fecha de consulta: 15/02/2017]

[15] **Lozano Equisoain, Daniel.** *Arduino Práctico.* Madrid : Grupo Anaya, 2017.

[16] **Group, ABB.** *Especificaciones de producto IRB 120.*

[17] **ABB.** ABB. [En línea] http://new.abb.com/es. [Fecha de consulta: 20/02/2017]

[18] **SMC`sCorporate.** *“Catalogo pinzas eléctricas CAT.EUS100-77E-ES".*

[19] **InvenSense Inc.** *MPU-6000 and MPU-6050.* California : s.n., 2013. PS-MPU-6000A-00.

[20] **ANALOGDEVICES.** *Digital Accelerometer ADXL345.* s.l. : Trademaks, 2009.

[21] **García, Ángel Franco.** Física con ordenador. [En línea] http://www.sc.ehu.es/sbweb/fisica/cinematica/circular/circular.htm. [Fecha de consulta: 03/04/2017]

[22] **Ferraté, G.** *Robótica Industrial.* Barcelona : Marcombo, 1986.

[23] **ingenio, electro.** Electro ingenio. [En línea] http://www.electroingenio.com/arduino/tipos-de-sensores-en-modulos-arduino/.[Fecha de consulta: 16/04/2017]

[24] **Fisica con ordenador.** [En línea] http://www.sc.ehu.es/sbweb/fisica/cinematica/circular/circular.htm. [Fecha de consulta: 16/04/2017]

11 ANEXOS

En este apartado se presentarán los códigos utilizados en la realización de este trabajo fin de grado tanto de Arduino, MatLab y RAPID.

11.1 Programas Arduino.

Primero se mostrarán los programas de calibración de los sensores y por último el código utilizado para obtener la inclinación en grados.

11.1.1 Calibración MPU-6050.

Programa utilizado para realizar la calibración de los dos sensores MPU6050, destacar que para realizar esta calibración primero se realizará para un sensor y a continuación será necesario modificar la línea 9, introduciéndole un comentario (//) y descomentando la línea 10.

Código:

```
/ Librerias I2C para controlar el mpu6050
// la libreria MPU6050.h necesita I2Cdev.h, I2Cdev.h necesita Wire.h
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

// La dirección del MPU6050 puede ser 0x68 o 0x69, dependiendo
// del estado de AD0. Si no se especifica, 0x68 estará implicito
MPU6050 sensor (0x69);
//MPU6050 sensor (0x68) //hay que realizar la calibración dos veces una p
ara cada sensor
// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x
,y,z
int ax, ay, az;
int gx, gy, gz;

//Variables usadas por el filtro pasa bajos
long f_ax,f_ay, f_az;
int p_ax, p_ay, p_az;
long f_gx,f_gy, f_gz;
int p_gx, p_gy, p_gz;
int counter=0;

//Valor de los offsets
int ax_o,ay_o,az_o;
int gx_o,gy_o,gz_o;

void setup() {
  Serial.begin(57600);   //Iniciando puerto serial
  Wire.begin();           //Iniciando I2C
  sensor.initialize();    //Iniciando el sensor

  if (sensor.testConnection()) Serial.println("Sensor iniciado
correctamente");

  // Leer los offset los offsets anteriores
  ax_o=sensor.getXAccelOffset();
  ay_o=sensor.getYAccelOffset();
  az_o=sensor.getZAccelOffset();
```

```
  gx_o=sensor.getXGyroOffset();
  gy_o=sensor.getYGyroOffset();
  gz_o=sensor.getZGyroOffset();

  Serial.println("Offsets:");
  Serial.print(ax_o); Serial.print("\t");
  Serial.print(ay_o); Serial.print("\t");
  Serial.print(az_o); Serial.print("\t");
  Serial.print(gx_o); Serial.print("\t");
  Serial.print(gy_o); Serial.print("\t");
  Serial.print(gz_o); Serial.print("\t");
  Serial.println("\nEnvie cualquier caracter para empezar la
calibracion\n");
  // Espera un caracter para empezar a calibrar
  while (true){if (Serial.available()) break;}
  Serial.println("Calibrando, no mover IMU");

}

void loop() {
  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  // Filtrar las lecturas
  f_ax = f_ax-(f_ax>>5)+ax;
  p_ax = f_ax>>5;

  f_ay = f_ay-(f_ay>>5)+ay;
  p_ay = f_ay>>5;

  f_az = f_az-(f_az>>5)+az;
  p_az = f_az>>5;

  f_gx = f_gx-(f_gx>>3)+gx;
  p_gx = f_gx>>3;

  f_gy = f_gy-(f_gy>>3)+gy;
  p_gy = f_gy>>3;

  f_gz = f_gz-(f_gz>>3)+gz;
  p_gz = f_gz>>3;

  //Cada 100 lecturas corregir el offset
  if (counter==100){
    //Mostrar las lecturas separadas por un [tab]
    Serial.print("promedio:"); Serial.print("\t");
    Serial.print(p_ax); Serial.print("\t");
    Serial.print(p_ay); Serial.print("\t");
    Serial.print(p_az); Serial.print("\t");
    Serial.print(p_gx); Serial.print("\t");
    Serial.print(p_gy); Serial.print("\t");
    Serial.println(p_gz);

    //Calibrar el acelerometro a 1g en el eje y (ajustar el offset)
    if (p_ax>0) ax_o--;
    else {ax_o++;}
    if (p_ay>0) ay_o--;
    else {ay_o++;}
    if (p_az-16384>0) az_o--;
    else {az_o++;}
```

```
    sensor.setXAccelOffset(ax_o);
    sensor.setYAccelOffset(ay_o);
    sensor.setZAccelOffset(az_o);

    //Calibrar el giroscopio a 0°/s en todos los ejes (ajustar el offset)
    if (p_gx>0) gx_o--;
    else {gx_o++;}
    if (p_gy>0) gy_o--;
    else {gy_o++;}
    if (p_gz>0) gz_o--;
    else {gz_o++;}

    sensor.setXGyroOffset(gx_o);
    sensor.setYGyroOffset(gy_o);
    sensor.setZGyroOffset(gz_o);

    counter=0;
  }
  counter++;
}
```

11.1.2 Calibración ADXL 345.

Este es el programa utilizado para realizar la calibración del ADXL345, destacar que necesitaremos obtener los valores de los offset de los sensores y posteriormente cambiarlos en el programa de ejecución.

Código:

```
#include <Wire.h>
#include "ADXL345lib.h"
#include <math.h>

#define SAMPLESIZE 15

Accelerometer acc;

int8_t gatherAverage(int16_t * average_x, int16_t * average_y, int16_t *
average_z)
{
    Serial.println("Gathering Average...");

    int32_t accum_x = 0, accum_y = 0, accum_z = 0;

    for (uint8_t i = 0; i < SAMPLESIZE; ++i)
    {
        int16_t x, y, z;

        if (acc.readRaw(&x, &y, &z) != 0)
        {
            Serial.println("Failed to read sensor. End.");
            return -1;
        }

        accum_x += x;
        accum_y += y;
        accum_z += z;

        // sensor runs at 25hz by default
```

```
        // wait atleast 1/25th of a second before attempting
        // another read or we will read the same value
        delay(50);
    }

    // average values
    *average_x = accum_x / SAMPLESIZE;
    *average_y = accum_y / SAMPLESIZE;
    *average_z = accum_z / SAMPLESIZE;


    Serial.println("Averages Gathered:");
    Serial.print("X: ");
    Serial.print(*average_x);
    Serial.print(" Y: ");
    Serial.print(*average_y);
    Serial.print(" Z: ");
    Serial.println(*average_z);

    return 0;
}

void setup()
{
    Serial.begin(57600);

    // set the I2C address of the accelerometer
    //
    // With the OSEPP Accelerometer use:
    //      OSEPP_ACC_SW_ON    - Switch is in the ON position
    //      OSEPP_ACC_SW_OFF   - Switch is in the OFF position
    //
    // begin will return 0 on success
    if (acc.begin(OSEPP_ACC_SW_ON) != 0)
    {
        Serial.println("Error connecting to accelerometer");
        return;
    }

    // use the most sensitive mode for calibration
    // at this sensitivity we have 256 values / g
    acc.setSensitivity(ADXL345_RANGE_PM2G);
    // clear any existing offsets
    acc.setOffsets(0, 0, 0);


    Serial.println("This example will calculate offset values");
    Serial.println("to calibrate the accelerometer. Record");
    Serial.println("these values and enter them in your own");
    Serial.println("projects using this library.");
    Serial.println();

    Serial.println("Firmy hold the accelerometer upside down on a flat");
    Serial.println("surface and type \"k\" when ready.");
    Serial.println();


    while (true)
    {
        if (Serial.available())
        {
```

```
            char inByte = Serial.read();

            if (inByte == 'k' || inByte == 'K')
                break;
        }
    }

    int16_t average_x, average_y, average_z;

    if (gatherAverage(&average_x, &average_y, &average_z) != 0)
        return;

    Serial.println();
    Serial.println("Cacluating Offsets...");

    int8_t offset_x, offset_y, offset_z;

    // we have to divide by four because the offsent sensitivity
    // is 15.6mg vs 3.9mg of the 2g sensitivity
    offset_x = 0 - round((double)average_x / 4);
    offset_y = 0 - round((double)average_y / 4);
    offset_z = 0 - round((double)(average_z + 256) / 4);

    Serial.println("Save these values for use in your program:");
    Serial.print("X: ");
    Serial.print(offset_x);
    Serial.print(" Y: ");
    Serial.print(offset_y);
    Serial.print(" Z: ");
    Serial.println(offset_z);


    Serial.println();
    Serial.print("Setting Offsets with \"acc.setOffsets(");
    Serial.print(offset_x);
    Serial.print(", ");
    Serial.print(offset_y);
    Serial.print(", ");
    Serial.print(offset_z);
    Serial.print(");\"");
    Serial.println();
    Serial.println();

    acc.setOffsets(offset_x, offset_y, offset_z);

    if (gatherAverage(&average_x, &average_y, &average_z) != 0)
        return;
}

void loop()
{

}
```

11.1.3 Programa de ejecución

Mostrará el ángulo de inclinación de cada eslabón. Esta información pasara a MatLab para procesar los datos.

Código:

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
#include "ADXL345lib.h"
#include <ADXL345.h>

MPU6050 sensor1 (0x68);
MPU6050 sensor2 (0x69);
ADXL345 accel;

// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x
,y,z
int ax1, ay1, az1;
int gx1, gy1, gz1;
int ax2, ay2, az2;
int gx2, gy2, gz2;

unsigned long time;
long tiempo_prev;
float dt;
float ang_x, ang_y;
float ang_x_prev, ang_y_prev;
float ang_x2, ang_y2;
float ang_x_prev2, ang_y_prev2;
float RadianX, RadianY;
float pos, giro;
float gz_prev,giro_prev;

void setup() {
  Serial.begin(57600);    //Iniciando puerto serial
  Wire.begin();           //Iniciando I2C
  sensor1.initialize();    //Iniciando el sensor 1
  sensor2.initialize();    //Iniciando el sensor 2
  accel.init(-6, -3, -118); //Iniciando el sensor 3


}

void loop() {
  // Leer las aceleraciones y velocidades angulares
  sensor1.getAcceleration(&ax1, &ay1, &az1);
  sensor1.getRotation(&gx1, &gy1, &gz1);
  sensor2.getAcceleration(&ax2, &ay2, &az2);
  sensor2.getRotation(&gx2, &gy2, &gz2);

  AccelRotation accelRot;
  accelRot = accel.readPitchRoll();
  dt = (millis()-tiempo_prev)/1000.0;
  tiempo_prev=millis();
  time = millis();
  //Calcular los ángulos con acelerometro
  //float accel_ang_x=atan(ay1/sqrt(pow(ax1,2) +
pow(az1,2)))*(180.0/3.14159265359);
  //float accel_ang_y=atan(-ax1/sqrt(pow(ay1,2) +
pow(az1,2)))*(180.0/3.14159265359);

  // 25 - 26
  float accel_ang_x  = atan2(ay1, az1) * (180.0/3.14159265359);
```

```
  float accel_ang_y  = atan(-ax1 / sqrt(pow(ay1,2) +
pow(az1,2)))*(180.0/3.14159265359);

  //28 - 29
  //float accel_ang_x  = atan(ay1 / sqrt(pow(ax1,2) + pow(az1,2)) *
(180.0/3.14159265359);
  //float accel_ang_y  = (-ax1, az1) * (180.0/3.14159265359);


  //Calcular angulo de rotación con giroscopio y filtro complemento
  ang_x = 0.97*(ang_x_prev+(gx1/131)*dt) + 0.03*accel_ang_x;
  ang_y = 0.97*(ang_y_prev+(gy1/131)*dt) + 0.03*accel_ang_y;

  ang_x_prev=ang_x;
  ang_y_prev=ang_y;

  //float accel_ang_x2=atan(ay2/sqrt(pow(ax2,2) +
pow(az2,2)))*(180.0/3.14159265359);
//  float accel_ang_y2=atan(-
ax2/sqrt(pow(ay2,2) + pow(az2,2)))*(180.0/3.14159265359);
// 25 - 26
  float accel_ang_x2  = atan2(ay2, az2) * (180.0/3.14159265359);
  float accel_ang_y2  = atan(-ax2 / sqrt(pow(ay2,2) +
pow(az2,2)))*(180.0/3.14159265359);

  //28 - 29
  //float accel_ang_x  = atan(ay2 / sqrt(pow(ax2,2) + pow(az2,2)) *
(180.0/3.14159265359);
  //float accel_ang_y  = (-ax2, az2) * (180.0/3.14159265359);
  //Calcular angulo de rotación con giroscopio y filtro complemento
  ang_x2 = 0.97*(ang_x_prev2+(gx2/131)*dt) + 0.03*accel_ang_x2;
  ang_y2 = 0.97*(ang_y_prev2+(gy2/131)*dt) + 0.03*accel_ang_y2;

  ang_x_prev2=ang_x2;
  ang_y_prev2=ang_y2;
  // Cálculo de la posición 1 a través del giroscopio
   giro = (gz1/131)*dt + giro_prev;
  gz_prev = gz1;
  giro_prev = giro;
  //Mostrar las lecturas separadas por un [tab]
  //Serial.print("a[x y z](m/s2)\t");

  Serial.print(time);Serial.print("\t");

  Serial.print(giro);Serial.print("\t");
  Serial.print(ang_x);Serial.print("\t");
 // Serial.print(ang_y);Serial.print("\t");


  Serial.print(ang_x2);Serial.print("\t");
  Serial.print(ang_y2);Serial.print("\t");

 // Serial.print(accelRot.pitch);Serial.print("\t");
  Serial.println(accelRot.roll);Serial.print("\t");


  delay(10);
}
```

11.2 MatLab

En este apartado se presentarán los programas utilizados por MatLab.

11.2.1 Interfaz

Interfaz que mostrara las gráficas de velocidad y aceleración de cada eslabón del brazo robótico ABB IRB 120.

Código:

```
function varargout = InterfazV03(varargin)
% INTERFAZV03 MATLAB code for InterfazV03.fig
%      INTERFAZV03, by itself, creates a new INTERFAZV03 or raises the existing
%      singleton*.
%
%      H = INTERFAZV03 returns the handle to a new INTERFAZV03 or the handle
to
%      the existing singleton*.
%
%      INTERFAZV03('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in INTERFAZV03.M with the given input
arguments.
%
%       INTERFAZV03('Property','Value',...) creates a new INTERFAZV03 or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before InterfazV03_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to InterfazV03_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES


% Edit the above text to modify the response to help InterfazV03


% Last Modified by GUIDE v2.5 03-Mar-2017 11:50:34


% Begin initialization code - DO NOT EDIT
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @InterfazV03_OpeningFcn, ...
                   'gui_OutputFcn',  @InterfazV03_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before InterfazV03 is made visible.
function InterfazV03_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to InterfazV03 (see VARARGIN)

% Choose default command line output for InterfazV03
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);



axes(handles.axes7)
handles.imagen=imread('logo_ujaen.jpg');
```

```
imagesc(handles.imagen)
axis off
% UIWAIT makes InterfazV03 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = InterfazV03_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
delete(instrfind({'port'},{'COM3'}));
baudrate = 57600;
global s T w1 w2 w3 w4 w5 v1 v2 v3 v4 v5 A1 A2 A3 A4 A5 yw1 yw2 yw3 yw4 yw5
yv1 yv2 yv3 yv4 yv5 yA1 yA2 yA3 yA4 yA5 theta1 theta2 theta3 theta4 theta5 theta6;
theta1 = [];
theta2 = [];
theta3 = [];
theta4 = [];
theta5 = [];
theta6 = 0;
T = [];
vm = [];
va = [];
alpham = [];
alphaa = [];

Am = [];
```

```
Aa = [];
wm = [];
wa = [];
Anm = [];
Ana = [];
theta3C = [];
n = 1;
s = serial('com3','BaudRate',baudrate); % change the COM Port number as needed
fopen(s)
t = str2num(char(get(handles.edit1,'String')));
tic;
while toc<=t
AR = fscanf (s, '%E\t');
T(n) = AR(1)/1000;
theta1(n) = AR(2);
theta2(n) = AR(3);
theta3(n) = AR(4);
theta4(n) = AR(5);
theta5(n) = AR(6);
n=n+1;
toc;
end
fclose(s);
% for x = 1:(n*0.5)
%     theta2(x) = 0;
%     theta3(x) = 0;
% end
for x = 1:n-1
   theta3C(x) = theta2(x) - theta3(x);
   theta5C(x) = theta3C(x) - theta5(x);
end
fclose(s);
%%  Velocidad ángular
for x = 2:n-1
   w1(x) = ((theta1(x)-theta1(x-1))/(T(x)-T(x-1)))*0.0174533;
   w2(x) = ((theta2(x)-theta2(x-1))/(T(x)-T(x-1)))*0.0174533;
   w3(x) = ((theta3C(x)-theta3C(x-1))/(T(x)-T(x-1)))*0.0174533;
```

```
    w4(x) = ((theta4(x)-theta4(x-1))/(T(x)-T(x-1)))*0.0174533;
    w5(x) = ((theta5C(x)-theta5C(x-1))/(T(x)-T(x-1)))*0.0174533;
end
%%  Velocidad lineal
%distancias de los eslabones
m1 = 0.090;
m2 = 0.270;
m3 = 0.302;
m4 = 0.070;
m5 = 0.072;
v1 = w1 * m1;
v2 = w2 * m2;
v3 = w3 * m3;
v4 = w4 * m4;
v5 = w5 * m5;
%%  Aceleración
An1 = w1.^2 * m1;
An2 = w2.^2 * m2;
An3 = w3.^2 * m3;
An4 = w4.^2 * m4;
An5 = w5.^2 * m5;
for x = 2:n-1
    alpha1(x) = ((w1(x)-w1(x-1))/(T(x)-T(x-1)))* m1;
    alpha2(x) = ((w2(x)-w2(x-1))/(T(x)-T(x-1)))* m2;
    alpha3(x) = ((w3(x)-w3(x-1))/(T(x)-T(x-1)))* m3;
    alpha4(x) = ((w4(x)-w4(x-1))/(T(x)-T(x-1)))* m4;
    alpha5(x) = ((w5(x)-w5(x-1))/(T(x)-T(x-1)))* m5;
end
A1 = An1 + alpha1;
A2 = An2 + alpha2;
A3 = An3 + alpha3;
A4 = An4 + alpha4;
A5 = An5 + alpha5;
```

```
%% Flitrado de datos
d1 = designfilt('lowpassiir','FilterOrder',12, ...
   'HalfPowerFrequency',0.15,'DesignMethod','butter');
yw1 = filtfilt(d1,w1);
yw2 = filtfilt(d1,w2);
yw3 = filtfilt(d1,w3);
yw4 = filtfilt(d1,w4);
yw5 = filtfilt(d1,w5);
yv1 = filtfilt(d1,v1);
yv2 = filtfilt(d1,v2);
yv3 = filtfilt(d1,v3);
yv4 = filtfilt(d1,v4);
yv5 = filtfilt(d1,v5);
yA1 = filtfilt(d1,A1);
yA2 = filtfilt(d1,A2);
yA3 = filtfilt(d1,A3);
yA4 = filtfilt(d1,A4);
yA5 = filtfilt(d1,A5);


function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
%% Velocidad Angular
global T w1 w2 w3 w4 w5 popup_sel_index;
```

```
if get(handles.checkbox1,'Value') == 1
    switch popup_sel_index
        case 1
            plot(handles.axes1,T,w1);
            hold(handles.axes1);
        case 2
            plot(handles.axes1,T,w2);
            hold(handles.axes1);
        case 3
            plot(handles.axes1,T,w3);
            hold(handles.axes1);
        case 4
            plot(handles.axes1,T,w4);
            hold(handles.axes1);
        case 5
            plot(handles.axes1,T,w5);
            hold(handles.axes1);
    end
title(handles.axes1,'Velocidad Angular')
xlabel(handles.axes1,'Tiempo')
ylabel(handles.axes1,'rad/s')
end

function checkbox2_Callback(hObject, eventdata, handles)
%% Velocidad Lineal
global T v1 v2 v3 v4 v5 popup_sel_index;
if get(handles.checkbox2,'Value') == 1
    switch popup_sel_index
        case 1
            plot(handles.axes1,T,v1);
            hold(handles.axes1);
        case 2
            plot(handles.axes1,T,v2);
            hold(handles.axes1);
        case 3
            plot(handles.axes1,T,v3);
            hold(handles.axes1);
```

```
        case 4
            plot(handles.axes1,T,v4);
            hold(handles.axes1);
        case 5
            plot(handles.axes1,T,v5);
            hold(handles.axes1);
    end
title(handles.axes1,'Velocidad')
xlabel(handles.axes1,'Tiempo')
ylabel(handles.axes1,'m/s')
end

function checkbox3_Callback(hObject, eventdata, handles)
global T yw1 yw2 yw3 yw4 yw5 popup_sel_index;
if get(handles.checkbox3,'Value') == 1
    switch popup_sel_index
        case 1
            plot(handles.axes1,T,yw1,'LineWidth',2);
            hold(handles.axes1);
        case 2
            plot(handles.axes1,T,yw2,'LineWidth',2);
            hold(handles.axes1);
        case 3
            plot(handles.axes1,T,yw3,'LineWidth',2);
            hold(handles.axes1);
        case 4
            plot(handles.axes1,T,yw4,'LineWidth',2);
            hold(handles.axes1);
        case 5
            plot(handles.axes1,T,yw5,'LineWidth',2);
            hold(handles.axes1);
    end
title(handles.axes1,'Velocidad Angular')
xlabel(handles.axes1,'Tiempo')
ylabel(handles.axes1,'rad/s')
end
```

```
% --- Executes on button press in checkbox4.
function checkbox4_Callback(hObject, eventdata, handles)
global T yv1 yv2 yv3 yv4 yv5 popup_sel_index;
if get(handles.checkbox4,'Value') == 1
    switch popup_sel_index
        case 1
            plot(handles.axes1,T,yv1,'LineWidth',2);
            hold(handles.axes1);
        case 2
            plot(handles.axes1,T,yv2,'LineWidth',2);
            hold(handles.axes1);
        case 3
            plot(handles.axes1,T,yv3,'LineWidth',2);
            hold(handles.axes1);
        case 4
            plot(handles.axes1,T,yv4,'LineWidth',2);
            hold(handles.axes1);
        case 5
            plot(handles.axes1,T,yv5,'LineWidth',2);
            hold(handles.axes1);
    end
title(handles.axes1,'Velocidad')
xlabel(handles.axes1,'Tiempo')
ylabel(handles.axes1,'m/s')
end


% --- Executes on button press in checkbox6.
function checkbox6_Callback(hObject, eventdata, handles)
global T yA1 yA2 yA3 yA4 yA5 popup_sel_index;
if get(handles.checkbox6,'Value') == 1
    switch popup_sel_index
        case 1
            plot(handles.axes1,T,yA1,'LineWidth',2);
            hold(handles.axes1);
```

```
        case 2
            plot(handles.axes1,T,yA2,'LineWidth',2);
            hold(handles.axes1);
        case 3
            plot(handles.axes1,T,yA3,'LineWidth',2);
            hold(handles.axes1);
        case 4
            plot(handles.axes1,T,yA4,'LineWidth',2);
            hold(handles.axes1);
        case 5
            plot(handles.axes1,T,yA5,'LineWidth',2);
            hold(handles.axes1);
    end
title(handles.axes1,'Aceleración')
xlabel(handles.axes1,'Tiempo')
ylabel(handles.axes1,'m/s^2')
end
% --- Executes on button press in checkbox5.
function checkbox5_Callback(hObject, eventdata, handles)
global T A1 A2 A3 A4 A5 popup_sel_index;
if get(handles.checkbox5,'Value') == 1
    switch popup_sel_index
        case 1
            plot(handles.axes1,T,A1);
            hold(handles.axes1);
        case 2
            plot(handles.axes1,T,A2);
            hold(handles.axes1);
        case 3
            plot(handles.axes1,T,A3);
            hold(handles.axes1);
        case 4
            plot(handles.axes1,T,A4);
            hold(handles.axes1);
        case 5
            plot(handles.axes1,T,A5);
            hold(handles.axes1);
```

```
    end
title(handles.axes1,'Aceleración')
xlabel(handles.axes1,'Tiempo')
ylabel(handles.axes1,'m/s^2')

end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
cla(handles.axes1);




% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
Arduino;


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
Imagen;


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
Tabla;


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
cinematica1;
```

```
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
quit;



% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
global popup_sel_index;
popup_sel_index = get(handles.popupmenu1,'Value');

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if                ispc                &&                isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

11.2.2 Botón Arduino

Abrirá el conjunto de códigos necesarios de Arduino para ejecutar el programa.

Código:

```
function varargout = Arduino(varargin)
% ARDUINO MATLAB code for Arduino.fig
%      ARDUINO, by itself, creates a new ARDUINO or raises the existing
%      singleton*.
%
%      H = ARDUINO returns the handle to a new ARDUINO or the handle to
%      the existing singleton*.
%
%      ARDUINO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ARDUINO.M with the given input arguments.
%
%      ARDUINO('Property','Value',...) creates a new ARDUINO or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Arduino_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Arduino_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Arduino

% Last Modified by GUIDE v2.5 19-Dec-2016 20:33:17

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Arduino_OpeningFcn, ...
                   'gui_OutputFcn',  @Arduino_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
```

```
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Arduino is made visible.
function Arduino_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Arduino (see VARARGIN)

% Choose default command line output for Arduino
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Arduino wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Arduino_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
winopen('tres.ino');


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
winopen('calibracionMPU.ino');


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
winopen('calibracionADXL.ino');
```

11.2.3 Botón Robot

Muestra la figura del robot, con todos sus ejes de coordenadas.

Código:

```
function varargout = Imagen(varargin)
% IMAGEN MATLAB code for Imagen.fig
%      IMAGEN, by itself, creates a new IMAGEN or raises the existing
%      singleton*.
%
%      H = IMAGEN returns the handle to a new IMAGEN or the handle to
%      the existing singleton*.
%
%      IMAGEN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in IMAGEN.M with the given input arguments.
%
%      IMAGEN('Property','Value',...) creates a new IMAGEN or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Imagen_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Imagen_OpeningFcn via varargin.
```

```
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Imagen

% Last Modified by GUIDE v2.5 19-Dec-2016 20:40:02

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Imagen_OpeningFcn, ...
                   'gui_OutputFcn',  @Imagen_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before Imagen is made visible.
function Imagen_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to Imagen (see VARARGIN)

% Choose default command line output for Imagen
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
axes(handles.axes1)
handles.imagen=imread('ABB2.png');
imagesc(handles.imagen)
axis off

% UIWAIT makes Imagen wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command line.
function varargout = Imagen_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

11.2.4 Botón Datos

Mostrará una table con los datos obtenidos de forma numérica.

Código:

```
function varargout = Tabla(varargin)
% TABLA MATLAB code for Tabla.fig
%      TABLA, by itself, creates a new TABLA or raises the existing
%      singleton*.
%
%      H = TABLA returns the handle to a new TABLA or the handle to
%      the existing singleton*.
```

```
%
%      TABLA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TABLA.M with the given input arguments.
%
%      TABLA('Property','Value',...) creates a new TABLA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Tabla_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Tabla_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Tabla

% Last Modified by GUIDE v2.5 03-Mar-2017 12:07:59

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Tabla_OpeningFcn, ...
                   'gui_OutputFcn',  @Tabla_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
% End initialization code - DO NOT EDIT


% --- Executes just before Tabla is made visible.
function Tabla_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Tabla (see VARARGIN)

% Choose default command line output for Tabla
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Tabla wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Tabla_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
global T w1 w2 w3 w4 w5 datos;
datos = [T' w1' w2' w3' w4' w5'];
```

```
set(handles.uitable1,'data',datos);



% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
global T v1 v2 v3 v4 v5 datos;
datos = [T' v1' v2' v3' v4' v5']
set(handles.uitable1,'data',datos);



% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
global T A1 A2 A3 A4 A5 datos;
datos = [T' A1' A2' A3' A4' A5']
set(handles.uitable1,'data',datos);

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global datos;
if get(handles.pushbutton4,'Value') == 1
   fid = fopen('exportacion_datos_text2.txt','w');
fprintf(fid,'%f  %f  %f\n',datos);
fclose(fid);
type('exportacion_datos_text2.txt')
end
```

11.2.5 Botón Cinemática

Este desplegara otra ventana donde se calculara la posición en el espacio de trabajo.

Código:

```
function varargout = cinematica1(varargin)
% CINEMATICA1 MATLAB code for cinematica1.fig
%      CINEMATICA1, by itself, creates a new CINEMATICA1 or raises the
existing
%      singleton*.
```

```
%
%      H = CINEMATICA1 returns the handle to a new CINEMATICA1 or the
handle to
%      the existing singleton*.
%
%      CINEMATICA1('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in CINEMATICA1.M with the given input
arguments.
%
%      CINEMATICA1('Property','Value',...) creates a new CINEMATICA1 or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before cinematica1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to cinematica1_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help cinematica1

% Last Modified by GUIDE v2.5 19-Dec-2016 23:16:29

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @cinematica1_OpeningFcn, ...
                   'gui_OutputFcn',  @cinematica1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before cinematica1 is made visible.
function cinematica1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to cinematica1 (see VARARGIN)

% Choose default command line output for cinematica1
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes cinematica1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = cinematica1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% global  theta1 theta2 theta3 theta4 theta5 theta6 px py pz q1 q2 q3 q4;
theta1 = [];
theta2 = [];
theta3 = [];
theta4 = [];
theta5 = [];
theta6 = 0;
theta3C = [];
theta5C = [];
delete(instrfind({'port'},{'COM3'}));
baudrate = 57600;
s = serial('com3','BaudRate',baudrate);
fopen(s);
n = 1;
tic;
while n<300
AR = fscanf (s, '%E\t');
theta1(n) = AR(2);
theta2(n) = AR(3);
theta3(n) = AR(4);
theta4(n) = AR(5);
theta5(n) = AR(6);
n=n+1;
toc;
end
fclose(s);
for x = 1:n-1
    theta3C(x) = theta2(x) - theta3(x);
    theta5C(x) = theta3C(x) - theta5(x);
end
fclose(s);
theta1 = theta1(n-5);
theta2 = theta2(n-5);
theta3C = theta3(n-5);
theta4 = theta4(n-5);
theta5C = theta5(n-5);
%Constantes geométricas del Robot IRB 120
%Eslabon1
d1=290;
```

```
a1=0;
alpha1=-90;
%Eslabon2
d2=0;
a2=270;
alpha2=0;
theta2=theta2-90;
%Eslabon3
d3=0;
a3=70;
alpha3=-90;
%Eslabon4
d4=302;
a4=0;
alpha4=90;
%Eslabon5
d5=0;
a5=0;
alpha5=-90;
%Eslabon6
d6=72;
a6=0;
alpha6=0;
theta6=theta6-180;

IRB120_DH=[theta1 d1 a1 alpha1;theta2 d2 a2 alpha2;theta3C d3 a3
alpha3;theta4 d4 a4 alpha4;theta5C d5 a5 alpha5;theta6 d6 a6 alpha6];
%Matrices geométricamente homogéneas
A01=Matriz_homogenea(1,IRB120_DH);
A12=Matriz_homogenea(2,IRB120_DH);
A23=Matriz_homogenea(3,IRB120_DH);
A34=Matriz_homogenea(4,IRB120_DH);
A45=Matriz_homogenea(5,IRB120_DH);
A56=Matriz_homogenea(6,IRB120_DH);
A06=A01*A12*A23*A34*A45*A56;
%Coordenadas del extremo
px=A06(1,4);
py=A06(2,4);
pz=A06(3,4);
%Cuaternios
q1=0.5*sqrt(A06(1,1)+A06(2,2)+A06(3,3)+1);
q2=sign(A06(3,2)-A06(2,3))*0.5*sqrt(A06(1,1)-A06(2,2)-A06(3,3)+1);
q3=sign(A06(1,3)-A06(3,1))*0.5*sqrt(-A06(1,1)+A06(2,2)-A06(3,3)+1);
q4=sign(A06(2,1)-A06(1,2))*0.5*sqrt(-A06(1,1)-A06(2,2)+A06(3,3)+1);
Coordenadas_extremo=[px,py,pz,q1,q2,q3,q4];
set(handles.edit1,'string',px);
set(handles.edit8,'string',py);
set(handles.edit9,'string',pz);
set(handles.edit10,'string',q1);
set(handles.edit11,'string',q2);
set(handles.edit12,'string',q3);
set(handles.edit13,'string',q4);
fclose(s);
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double
```

```
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a
double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a
double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a
double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a
double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a
double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a
double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a
double


% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a
double


% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as
a double


% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as
a double


% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of edit12 as
a double


% --- Executes during object creation, after setting all properties.
```

```
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of edit13 as
a double


% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

11.2.5.1 Matriz Homogénea

Matriz necesaria para el cálculo de la cinemática

Código:

```
function [MH]=Matriz_homogenea(Eslabon,IRB120_DH)
theta=IRB120_DH(Eslabon,1);
d=IRB120_DH(Eslabon,2);
a=IRB120_DH(Eslabon,3);
alpha=IRB120_DH(Eslabon,4);
MH=[cosd(theta)  -sind(theta)*cosd(alpha)  sind(theta)*sind(alpha)  a*cosd(theta)
;sind(theta) cosd(alpha)*cosd(theta) -cosd(theta)*sind(alpha) a*sind(theta) ; 0 sind(alpha)
cosd(alpha) d ; 0 0 0 1] ;
```

End

11.3 RAPID

En este apartado de adjuntara los programas diseñados en RAPID, es decir estos programas son los movimientos que realiza el robot.

Código:

```
MODULE Jorge(SYSMODULE)

    CONST robtarget Pra1_1:=[[260.30,293.37,128.94],[0.000758785,0.00502152,0.999982,0.00322058],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget Pra1_2:=[[450.30,293.37,128.94],[0.000758785,0.00502152,0.999982,0.00322058],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    PERS wobjdata wobj3:=[FALSE,TRUE,"",[[121.869,-301.53,20.8388],[0.999986,-0.00520122,2.33883E-06,7.22947E-06]],[[0,0,0],[1,0,0,0]]];

    CONST robtarget PO12:=[[356.25,-18.60,143.95],[0.26154,0.00772282,0.96516,-0.00161932],[-1,-1,-1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget PO32:=[[431.99,-108.83,209.91],[0.188191,-0.67773,0.687211,-0.181679],[-1,-2,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget PO22:=[[431.99,-108.83,209.91],[0.188191,-0.67773,0.687211,-0.181679],[-1,-2,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P3_11:=[[245.73,264.69,295.59],[0.26243,-0.667377,-0.6343,-0.288795],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P3_31:=[[153.32,197.33,221.93],[0.037838,-0.0221825,0.92046,0.388367],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P3_21:=[[153.32,197.33,221.93],[0.037838,-0.0221825,0.92046,0.388367],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P3_51:=[[243.87,128.56,150.33],[0.305367,-0.641388,0.657496,0.251141],[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P3_41:=[[143.87,128.56,150.33],[0.305367,-0.641388,0.657496,0.251141],[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    CONST robtarget P4_12:=[[245.74,36.38,143.82],[0.283454,-0.647175,0.661538,0.251368],[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    PROC teste()
    MoveAbsJ calib_pos, v1000, z50, tPinza;
    MoveJ Pra1_1, v1000, z50, tPinza\WObj:=wobj_cuadricula;
    MoveJ Pra1_2, v1000, z50, tPinza\WObj:=wobj_cuadricula;
  ENDPROC
    PROC Soldadura2()
      CONST robtarget P3_1:=[[174.30,454.72,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
        CONST robtarget PL_10:=[[154.77,563.27,100],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
      CONST robtarget PL_11:=[[154.77,563.27,50],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
      CONST robtarget PL_12:=[[154.77,563.27,24.65],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
      CONST robtarget P4_1:=[[246.06,197.92,278],[0.283473,-0.647191,-0.661539,-0.251304],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
      CONST robtarget P4_2:=[[150,147.95,228],[0.00080288,0.00142847,0.999994,0.00320539],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
      CONST robtarget P4_3:=[[246.06,61,142],[0.283473,-0.647191,0.661539,0.251304],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
CONST robtarget P5_1:=[[246,314,142],[0.189,0.67034,0.66546,-0.26848],[0,-1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P5_2:=[[160,220,228],[0.00080288,0.00142847,0.999994,0.00320539],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P5_3:=[[246.06,165,278],[0.189,0.67034,-0.66546,0.26848],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];


MoveAbsJ calib_pos, v1000, z50, tPinza;
Abrir_Pinza;
!! LASER
!    MoveJ P3_1, v1000, z50, tPinza\WObj:=wobj_cuadricula; !!punto para que llegue a destino
!    MoveJ PL_10, v300, z50, tPinza\WObj:=wobj_cuadricula;
!    MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;
!    MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;
!    Cerrar_Pinza_Linea7; !!Obtención del laser
!    MoveL PL_11, v20, fine, tPinza\WObj:=wobj_cuadricula;
MoveJ PL_10, v1000, z50, tPinza\WObj:=wobj_cuadricula;
!! Soldadura a la inversa
MoveJ [[239.38,261.76,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100, z50,
tPinza\WObj:=wobj_cuadricula;
```

```
MoveJ [[239.38,261.76,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL P5_3, v100, z50, tPinza\WObj:=wobj_cuadricula; !!soldadura

!MoveL P5_2, v100, z50, tPinza\WObj:=wobj_cuadricula;

!MoveL P5_1, v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveC P5_2, P5_1, v100, fine, tPinza\WObj:=wobj_cuadricula;!!soldadura

!! Alejar laser

MoveL Offs (P5_1, 0,0,300),v100,z50,tPinza\WObj:=wobj_cuadricula;

!MoveC [[246.05,197.90,207.11],[0.189051,0.670433,0.665392,-0.268389],[0,-1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], [[246.05,197.90,207.11],[0.189051,0.670433,0.665392,-0.268389],[0,-1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z10, tPinza\WObj:=wobj_cuadricula;

!!soltar laser

MoveL PL_10, v300, z50, tPinza\WObj:=wobj_cuadricula;

!  MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;

!  MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;

!  Abrir_Pinza; !! deja el laser

!  MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;

!  MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;
```

```
ENDPROC

  PROC Soldadura3()

    !CONST robtarget P3_1:=[[239.38,261.76,275.03],[0.31021,-0.63195,-0.64304,-0.30151],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P3_1:=[[174.30,454.72,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget PL_10:=[[154.77,563.27,100],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget PL_11:=[[154.77,563.27,50],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget PL_12:=[[154.77,563.27,28],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P4_1:=[[246.06,197.92,280],[0.283473,-0.647191,-0.661539,-0.251304],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P4_2:=[[150,147.95,230],[0.00080288,0.00142847,0.999994,0.00320539],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P4_3:=[[246.06,61,144],[0.283473,-0.647191,0.661539,0.251304],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P5_1:=[[246,338,148],[0.189,0.67034,0.66546,-0.26848],[0,-1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P5_2:=[[142,260,220],[0.00080288,0.00142847,0.999994,0.00320539],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget P5_3:=[[246.06,198,274],[0.189,0.67034,-0.66546,0.26848],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
MoveAbsJ calib_pos, v100, z50, tPinza;

Abrir_Pinza;

MoveJ P3_1, v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[254.46,448.28,132.45],[0.47261,0.51568,-0.51584,0.49459],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[254.46,448.28,32.45],[0.47261,0.51568,-0.51584,0.49459],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v50, fine, tPinza\WObj:=wobj_cuadricula;

Cerrar_Pinza_Linea2;

MoveL [[254.46,448.28,132.45],[0.47261,0.51568,-0.51584,0.49459],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, fine, tPinza\WObj:=wobj_cuadricula;

MoveL [[239.38,258,200],[0.45523,0.526392,-0.523207,0.491866],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[239.38,258.76,110],[0.45523,0.526392,-0.523207,0.491866],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v50, fine, tPinza\WObj:=wobj_cuadricula;

Abrir_Pinza; !!Coloca la pieza encima de la "tabla"

WaitTime 1;

MoveL [[239.38,261.76,200],[0.45523,0.526392,-0.523207,0.491866],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v50, fine, tPinza\WObj:=wobj_cuadricula;

MoveL [[239.38,261.76,300],[0.45523,0.526392,-0.523207,0.491866],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000, z50, tPinza\WObj:=wobj_cuadricula;
```

```
MoveJ P3_1, v100, z50, tPinza\WObj:=wobj_cuadricula; !!punto para que llegue a destino

MoveJ PL_10, v300, z50, tPinza\WObj:=wobj_cuadricula;

MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;

MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;

Cerrar_Pinza_Linea7; !!Obtención del laser

MoveL PL_11, v20, fine, tPinza\WObj:=wobj_cuadricula;

MoveJ PL_10, v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveJ [[239.38,261.76,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100, z50,
tPinza\WObj:=wobj_cuadricula;

MoveJ [[239.38,261.76,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100, z50,
tPinza\WObj:=wobj_cuadricula;

MoveL P4_1, v100, z50, tPinza\WObj:=wobj_cuadricula; !!soldadura

SetDO D652_10_DO1, 1;

MoveC P4_2, P4_3, v100, fine, tPinza\WObj:=wobj_cuadricula;!!soldadura

!!alejar el robot

SetDO D652_10_DO1, 0;

MoveL P4_12, v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[246.06,61,250],[0.283473,-0.647191,0.661539,0.251304],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v50, fine,
```

```
tPinza\WObj:=wobj_cuadricula;

        MoveL       [[246.06,61,400],[0.283473,-0.647191,0.661539,0.251304],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],      v50,      fine,
tPinza\WObj:=wobj_cuadricula;

        MoveL PL_10, v300, z50, tPinza\WObj:=wobj_cuadricula;

        MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;

        MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;

        Abrir_Pinza; !! deja el laser

        MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;

        MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;

        !! Cojer la pieza para darle la vuelta

        MoveL       [[254.46,448.28,200],[0.47261,0.51568,-0.51584,0.49459],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],      v100,      fine,
tPinza\WObj:=wobj_cuadricula;

        MoveL      [[239.38,261.76,200],[0.45523,0.526392,-0.523207,0.491866],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],      v1000,      z50,
tPinza\WObj:=wobj_cuadricula;

        MoveL       [[239.38,258.76,105],[0.45523,0.526392,-0.523207,0.491866],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],      v50,      fine,
tPinza\WObj:=wobj_cuadricula;

        Cerrar_Pinza_Linea7;

        MoveL       [[239.38,261.76,200],[0.45523,0.526392,-0.523207,0.491866],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],      v50,      fine,
tPinza\WObj:=wobj_cuadricula;
```

```
MoveL [[239.38,261.76,300],[0.45523,0.526392,-0.523207,0.491866],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[183.59,453.02,281.83],[0.309563,0.463325,-0.809593,-0.184552],[0,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[179.00,89.58,279.47],[0.309571,0.463323,-0.809588,-0.184569],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[246.20,12.90,249.12],[0.177504,-0.676176,-0.658961,-0.277575],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[228.89,131.31,308.12],[0.496342,-0.529853,-0.47464,-0.497611],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[229.06,132.05,193.74],[0.496387,-0.529851,-0.474625,-0.497583],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[239.38,138.99,105],[0.487645,-0.482556,-0.522664,-0.506126],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v50, fine, tPinza\WObj:=wobj_cuadricula;

Abrir_Pinza;

WaitTime 1;

MoveL [[239.38,138.99,300],[0.487645,-0.482556,-0.522664,-0.506126],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[295.74,32.31,223.36],[0.0958168,-0.756458,-0.457721,-0.457255],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50, tPinza\WObj:=wobj_cuadricula;

MoveL [[242.07,89.07,279.87],[0.373612,-0.573796,-0.591514,-0.425774],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v100, z50,
```

```
tPinza\WObj:=wobj_cuadricula;


        MoveL    [[246.20,12.90,249.12],[0.177504,-0.676176,-0.658961,-0.277575],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],    v100,    z50,
tPinza\WObj:=wobj_cuadricula;

        !MoveL   [[179.00,89.58,279.47],[0.309571,0.463323,-0.809588,-0.184569],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],   v1000,   z50,
tPinza\WObj:=wobj_cuadricula;

        !! LASER

        MoveJ P3_1, v100, z50, tPinza\WObj:=wobj_cuadricula; !!punto para que llegue a destino

        MoveJ PL_10, v300, z50, tPinza\WObj:=wobj_cuadricula;

        MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;

        MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;

        Cerrar_Pinza_Linea7; !!Obtención del laser

        MoveL PL_11, v20, fine, tPinza\WObj:=wobj_cuadricula;

        MoveJ PL_10, v100, z50, tPinza\WObj:=wobj_cuadricula;

        !! Soldadura a la inversa

        MoveJ    [[239.38,261.76,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100,    z50,
tPinza\WObj:=wobj_cuadricula;

        MoveJ    [[239.38,261.76,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100,    z50,
tPinza\WObj:=wobj_cuadricula;
```

```
    MoveL P5_3, v100, z50, tPinza\WObj:=wobj_cuadricula; !!soldadura
    SetDO D652_10_DO1, 1;
    MoveC P5_2, P5_1, v100, fine, tPinza\WObj:=wobj_cuadricula;!!soldadura
    !! Alejar laser
    SetDO D652_10_DO1, 0;
    MoveL Offs (P5_1, 0,0,300),v100,z50,tPinza\WObj:=wobj_cuadricula;
    !!soltar laser
    MoveL PL_10, v300, z50, tPinza\WObj:=wobj_cuadricula;
    MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;
    MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;
    Abrir_Pinza; !! deja el laser
    MoveL PL_12, v20, fine, tPinza\WObj:=wobj_cuadricula;
    MoveL PL_11, v50, z50, tPinza\WObj:=wobj_cuadricula;


  ENDPROC
 PROC Soldadura1()
    CONST                                                                                    robtarget
P1:=[[174.30,454.72,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
CONST robtarget P2:=[[174.30,454.72,150],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P3:=[[174.30,454.72,82],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P4:=[[221.16,248.63,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P5:=[[221.16,248.63,82],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P6:=[[174.30,22.66,400],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P7:=[[174.30,22.66,150],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P8:=[[174.30,22.66,82],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P9:=[[221.16,248.63,164],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P10:=[[154.77,563.27,100],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P11:=[[154.77,563.27,50],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P12:=[[154.77,563.27,28],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L1:=[[157.27,280.10,400],[0.0,0.00143,0.99999,0.00315],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L1_1:=[[157.27,280.10,200],[0.0,0.00143,0.99999,0.00315],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L1_2:=[[157.27,280.10,95],[0.0,0.00143,0.99999,0.00315],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
CONST robtarget L1_3:=[[157.27,218.10,95],[0.0,0.00143,0.99999,0.00315],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L1_4:=[[157.27,160,95],[0.00274,-0.70932,0.70488,0.00174],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
!CONST robtarget L2:=[[210.20,178.75,300],[0.00274,-0.70932,0.70488,0.00174],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
!CONST robtarget L2_1:=[[209.20,176.75,200],[0.00274,-0.70932,0.70488,0.00174],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L2_2:=[[200,176,95],[0.00274,-0.70932,0.70488,0.00174],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L2_3:=[[266,176,95],[0.00274,-0.70932,0.70488,0.00174],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L2_4:=[[266.42,176,300],[0.00274,-0.70932,0.70488,0.00174],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L3:=[[312.68,224.04,300],[0.01271,-0.99965,-0.02251,-0.00636],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L3_1:=[[313,224.04,200],[0.01271,-0.99965,-0.02251,-0.00636],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L3_2:=[[313,224.04,95],[0.01271,-0.99965,-0.02251,-0.00636],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L3_3:=[[313,281.40,95],[0.01271,-0.99965,-0.02251,-0.00636],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L3_4:=[[313,340,95],[0.00469,-0.70087,-0.71314,-0.01336],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
!CONST robtarget L4:=[[269.41,324.76,300],[0.00469,-0.70087,-0.71314,-0.01336],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
!CONST robtarget L4_1:=[[269.41,324.76,200],[0.00469,-0.70087,-0.71314,-0.01336],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L4_2:=[[269.41,324.76,95],[0.00469,-0.70087,-0.71314,-0.01336],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L4_3:=[[200.93,324.20,95],[0.00469,-0.70087,-0.71314,-0.01336],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget L4_4:=[[200.93,324.20,300],[0.00469,-0.70087,-0.71314,-0.01336],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
MoveAbsJ calib_pos, v1000, z50, tPinza;
Abrir_Pinza;
!SetDO D652_10_DO1, 0;
MoveJ P1, v1000, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P2, v200, fine, tPinza\WObj:=wobj_cuadricula;
MoveL P3, v20, fine, tPinza\WObj:=wobj_cuadricula;
Cerrar_Pinza_Linea6;
MoveL P2, v50, fine, tPinza\WObj:=wobj_cuadricula;
MoveJ P1, v300, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P4, v300, z50, tPinza\WObj:=wobj_cuadricula;
MoveJ P5, v50, fine, tPinza\WObj:=wobj_cuadricula;
Abrir_Pinza;
WaitTime 1;
MoveL P4, v300, z50, tPinza\WObj:=wobj_cuadricula;
MoveJ P6, v300, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P7, v50, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P8, v20, fine, tPinza\WObj:=wobj_cuadricula;
```

```
Cerrar_Pinza_Linea6;
MoveL P7, v50, z50, tPinza\WObj:=wobj_cuadricula;
MoveJ P6, v300, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P4, v300, fine, tPinza\WObj:=wobj_cuadricula;
MoveJ P9, v50, fine, tPinza\WObj:=wobj_cuadricula;
Abrir_Pinza;
WaitTime 1;
MoveL P4, v300, fine, tPinza\WObj:=wobj_cuadricula;
MoveJ P10, v300, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P11, v50, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P12, v20, fine, tPinza\WObj:=wobj_cuadricula;
Cerrar_Pinza_Linea7;
MoveL P11, v20, fine, tPinza\WObj:=wobj_cuadricula;
MoveJ P10, v100, z50, tPinza\WObj:=wobj_cuadricula;
MoveJ L1, v300,z50, tPinza\WObj:=wobj_cuadricula;
MoveL L1_1, v200, z50, tPinza\WObj:=wobj_cuadricula;
MoveL L1_2, v20, fine, tPinza\WObj:=wobj_cuadricula;
```

```
SetDO D652_10_DO1, 1;
MoveL L1_3, v10, fine, tPinza\WObj:=wobj_cuadricula;
SetDO D652_10_DO1, 0;
MoveL L1_4, v100, fine, tPinza\WObj:=wobj_cuadricula;
!MoveJ L2, v100,z50, tPinza\WObj:=wobj_cuadricula;
!MoveL L2_1, v20, z50, tPinza\WObj:=wobj_cuadricula;
MoveL L2_2, v20, fine, tPinza\WObj:=wobj_cuadricula;
SetDO D652_10_DO1, 1;
MoveL L2_3, v10, fine, tPinza\WObj:=wobj_cuadricula;
SetDO D652_10_DO1, 0;
MoveL L2_4, v100, fine, tPinza\WObj:=wobj_cuadricula;
MoveJ L3, v100,z50, tPinza\WObj:=wobj_cuadricula;
MoveL L3_1, v20, z50, tPinza\WObj:=wobj_cuadricula;
MoveL L3_2, v20, fine, tPinza\WObj:=wobj_cuadricula;
SetDO D652_10_DO1, 1;
MoveL L3_3, v10, fine, tPinza\WObj:=wobj_cuadricula;
SetDO D652_10_DO1, 0;
```

```
MoveL L3_4, v100, fine, tPinza\WObj:=wobj_cuadricula;
!MoveJ L4, v100,z50, tPinza\WObj:=wobj_cuadricula;
!MoveL L4_1, v20, z50, tPinza\WObj:=wobj_cuadricula;
MoveL L4_2, v20, fine, tPinza\WObj:=wobj_cuadricula;
SetDO D652_10_DO1, 1;
MoveL L4_3, v10, fine, tPinza\WObj:=wobj_cuadricula;
SetDO D652_10_DO1, 0;
MoveL L4_4, v100, fine, tPinza\WObj:=wobj_cuadricula;
MoveJ P10, v300, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P11, v50, z50, tPinza\WObj:=wobj_cuadricula;
MoveL P12, v20, fine, tPinza\WObj:=wobj_cuadricula;
Abrir_Pinza;
MoveL P12, v20, fine, tPinza\WObj:=wobj_cuadricula;
MoveL P11, v50, z50, tPinza\WObj:=wobj_cuadricula;
```

```
    ENDPROC


PROC Mov_1()
    MoveAbsJ calib_pos, v100, fine, tPinza;
    WaitTime 2;
    MoveJ  [[601.96,0.01,241.73],[0.342123,-2.46697E-05,0.939655,-6.90581E-06],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],  v500,  z100,
tPinza;
    WaitTime 2;
    MoveJ   [[53.64,0.01,891.51],[0.90446,-1.27093E-05,0.426558,-2.71928E-05],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],   v500,   z100,
tPinza;
    WaitTime 2;



ENDPROC


PROC Mov_2()
    MoveAbsJ calib_pos, v100, z50, tPinza;
    WaitTime 2;
```

```
        MoveJ [[394.70,0.01,337.32],[0.341149,-1.94505E-05,0.940009,-9.33804E-07],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v200, z50,
tPinza;

        WaitTime 2;

        MoveJ [[220.97,0.01,955.66],[0.906827,-1.56176E-05,0.421504,-1.37272E-05],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v200, z50,
tPinza;

        WaitTime 2;


    ENDPROC


    PROC D_jorge1()


            CONST                                                                                                                    robtarget
Pr5_Pcoger:=[[174.30,459.87,79.62],[0.000807884,0.00141511,0.999994,0.00320774],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

             CONST string Pr5_TPText:="Pieza para colocar";

             CONST string Pr5_TPFK1:="Pos. 1";

             CONST string Pr5_TPFK2:="Pos. 2";

             CONST string Pr5_TPFK3:="Pos. 3";

             CONST string Pr5_TPFK4:="Pos. 4";
```

```
VAR robtarget Pr5_POrigen;              !Punto de origen donde empezar el movimiento seleccion de piezas
VAR robtarget Pr5_Pcoger_posicion;      !Punto de agarre de pieza
VAR robtarget Pr5_Pcoger_aprox_posicion;    !Punto de aproximación a la pieza a la hora de coger
VAR robtarget Pr5_Pcoger_alejar_posicion;   !Punto de separación de la pieza para evitar choques
VAR robtarget Pr5_Pdejar;               !Punto de depositar pieza en superficie
VAR robtarget Pr5_Pdejar_altura;        !Punto de depositar pieza en altura especifica
VAR robtarget Pr5_Pdejar_aprox;         !Punto de aproximacion a depositar pieza a altura especifica
VAR robtarget Pr5_Pdejar_alejar;        !Punto de separación de la pieza para evitar choques segun la altura
VAR num Pr5_posicion:=0;                !Posición de la pieza a coger
VAR num Pr5_altura:=3;                  !Altura de dejar pieza

MoveAbsJ calib_pos, v200, z50, tPinza;      !Movimiento de ejes a 0

WHILE Pr5_altura>-1 DO                  !Bucle que ejecuta el programa hasta que las cuatro piezas estan apiladas

TPReadFK Pr5_posicion, Pr5_TPText, Pr5_TPFK1, Pr5_TPFK2, Pr5_TPFK3, Pr5_TPFK4, stEmpty; !Pregunta de selección de pieza
```

```
Pr5_POrigen:=Offs(Pr5_Pcoger,0,0,100);                    !Se define el punto como un desplazamiento del punto de cogida de la primera pieza

Pr5_Pcoger_posicion:=Offs(Pr5_Pcoger,(Pr5_posicion-1)*100,0,0);          !Se define este punto dependiendo de la posicion de la pieza a coger, separadas entre si 100 mm

Pr5_Pcoger_aprox_posicion:=Offs(Pr5_Pcoger_posicion,0,0,15);          !Se define este punto como una aproximacion a 15 mm de la pieza para hacer el movimiento de aproximacion con delicadeza

Pr5_Pcoger_alejar_posicion:=Offs(Pr5_Pcoger_posicion,0,0,(Pr5_altura*80)+100); !Se define la altura a la que debe subir la pieza despues de ser cogida para evitar choques

Pr5_Pdejar:=Offs(Pr5_Pcoger,100,-200,0);                  !Se define el punto como un desplazamiento del punto de cogida de la primera pieza

Pr5_Pdejar_altura:=Offs(Pr5_Pdejar,0,0,Pr5_altura*80);          !Se define la altura a la que se deposita el objeto a partir del punto de depositar pieza en superficie

Pr5_Pdejar_aprox:=Offs(Pr5_Pdejar_altura,0,0,15);          !Se define este punto como una aproximacion a 15 mm de la pieza para hacer el movimiento de aproximacion con delicadeza

Pr5_Pdejar_alejar:=Offs(Pr5_Pdejar_altura,0,0,100);          !Se define la altura a la que debe subir la pieza despues de ser cogida para evitar choques

  MoveJ Pr5_Pdejar_alejar, v200, z50, tPinza\WObj:=wobj_cuadricula;          !Posición origen de selección de pieza

  MoveL Pr5_Pdejar_aprox, v100, z10, tPinza\WObj:=wobj_cuadricula;      !Aproximación rapida a pieza elegida

  MoveL Pr5_Pdejar_altura, v20, fine, tPinza\WObj:=wobj_cuadricula;          !Aproximación lenta a pieza elegida

  Cerrar_Pinza_Linea6;

  MoveL Pr5_Pdejar_aprox, v20, z10, tPinza\WObj:=wobj_cuadricula;
```

```
        Movel Pr5_Pdejar_alejar, v100, z50, tPinza\WObj:=wobj_cuadricula;          !Movimiento a posición de dejar pieza pero a una altura adecuada para evitar choques

        Movej Pr5_Pcoger_alejar_posicion, v100, fine, tPinza\WObj:=wobj_cuadricula;          !Aproximacion rapida a la posicion para depositar

        MoveL Pr5_Pcoger_aprox_posicion, v100, fine, tPinza\WObj:=wobj_cuadricula;          !Aproximacion delicada de la pieza a superficie para depositar

        MoveL Pr5_Pcoger_posicion, v20, fine, tPinza\WObj:=wobj_cuadricula;

        Abrir_Pinza;

        WaitTime 1;                                          !Tiempo de espera de 1 segundo para evitar movimientos bruscos al abrir pinza

        MoveL Pr5_Pcoger_aprox_posicion, v20, fine, tPinza\WObj:=wobj_cuadricula;          !Alejamiento de la pieza delicado

        MoveL Pr5_Pcoger_alejar_posicion, v100, z100, tPinza\WObj:=wobj_cuadricula;          !Alejamiento de la pieza rapido

        MoveL Offs(Pr5_Pdejar_alejar,0,200,0),v100, z100, tPinza\WObj:=wobj_cuadricula; !Movimiento horizontal para colocar pieza encima de la fila de selección de pieza

        MoveJ Pr5_Pdejar_alejar, v200, z50, tPinza\WObj:=wobj_cuadricula;          !Posición origen de selección de pieza


          decr Pr5_altura;                                   !Incremento del valor de la altura para apilar piezas

        ENDWHILE

        MoveAbsJ calib_pos, v200, z50, tPinza;                    !Al terminar de apilar posicion de ejes 0

      ENDPROC!Amontonar piezas

      ENDMODULE
```