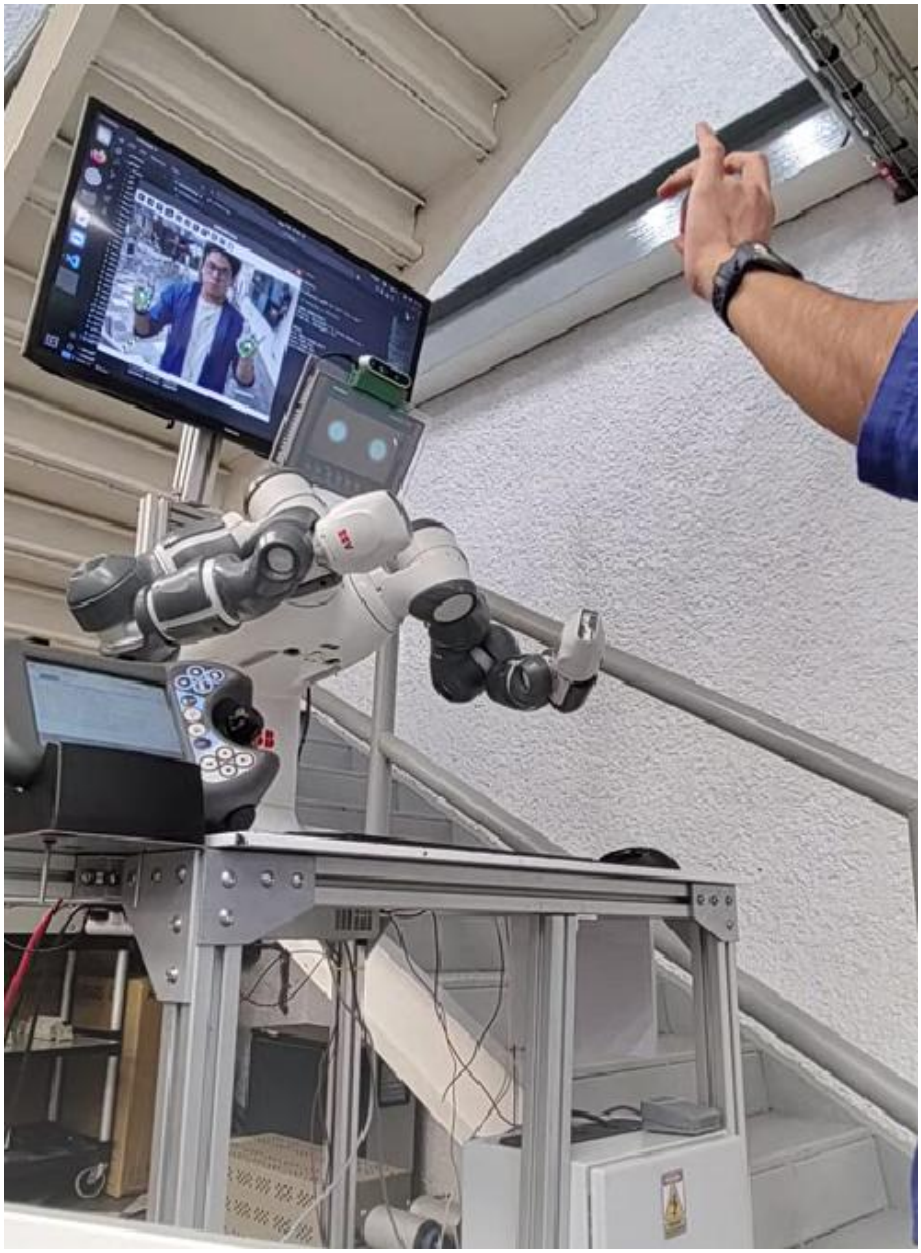
MANUAL DE USO DEL SISTEMA DE CONTROL DEL ROBOT ABB YuMi CON RECONOCIMIENTO DE MANOS



Autor: Ing. Carlos Martínez García

1. Introducción

Este sistema permite controlar de forma **intuitiva** los brazos del robot ABB YuMi IRB 14000 usando el seguimiento en tiempo real de las manos del operador con un ligero retraso debido al tiempo de procesamiento, capturadas por una cámara **Intel RealSense** y procesadas con **Python**, **MediaPipe** y con una **Jetson Nano** como computadora para el procesamiento de las imágenes.

El sistema envía al robot:

- **Coordenadas 3D** de cada mano para mover los TCPs de cada brazo.
- Señales "ABIERTO" y "CERRADO" para controlar la apertura y cierre de los **Smart Grippers**, usando el gesto de cerrar o abrir la mano.

Cuando no se detecta la mano durante **10 segundos**, el sistema devuelve automáticamente el brazo a una **posición HOME segura** y abre el gripper.

2. Componentes del sistema

Elemento	Descripción
Robot	ABB YuMi IRB 14000 con controlador IRC5.
Cámara	Intel RealSense D435i (o similar) para captura de color y profundidad.
Host visión	NVIDIA Jetson Nano con Ubuntu Linux.
Software visión	Python 3, MediaPipe, OpenCV, PyRealSense2.
Red	Conexión TCP/IP Jetson ↔ Controlador IRC5.
Programas RAPID	Módulos MainModule_Left.mod y MainModule_Right.mod.

3. Requisitos previos

Jetson Nano con:

- Python 3.
- Dependencias: pip install opencv-python mediapipe pyrealsense2.

Conexión de red Jetson ↔ IRC5 via cable Ethernet:

- Configura IP fija, por ejemplo:
  - Jetson: 192.168.125.2
  - IRC5: 192.168.125.1.

Archivos RAPID cargados en cada brazo del YuMi:

- Brazo izquierdo escucha en puerto 30010.
- Brazo derecho escucha en puerto 30011.

FlexPendant calibrado (g_Calibrate para Smart Grippers).

4. Estructura de archivos

- **rs2YuMiv2.8.py**
  Script Python principal. Ejecuta la detección, transforma coordenadas y envía datos por socket.
- **posImitatorL.mod**
  Programa RAPID para el brazo izquierdo. Acepta comandos de posición y gripper.
- **posImitatorR.mod**
  Programa RAPID para el brazo izquierdo. Acepta comandos de posición y gripper.

5. Funcionamiento del sistema

1. La cámara RealSense captura imagen RGB y profundidad.

2. MediaPipe Hands detecta la posición de la muñeca y dedos.

3. El script Python transforma la posición (X, Y) de MediaPipe al sistema de coordenadas del robot:
    - Ajusta escala e introduce un offset en base a las coordenadas X & Y correspondientes a posiciones seguras de cada brazo del robot YuMi (dichas posiciones son alcanzadas por el robot de manera automática cuando se ejecuta el código del YuMi desde el FlexPendant y se mantienen en dicha posición hasta que el sistema detecta una o más manos de las cuales tomar sus coordenadas X, Y para imitar).

    - Cambia ejes según la orientación del Work Object.

    - NOTA: El sistema inicialmente se planeo para que los brazos también detectaran cambios en la profundidad (o eje Z) en las manos del operador y respondieran de manera correspondiente, sin embargo, por motivos de seguridad y practicidad, se decidió ajustar la funcionalidad del sistema para que solamente imite 2 dimensiones del movimiento de la mano del usuario (X,Y) y cambiar así solamente 2 dimensiones de cada brazo del robot (Y & Z del cada flange respectivamente)

4. Envía de las coordenadas:
    - Cada ~frame una nueva coordenada del TCP.
    - Señales "ABIERTO" o "CERRADO" si detecta que el pulgar se acerca o aleja del dedo medio.
    - NOTA: En ocasiones el tráfico de mensajes a través de los puertos (sockets) puede verse lento o los mensajes de posiciones y de ABIERTO/CERRADO pueden interrumpirse entre sí si se realizan en un intervalo de tiempo muy pequeño o abruptamente rápido.

5. Si la mano se pierde >10 segundos:
    - ~~El script envía la posición HOME predefinida~~ (ANTES)
    - ~~Envía "ABIERTO" para soltar el gripper~~ (ANTES)
    - El script de Python envía periódicamente la última posición registrada de cada brazo para evitar que se llegue a un Timeout en los puertos de comunicación del FlexPendant.
6. El RAPID recibe:
    - Si el dato es "ABIERTO" o "CERRADO", actúa sobre el gripper.
    - Si es coordenada, convierte strings a num, aplica límite de rango (LimitTCP) y ejecuta MoveJ.

6. Parámetros clave

Parámetro	Descripción	Valor recomendado
ESCALA_X, ESCALA_Y	Factores de conversión de la escala MediaPipe a milimetros reales.	150, 150,
Work Object	Origen de cada brazo (posición de inicio).	Left: X=295.15 Y=260 Z=267 Right: X=287 Y=-200 Z=250
RESEND_INTERVAL	Tiempo sin manos visibles antes de regresar a HOME.	10 s
GRIPPER_THRESH	Distancia pulgar-dedo medio para abrir/cerrar.	50 px

CLARIFICACIONES Y EXPLICACIONES:

→ "Ajusta escala e introduce un offset en base a las coordenadas X & Y correspondientes a posiciones seguras..."

Cuando el robot arranca, cada brazo se mueve a una posición segura de inicio (HOME). **Esa posición sirve como origen** (***offset***) para convertir coordenadas de cámara a coordenadas del robot.

Como la cámara trabaja en "pixeles" y el robot en "milímetros", necesitas:
escala para convertir tamaño de movimiento de imagen a movimiento real
offset → desplazar el origen de la cámara al origen físico del brazo

En resumen: el **sistema convierte pixeles de cámara a milímetros del robot usando escala + offset basados en la posición inicial del brazo.**

Qué es el TCP (Tool Center Point) ?

Es el punto de referencia del robot ubicado en la herramienta (generalmente en la punta del gripper). Todas las posiciones del robot se expresan respecto a ese punto. Cuando envías X, Y, Z, estás diciendo "lleva el TCP a estas coordenadas".

Por qué usamos Ethernet TCP/IP entre Jetson y ABB ?

Porque el controlador IRC5 permite recibir comandos en tiempo real usando sockets TCP. Es la forma más estable y simple de enviar datos continuos (coordenadas + comandos de gripper). TCP garantiza orden, entrega y no pérdida de paquetes → fundamental en control robótico.

Qué es TCP/IP ?

Es una forma estándar de comunicar dos computadoras por red, con comunicación fiable. Es un conjunto de los protocolos:

IP: direcciona los dispositivos (IP: 192.168.125.1 etc.)

TCP: abre una conexión confiable entre dos equipos y envía datos en orden.

Por qué la cámara usa X–Y pero el robot usa Y–Z

MediaPipe te da las coordenadas (X, Y, Z) de la imagen. Pero el robot interpreta coordenadas en su propio sistema... Para el diseño del *ABB YuMi*, el plano de trabajo donde se mueven los brazos corresponde a los ejes Y–Z del robot.

Entonces haces un mapeo:
(X, Y de la cámara) ⇔ (Y, Z del robot)
Esto se hace por comodidad y seguridad.

7. Cómo usarlo

1) Inicia el controlador RAPID con MainModule_Left y MainModule_Right.
2) Verifica que los sockets acepten conexión (TPWrite "Esperando conexión...").
3) Corre la versión del script de Python más reciente guardado en la JetsonNano. Última versión funcional guardada en la Jetson (pero no respaldado en Teams): rs2YuMiv2.8.py
4) Oculte las manos hasta que el código de Python despliegue la imagen en tiempo real de la cámara en el monitor encima del robot.
5) Mueve tus manos frente a la cámara RealSense.
6) Observa cómo el YuMi sigue la posición de tus manos y abre/cierra el gripper.
7) Si sacas la mano por más de 10 s, los brazos deberán quedarse en la última posición registrada por el sistema de visión.

8. Mantenimiento

- Revisa conexión Ethernet directa: evita perder paquetes.
- Limpia la lente de la RealSense regularmente.
- Verifica offsets y límites en LimitTCP para evitar movimientos peligrosos.
- Siempre ten habilitado el botón de parada de emergencia.

9. Autor y versión

Proyecto: **YuMi Realtime Hand TCP Control**
Versión: **2.8.2**
Base: Python 3 + RAPID + Realsense SDK + MediaPipe.

Autor: Ing. Carlos Martínez García

Datos de Contacto:

- E-mail: mgcar_los@hotmail.com (por favor incluir asunto)
- Teléfono Celular: +52 464 176 7027 (WhatsApp preferencialmente)

Explicaciones sobre el lenguaje de programación RAPID:

Función: ***SocketPeek(stringVar)***
Checa si hay bytes esperando en la parte de recepción de datos y cuentos hay... Regresa:

0 ? no datos disponibles
>0 ? números de bytes listos para estar leído
<0 ? error de programa

Función: ***MoverJ(positionVar[], Velocity, Zone, toolName)***
ejecuta un movimiento NO rectilíneo del COMPONENTE «*toolName*» desde su posición ACTUAL hasta la posición de coordenadas en «*positionVAR*[]»
con una VELOCIDAD específica, en una ZONA/ÁREA específica.

Proceso de Conversión de las Coordenadas:
El sistema externo (cámara + Jetson) envía coordenadas en forma de texto, ya que es sencillo y robusto a través de la conexión TCP/IP.
Así, el robot recibe una cadena y la convierte en coordenadas enteras utilizando las siguientes funciones:

→ **Función**: ***StrPart(variableName, start, length)***
Extrae una subcadena de una cadena... por lo que puede extraer una parte de la cadena «*variableName*», desde *START* hasta *START+LENGHT* incluido.

→ **Función**: ***StrToVal(strVar, numVar)***
Convierte una cadena numérica en un número (devuelve *TRUE* si la conversión se realiza correctamente, *FALSE* si falla).

Qué es *ZONE* en la función *MoveJ() ?*

ZONE = precisión con la que el robot debe atravesar el punto de destino.
fine → el robot debe detenerse exactamente en el punto de destino
(sin redondeos, más lento, más seguro)

- z1 → permite un redondeo de 1 mm
- z10 → 10 mm
- z50 → 50 mm
- z100 → 100 mm

Una zona más alta = movimiento más suave y rápido, pero menos preciso

CoordenadasVar ***[[X,Y,Z], [Q1,Q2,Q3,Q4], [CF1,CF2,CF3,CF4], [E1,E2,E3,E4,E5,E6]]***

[X,Y,Z] <-- posición del TCP (punto central de la herramienta – Tool Central Point) en coordenadas cartesianas

[Q1,Q2,Q3,Q4] <-- orientación (o cuaternión) que representa la rotación de la herramienta en 3D

[CF1,CF2,CF3,CF4] <-- datos de configuración, contiene información sobre la postura del brazo del robot

[E1,E2,E3,E4,E5,E6] <-- ejes externos (9E+09 = ignorar este eje)

Más detalles sobre la parte de datos de configuración: contiene información sobre la postura del brazo robótico:

- CF1: Configuración del brazo (codo arriba/abajo, hombro izquierda/derech)
- CF2: Configuración de la muñeca
- CF3: Número de giros (revoluciones de los ejes)
- CF4: configuración cinemática externa

Los robots ABB necesitan esto para elegir una de las varias posturas posibles del robot que alcanzan el mismo punto

Más detalles sobre el eje externo: son datos para robots con:

- guías lineales
- posicionadores
- ejes rotativos adicionales

9E+09 significa: ignorar este eje / no se utiliza ningún eje externo

CLARIFICACIONES Y EXPLICACIONES:

Qué es el TCP (Tool Center Point) ?

Es el punto de referencia del robot ubicado en la herramienta (generalmente en la punta del gripper). Todas las posiciones del robot se expresan respecto a ese punto. Cuando envías X, Y, Z, estás diciendo “lleva el TCP a estas coordenadas”.

Por qué usamos Ethernet TCP/IP entre Jetson y ABB ?

Porque el controlador IRC5 permite recibir comandos en tiempo real usando sockets TCP. Es la forma más estable y simple de enviar datos continuos (coordenadas + comandos de gripper). TCP garantiza orden, entrega y no pérdida de paquetes → fundamental en control robótico.

Qué es TCP/IP ?

Es una forma estándar de comunicar dos computadoras por red, con comunicación fiable. Es un conjunto de los protocolos:

IP: direcciona los dispositivos (IP: 192.168.125.1 etc.)

TCP: abre una conexión confiable entre dos equipos y envía datos en orden.

Por qué la cámara usa X–Y pero el robot usa Y–Z

MediaPipe te da las coordenadas (X, Y, Z) de la imagen. Pero el robot interpreta coordenadas en su propio sistema... Para el diseño del *ABB YuMi*, el plano de trabajo donde se mueven los brazos corresponde a los ejes Y–Z del robot.

Entonces haces un mapeo:
(X, Y de la cámara) ⇔ (Y, Z del robot)
Esto se hace por comodidad y seguridad.

ANEXOS:

Código RAPID para brazo Derecho (última versión funcional):

```
MODULE MainModule
!!!
!! Documentacion completa sobre las funcciones y los processos, abajo de este codigo
!!!

    VAR socketdev serverSocket_R;
    VAR socketdev clientSocket_R;
    VAR string receivedData_R;
    VAR string x_str;
    VAR string y_str;
    VAR string z_str;
    VAR bool ok1;
    VAR bool ok2;
    VAR bool ok3;
    VAR num x_val;
    VAR num y_val;
    VAR num z_val;
    VAR num elapsed;
    VAR robtarget target_pos_R := [[0,0,0],[0.558278,-0.650056,0.366482,-0.362552],[1,2,-
1,4],[-127.013,9E+09,9E+09,9E+09,9E+09,9E+09]];
    ! target_pos_R  is initialized here just give it a shape, otherway it wouldn't that it must
have this shape/form/structure... like an array...
    CONST num MAX_X := 470;
    CONST num MIN_X := 270;
    CONST num MAX_Y := -50;
    CONST num MIN_Y := -270;
    CONST num MAX_Z := 380;
    CONST num MIN_Z := 170;
    CONST robtarget INICIAL_POS_R := [[294.55,-187.26,234.26],[0.558278,-
0.650056,0.366482,-0.362552],[1,2,-1,4],[-127.013,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST num PUERTO_BRAZO_R := 30011;
```

```
!Función para limitar coordenadas de TCP en rango seguro
FUNC robtarget LimitTCP(robtarget tcp)
    VAR robtarget safeTCP;
    safeTCP := tcp;
    IF safeTCP.trans.x > MAX_X THEN safeTCP.trans.x := MAX_X; ENDIF
    IF safeTCP.trans.x < MIN_X THEN safeTCP.trans.x := MIN_X; ENDIF
    IF safeTCP.trans.y > MAX_Y THEN safeTCP.trans.y := MAX_Y; ENDIF
    IF safeTCP.trans.y < MIN_Y THEN safeTCP.trans.y := MIN_Y; ENDIF
    IF safeTCP.trans.z > MAX_Z THEN safeTCP.trans.z := MAX_Z; ENDIF
    IF safeTCP.trans.z < MAX_Z THEN safeTCP.trans.z := MIN_Z; ENDIF
    RETURN safeTCP;
ENDFUNC

PROC main()
    ! Initializar a la posicion iniciale (y segura)
    MoveJ INICIAL_POS_R, v1000, z50, tool0;

    ! Crear y configurar el socket por la recepcion de los datos
    SocketCreate serverSocket_R;
    SocketBind serverSocket_R, "192.168.125.1", PUERTO_BRAZO_R;
    SocketListen serverSocket_R;

    TPWrite "Esperando conexión en el puerto 30011...";
    g_Calibrate;   ! Calibramos el gripper antes de usarlo

    WHILE TRUE DO
        SocketAccept serverSocket_R, clientSocket_R;
        TPWrite "Conexión establecida con el cliente (brazo de derecha) ";

        WHILE TRUE DO
            ! Verificar el estado del socket antes de recibir datos
            IF SocketGetStatus(clientSocket_R) = ERR_SOCK_CLOSED THEN
                ! Checar si disconectado, si es.. conectar de vuelta..
                TPWrite "Conexión cerrada. Esperando nueva conexión...";
                EXIT;
            ENDIF




            elapsed := 0;
            WHILE SocketPeek(clientSocket_R) = 0 DO
                ! SocketPeek(clientSocket_R) checa si esta recibiendo bytes datas y cuentos
                ! Checar aqui la conexión, si =0 significa que no hay bytes, y entonces, no
conexión..
```

```
            WaitTime 100;
            elapsed := elapsed + 100;
            ! Esperar por intentar volver a conectarse
            IF elapsed >= 100000000 THEN
                ! Si no conexión despues de 100000000ms, el programa se apaga
                TPWrite "Timeout esperando datos, esperando nueva conexión...";
                EXIT;
            ENDIF
        ENDWHILE


        ! Recibir datos
        SocketReceive clientSocket_R \Str := receivedData_R;

        IF receivedData_R = "ABIERTO" THEN
            g_GripOut;
            TPWrite "Gripper DERECHO Abierto";

        ELSEIF receivedData_R = "CERRADO" THEN
            g_GripIn;
            TPWrite "Gripper DERECHO Cerrado";
        ELSE
            ! Verificar que la cadena tiene datos válidos y mas de 12 (3 coordenadas 3D y
2 separaciones como " ; " + 1 especial al gripper de derecha)
            IF StrLen(receivedData_R) = 11 THEN
                ! Convertir la cadena en valores numéricos
                x_str := StrPart(receivedData_R,1,3); ! Extraer el la primer cadena numerica
(de 1 a 1+3), correspondiente a las primeras coordenadas
                ok1 := StrToVal(x_str,x_val);        ! Convertir esta cadena numerica en
numeros/cifras
                y_str := StrPart(receivedData_R,5,3);
                ok2 := StrToVal(y_str,y_val);
                z_str := StrPart(receivedData_R,9,3);
                ok3 := StrToVal(z_str,z_val);
                TPWrite "Valores X,Y,Z recibidos: (" + x_str +" ; " + y_str + "  ; " + z_str +
") ";

                ! Asignar los valores a la estructura de robtarget, SOLO SI la conversion es
validada (los 3 ok=TRUE)
                IF ok1 AND ok2 AND ok3 THEN
                    target_pos_L.trans.x := x_val;
                    target_pos_L.trans.y := y_val;
                    target_pos_L.trans.z := z_val;
                    target_pos_L := LimitTCP(target_pos_L); ! Respectar las limites

                    ! Mover el brazo (tool0) a la posición recibida (target_pos_L) con una
velocidad de 600mm/s en la zona 50
```

```
                    MoveJ target_pos_L, v600, z50, tool0;
                ENDIF

            ELSEIF StrLen(receivedData_R) = 12 THEN
                ! Convertir la cadena en valores numéricos
                x_str := StrPart(receivedData_R,1,3);
                ok1 := StrToVal(x_str,x_val);
                y_str := StrPart(receivedData_R,5,4);
                ok2 := StrToVal(y_str,y_val);
                z_str := StrPart(receivedData_R,10,3);
                ok3 := StrToVal(z_str,z_val);
                TPWrite "Valores X,Y,Z recibidos: (" + x_str +" ; " + y_str + "  ; " + z_str +
") ";

                ! Asignar los valores a la estructura de robtarget, SOLO SI la conversion es
validada (los 3 ok=TRUE)
                IF ok1 AND ok2 AND ok3 THEN
                    target_pos_L.trans.x := x_val;
                    target_pos_L.trans.y := y_val;
                    target_pos_L.trans.z := z_val;
                    target_pos_L := LimitTCP(target_pos_L); ! Respectar las limites

                    ! Mover el brazo (tool0) a la posición recibida (target_pos_L) con una
velocidad de 600mm/s en la zona 50
                    MoveJ target_pos_L, v600, z50, tool0;
                ENDIF

            ELSE
                TPWrite "ERR: Datos inválidos -> ";
            ENDIF
        ENDIF
      ENDWHILE
    ENDWHILE
  ENDPROC
ENDMODULE




! Here is some documentation :
!
!  --> function :   SocketPeek(stringVar)
!     it checks how many bytes are waiting in the receive buffer, without removing them..
!     so it checks if it's receiving some bytes and how many... to check the coordonates data
reception
!     It returns:
!         0  ? no data available
```

```
!        >0 ? number of bytes ready to read
!        <0 ? error codes
!
!
!  --> function :   MoverJ(positionVar[], Velocity, Zone, toolName)
!     executes a NON-straight movement of the COMPONENT 'toolNAme'
!     from its CURRENT position TO the coordonates position in 'positionVAR[]'
!     with a specific VELOCITY, into a specific ZONE/AREA
!
!
!  --> Coordonates Conversion Process :
!    The external system (camera+Jetson) sends coordinates as text as it's simple and robust over TCP/IP liaison
!    So the robot receives a string and as to convert it into integer coordonates... using the following fuctions :
!
!  --> function :   StrPart(variableName, start, length)
!        Extracts substring from a string... so can extract a part of string 'variableName', from START to START+LENGHT included
!  --> function :   StrToVal(strVar, numVar)
!        Converts a numeric string into a number     (returns TRUE if conversion succeed, FALSE if failed)
!

!
!  --> What is zone in MoveJ() function ? (z50, fine, z1, z10, ...)
!     Zone = how precisely the robot must pass through the target point.
!        fine → robot must stop exactly at the target (no rounding, slowest, safest)
!        z1 → allow 1 mm rounding
!        z10 → 10 mm
!        z50 → 50 mm
!        z100 → 100 mm
!    Higher zone = smoother & faster motion, but less precise
!
!
!  --> CoordonatesVar := [ [X,Y,Z], [Q1,Q2,Q3,Q4], [CF1,CF2,CF3,CF4], [E1,E2,E3,E4,E5,E6] ]
!                    [X,Y,Z]               <-- position of the TCP (Tool Center Point) in cartesian
!                    [Q1,Q2,Q3,Q4]          <-- orientation (or quaternion) it represents the tool's rotation in 3D
!                    [CF1,CF2,CF3,CF4]       <-- configuration data, it contains robot arm posture information
!                    [E1,E2,E3,E4,E5,E6]     <-- external axes (9E+09 = ignore this axis)
!
!
!  --> More details about the configuration data part: it contains robot arm posture information:
```

```
!        CF1: Arm configuration (elbow up/down, shoulder left/right)
!        CF2: Wrist configuration
!        CF3: Turn number (revolutions of axes)
!        CF4: external kinematic configuration
!     ABB robots need this to choose one of several possible robot postures that reach the
same point
!
!
!  --> More details about the external axis: it's data for robots with:
!        linear tracks
!        positioners
!        extra rotary axes
!     9E+09 means: ignore this axis / no external axis used
!
!
```

Código RAPID para brazo Izquierdo (última versión funcional):

```
MODULE MainModule
!! Documentacion completa sobre las funcciones y los processos, abajo de este codigo
!!!

    VAR socketdev serverSocket_L;
    VAR socketdev clientSocket_L;
    VAR string receivedData_L;
    VAR string x_str;
    VAR string y_str;
    VAR string z_str;
    VAR bool ok1;
    VAR bool ok2;
    VAR bool ok3;
    VAR num x_val;
    VAR num y_val;
    VAR num z_val;
    VAR num elapsed;
    VAR robtarget target_pos_L := [[0,0,0],[0.385214,-0.336824,0.640231,-0.572944],[-
1,0,1,4],[138.956,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    ! target_pos_L  is initialized here just give it a shape, otherway it wouldn't that it must have this shape/form/structure... like an array...
    CONST num MAX_X := 470;
    CONST num MIN_X := 215;
    CONST num MAX_Y := 450;
    CONST num MIN_Y := 245;
    CONST num MAX_Z := 550;
    CONST num MIN_Z := 170;
    CONST robtarget INICIAL_POS_L := [[338.80,313.86,265.49],[0.385214,-0.336824,0.640231,-0.572944],[-1,0,1,4],[138.956,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST num PUERTO_BRAZO_L := 30010;

    ! Función para limitar coordenadas de TCP en rango seguro
    FUNC robtarget LimitTCP(robtarget tcp)
        VAR robtarget safeTCP;
        safeTCP := tcp;
        IF safeTCP.trans.x > MAX_X THEN safeTCP.trans.x := MAX_X; ENDIF
        IF safeTCP.trans.x < MIN_X THEN safeTCP.trans.x := MIN_X; ENDIF
        IF safeTCP.trans.y > MAX_Y THEN safeTCP.trans.y := MAX_Y; ENDIF
        IF safeTCP.trans.y < MIN_Y THEN safeTCP.trans.y := MIN_Y; ENDIF
        IF safeTCP.trans.z > MAX_Z THEN safeTCP.trans.z := MAX_Z; ENDIF
        IF safeTCP.trans.z < MAX_Z THEN safeTCP.trans.z := MIN_Z; ENDIF
        RETURN safeTCP;
    ENDFUNC


    PROC main()
        ! Initializar a la posicion iniciale (y segura)
        MoveJ INICIAL_POS_L, v1000, z50, tool0;

        ! Crear y configurar el socket por la recepcion de los datos
        SocketCreate serverSocket_L;
        SocketBind serverSocket_L, "192.168.125.1", PUERTO_BRAZO_L;
        SocketListen serverSocket_L;

        TPWrite "Esperando conexión en el puerto 30010...";
        g_Calibrate;   ! Calibramos el Gripper antes de usarlo

        WHILE TRUE DO
            SocketAccept serverSocket_L, clientSocket_L;
            TPWrite "Conexión establecida con el cliente (brazo de izquierda) ";

            WHILE TRUE DO
                ! Verificar el estado del socket antes de recibir datos
                IF SocketGetStatus(clientSocket_L) = ERR_SOCK_CLOSED THEN
                    ! Checar si disconectado, si es.. conectar de vuelta..
                    TPWrite "Conexión cerrada. Esperando nueva conexión...";
```

```
            EXIT;
        ENDIF

        elapsed := 0;
        WHILE SocketPeek(clientSocket_L) = 0 DO
            ! SocketPeek(clientSocket_L) checa si esta recibiendo bytes datas y cuentos
            ! Checar aqui la conexión, si =0 significa que no hay bytes, y entonces, no
conexión..
            WaitTime 100;
            elapsed := elapsed + 100;
            ! Esperar por intentar volver a conectarse
            IF elapsed >= 100000000 THEN
                ! Si no conexión despues de 100000000ms, el programa se apaga
                TPWrite "Timeout esperando datos, esperando nueva conexión...";
                EXIT;
            ENDIF
        ENDWHILE


        ! Recibir datos
        SocketReceive clientSocket_L \Str := receivedData_L;

        IF receivedData_L = "ABIERTO" THEN
            g_GripOut;
            TPWrite "Gripper IZQUIERDO Abierto";
        ELSEIF receivedData_L = "CERRADO" THEN
            g_GripIn;
            TPWrite "Gripper IZQUIERDO Cerrado";
        ELSE
            ! Verificar que la cadena tiene datos válidos y mas de 11 (3 coordenadas 3D y
2 separaciones como " ; " )
            IF StrLen(receivedData_L) >= 11 THEN
                ! Convertir la cadena en valores numéricos
                x_str := StrPart(receivedData_L,1,3); ! Extraer el la primer cadena numerica
(de 1 a 1+3), correspondiente a las primeras coordenadas
                ok1 := StrToVal(x_str,x_val);       ! Convertir esta cadena numerica en
numeros/cifras
                y_str := StrPart(receivedData_L,5,3);
                ok2 := StrToVal(y_str,y_val);
                z_str := StrPart(receivedData_L,9,3);
                ok3 := StrToVal(z_str,z_val);
                TPWrite "Valores X,Y,Z recibidos: (" + x_str +" ; " + y_str + "  ; " + z_str +
") ";
```

```
                    ! Asignar los valores a la estructura de robtarget, SOLO SI la conversion es
validada (los 3 ok=TRUE)
                    IF ok1 AND ok2 AND ok3 THEN
                        target_pos_L.trans.x := x_val;
                        target_pos_L.trans.y := y_val;
                        target_pos_L.trans.z := z_val;
                        target_pos_L := LimitTCP(target_pos_L); ! Respectar las limites

                        ! Mover el brazo (tool0) a la posición recibida (target_pos_L) con una
velocidad de 600mm/s en la zona 50
                        MoveJ target_pos_L, v600, z50, tool0;
                    ENDIF

                ELSE
                    TPWrite "ERR: Datos inválidos -> "+ receivedData_L;
                ENDIF
            ENDIF
        ENDWHILE
    ENDWHILE
  ENDPROC
ENDMODULE



! Here is some documentation :
!
!  --> function :   SocketPeek(stringVar)
!     it checks how many bytes are waiting in the receive buffer, without removing them..
!     so it checks if it's receiving some bytes and how many... to check the coordonates data
reception
!     It returns:
!        0  ? no data available
!        >0 ? number of bytes ready to read
!        <0 ? error codes
!
!
!  --> function :   MoverJ(positionVar[], Velocity, Zone, toolName)
!      executes a NON-straight movement of the COMPONENT 'toolNAme'
!      from its CURRENT position TO the coordonates position in 'positionVAR[]'
!      with a specific VELOCITY, into a specific ZONE/AREA
!
!
!  --> Coordonates Conversion Process :
!     The external system (camera+Jetson) sends coordinates as text as it's simple and robust
over TCP/IP liaison
```

```
!     So the robot receives a string and as to convert it into integer coordonates... using the following fuctions :
!
!  --> function :   StrPart(variableName, start, length)
!        Extracts substring from a string... so can extract a part of string 'variableName', from START to START+LENGHT included
!  --> function :   StrToVal(strVar, numVar)
!        Converts a numeric string into a number     (returns TRUE if conversion succeed, FALSE if failed)
!
!
!  --> What is zone in MoveJ() function ? (z50, fine, z1, z10, ...)
!      Zone = how precisely the robot must pass through the target point.
!        fine → robot must stop exactly at the target (no rounding, slowest, safest)
!        z1 → allow 1 mm rounding
!        z10 → 10 mm
!        z50 → 50 mm
!        z100 → 100 mm
!     Higher zone = smoother & faster motion, but less precise
!
!
!  --> CoordonatesVar := [ [X,Y,Z], [Q1,Q2,Q3,Q4], [CF1,CF2,CF3,CF4], [E1,E2,E3,E4,E5,E6] ]
!                    [X,Y,Z]               <-- position of the TCP (Tool Center Point) in cartesian
!                    [Q1,Q2,Q3,Q4]           <-- orientation (or quaternion) it represents the tool's rotation in 3D
!                    [CF1,CF2,CF3,CF4]       <-- configuration data, it contains robot arm posture information
!                    [E1,E2,E3,E4,E5,E6]     <-- external axes (9E+09 = ignore this axis)
!
!
!  --> More details about the configuration data part: it contains robot arm posture information:
!        CF1: Arm configuration (elbow up/down, shoulder left/right)
!        CF2: Wrist configuration
!        CF3: Turn number (revolutions of axes)
!        CF4: external kinematic configuration
!      ABB robots need this to choose one of several possible robot postures that reach the same point
!
!
!  --> More details about the external axis: it's data for robots with:
!        linear tracks
!        positioners
!        extra rotary axes
!      9E+09 means: ignore this axis / no external axis used
!
```

!