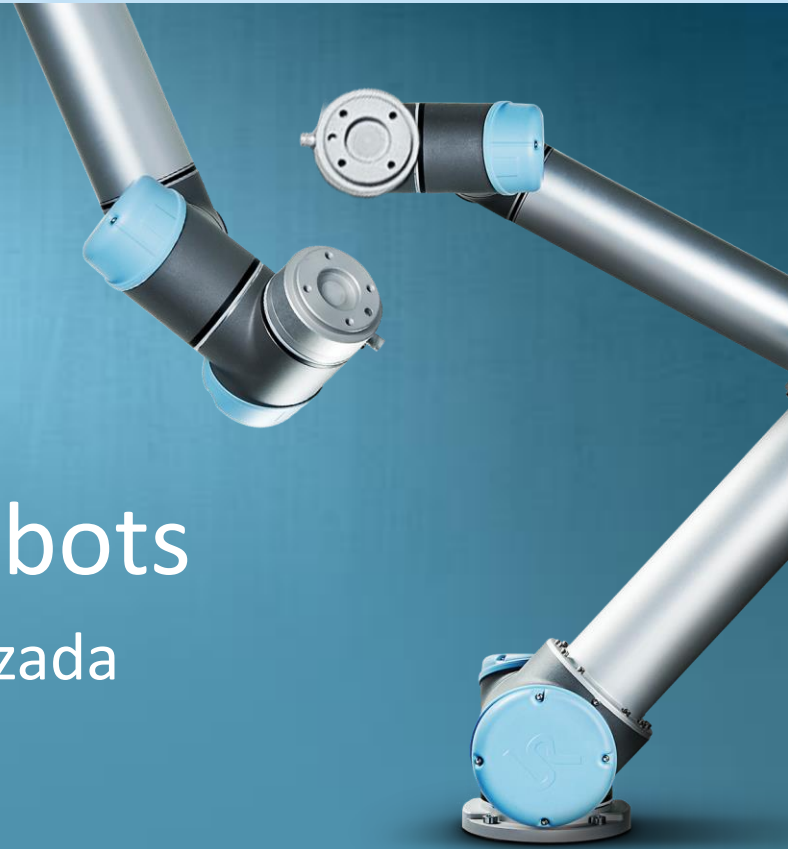
UNIVERSAL ROBOTS



# Universal Robots

## Formación Avanzada

# Bienvenido a la Formación Avanzada

- Experiencia práctica
- Memoria USB con URSim
- Test final de 30 minutos

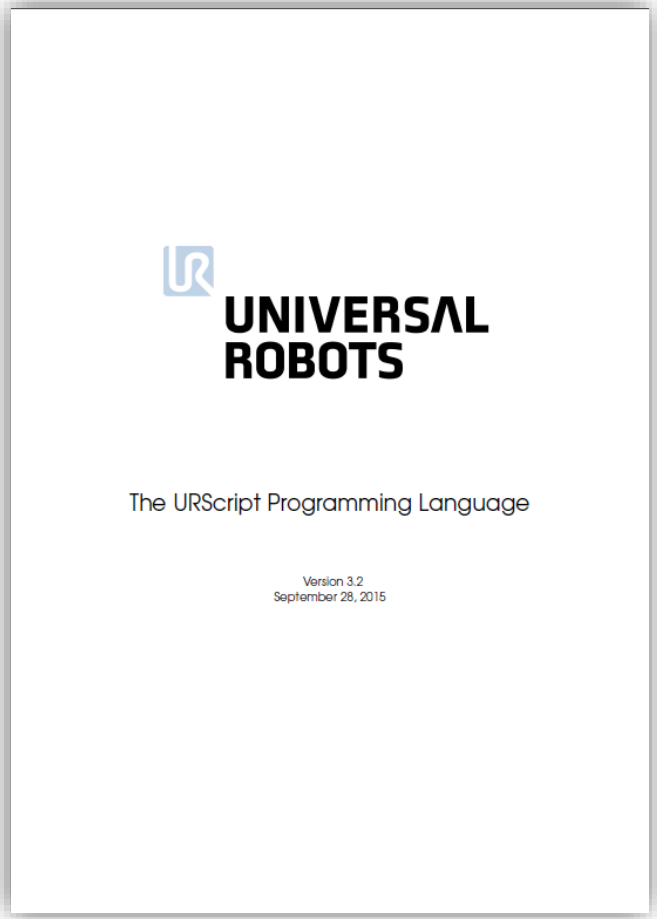
# Presentación del Profesor

Nombre: *Jordi Saboya*

Rol: *Tech Support*

## ¿Qué es el URScript?

- URScript
  - Lenguaje de alto nivel desarrollado por UR
  - Puede ser usado en lugar de crear los programas con PolyScope GUI
  - Similar al lenguaje de programación Python
  - El manual de Script contiene la definición de todas las funciones script disponibles
  - Los programas de Polyscope  son convertidos a URScript antes de su ejecución
  - Potencialmente se puede usar de diferentes formas



## Cómo usar URScript

- Podemos hacer uso de URScript de diversas formas:
  - Asignación
  - Código de script – Line
  - Código de script – File
  - Llamada de funciones
  - Interfaces Cliente

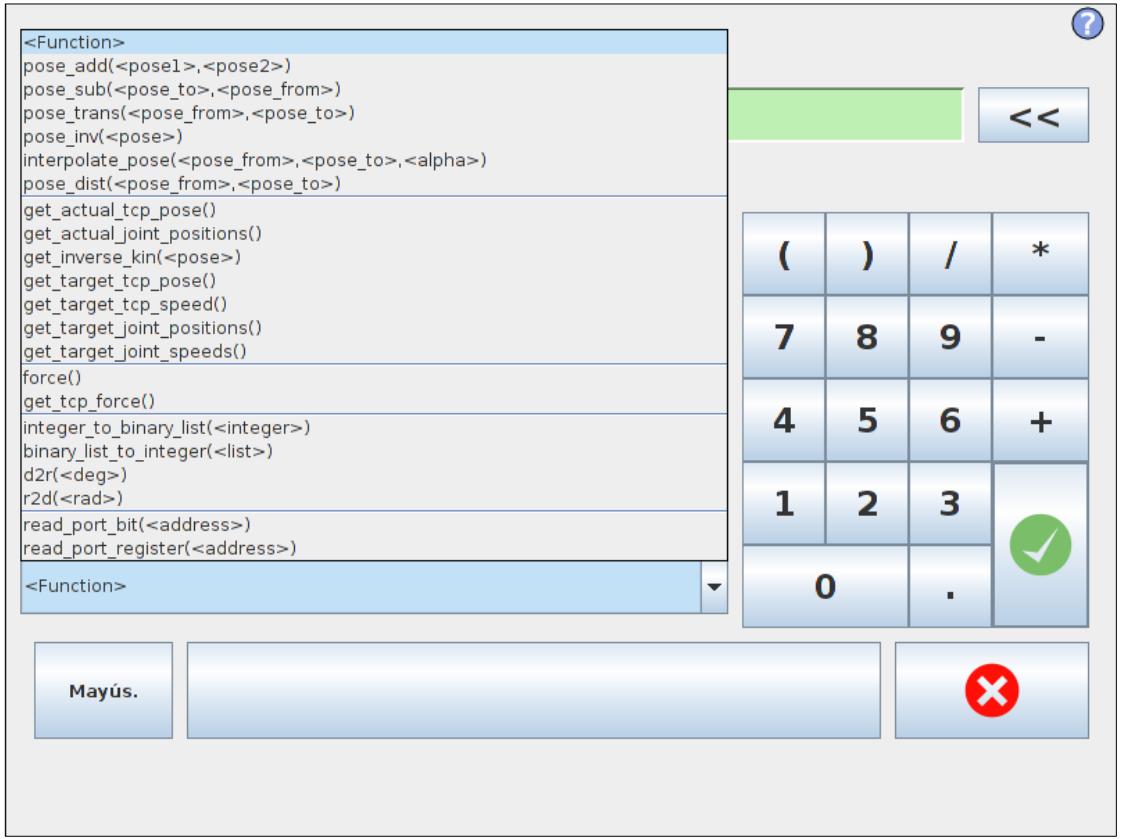
## Asignación

- Ejecuta una instrucción script
- Asigna el valor devuelto a una variable
- Permite que se monitoricen en la pestaña de variables

## Editor de expresiones

- Funciones script más comunes aparecen en la lista
- Facilita el uso de funciones, evitando tener que teclear la función completa
- Ejemplo de programa usando la función *force()*

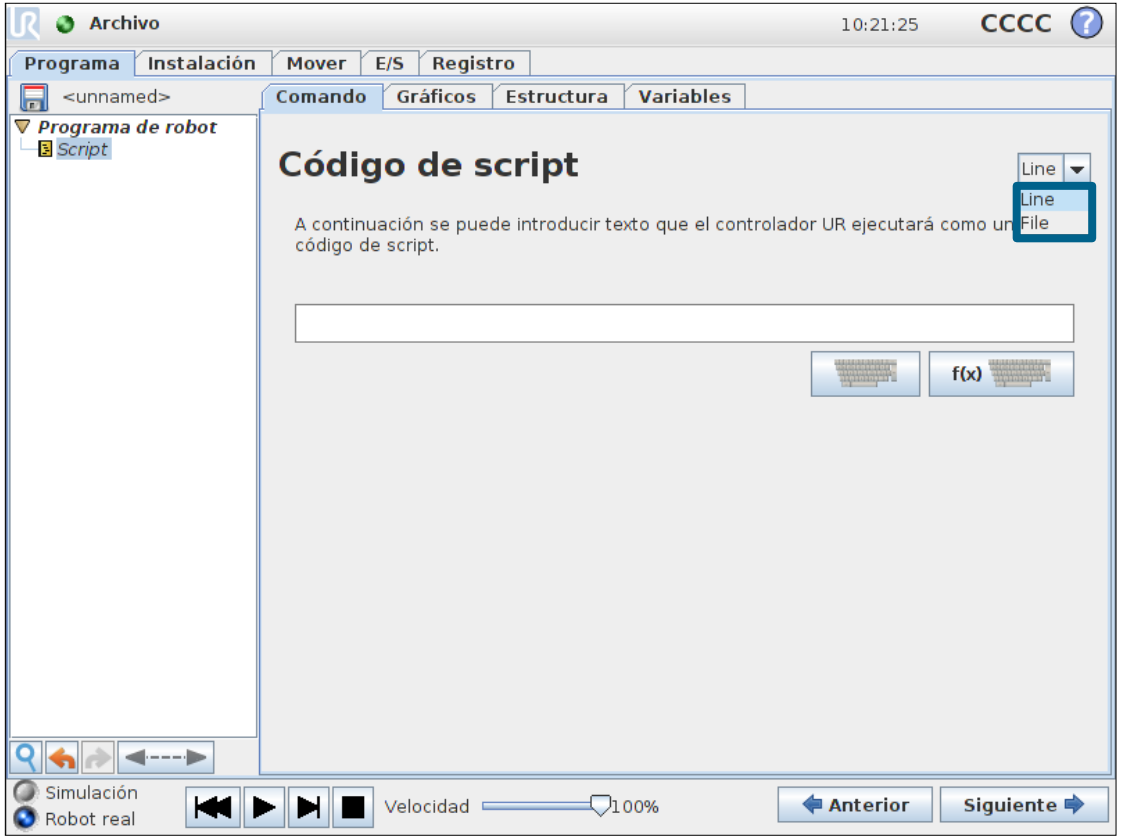
Programa de robot
MoveL
Punto_de_paso_1
If force() < 30
Punto_de_paso_2



## Código de script - Line

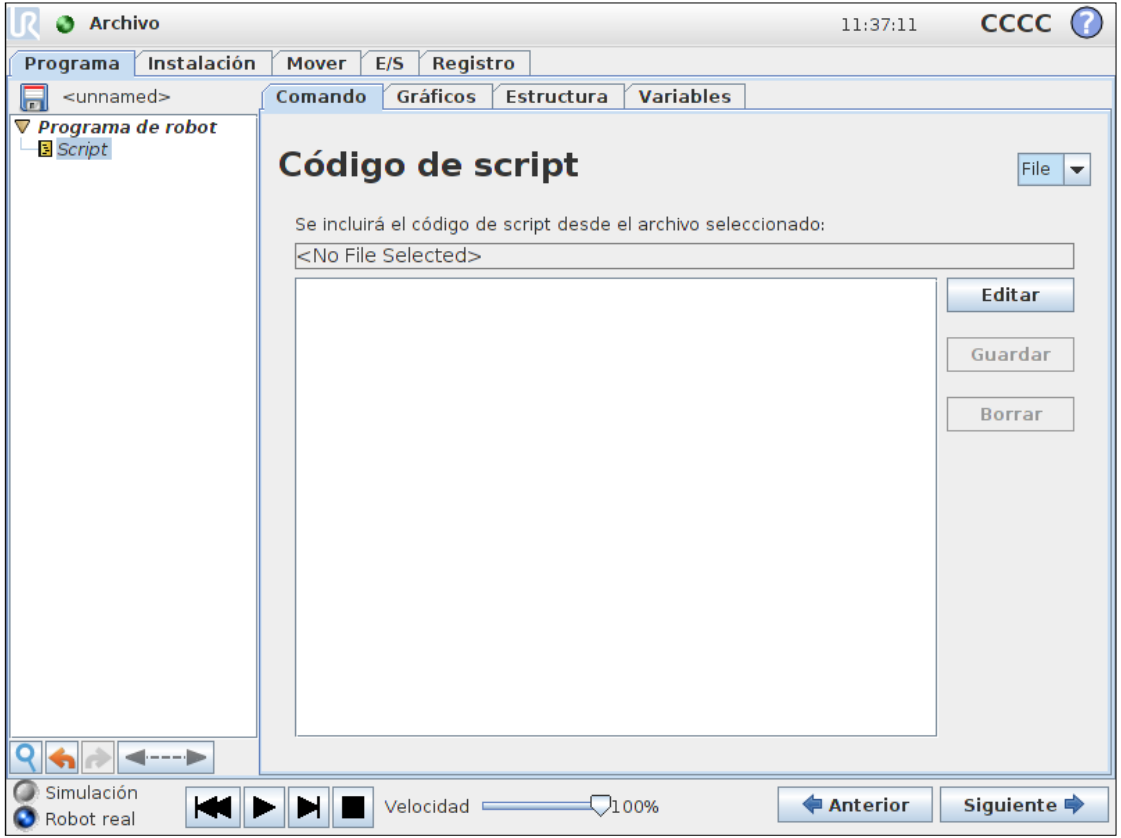
- Ejecuta una única línea de URScript
  - Las variables definidas en script no aparecen en la pestaña de variables
  - Las posiciones en script no aparecen en la pestaña de gráficos

```
Programa de robot
  movel(p[0,-0.4,0.3,0,3.14,0])
  sleep(1)
  movel(p[0.2,-0.4,0.3,0,3.14,0])
```



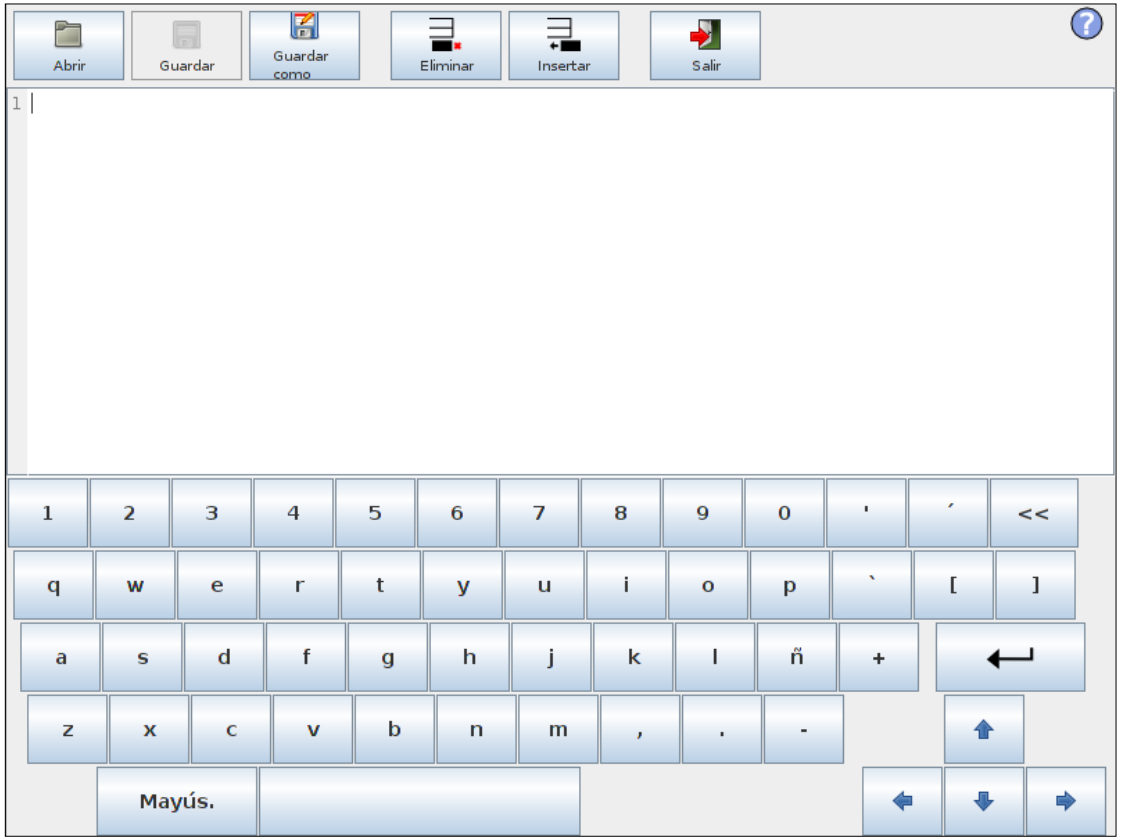
## Código de script - File

- Cargar y ejecutar un fichero script existente
- Ejecuta múltiples líneas de script
- Posibilidad de transferir los ficheros script por ethernet, será tratado posteriormente en el curso
- Seleccionar botón Editar



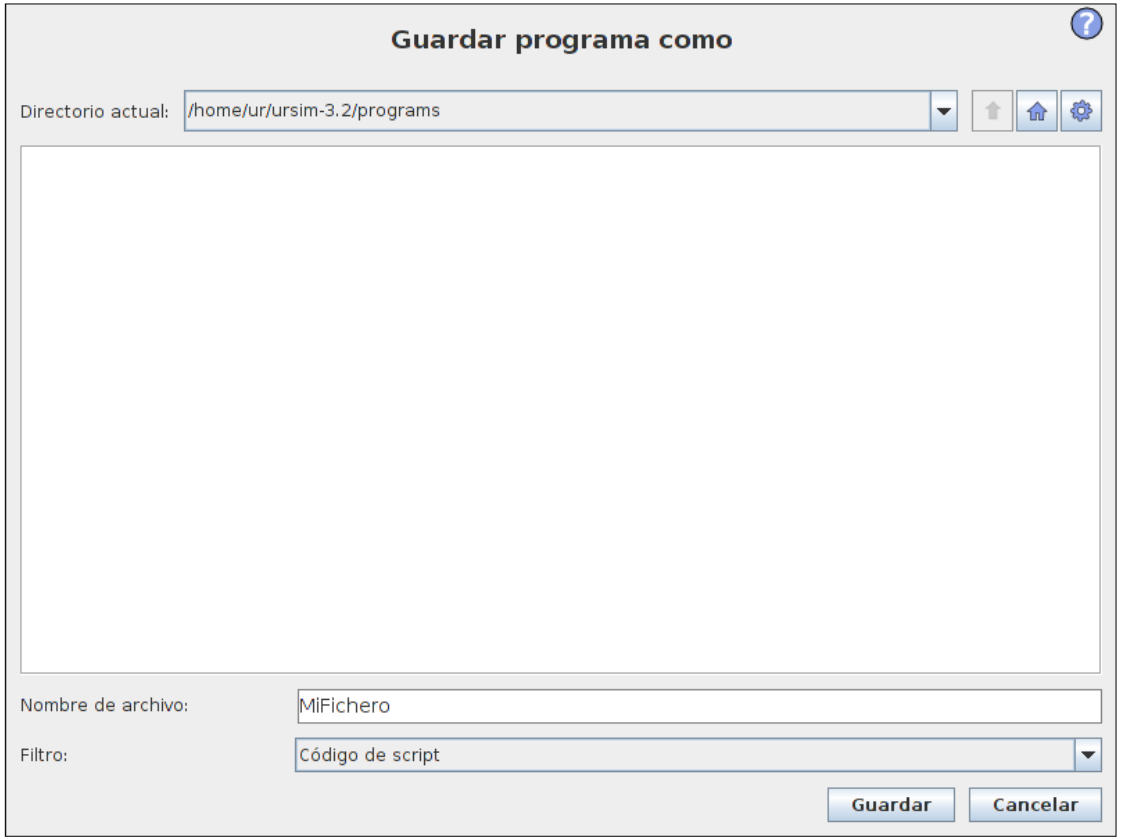
## Código de script - File

- Seleccionar Guardar como



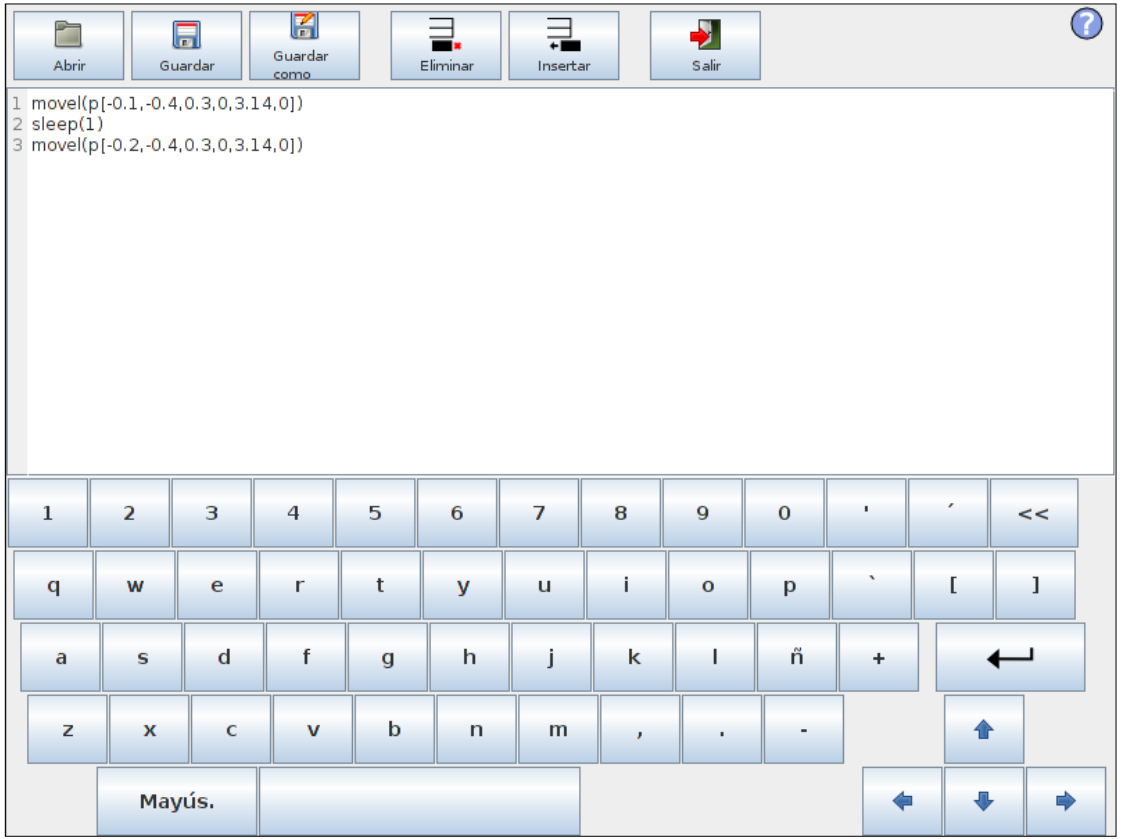
## Código de script - File

- En las opciones de Filtro se pueden seleccionar Todos los archivos o Código de script
- Si se selecciona Todos los archivos, se debe nombrar el archivo con la extensión *.script*
- Guardar el archivo



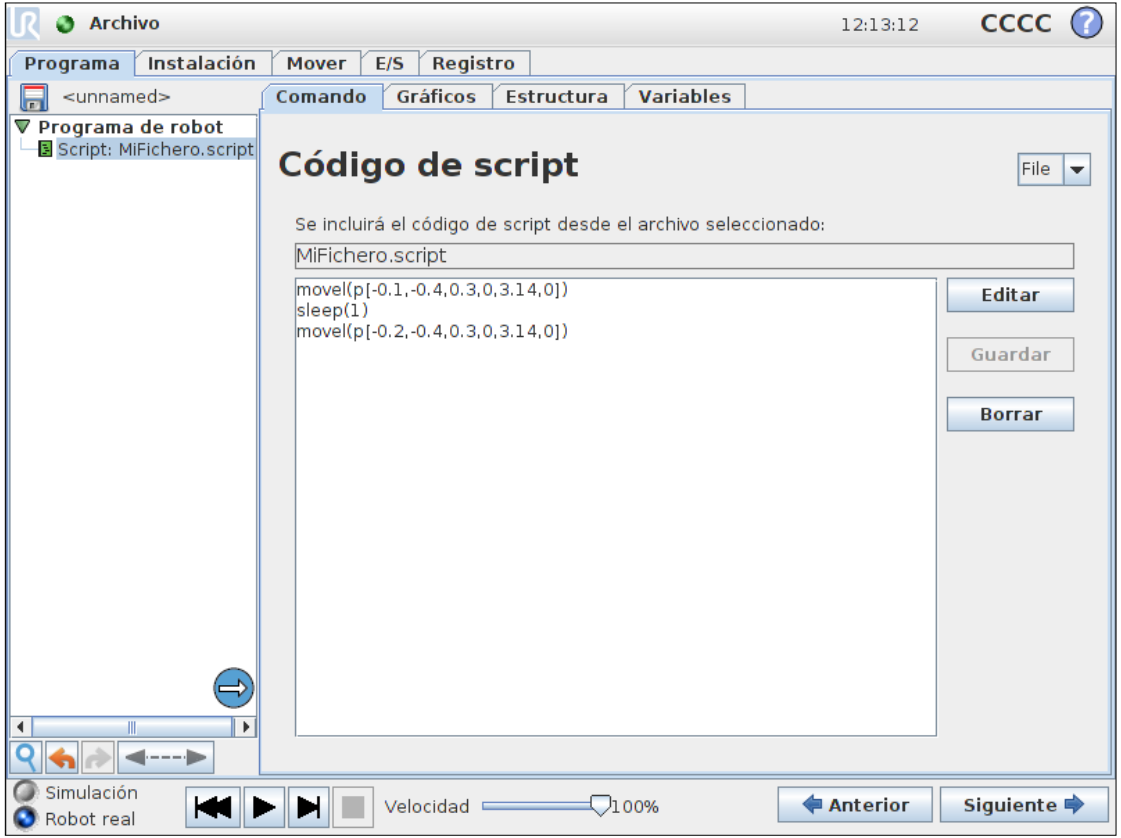
## Código de script - File

- Insertar tres nuevas líneas
- Escribir una línea de código script en cada una de ellas
- Guardar & Salir



## Código de script - File

- Ejecutar el programa



## Código de script - Funciones

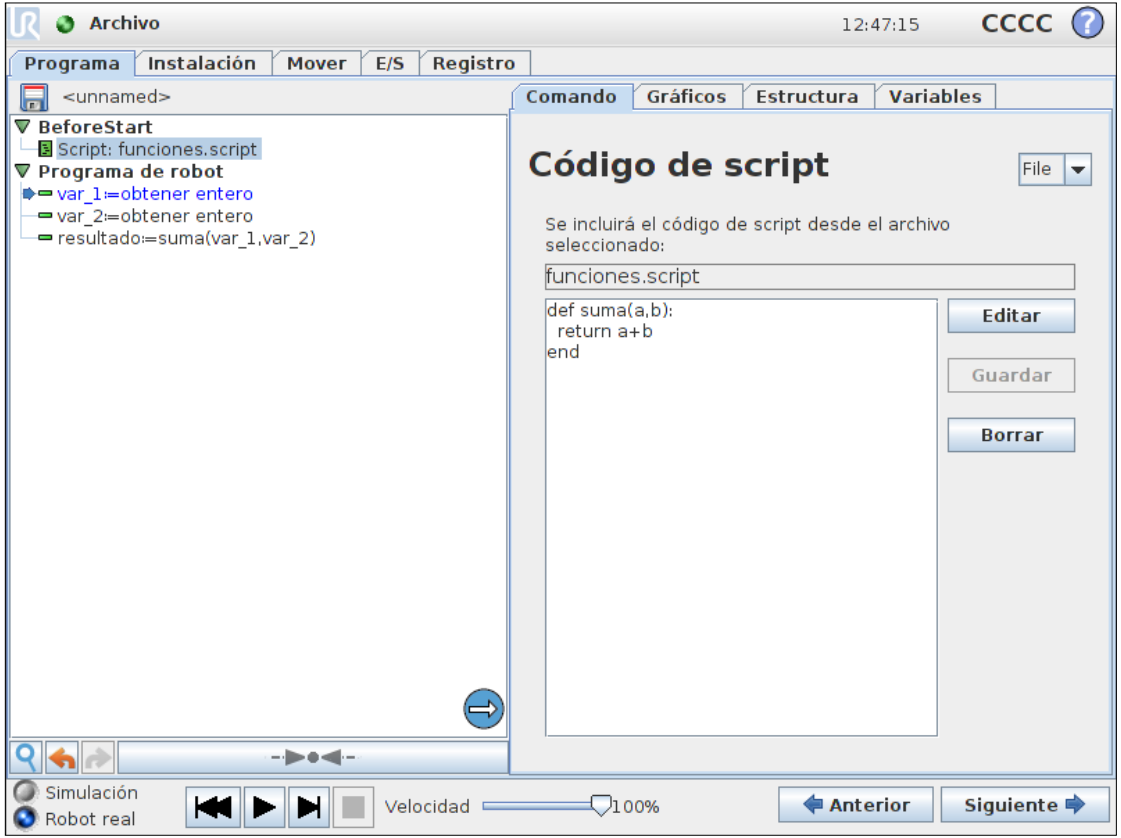
- Una función en script se declara como
  - def name()
    - script
    - …
  - end

- Una función puede tomar argumentos y devolver valores procesados
  - def name(argument)
    - …
    - …
    - return
  - end

- Una función puede ser llamada en múltiples ocasiones dentro del programa

## Código de script - Funciones

- Función para la suma de dos valores que devuelve el resultado
  - Crear un fichero script con la función
    - Nombrar la función
    - Definir dos argumentos
    - Añadir return
  - Declarar dos variables mediante una asignación de operario
  - Declarar la variable y asignarle el resultado de la función según los dos argumentos



## Interfaces Cliente

- Los códigos de script pueden ser enviados directamente al controlador a través de Ethernet desde un dispositivo externo
- Existen 3 interfaces de clientes ejecutándose automáticamente desde el arranque del robot
- Las interfaces de clientes serán tratados posteriormente en el curso

## Ejercicio práctico

- Crear un breve fichero URScript que:
  - Defina una función
  - Reciba un argumento
  - Procese el valor recibido
  - Devuelva el valor procesado
- Cargar este fichero en un programa
- Llamar a la función desde una asignación
- Ejecutar el programa
- Verificar los resultados

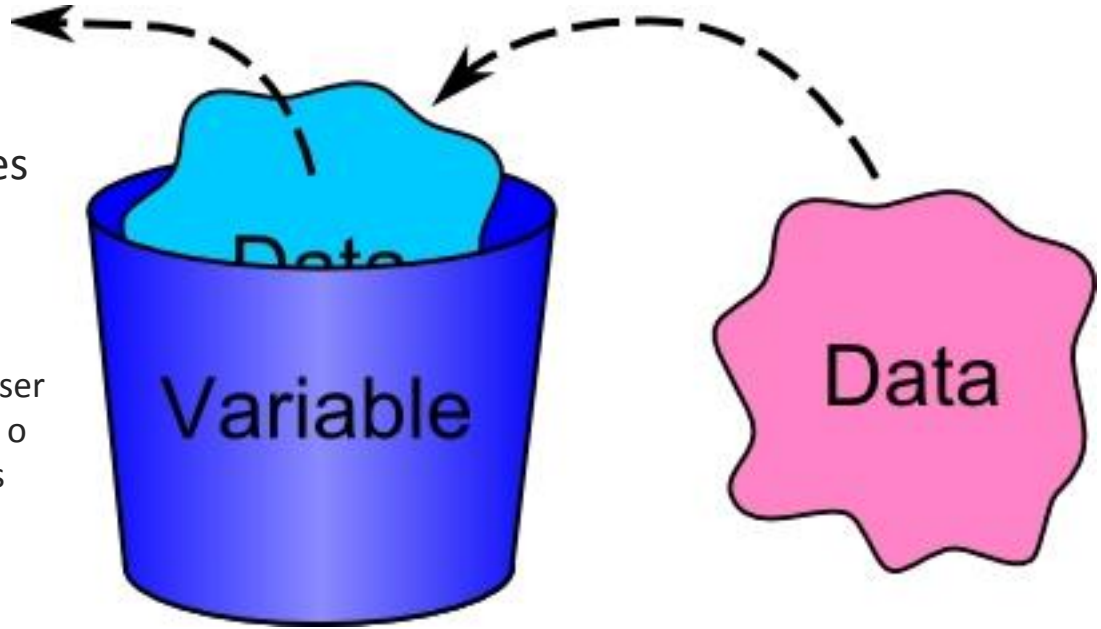
## ¿Qué es una variable?

Recordatorio Formación Principal

- Una variable es un espacio de almacenamiento (contenedor)
  - Su contenido puede cambiar

- Lectura/Escritura de variables
  - Su valor puede ser sobrescrito
  - Su valor puede ser leído

  - El valor de las variables puede ser comparado con otras variables o con los estados de los sensores



## Tipos de variable

Recordatorio
Formación Principal

| Tipo de variable | Valor |
|---|---|
| `boolean` | True / False |
| `integer` | Números enteros (32 bit) |
| `floating point` | Números reales (decimal) |
| `string` | Texto (caracteres ASCII) |
| `pose` | Variable de posición p[x,y,z,rx,ry,rz] |
| `list` | Lista de variables |

## Ámbito de las variables

Recordatorio Formación Principal

| Ámbito | Localización |
|---|---|
| `local` | Programa |
| `global` | Instalación |

- Variables Locales
  - Declaradas en el programa
  - Accesibles desde el mismo programa
  - Su valor se pierde cuando se desconecta la alimentación
- Variables Globales
  - Declaradas en Instalación
  - Accesibles desde todos los programas que usan la misma Instalación
  - Su valor se almacena en un archivo en la memoria

## Comando *Asignación*

- Opciones
  - Definir nombre de variable
  - Declarar tipo de variable
  - Asignar valor a la variable


```
Programa de robot
    var_1 = True
    Espera 0.5
    var_1 = False
    Espera 0.5
```

- Guardar programa de ejemplo como var_bool.urp

## Variable tipo Pose

- Definición
  - Es un vector que describe la posición y orientación de un punto en el espacio cartesiano
  - Es la combinación de
    - vector de posición (x,y,z)
    - vector de rotación (rx,ry,rz)

```
pose_var = p[ x , y , z , rx , ry , rz ]
```

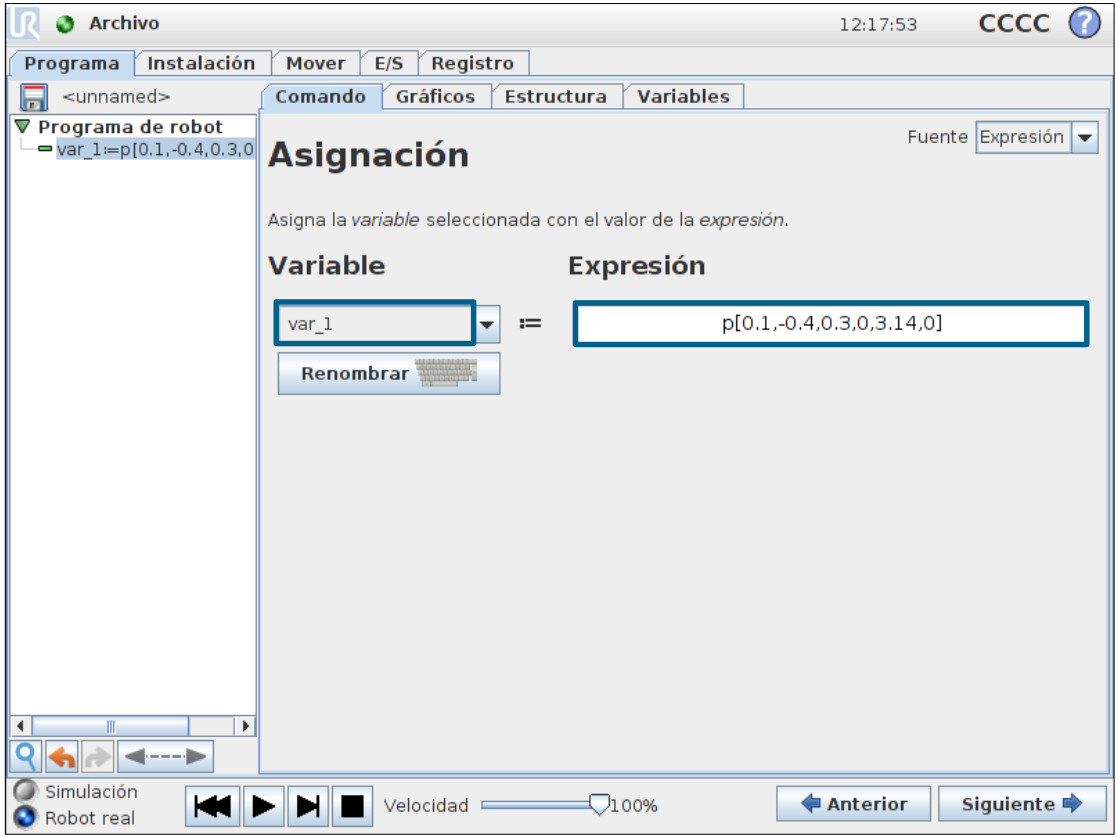
nombre de la variable — posición PCH — vector de rotación

  - Unidedes
    - x,y,z = metros
    - rx,ry,rz = radianes

## Trabajando con Variables tipo Pose

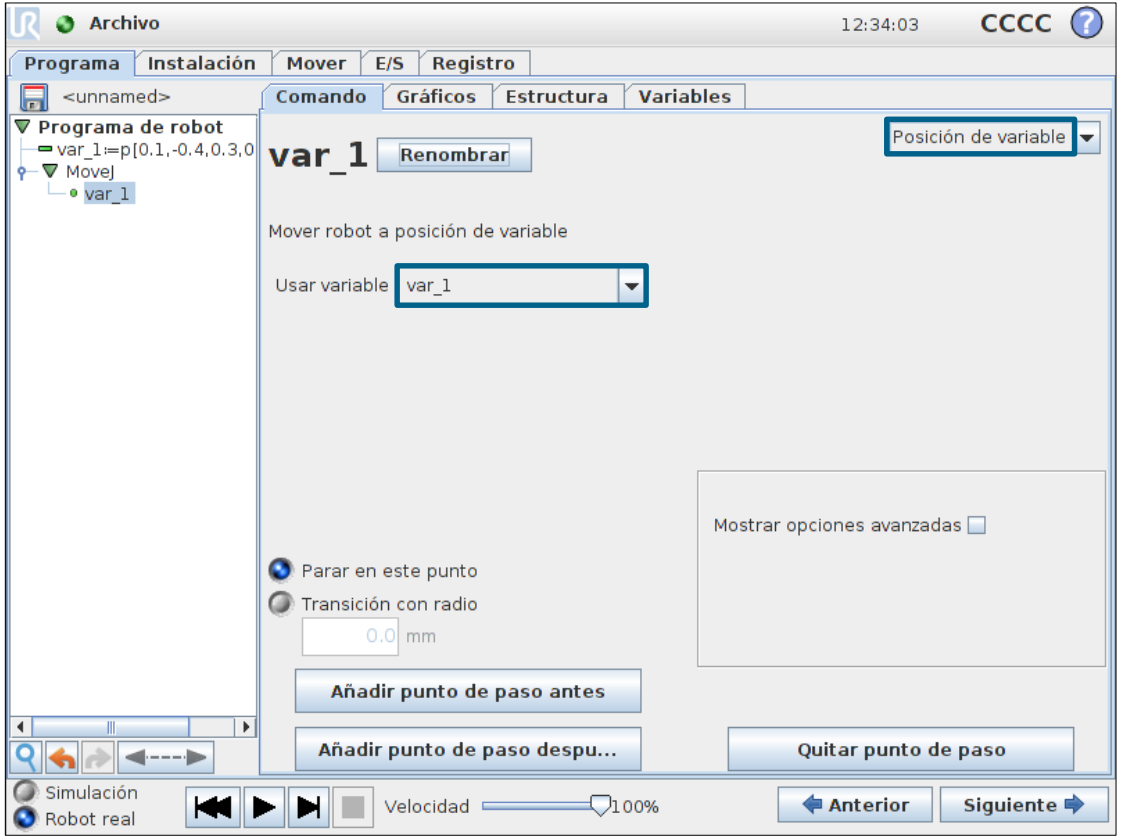
- Entrada manual de pose
  - Insertar una Asignación
  - Definir nombre de la variable
  - Introducir expresión como pose

    Ejemplo:
    - x = 100 mm
    - y = -400 mm
    - z = 300 mm
    - rx = 0
    - ry = 3.14
    - rz = 0



## Trabajando con Variables tipo Pose

- Mover a una pose
  - Insertar Movimiento
  - Seleccionar Posición de variable
  - Usar variable




- *NOTA: Si la primera posición en el programa es variable o relativa, la función de AutoMover no se ejecutará*

## Funciones script con variables Pose

- Funciones script disponibles para trabajar con variables tipo Pose

| Código Script | Función |
|---|---|
| **get_actual_tcp_pose()** | Devuelve la posición actual medida del PCH |
| get_actual_tcp_speed() | Devuelve la velocidad actual medida del PCH |
| get_inverse_kin() | Devuelve valores de cinemática inversa |
| get_target_tcp_pose() | Devuelve la posición objetivo actual del PCH |
| get_target_tcp_speed() | Devuelve la velocidad objetivo actual del PCH |
| interpolate_pose(*p_from, p_to, alpha*) | Interpolación lineal de la posición y orientación del PCH |
| **pose_add(*p_1, p_2*)** | Suma de Poses |
| pose_dist(*p_from, p_to*) | Devuelve la distancia entre dos Poses |
| pose_inv(*p_from*) | Devuelve la inversa de una Pose |
| pose_sub(*p_to, p_from*) | Sustracción de Poses |
| **pose_trans(*p_from, p_to*)** | Transformación de Poses |

## Ejemplo: get_actual_tcp_pose()

- Cómo leer la posición actual y mover a una posición de seguridad
  - Añadir secuencia BeforeStart
  - Leer la posición actual y almacenarla en una variable
  - Guardar el valor de “z” en otra variable
  - Definir una nueva variable pose
    - Con el mismo valor a excepción de “z”
    - Asignar a “z” el valor de 400 mm
  - Mover robot a la variable “pos_segura”
    - Reducir velocidad

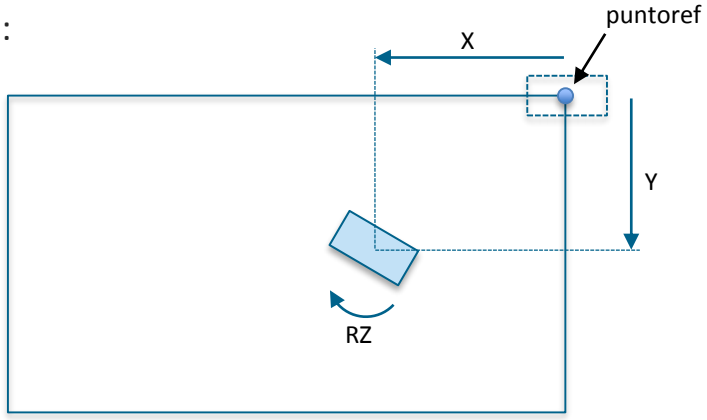
```
BeforeStart
    pa = get_actual_tcp_pose()
    z = pa[2]
    pos_segura = p[pa[0], pa[1], 0.4, pa[3], pa[4], pa[5]]
    Esperar 1
    MoveL
        pos_segura
Programa de robot
    Detener
```

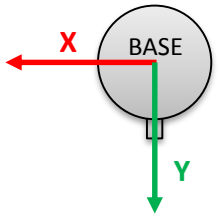
| Elmentos Pose | |
|---|---|
| **p[x,y,z,rx,ry,rz]** | **No. índice** |
| x | [0] |
| y | [1] |
| z | [2] |
| rx | [3] |
| ry | [4] |
| rz | [5] |

## Ejemplo: pose_add()

- Cómo mover desde una posición de referencia a una posición de coger variable
  - Datos simulados de equipo de visión
  - Desplazamiento del objeto desde una posición de referencia:
    - x = 250 mm
    - y = 120 mm
    - rz = 31 grados
  - Definir la variable Pose en función del desplazamiento
  - Añadir a la variable “puntoref” usando *pose_add()*
    - *pose_add()* usa la Base como referencia

*Función BASE como referencia*



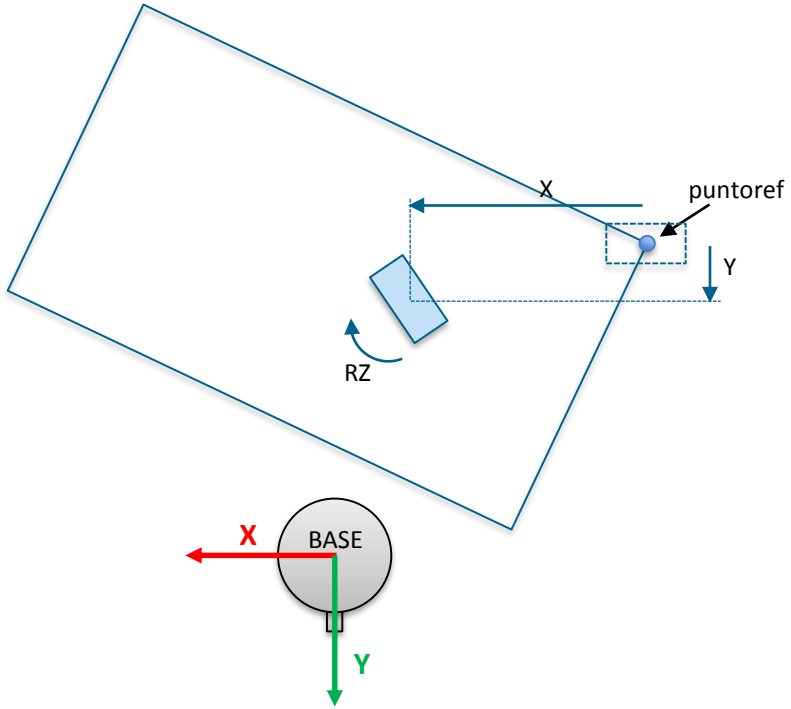




```
Programa de robot
  MoveJ
    puntoref
  Esperar DI[0] = HI
  Carpeta calculo de posicion
    x = 250
    y = 120
    rz = 31
    desplazamiento = p[(x/1000),(y/1000),0,0,0,d2r(rz)]
    pos_coger = pose_add(puntoref, desplazamiento)
  MoveL
    pos_coger
```

## Ejemplo: pose_trans()

- En una aplicación real es poco probable qué el plano del equipo de visión se encuentre alineado con el plano de la base
- Si utilizamos aquí pose_add(), será muy difícil alcanzar la posición objetivo correcta
- Necesitamos transformar nuestro X e Y en el plano de destino
- Aquí es dónde pose_trans() resulta necesario

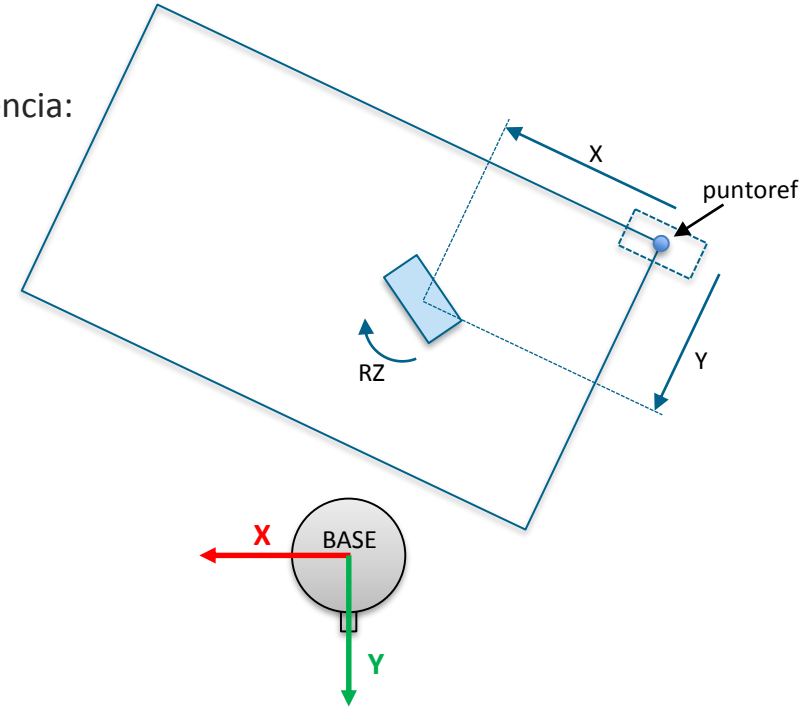
*Función BASE como referencia*



## Ejemplo: pose_trans()

- Cómo mover desde una posición de referencia a una posición de coger variable
  - Datos simulados de equipo de visión
  - Desplazamiento del objeto desde una posición de referencia:
    - x = 250 mm
    - y = 120 mm
    - rz = 31 grados
  - Definir la variable Pose en función del desplazamiento
  - Añadir a la variable “puntoref” usando *pose_trans()*

*PUNTOREF como referencia*



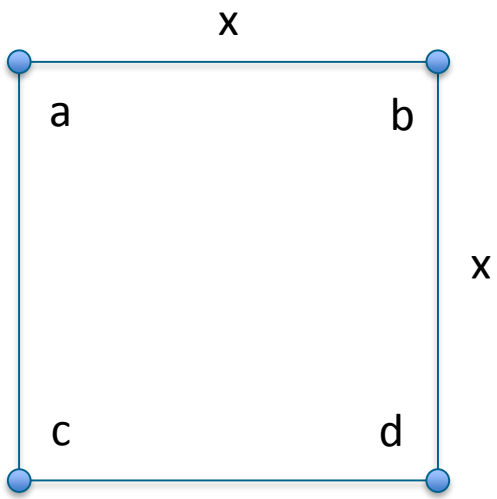

```
Programa de robot
  MoveJ
    puntoref
  Esperar DI[0] = HI
  Carpeta calculo de posicion
    x = 250
    y = 120
    rz = 31
    desplazamiento = p[(x/1000),(y/1000),0,0,0,d2r(rz)]
    pos_coger = pose_trans(puntoref, desplazamiento)
  MoveL
    pos_coger
```

## Ejercicio práctico parte 1

- A partir de la posición actual del robot, crear una trayectoria cuadrada de 10 cm de lado:

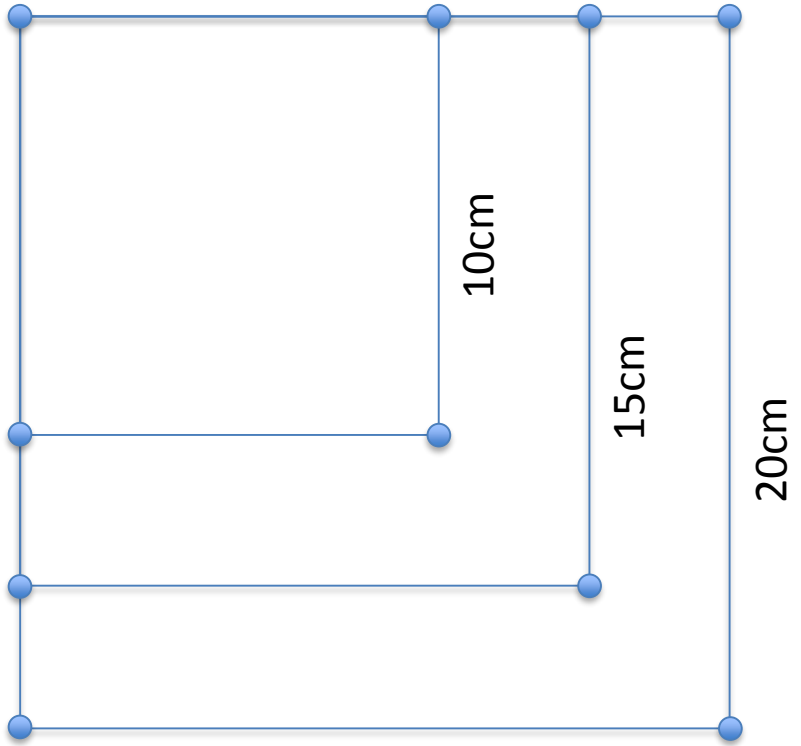
- Usar:
  - get_actual_tcp_pose()
  - pose_trans()
  - MoveL

- Usar la variable x para definir la longitud del lado



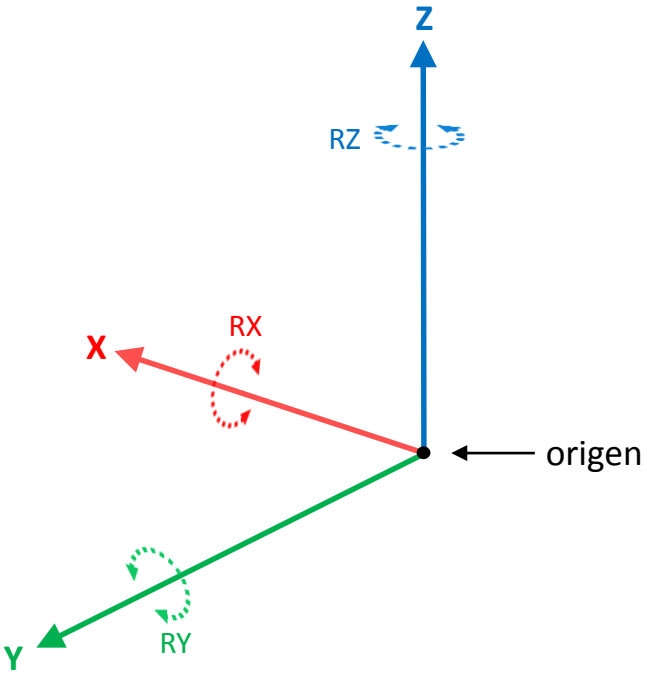
## Ejercicio práctico parte 2

- Usar el programa de la primera parte del ejercicio y las mismas variables
- Mover alrededor del cuadrado 3 veces
- Añadir en cada pasada 5 cm a la longitud del lado



## ¿Qué es una Función?

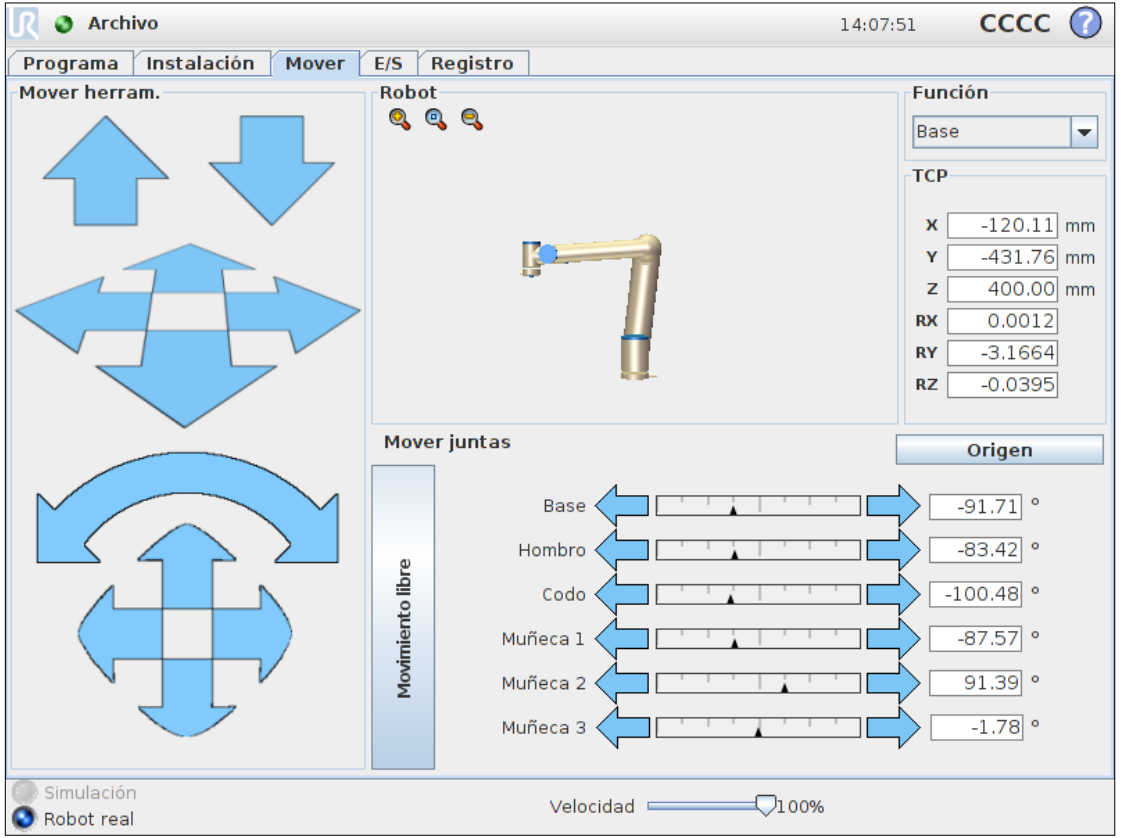
- Las funciones son sistemas de coordenadas cartesianos
  - Un sistemas de coordenadas cartesiano define un plano o espacio mediante ejes desde un punto fijo denominado origen




  - Las funciones son guardadas como variables tipo Pose

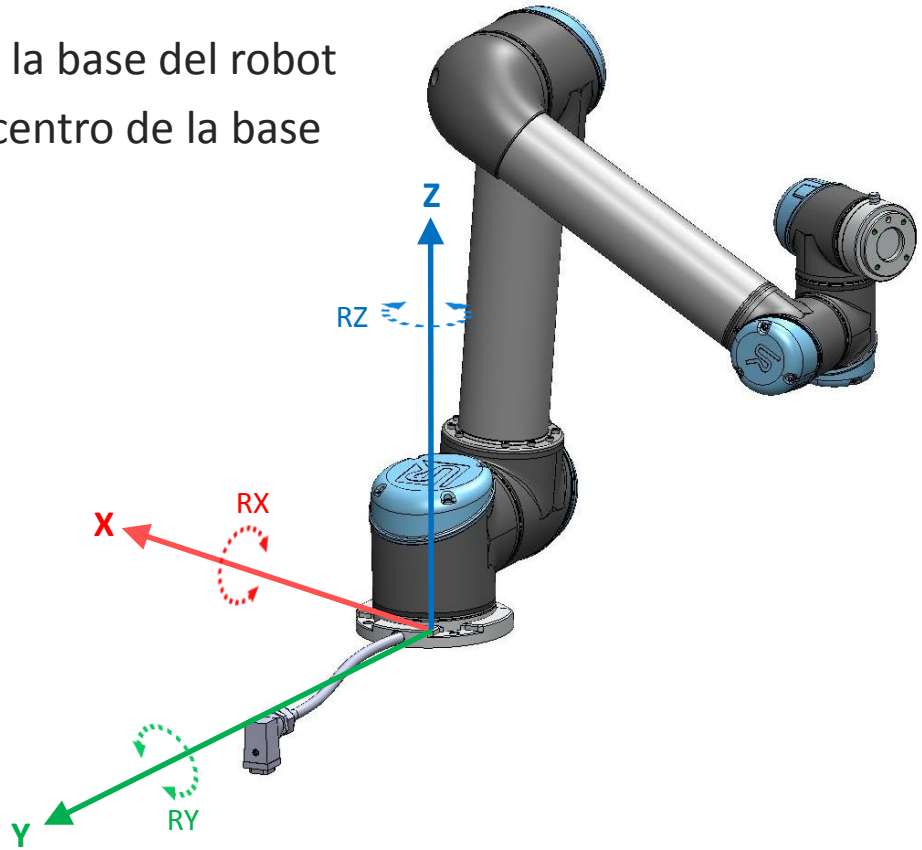
## ¿Qué es una Función?

- El robot opera de forma relativa a Funciones
  - Función Base
  - Función Herramienta
- Posición de la Herramienta
  - X, Y, Z = posición del PCH
    - Unidad: mm / pulgadas
  - Rx, Ry, Rz = orientación
    - Unidades: radianes
  - *NOTA: La orientación es dada por un vector de rotación, donde la longitud del vector es el ángulo es el ángulo a rotar en grados*



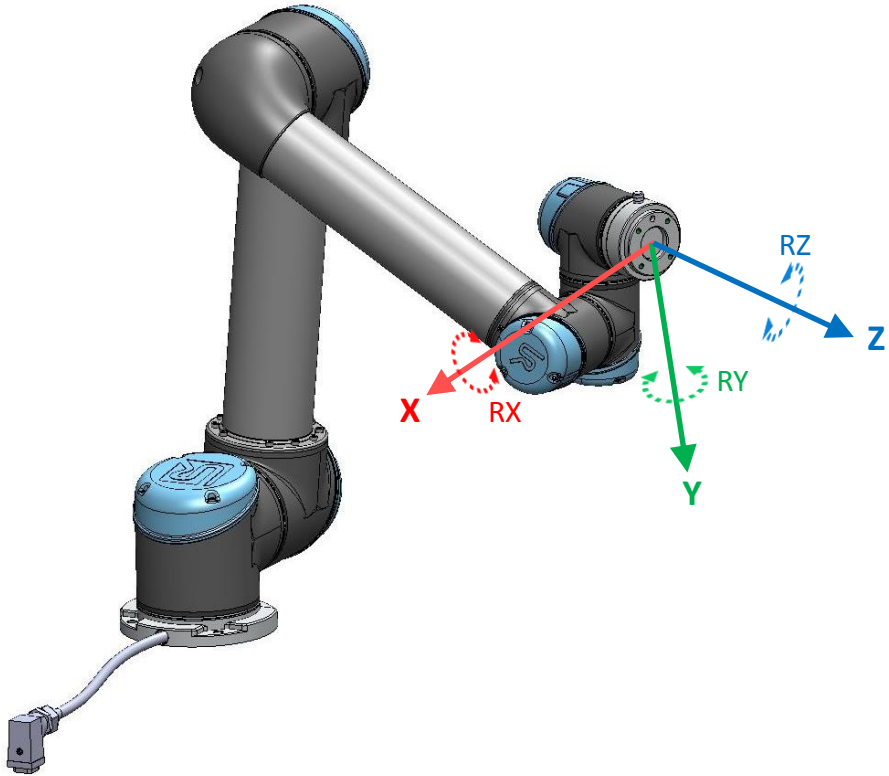
## Función Base

- Localizada en la base del robot
- Origen en el centro de la base



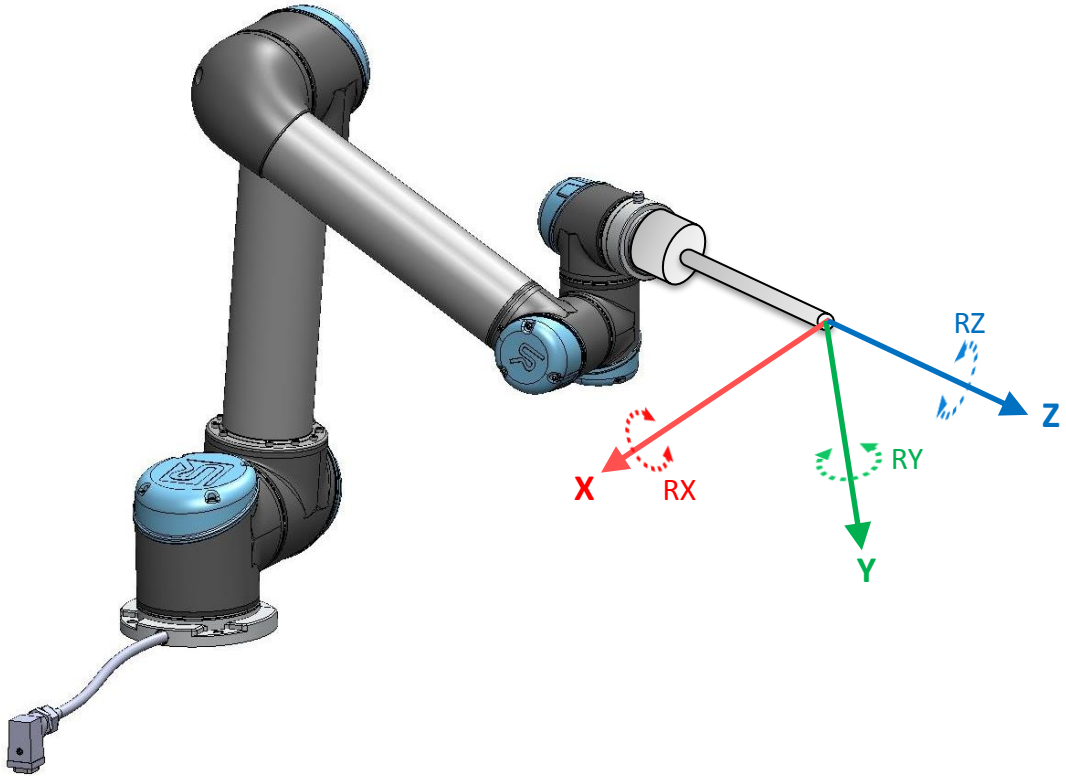
## Función Herramienta

- Origen en el centro de la brida de la herramienta



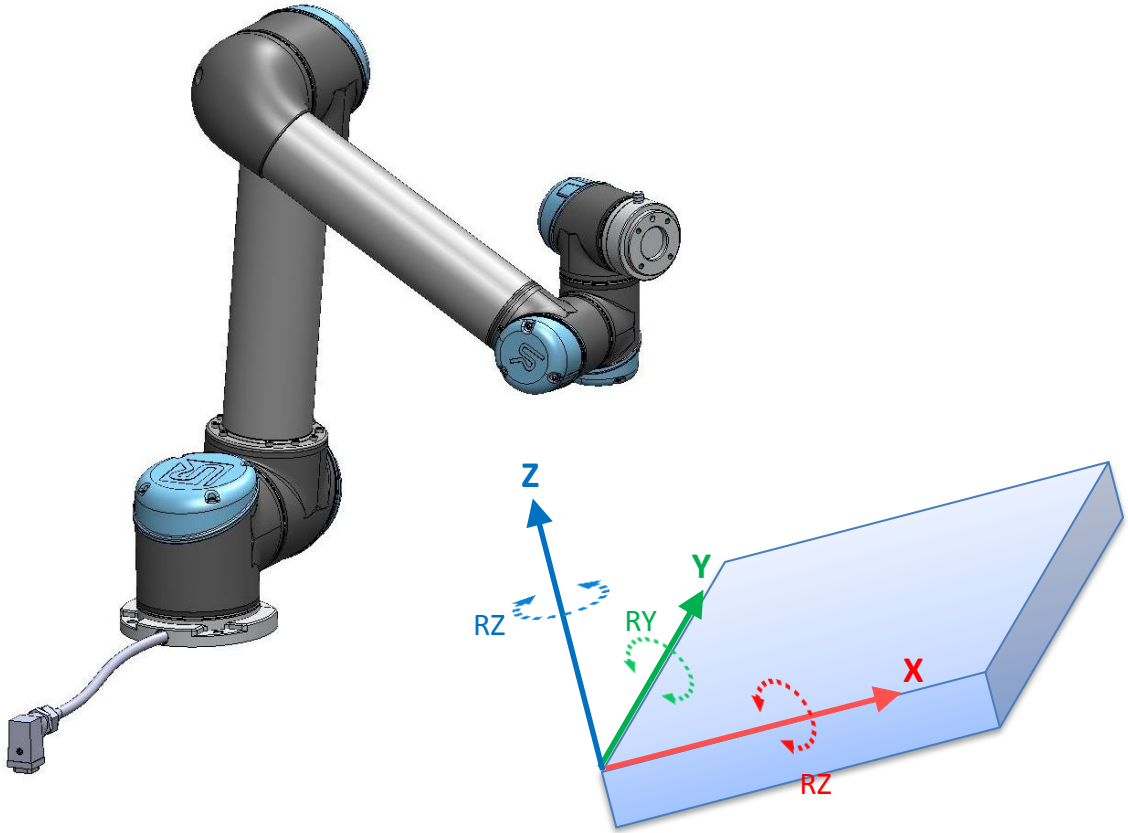
## Función Herramienta

- Origen en el PCH



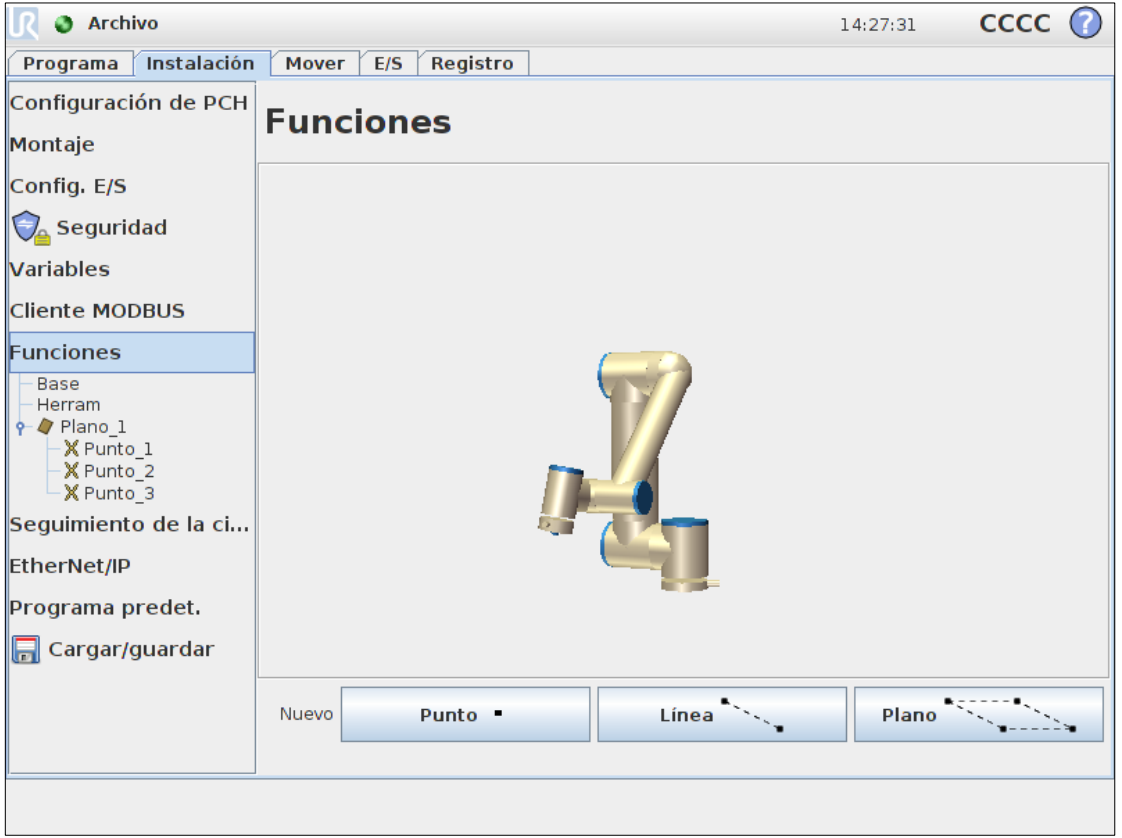
## Función definida por el usuario

- Definida por el operador



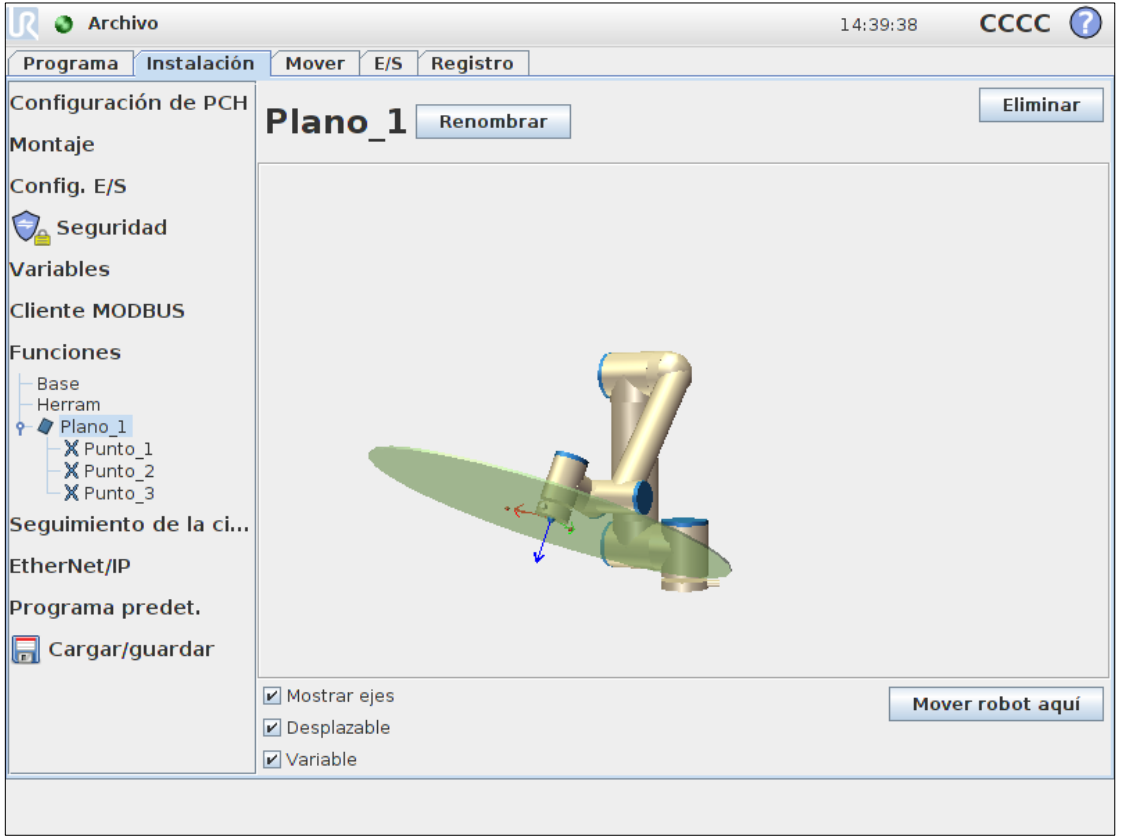
## Definición de una Función

- Funciones
  - En PolyScope un Plano se define como una Función
  - Se pueden definir múltiples funciones
  - Tipos de funciones:
    - Punto
    - Línea
    - Plano
- Añadir Función
  - Plano



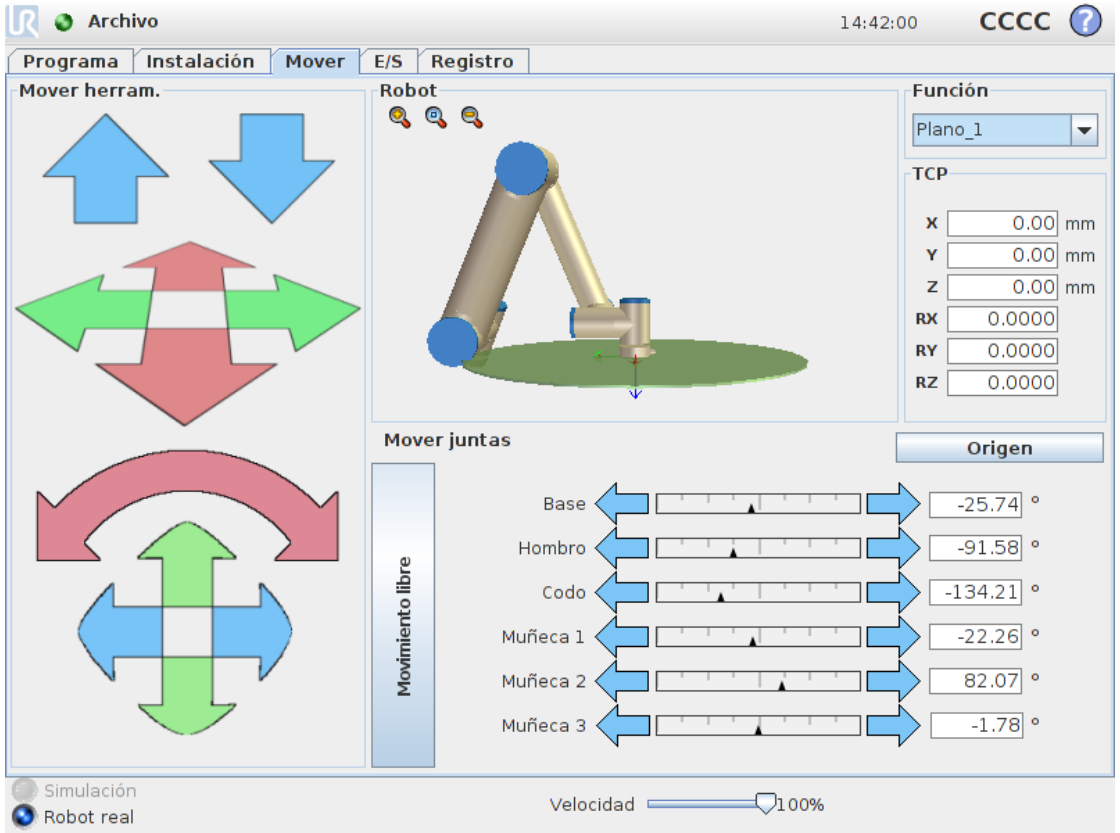
## Definición de una Función

- Propiedades
  - Mostrar ejes
    - Codificados por colores
  - Desplazable
    - Habilita la Función en la pestaña Mover
  - Variable
    - Crea una copia de la Función
    - Se puede modificar desde el programa
  - Mover robot aquí
    - Posición del PCH perpendicular a la Función



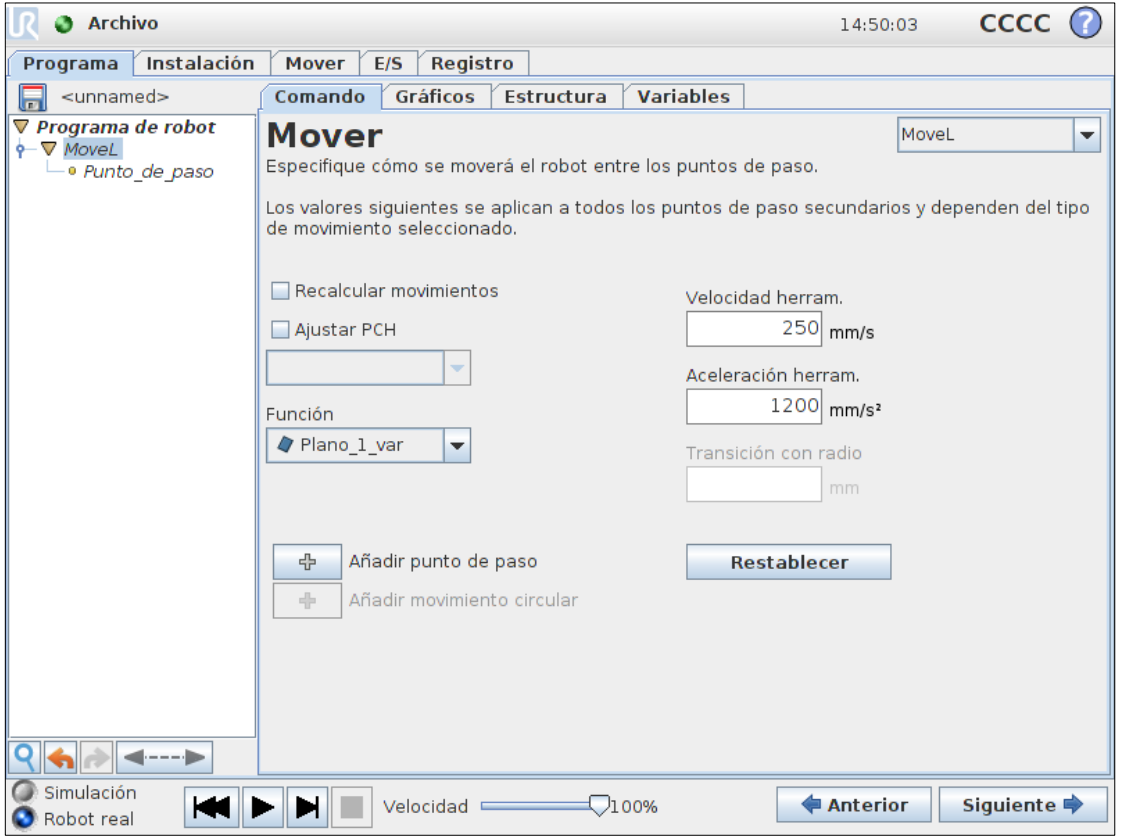
## Movimiento manual relativo a una Función

- Seleccionar Función
  - Desplazar el robot usando las flechas
  - Editor de posiciones



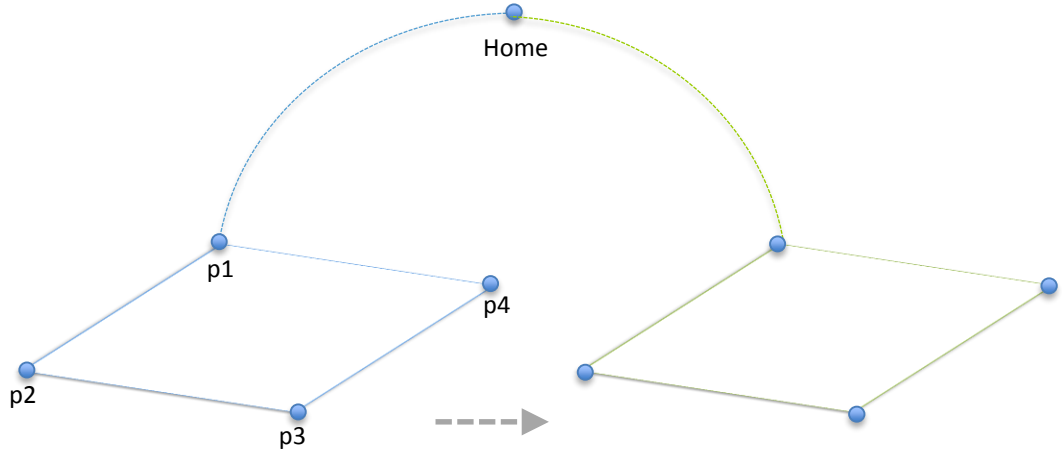
## Movimiento relativo a una Función

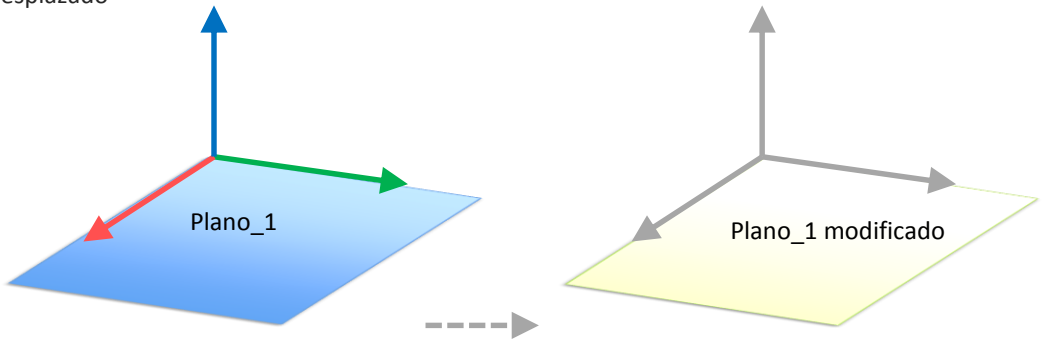
- Instrucción Movimiento
  - Función aplicable a
    - MoveL
    - MoveP
    - MoveC
  - Parámetros compartidos
    - Seleccionar la función definida por el usuario



## Trabajando con Funciones

- Crear un nuevo programa
  - Insertar MoveJ
    - Fijar posición *Home*
  - Insertar MoveL
    - Seleccionar **Plano_1** como Función
    - Fijar 4 puntos de paso
  - Ejecutar programa
  - Modificar Función
    - Seleccionar Función como Variable
    - Verificar que los puntos de paso se han desplazado



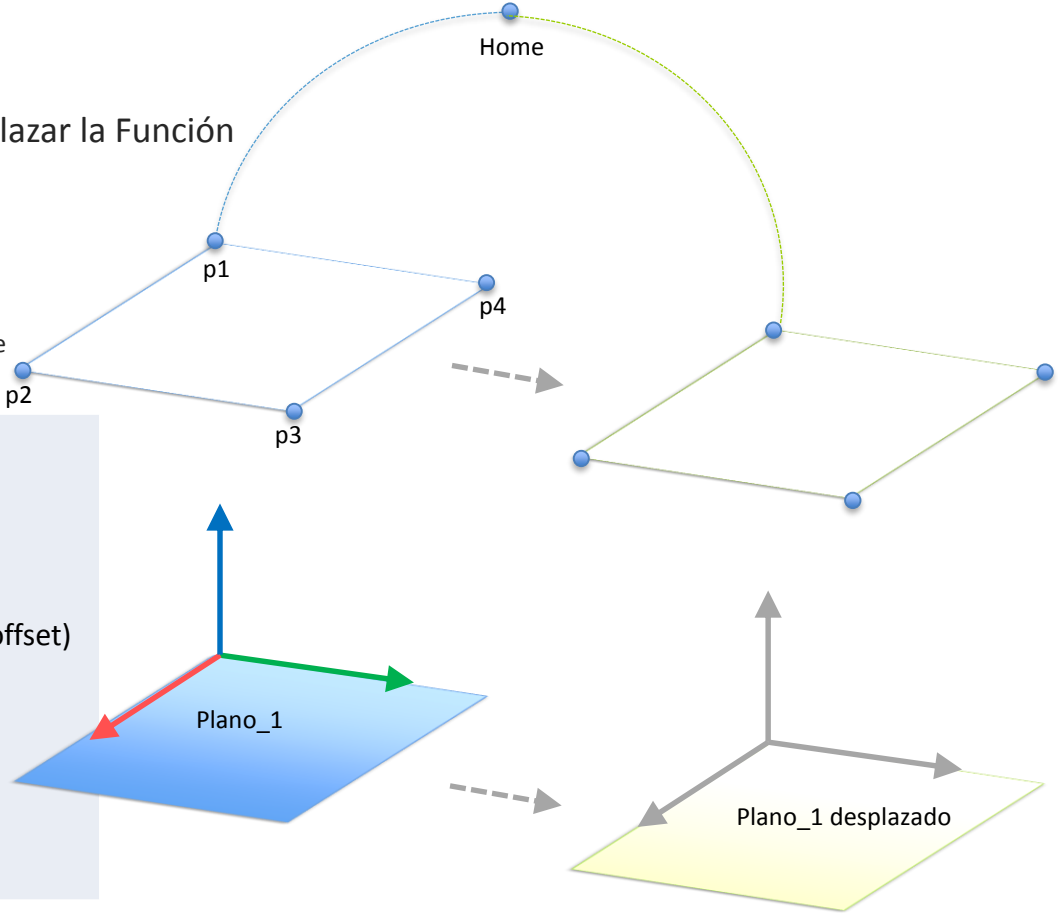




```
Programa de robot
  MoveJ
    Home
  MoveL
    Punto_de_paso_1
    Punto_de_paso_2
    Punto_de_paso_3
    Punto_de_paso_4
    Punto_de_paso_1
```

## Desplazamiento de Funciones

- Usar programa de ejemplo
  - Usar las funciones script para desplazar la Función
    - pose_add()
    - pose_trans()
  - Procedimiento
    - Definir  desplazamiento en “y”
    - Asignar valor a variable “offset” tipo Pose
    - Usar “offset” para desplazar la Función

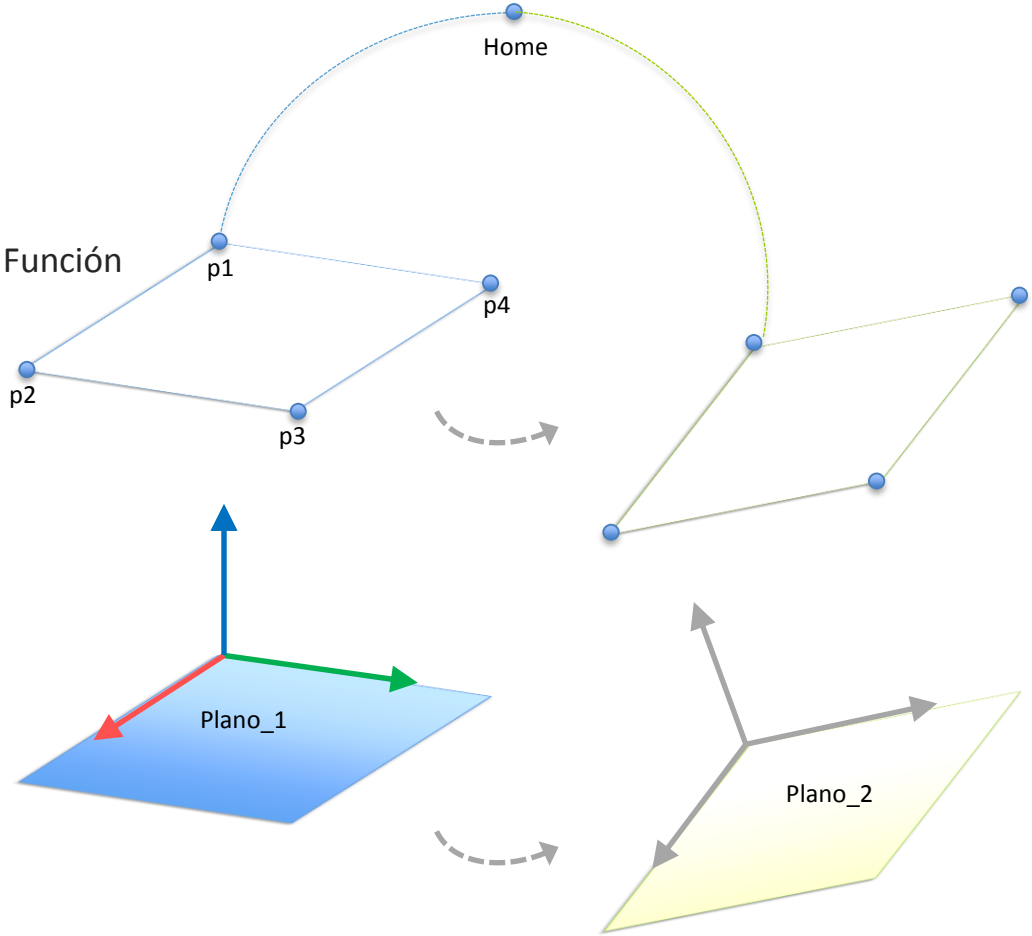
```
Programa de robot
    MoveJ
        Home
    y = 0.05
    offset = p[0,y,0,0,0,0]
    Plano_1_var = pose_trans(Plane_1_var,offset)
    MoveL
        Punto_de_paso_1
        Punto_de_paso_2
        Punto_de_paso_3
        Punto_de_paso_4
        Punto_de_paso_1
```



## Uso de múltiples Funciones

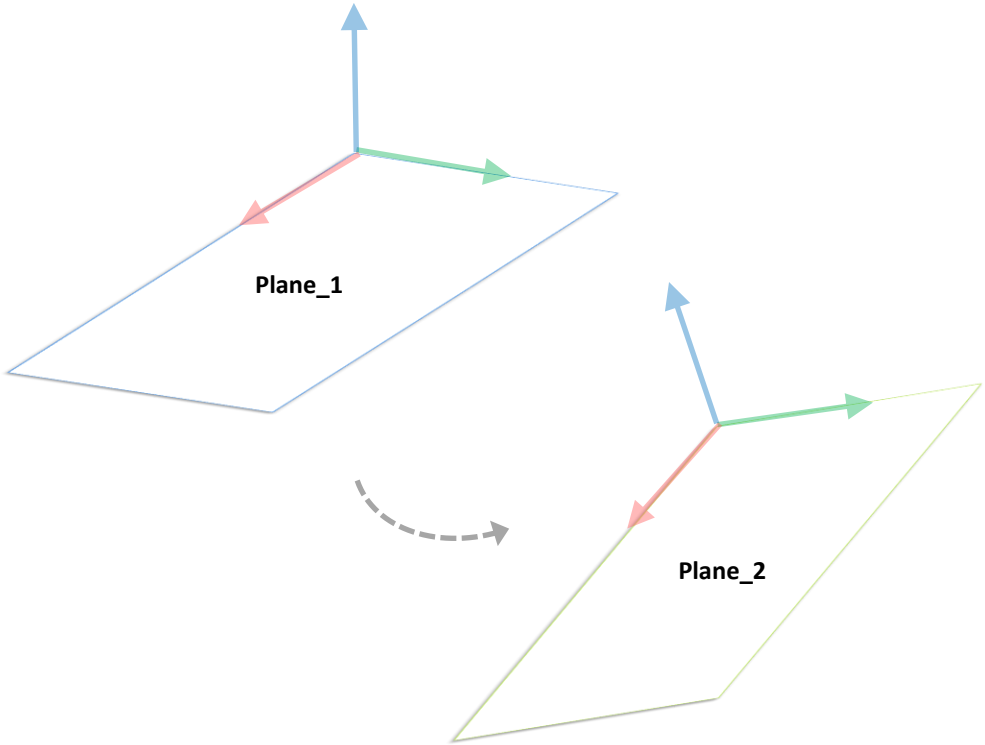
- Usar programa ejemplo
  - Crear dos Funciones
    - Plano_1
    - Plano_2
  - Usar una entrada para cambiar de Función
    - Si DI[0] = True, usar Plano_1
    - Si DI[0] = False, usar Plano_2

```
Programa de robot
  MoveJ
    Home
  IF DI[0] = True
    Plano_1_var = Plano_1
  ELSEIF DI[0] = False
    Plano_1_var = Plano_2
  MoveL
    Punto_de_paso_1
    Punto_de_paso_2
    Punto_de_paso_3
    Punto_de_paso_4
    Punto_de_paso_1
```



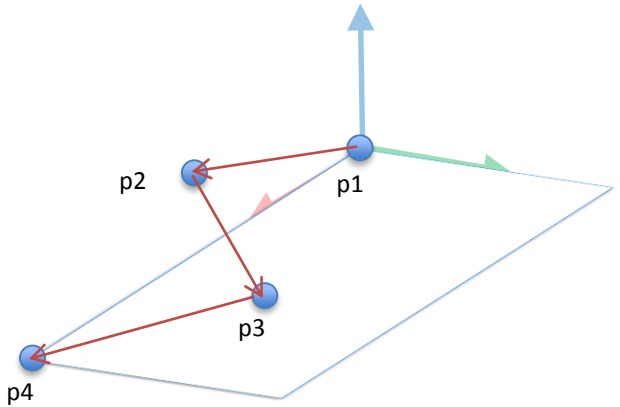
## Ejercicio práctico parte 1

- Crear dos funciones de plano:
  - Plano_1
  - Plano_2
- Marcarlas como variables
- Guardar instalación



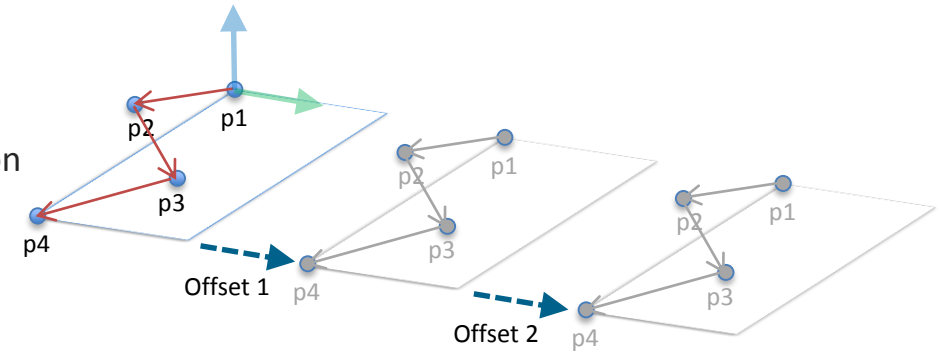
## Ejercicio práctico parte 2

- Crear un programa:
  - Añadir un movimiento MoveL
  - Seleccionar la función Plano_1_var en MoveL
  - Añadir 4 puntos de paso al MoveL
  - Fijar el primer punto de paso en el origen del Plano_1



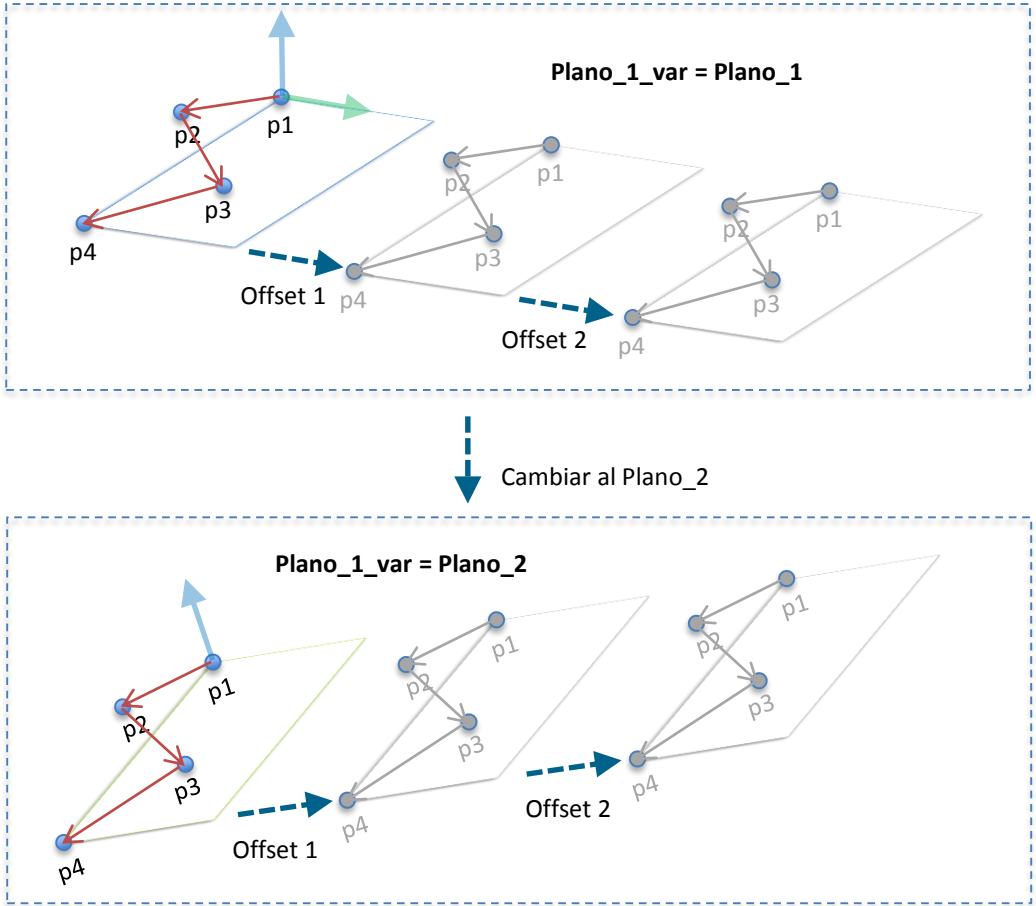
## Ejercicio práctico parte 3

- Cuando el MoveL finalice
  - Desplazar el Plano_1 +10 cm en dirección “y”
  - Repetir MoveL
  - Desplazar otros 10 cm en la misma dirección
  - Repetir MoveL



## Ejercicio práctico parte 4

- Cuando el tercer MoveL finalice
  - Asignar a Plano_1_var el valor de la segunda función (Plano_2)
  - Ejecutar los mismos movimientos usando la función Plano_2

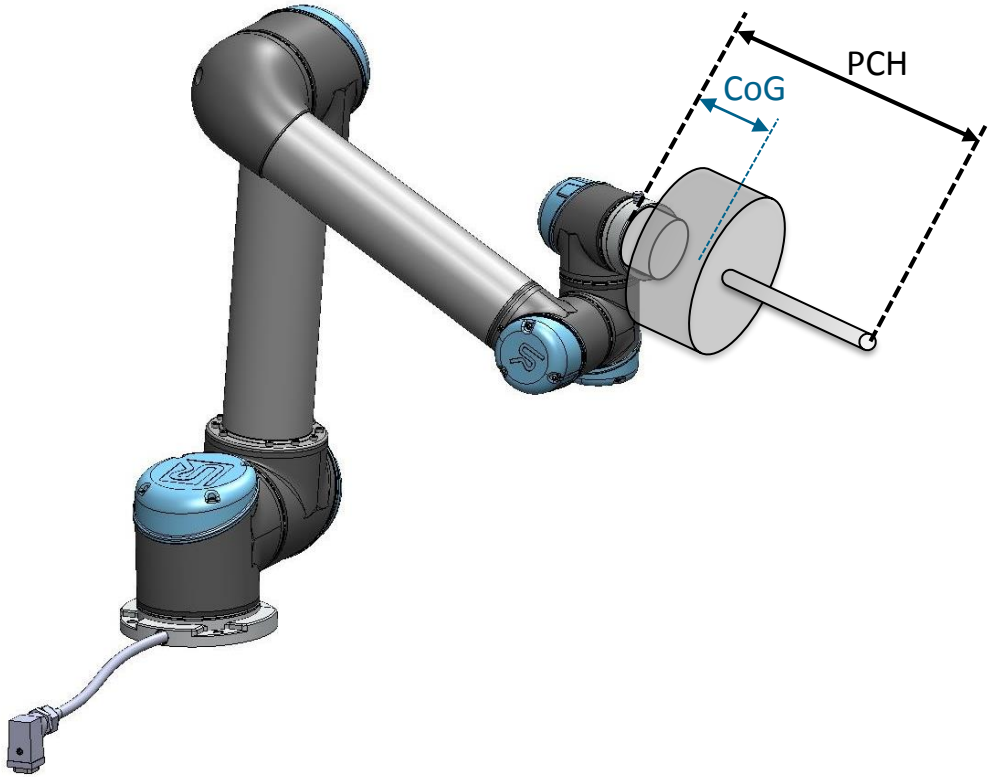


## Centro de Gravedad (CoG)

- Definición:
  - El centro de gravedad es la posición dónde la carga de la herramienta está equilibrada

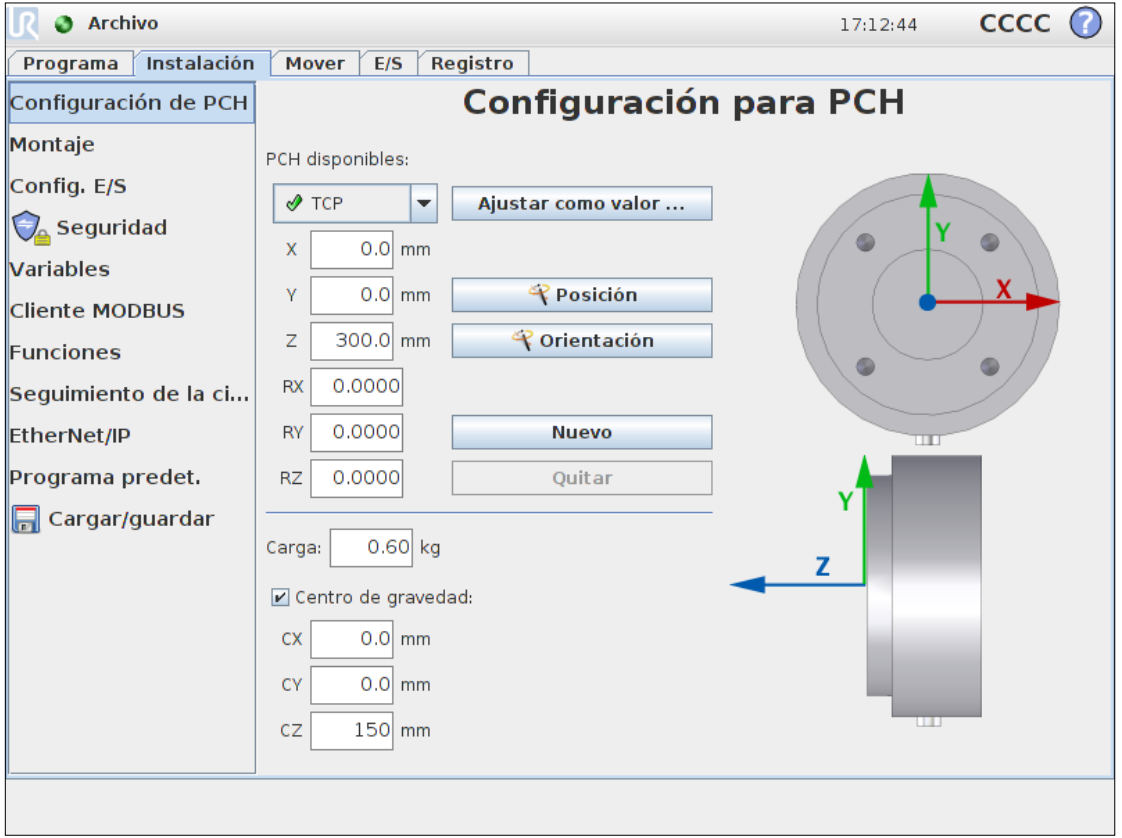
- Código Script
  - set_payload(*m, CoG*)
    - m = masa en kilogramos
    - CoG = Centro de Gravedad (CoGx, CoGy, CoGz) in metros

- NOTA:
  - Si no se define el CoG, se usará por defecto la posición del PCH como CoG



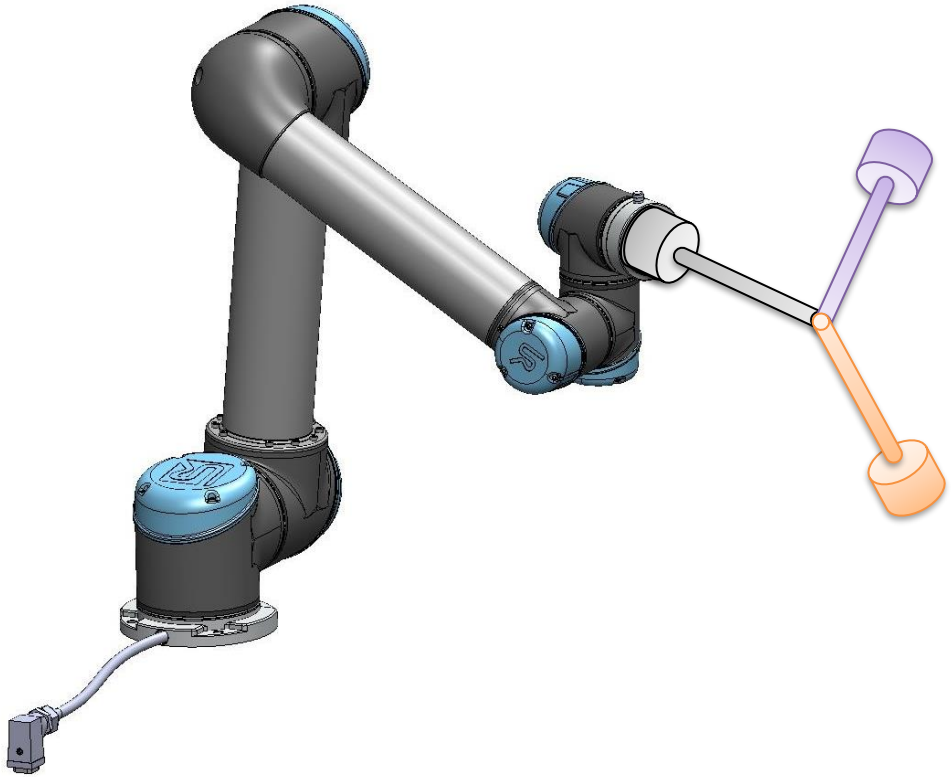
## Centro de Gravedad (CoG)

- Ajustar la Carga en la Configuración de PCH de la Interfaz Gráfica de Usuario
  - Procedimiento fácil para la definición del CoG
- Es necesario utilizar código script si el CoG cambia mucho durante la ejecución



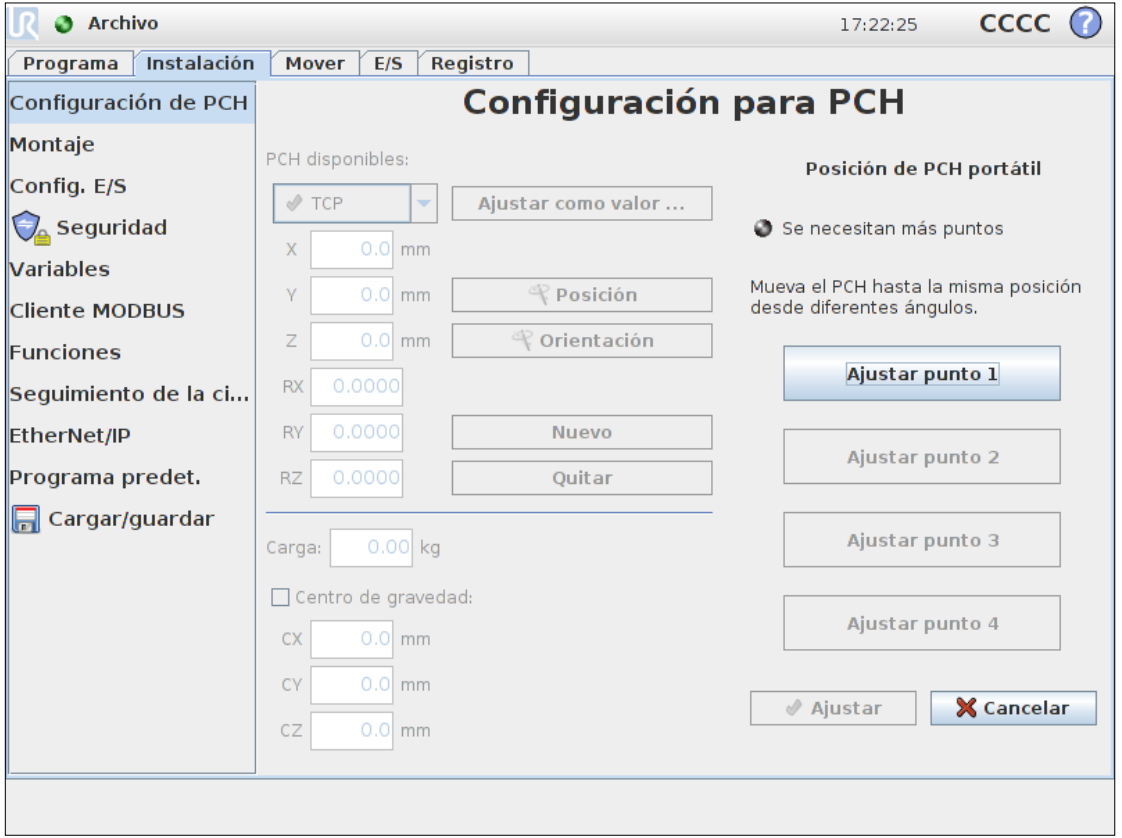
## Cálculo de la posición del PCH

- La posición del PCH se puede determinar moviendo la punta de la herramienta a la misma posición desde 3 ángulos diferentes
- Una vez ajustadas estas posiciones, los valores “x”, “y” y “z” del PCH serán calculados
- Añadir una cuarta posición mejora la precisión del cálculo



## Cálculo de la posición del PCH

- Seleccionar el botón de Posición para calcular la posición del PCH
- Ajustar 3 o 4 puntos
- El indicador LED mostrará el estado del cálculo del PCH y la calidad de cada punto

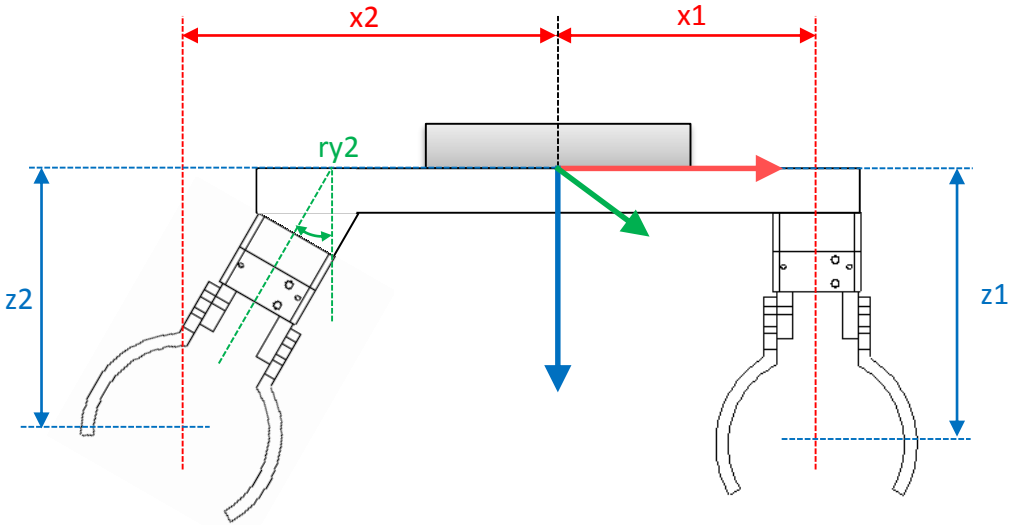


## Múltiples PCHs

- Definición de varios PCHs
  - En la pestaña de Instalación se pueden definir varios PCHs (sólo a partir de v3.1)
  - Como alternativa se pueden definir múltiples PCHs usando código script

- Código script
  - set_tcp(*pose*)
  - pose = p[x,y,z,rx,ry,rz]

- Instalación
  - Marcar función Base como *variable*



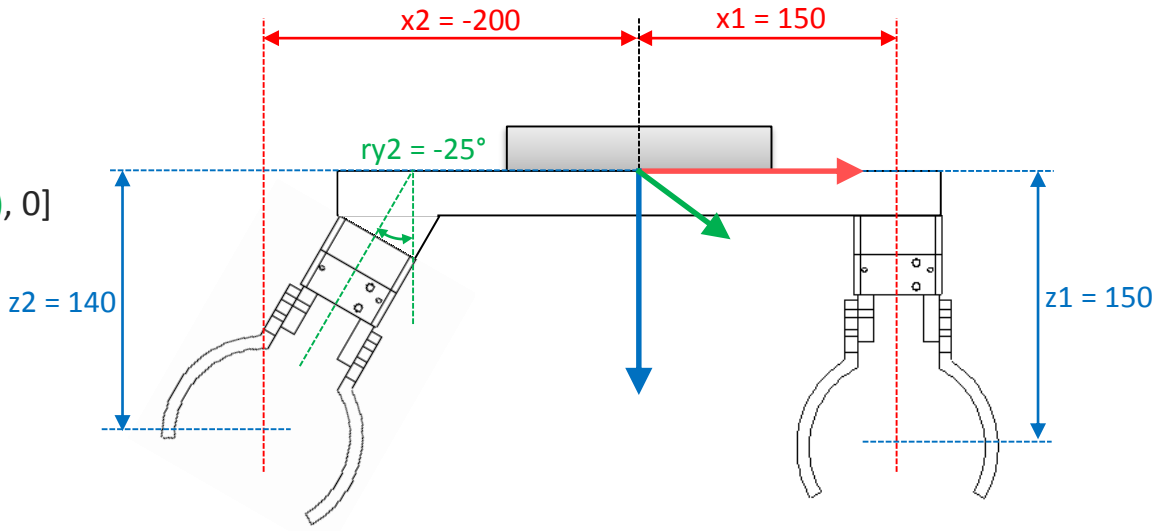

tcp2 = p[x2,0,z2,0,ry2,0]

tcp1 = p[x1,0,z1,0,0,0]

- NOTA:
  - Los cambios de TCP únicamente afectarán los movimientos MoveL y MoveP

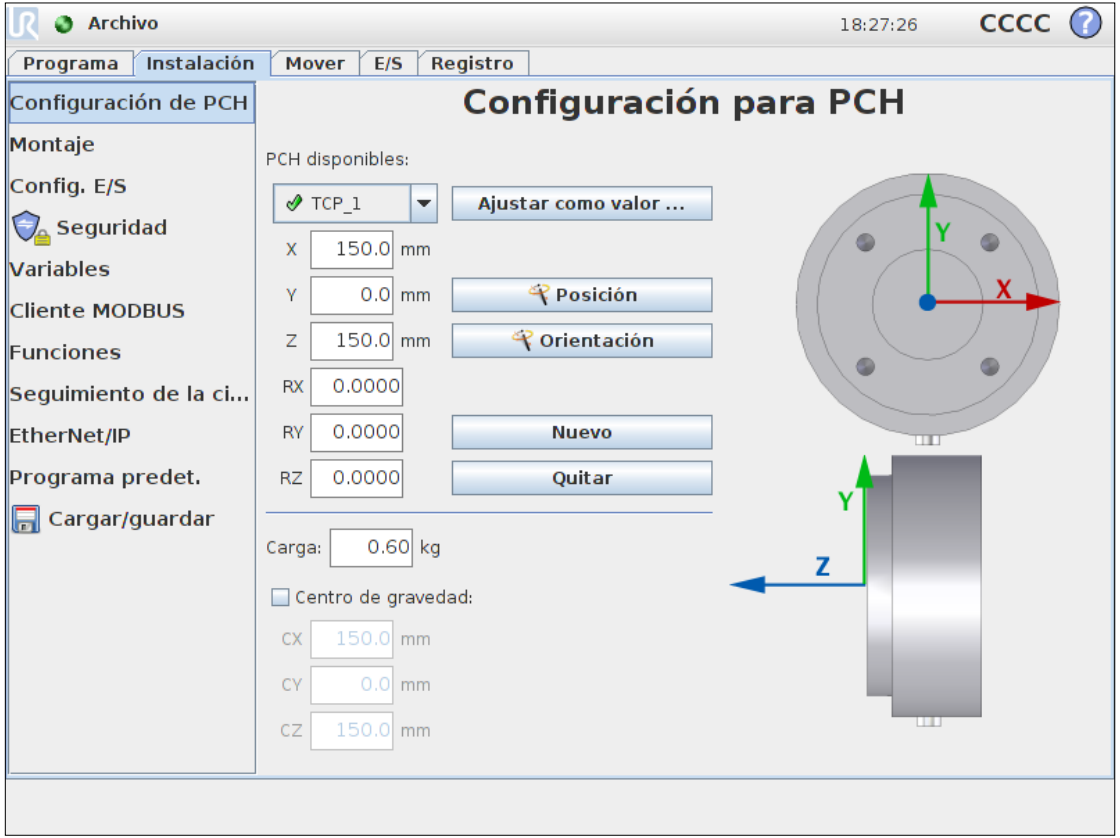
## Múltiples PCHs

- Valores tcp1
  - x1 = 150 mm.
  - z1 = 150 mm.
  - tcp1 = p[0.15, 0, 0.15, 0, 0, 0]
- Valores tcp2
  - x2 = -200 mm.
  - z2 = 140 mm.
  - ry2 = -25 °
  - tcp2 = p[-0.2, 0, 0.14, 0, d2r(-25), 0]



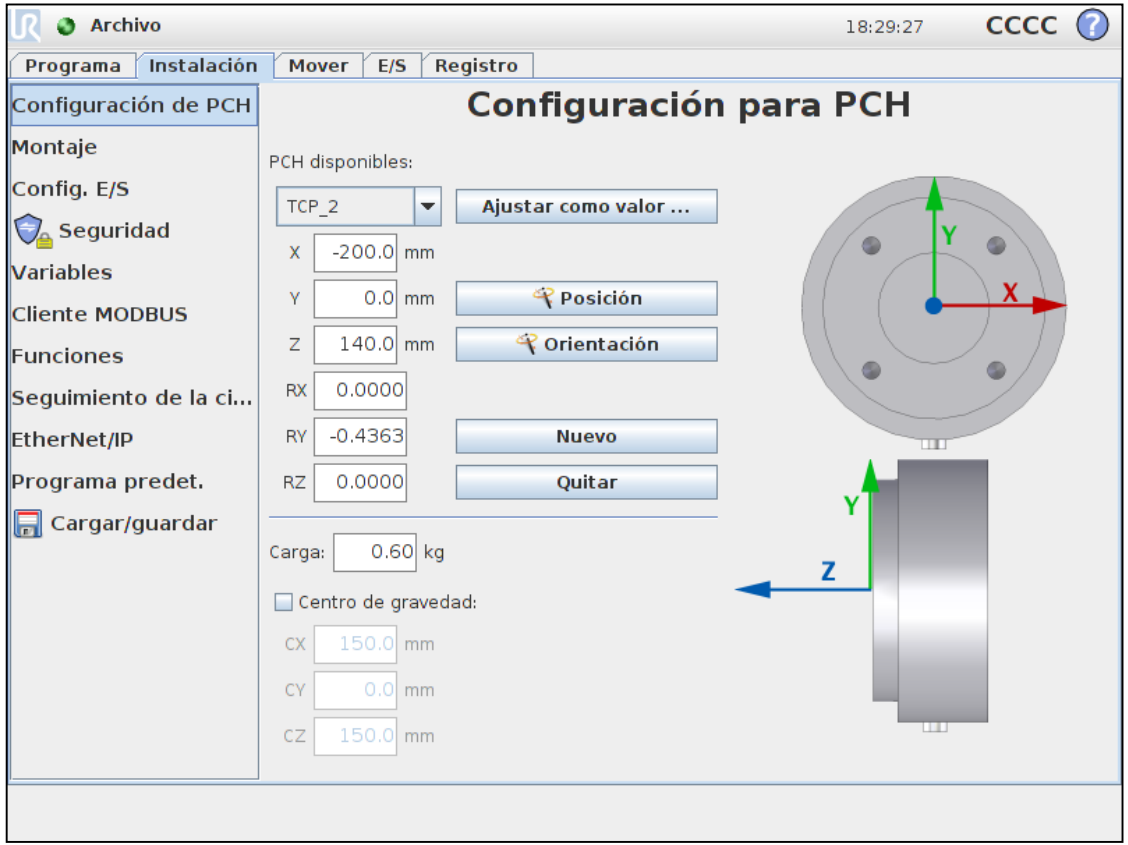
## Ajustar PCH

- Ajustar TCP_1 como valor por defecto
- TCP_1
  - x1 = 150 mm.
  - z1 = 150 mm.



## Ajustar PCH

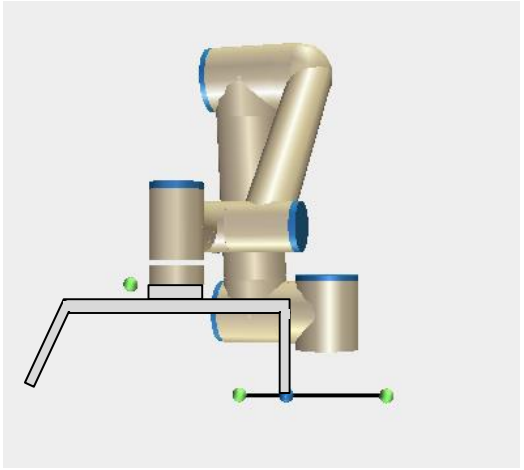
- Seleccionar Nuevo para añadir un segundo PCH
- TCP_2
  - x2 = -200 mm.
  - z2 = 140 mm.
  - ry2 = -25 °
- La rotación debe ser especificada en radianes
  - 25*pi/180 = 0.4363

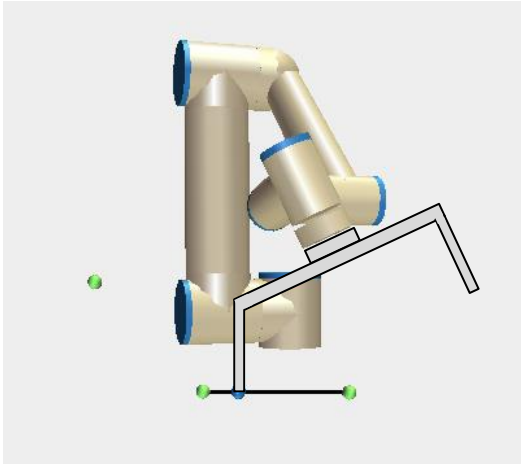


## Uso de múltiples PCHs en programa

- Programa de ejemplo
  - Marcar la función Base como *variable* en pestaña de Instalación
  - Declarar variable “tcp” (entero) para alternar el uso de PCHs
  - Seleccionar Función Base_var en Comando de MoveL

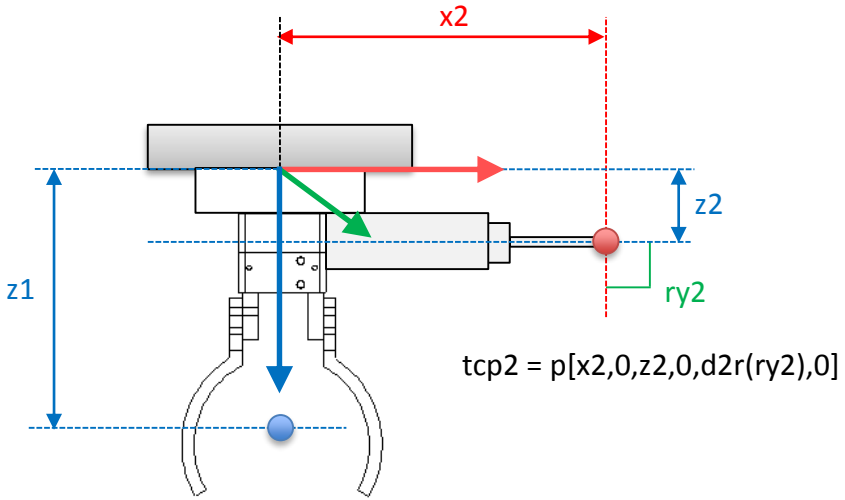
```
BeforeStart
    tcp = 1
    tcp1 = p[0.15,0,0.15,0,0,0]
    tcp2 = p[-0.2,0,0.14,0,d2r(-25),0]
    MoveJ
        Home
Programa de robot
    IF tcp = 1
        set_tcp(tcp1)
        tcp = 2
    ELSEIF tcp = 2
        set_tcp(tcp2)
        tcp = 1
    MoveL
        Punto_de_paso_1
        Punto_de_paso_2
        Punto_de_paso_1
```





## Ejercicio práctico parte 1

- Simular una instalación con dos herramientas, una garra y un atornillador
- Garra (tcp1)
  - Z Offset 20 cm
- Atornillador (tcp2)
  - X offset 20 cm
  - Z offset 5 cm
  - RY offset 90°
- Instalación
  - Recordar marcar Función Base como *variable*
- NOTA:
  - set_tcp() únicamente afecta a MoveL y MoveP



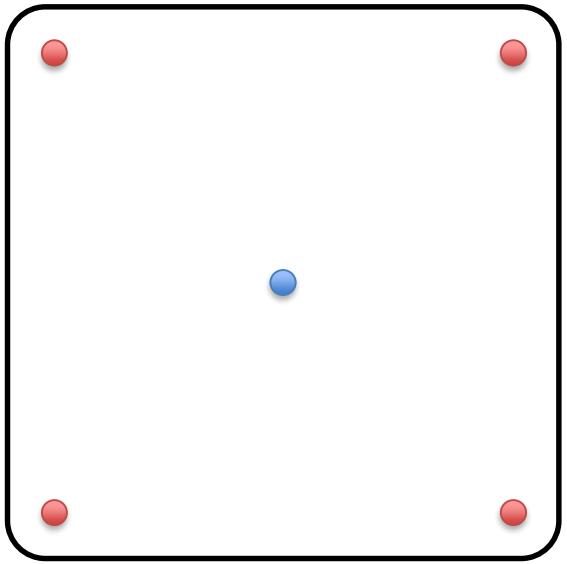
## Ejercicio práctico parte 2

- Simular con la garra coger y dejar una pieza de trabajo cuadrada, luego atornillarla en cuatro posiciones

- Ajustar TCP Garra (tcp1)
  - Elegir y fijar posición para coger pieza
  - Elegir y fijar posición para dejar pieza

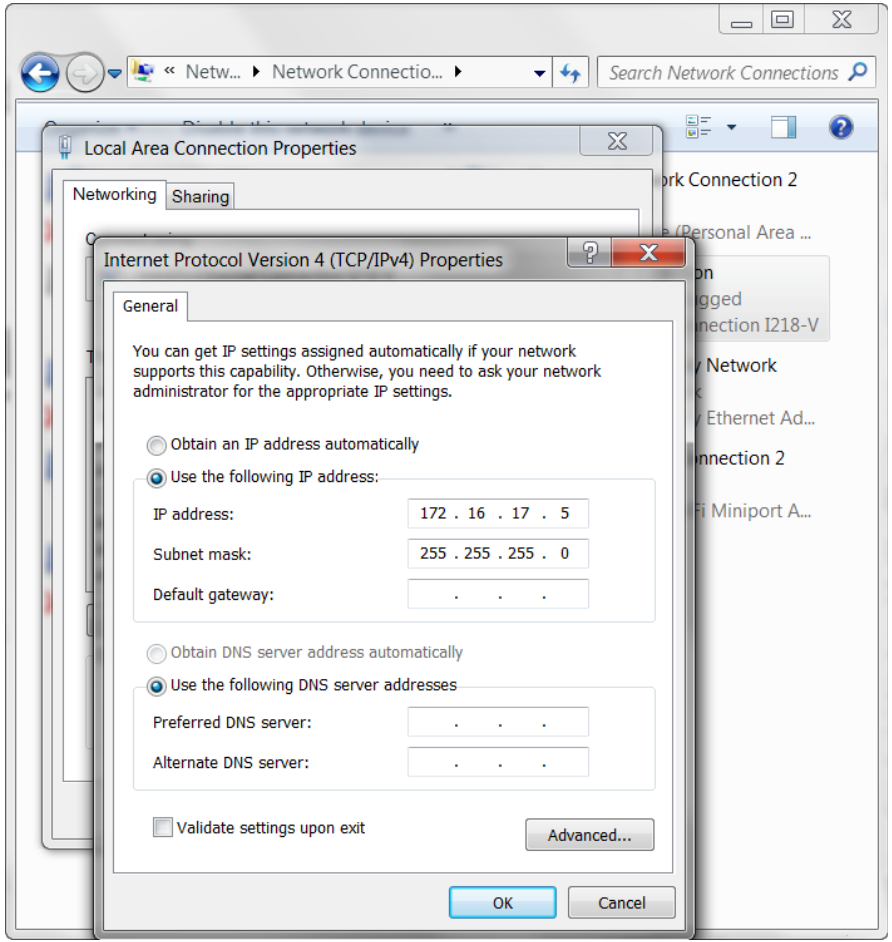
- Ajustar TCP Atornillador (tcp2)
  - Usar pose_trans() para definir 4 puntos de atornillado en:
  - x = ±7.5 cm e y = ±7.5 cm desde el punto de dejar pieza

- Crear un subprograma
  - MoveL a posición relativa Z+5cm en referencia de herramienta
  - MoveL a posición relativa Z-5cm en referencia de herramienta
  - Llamar en cada posición desde el programa principal



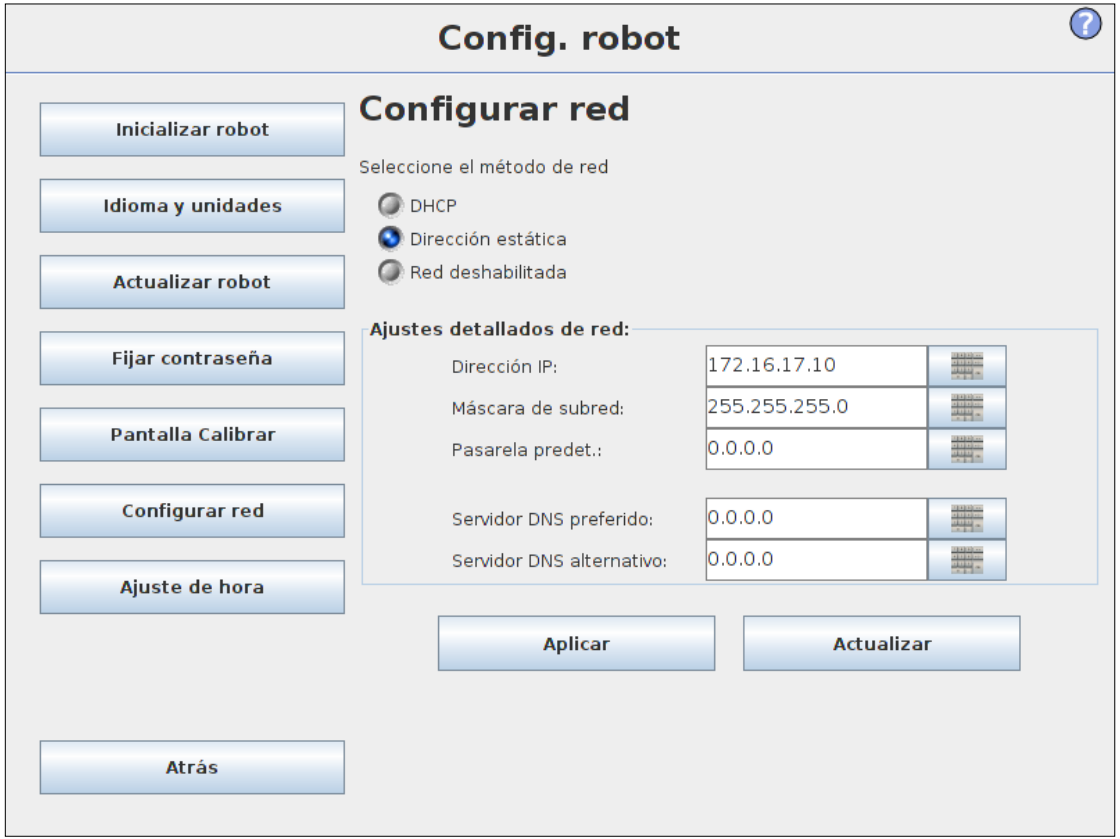
## Configuración de red - PC

- Para comunicar por TCP/IP, debemos configurar la red en nuestro PC y en el robot
- En el PC especificar en propiedades de la conexión:
  - Dirección IP
  - Máscara de subred
- La dirección IP en el robot y en el PC deben ser iguales a excepción del último número, en este caso:
  - 192.16.17.x
- Ambas máscaras de subred deben ser:
  - 255.255.255.0



## Configuración de red - Robot

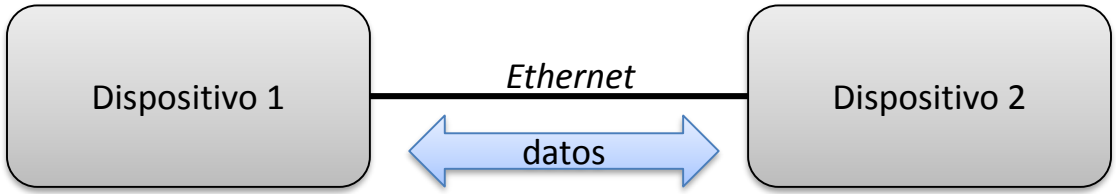
- Ir a Configuración del Robot, seleccionar Configurar red y establecer:
  - Dirección IP
  - Máscara de subred
- La dirección IP debe estar en el mismo rango que la del PC:
  - 192.16.17.x
- La máscara de subred debe ser:
  - 255.255.255.0



## ¿Qué es ModBus TCP?

Recordatorio Formación Principal

- ModBus TCP
  - Protocolo de comunicación basado en Ethernet
- Protocolo de comunicación
  - Un protocolo es un lenguaje común mediante el cual dos dispositivos pueden comunicarse
  - Posibilita la transmisión de datos entre los dispositivos

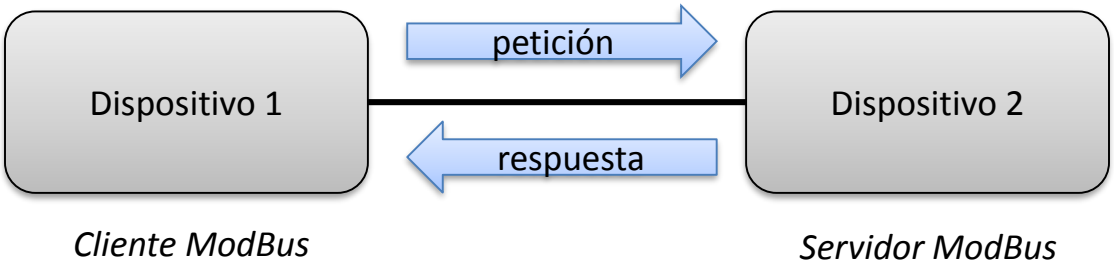



- Estructura Cliente / Servidor

## Cliente / Servidor

Recordatorio Formación Principal

- Servidor (Esclavo)
  - Uno de los dispositivos actúa como Servidor
  - A la escucha de peticiones desde el Cliente
- Cliente (Maestro)
  - Otro dispositivo en la red actúa como Cliente
  - Envía peticiones al Servidor




- Cada dispositivo debe tener configurada una dirección-IP única

## Tipos de datos

Recordatorio Formación Principal

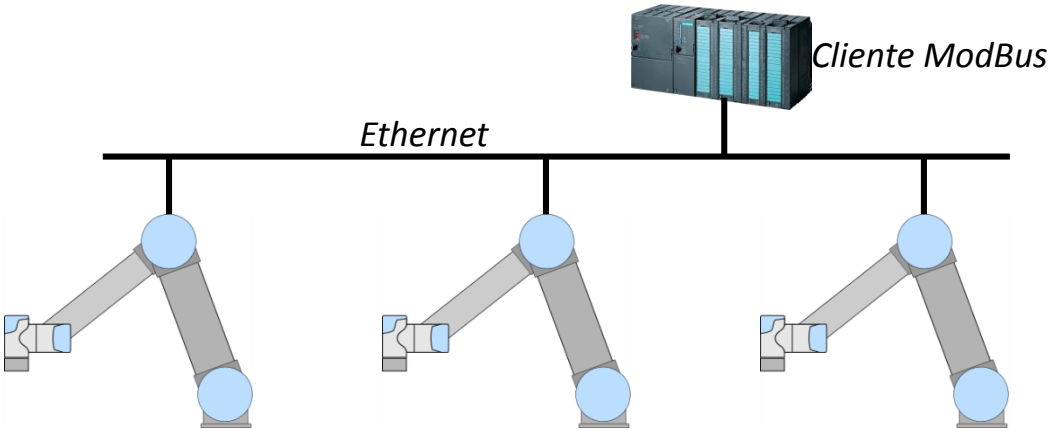
- Tipos de datos disponibles en ModBus TCP

| Tipo de dato | Valor | Rango de direcciones |
|---|---|---|
| Entradas digitales | ON/OFF | *Consultar la documentación del fabricante del dispositivo Servidor* |
| Salidas digitales | ON/OFF | |
| Entradas de registro | 16 bits | |
| Salidas de registro | 16 bits | |

- Rango de direcciones
  - Cada señal digital y cada registro poseen una única dirección
  - La dirección se especifica *siempre* en la documentación suministrada por el fabricante

## Servidor ModBus UR

- Formación Principal
  - Se trató el uso del robot como Cliente en la red ModBus y cómo conectar con un Servidor
- Formación Avanzada
  - Se tratará cómo usar el controlador del robot como Servidor y aceptar conexiones de Clientes

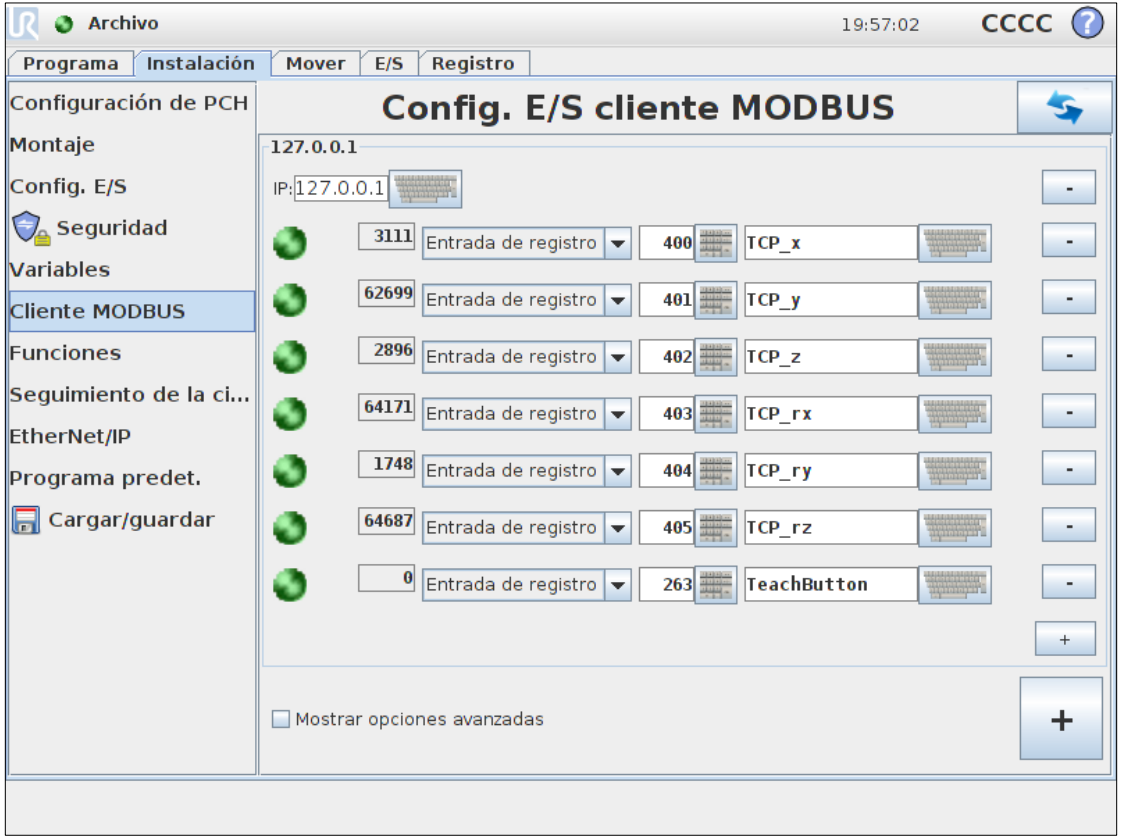


## Características del servidor ModBus

- El servidor ModBus se encuentra siempre activo en el Controlador UR
  - Dirección IP: Establecer en el menú de configuración del robot
  - No. puerto: 502 (*no necesita configuración*)
- Características:
  - El Servidor ModBus transmite información sobre el estado del robot, incluyendo:
    - Juntas: Ángulo de giro/Velocidad/Corriente/Temperatura
    - PCH: Posición/Velocidad/Offset
    - Estado del Programa – Parado por emergencia/En movimiento libre/etc.
  - E/S pueden ser leídas/escritas remotamente
    - Transmisión del estado de E/S
    - Las salidas pueden ser controladas por un dispositivo remoto
  - Registros de propósito general
    - 128 registros disponibles
- Listado completo de direcciones de registros disponibles en página de soporte

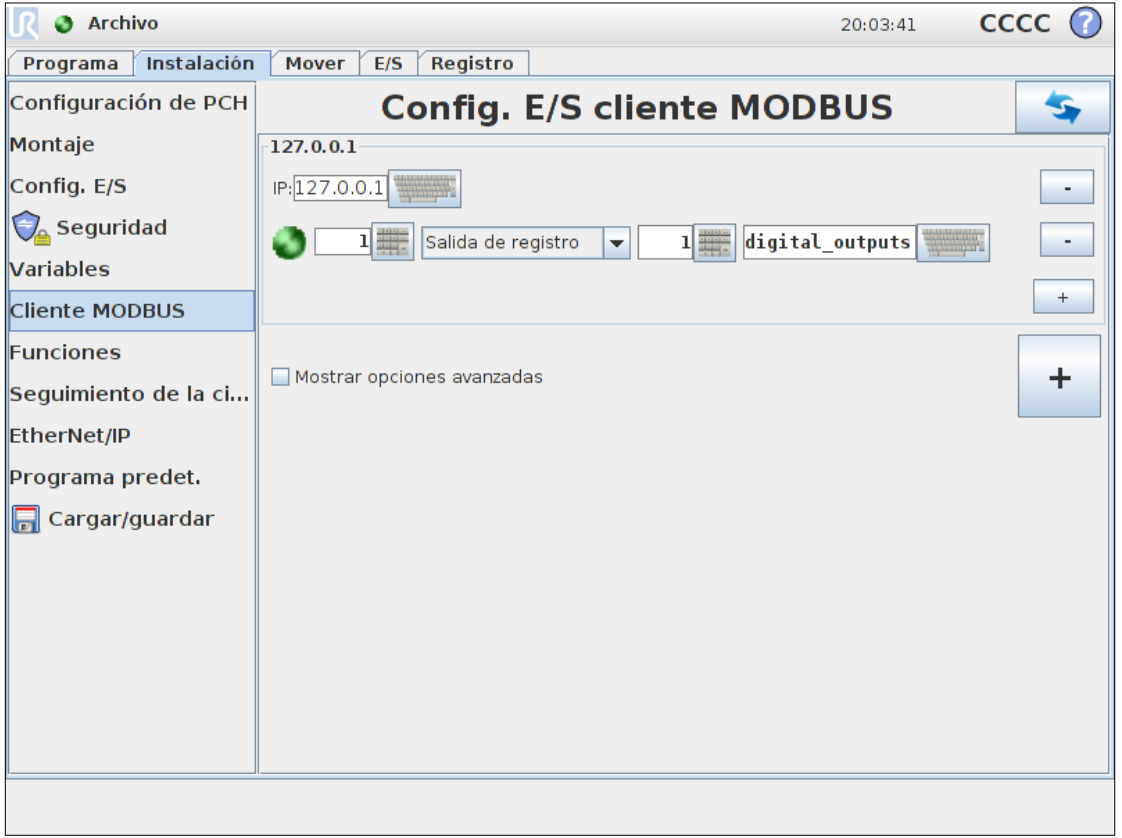
## Acceso al propio servidor del robot

- Acceso al servidor desde el propio robot
  - Dirección IP: 127.0.0.1
- Leer la posición del PCH
  - Direcciones de los registros:
    - 400 TCP-x en décimas de mm
    - 401 TCP-y en décimas de mm
    - 402 TCP-z en décimas de mm
    - 403 TCP-rx en mrad
    - 404 TCP-ry en mrad
    - 405 TCP-rz en mrad
  - = en Función Base
    - 263 isTeachButtonPressed



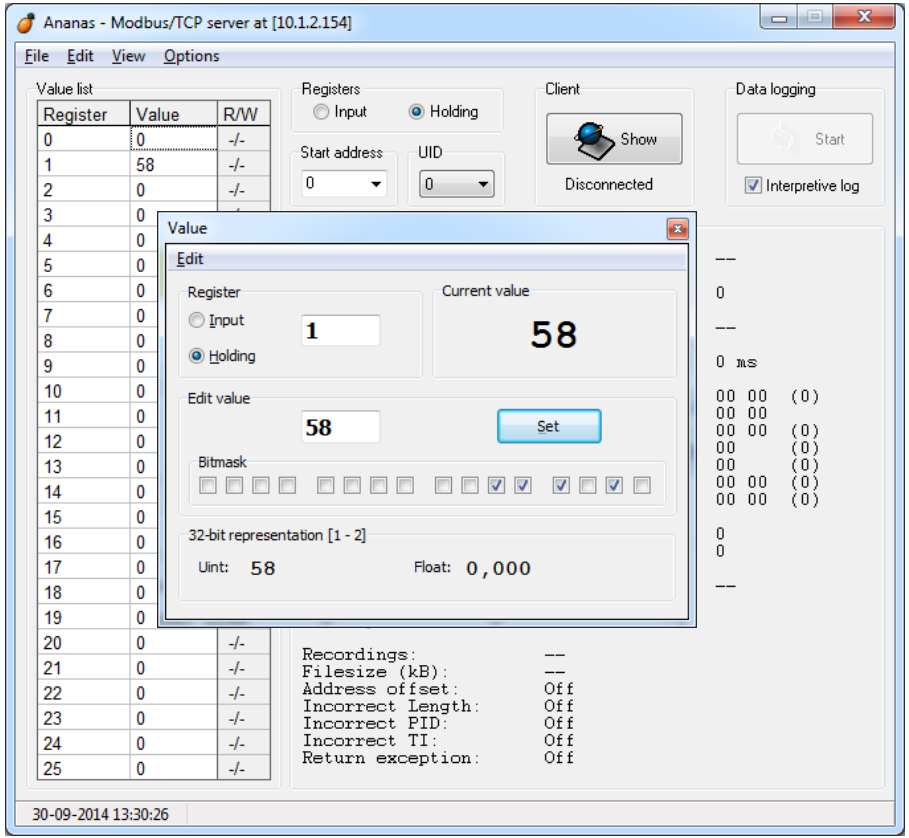
## Acceso a los registros de E/S

- Configurar registro con salidas digitales
  - Dirección del registro = 1
- Función
  - Permite la manipulación de múltiples canales en una única operación, ahorrando tiempo de procesado
- Formato
  - Registro de enteros de 16 bits
  - [BBBBBBBBTTxxxxxx]
    *B=box , T=tool, x=undef*



## Aplicaciones ModBus

- Aplicaciones potenciales para el uso del Servidor ModBus:
  - Controlar las E/S del robot desde un dispositivo remoto
- Cómo probar la comunicación
  - Instalar Ananas
    - Programa *freeware* para probar la comunicación mediante ModBus
  - Configurar Ananas como Cliente ModBus
  - Cambiar el estado de las salidas digitales en el controlador marcando/desmarcando bits o editando el valor entero



## Ejercicio práctico parte 1

- Escriba un programa que:
  - Lea el registro de salidas digitales
  - Convierta a una lista de binarios
  - Conmute la salida del canal 3
  - Convierta de nuevo la lista a un entero
  - Escriba el nuevo valor en el registro

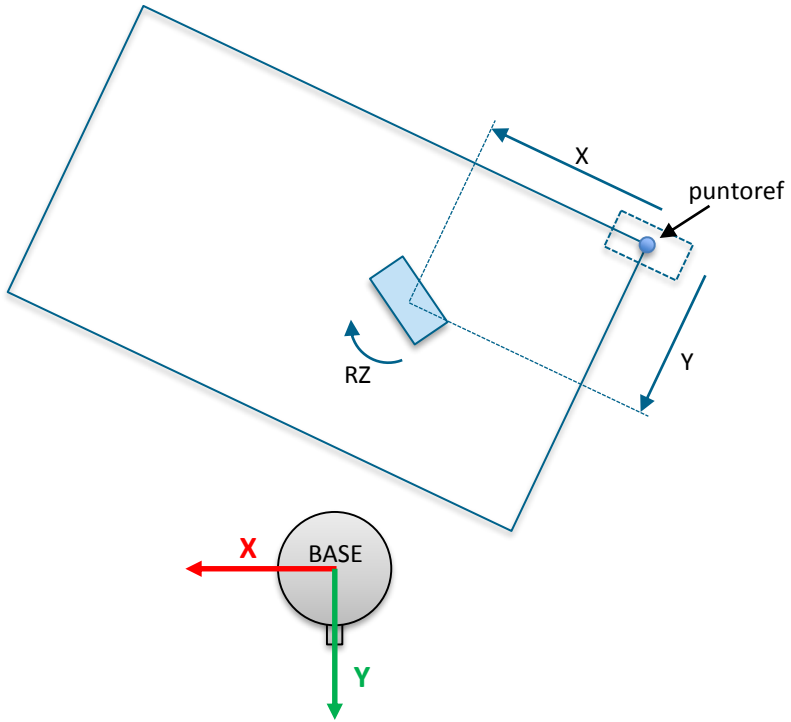
- Ejecute el programa y verifique el cambio en la salida

## Ejercicio práctico parte 2

- Escriba un programa que simule la recepción de coordenadas desde una cámara de visión mediante el Servidor ModBus y que sea capaz de moverse a la posición recibida:
  - Recibir las coordenadas de coger variables desde los registros de propósito general del Servidor ModBus
  - Mover a la posición usando pose_trans(), como en el ejemplo previo en la sección Variables

- Direcciones de los registros:
  - 128: nuevo dato disponible (0=no, 255=yes)
  - 129: x
  - 130: y
  - 131: rz

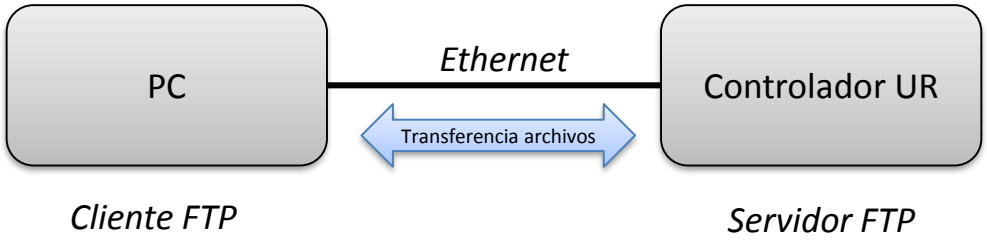
*puntoref como referencia*



## ¿Qué es el servidor FTP?

- Servidor FTP
  - Protocolo Cliente/Servidor para transmisión de archivos
  - Comunicación basada en Ethernet

- Propósito
  - Facilita la transferencia de archivos de programa
  - Facilita la realización de copias de seguridad



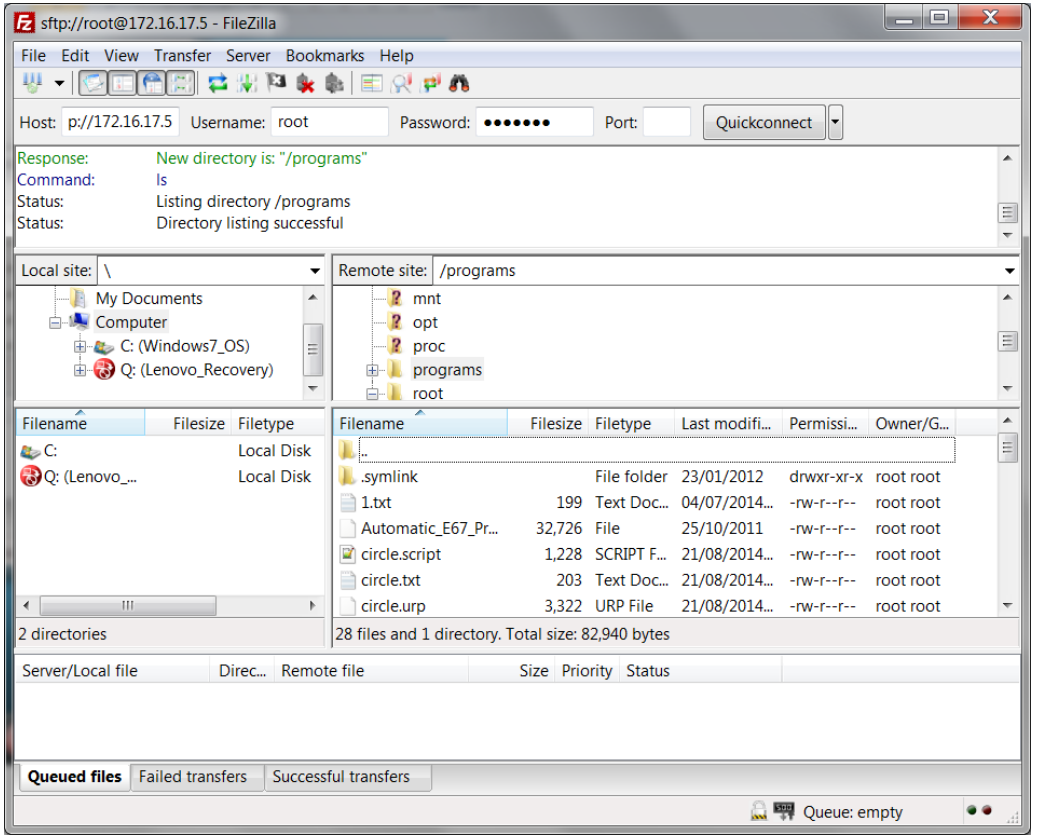

- Servidor en el Controlador UR
  - El servidor se encuentra siempre activo en el controlador
  - No es necesario ninguna configuración en el controlador

## Cliente FTP

- FileZilla
  - Cliente FTP *freeware*
  - filezilla-project.org
  - Conexión vía ssh o sftp

- Cómo conectar
  - Configuración FileZilla
    - Anfitrión: Dirección IP en el robot
    - Usuario: root
    - Contraseña: easybot
    - Puerto: 22

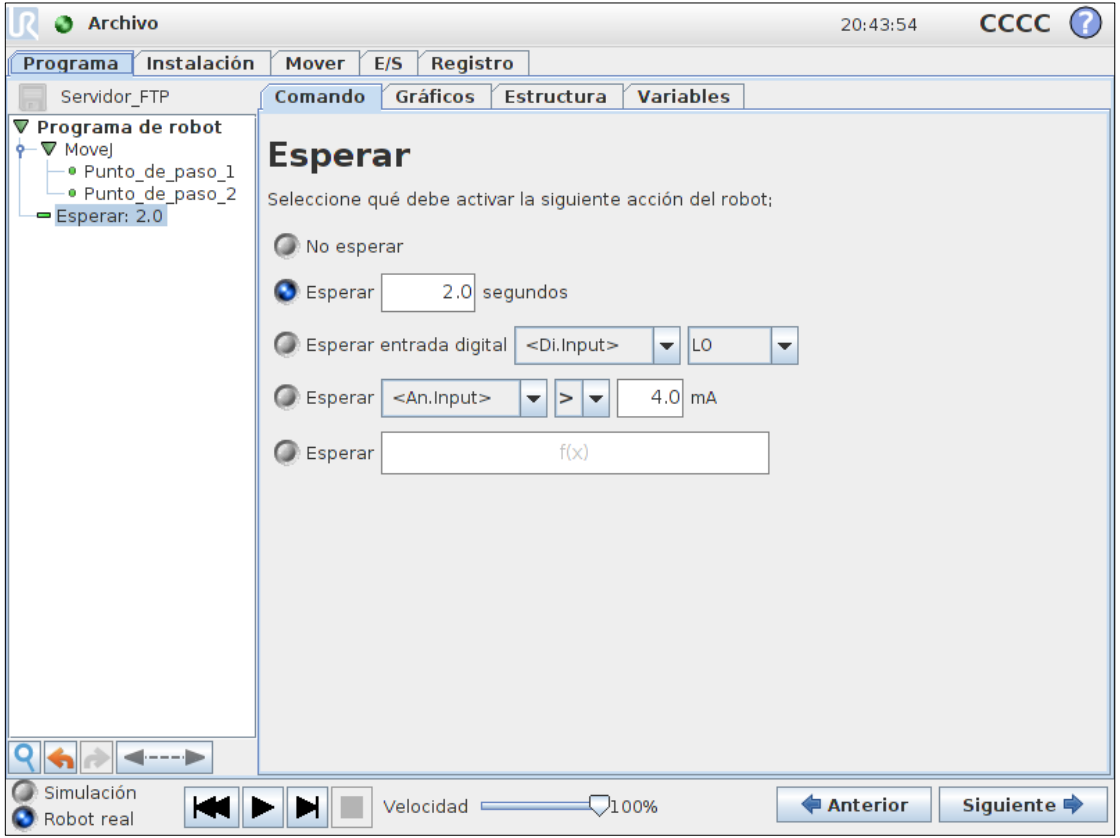
- Cómo transferir
  - Arrastrar y soltar ficheros entre PC y directorio /programs del robot




- ATENCIÓN: Los archivos del sistema del robot no están protegidos contra escritura, un uso incorrecto puede provocar un funcionamiento del robot

## Ejercicio práctico

- Realizar una copia de seguridad de todos los programas contenidos en el directorio */programs*
- Escribir un simple programa usando URSim en el PC
- Transferir el programa al directorio */programs*
- Ejecutar el programa en el robot



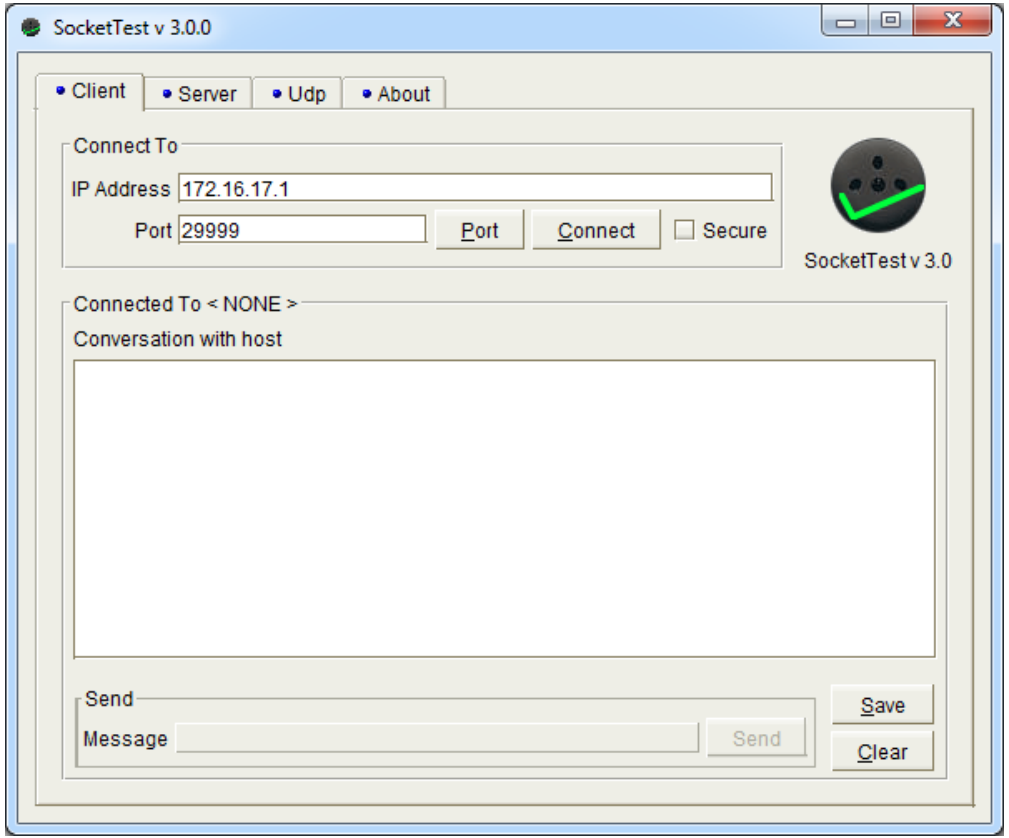
## ¿Qué es el Servidor Dashboard?

- El Servidor Dashboard (Panel de mando)
  - Permitir el control remoto del robot por TCP/IP socket a través del puerto 29999
  - Controlar la operación del robot, cargando/iniciando programas
  - Obtiener información sobre el estado del robot (programa en ejecución/cargado)
  - Mostrar/Cerrar mensajes emergentes
  - Establecer diferentes niveles de acceso (deshabilita diferentes partes del *GUI*)
  - La lista completa de comandos admitidos por el Servidor se puede encontrar en el sitio de soporte

## SocketTest

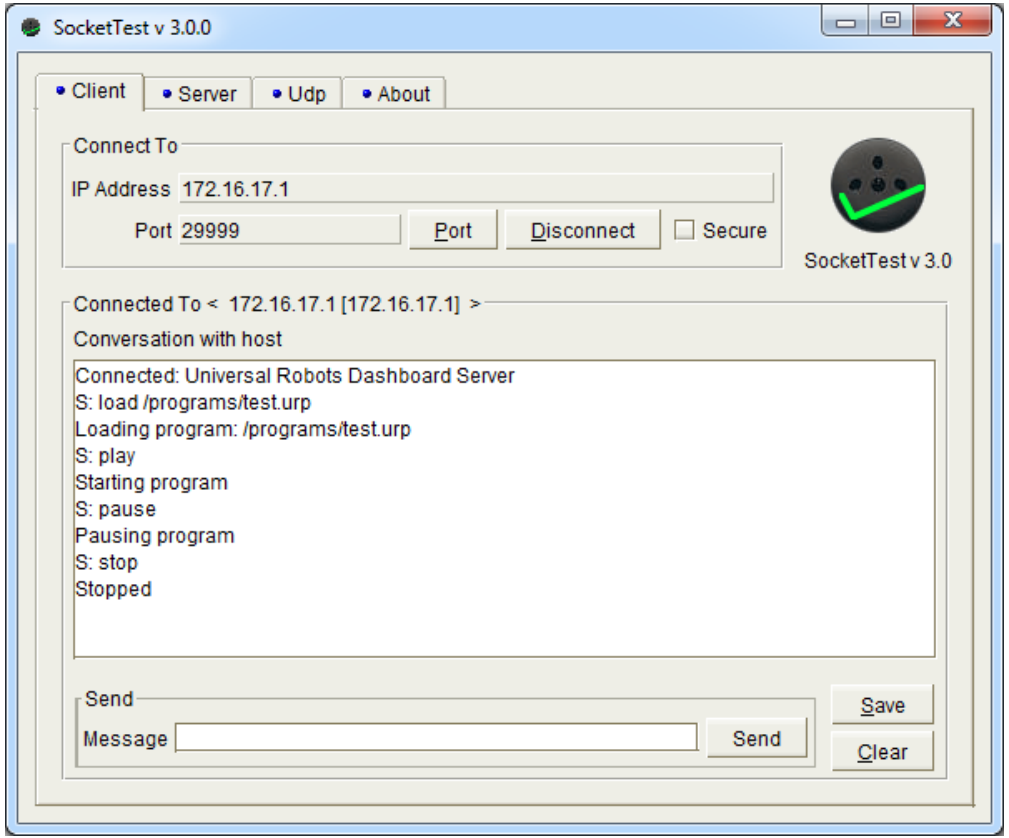
- SocketTest
  - Programa *freeware* para probar la comunicación por TCP/IP sockets
  - sockettest.sourceforget.net

- Cómo conectar
  - Configurar la dirección IP tanto en el PC como en el robot
  - Abrir SocketTest
    - Seleccionar la pestaña "*Client*"
    - Introducir la dirección IP del robot
    - Introducir no. puerto del Servidor Dashboard (29999)
    - Pulsar botón "*Connect*"



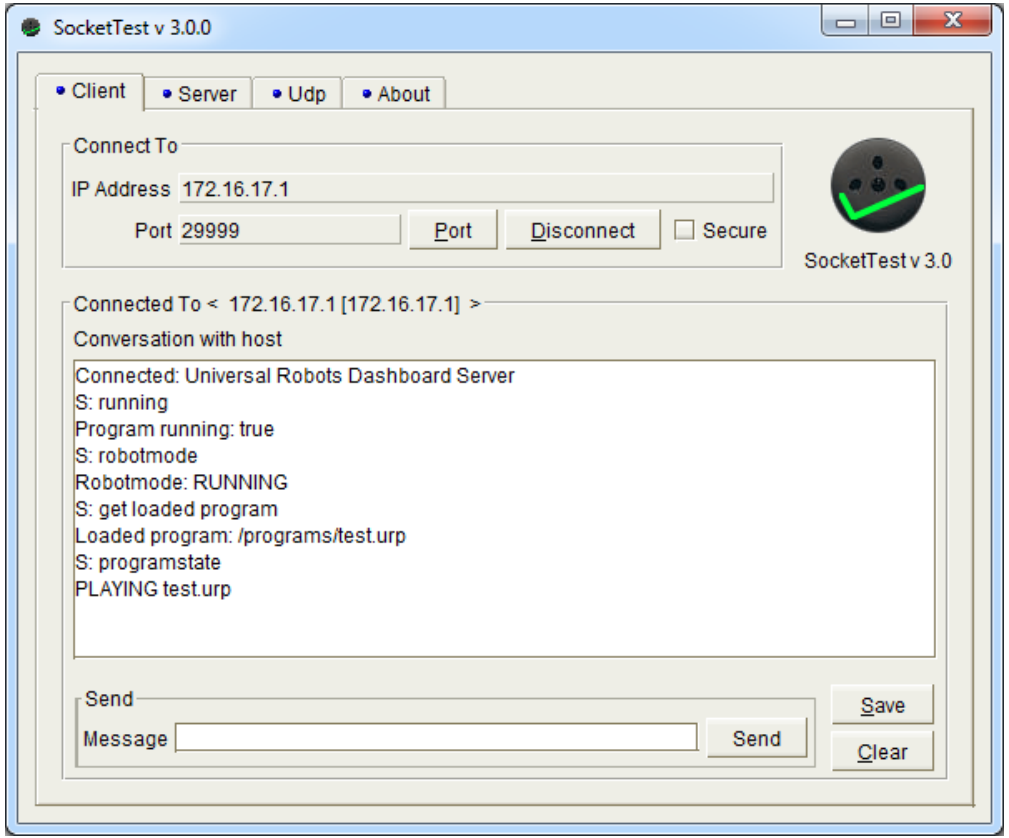
## Operación remota

- Controlar remotamente la operación del robot
  - *load <program.urp>*
  - *play* (arranca el programa cargado)
  - *stop* (detiene el programa en ejecución)
  - *pause* (pausa el programa en ejecución)
  - *quit* (cierra la conexión)
  - *shutdown* (desconecta el robot y el controlador)



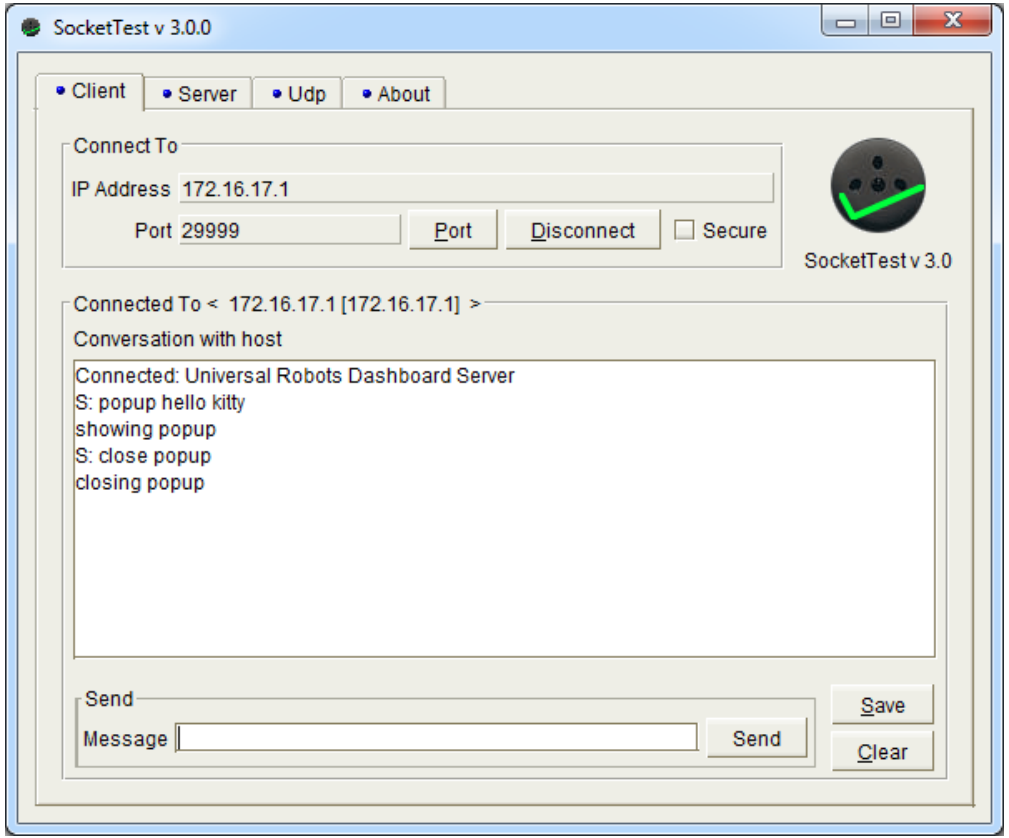
## Consultas

- Solicitar estado del robot
  - *running* (consulta estado de ejecución)
  - *robotmode* (consulta modo robot)
  - *Get loaded program* (devuelve que programa está cargado)
  - *isProgramSaved* (devuelve *true* cuando el programa está guardado o *false* en caso contrario)
  - *programState* (devuelve *PLAYING* si el programa está en ejecución o *STOPPED* si no hay programa cargado)



## Mensajes de aviso emergentes

- Abrir y cerrar mensajes emergentes en el *GUI*
  - *popup* <texto> (abre una ventana con el mensaje de aviso “texto”)
  - *close popup* (cierra la ventana del mensaje)

## Privilegios de usuario

- Comprobar las opciones disponibles en la pantalla de bienvenida
  - *setUserRole <role>*

| **Usuario (<*role*>)** | **Descripción** |
|---|---|
| *programmer* | El botón “Config. Robot” deshabilitado, “Modo Experto” habilitado (introduciendo la contraseña correcta) |
| *operator* | Únicamente disponibles los botones “Ejecutar programa” y “Apagar robot”, “Modo Experto” deshabilitado |
| *none* | Habilitados todos los botones, “Modo Experto” habilitado (introduciendo la contraseña correcta) |
| *locked* | Todos los botones y “Modo Experto” deshabilitados |

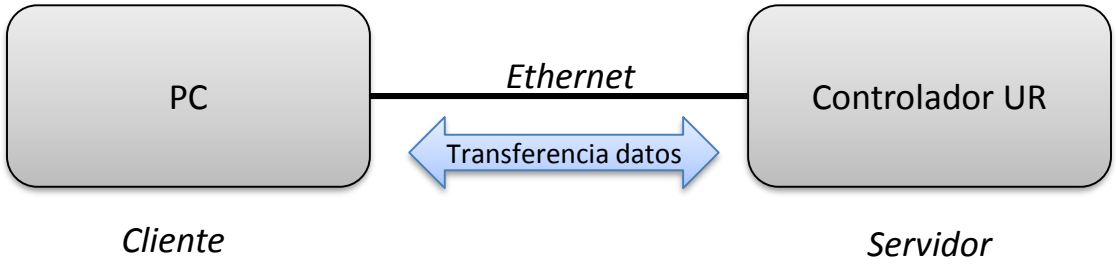
## Ejercicio práctico

- Mostrar la pantalla de bienvenida en el robot
- Conectar con el Servidor Dashboard
- Cargar el programa previamente transferido por FTP
- Ejecutar el programa

## ¿Qué son los Interfaces Cliente?

- ¿Qué son los Interfaces Cliente?
  - Los servidores se encuentran siempre activos en el Controlador UR

- Funcionalidad
  - Los servidores envían continuamente flujo de datos sobre el estado del robot
  - Los clientes pueden enviar instrucciones URScript a los servidores
    - (Los servidores están siempre a la espera de recibir instrucciones)



## Interfaces disponibles

- Interfaces disponibles

| | Primaria | Secundaria | Tiempo Real |
|---|---|---|---|
| Transmite | Estado del robot y mensajes adicionales | Mensajes estado del robot | Mensajes estado del robot |
| Recibe | Instrucciones URScript | Instrucciones URScript | Instrucciones URScript |
| No. Puerto | 30001 | 30002 | 30003 |

- La interface de cliente de Tiempo Real es más rápida que la Primaria y Secundaria

## Flujo de datos transmitido

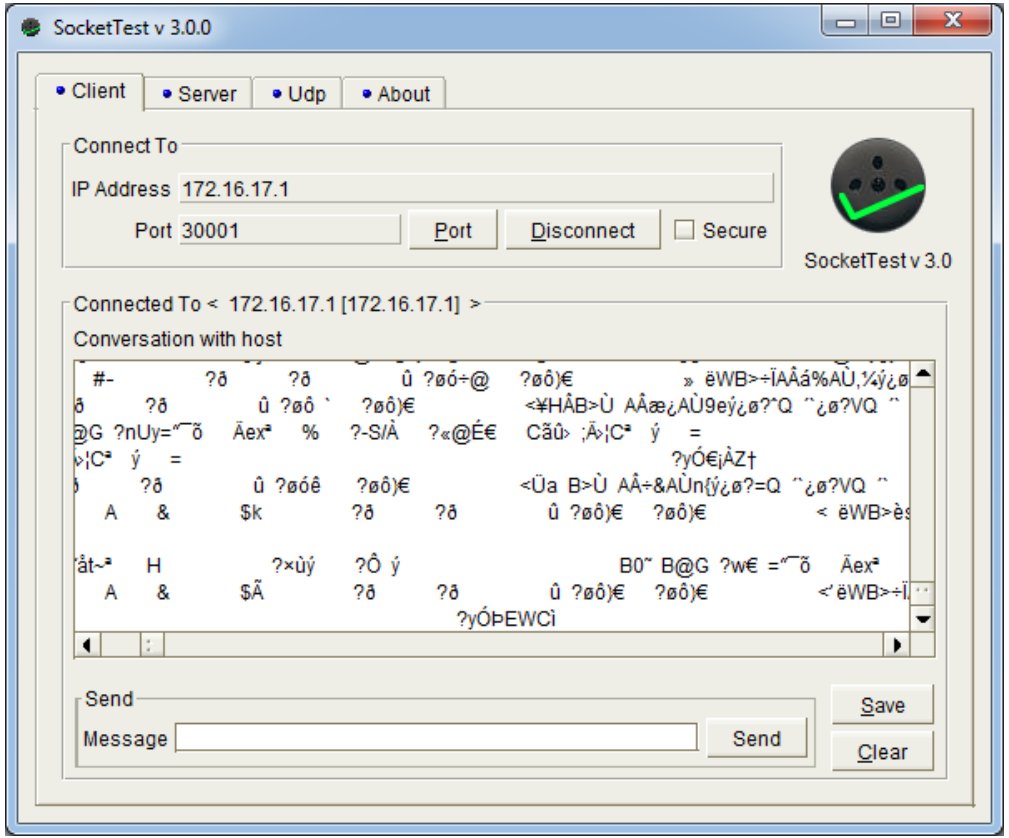
- Flujo de datos
  - El flujo de datos transmitido contiene una gran cantidad de datos
  - Es usado principalmente por el sistema para la comunicación entre la *GUI* y el controlador
  - Es necesario escribir un programa para extraer y procesar la información necesaria de los *bytes* recibidos
  - Se puede obtener más información en la página de soporte técnico
  - Supone mucho trabajo el recibir datos, así que no resulta siempre la mejor opción

## Envío de instrucciones

- Envío de instrucciones a través de los Interfaces Cliente
  - Las Interfaces Cliente resultan útiles para enviar instrucciones URScript desde equipos externos
  - Las instrucciones individuales se ejecutan en orden al recibirse
  - Se pueden enviar funciones para ser ejecutadas posteriormente
  - Es posible escribir un programa en URScript y ejecutarlo sin usar la *GUI*

## Envío de instrucciones

- Abrir SocketTest
  - Seleccionar pestaña “*Client*”
  - Fijar No. “*Port*” 30001
  - “*Connect*”




- Enviar instrucciones URScript, por ejemplo:
  - popup()
  - set_digital_out()
  - movel()
  - speedj()

- Consultar manual de URScript para una descripción detallada de la sintaxis correcta de estas instrucciones

## Ejercicio práctico

- Usando SocketTest enviar instrucciones al robot (o URSim) por Interfaces Cliente para hacer lo siguiente (consultar el manual de URScript por la sintaxis correcta)

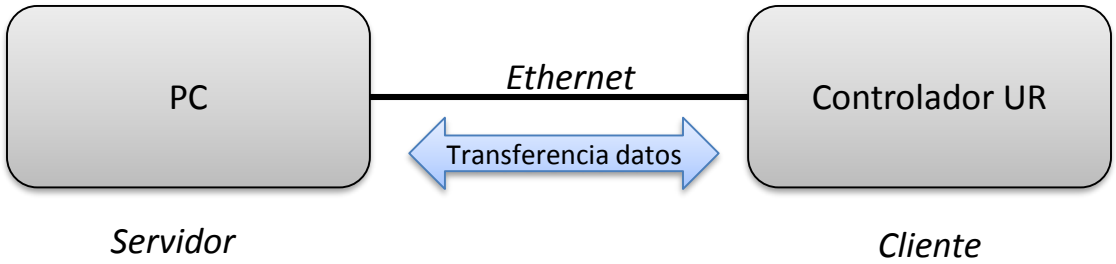
- Mover el PCH del robot en coordenadas BASE con los siguientes parámetros:
  - Velocidad Z = 0.3 m/s
  - Velocidad X, Y, RX, RY, RZ = 0 m/s
  - Duración = 1 segundo
  - Aceleración = 0.5 m/s²

- Mover el PCH del robot en coordenadas Herramienta con los estos parámetros:
  - Z = Z + 0.1 m (añadir 0.1 m al valor actual de “z” del PCH)
  - Mover linealmente en coordenadas de la herramienta
  - Velocidad = 0.2 m/s
  - Aceleración = 0.5 m/s²
  - Ayuda: Será necesario usar get_actual_tcp_pose() y pose_trans() dentro de una instrucción movel

## ¿Qué son los Sockets?

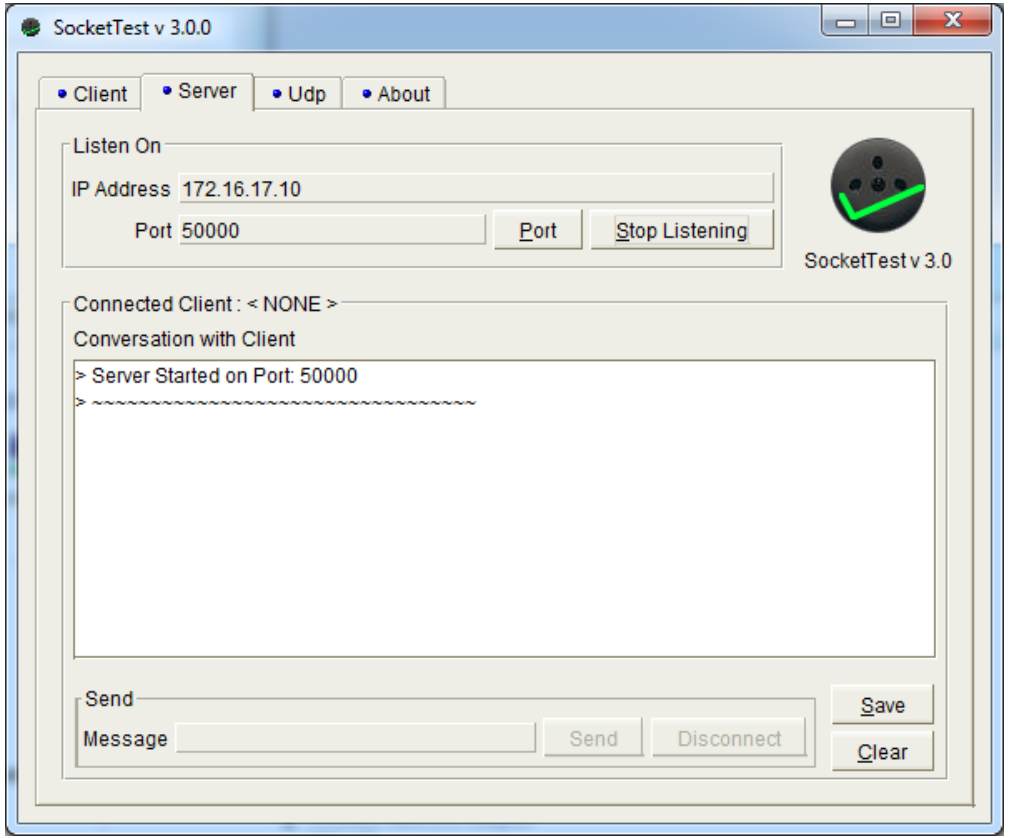
- Comunicación Socket
  - La comunicación por sockets es muy útil para intercambiar datos entre el robot y otros dispositivos
  - Robot = Cliente, otros dispositivos = Servidores

- Funcionalidad
  - El Servidor siempre está a la espera de una petición de conexión de un Cliente en el socket
  - Funciones script para abrir/cerrar sockets
  - Funciones script para enviar/recibir diferentes formatos de datos



## Iniciando el Servidor

- Abrir SocketTest
  - Seleccionar opción “*Server*”
  - Definir la dirección IP
  - Definir no. de puerto
  - “*Start Listening*”




- *La dirección IP debe ser la misma que la del PC*

## Abrir un Socket

- socket_open(*address*, *port*, *socket_name*)
  - dirección
  - puerto
  - nombre del socket
  - *Si un programa abre un único socket, el argumento socket_name no es necesario*
- Valor devuelto
  - TRUE si el socket queda abierto
  - FALSE si falla

**socket_open**(*address, port, socket_name='socket_0'*)

Open ethernet communication

Attempts to open a socket connection, times out after 2 seconds.

**Parameters**

| | |
|---|---|
| `address:` | Server address (string) |
| `port:` | Port number (int) |
| `socket_name:` | Name of socket (string) |

**Return Value**

False if failed, True if connection succesfully established

Programa de robot

```
socket_open("172.16.17.10", 50000)
```

## Enviar una cadena de caracteres (*string*)

- socket_send_string(*str*, *socket_name*)
  - cadena de caracteres
  - nombre del socket

- Valor devuelto
  - ninguno

Programa de robot
socket_send_string(”test”)

**socket_send_string**(*str*, *socket_name*=`'socket_0'`)

Sends a string to the server

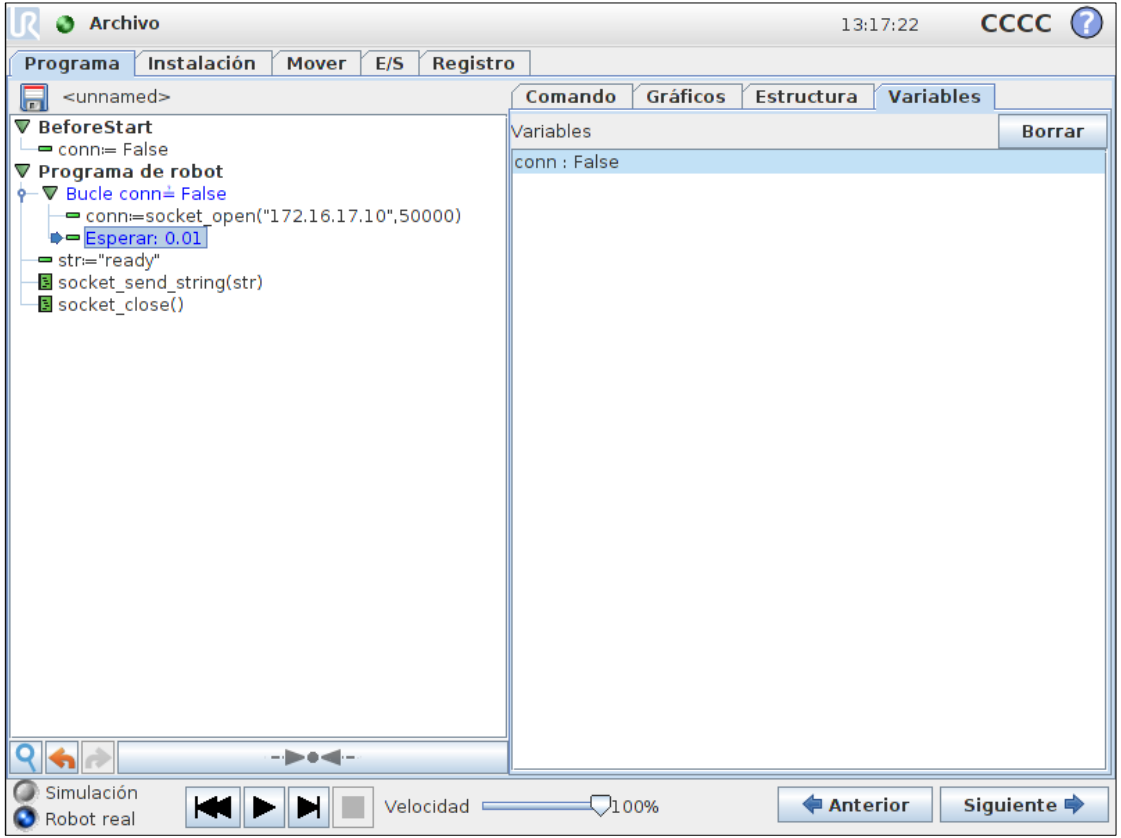
Sends the string <str> through the socket in ASCII coding. Expects no response.

**Parameters**

| | |
|---|---|
| `str:` | The string to send (ascii) |
| `socket_name:` | Name of socket (string) |

## Programa de ejemplo para enviar un *string*

- Iniciar el Servidor
- Enviar *string* al Servidor
  - Abrir socket
  - Esperar a socket abierto
  - Enviar *string*
  - Cerrar socket
- Comprobar resultado en el Servidor



## Recibir respuesta

- Socket_read_ascii_float (*n*, *socket_name*)
  - número
  - nombre del socket

- Valor devuelto
  - Lista de *floats* + 1 entero
    - (El primer elemento de la lista es un entero e indica el número de *floats* válidos recibidos en la lista)

Programa de robot
socket_read_ascii_float(3)

**socket_read_ascii_float**(*number*, *socket_name*='socket_0')

Reads a number of ascii formatted floats from the TCP/IP connected. A maximum of 30 values can be read in one command.

```
>>> list_of_four_floats = socket_read_ascii_float(4)
```

The format of the numbers should be in parantheses, and seperated by ",". An example list of four numbers could look like "( 1.414 , 3.14159, 1.616, 0.0 )".

The returned list contains the total numbers read, and then each number in succession. For example a read_ascii_float on the example above would return (4, 1.414, 3.14159, 1.616, 0.0).

A failed read or timeout after 2 seconds will return the list with 0 as first element and then "Not a number (nan)" in the following elements (ex. (0, nan., nan, nan) for a read of three numbers).

**Parameters**

`number:` The number of variables to read (int)

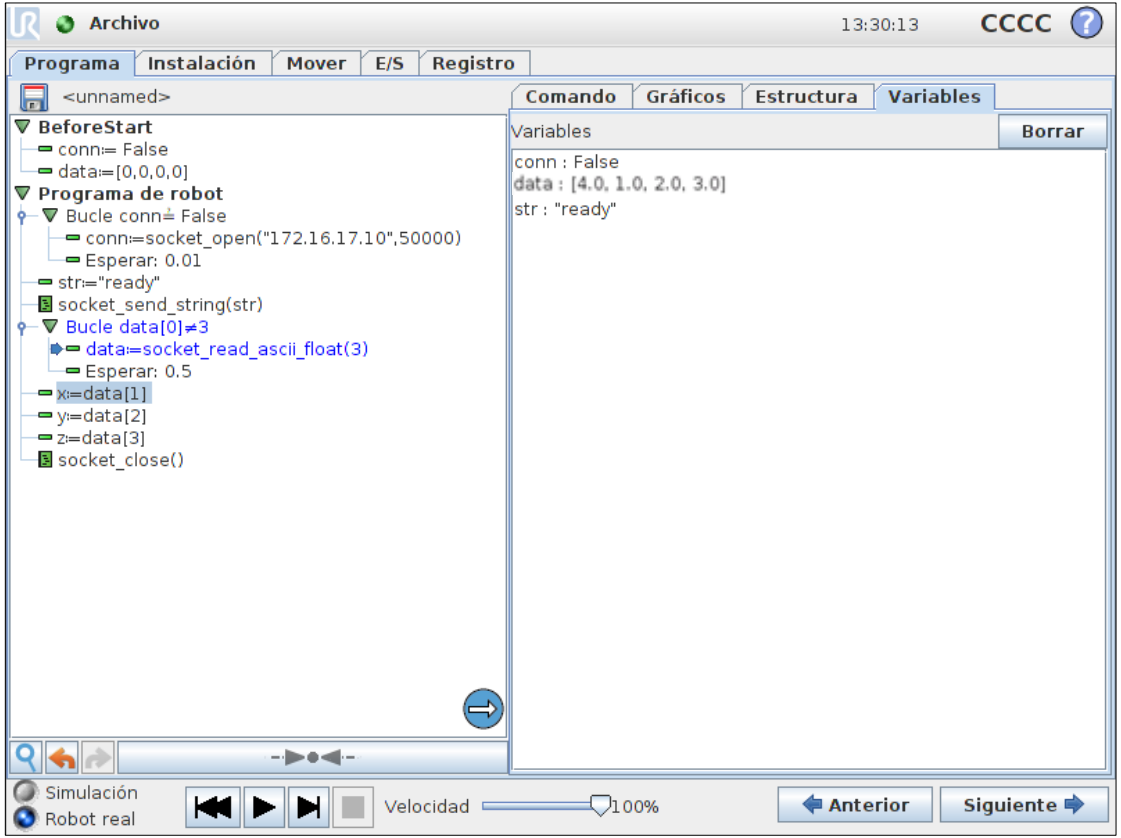
`socket_name:` Name of socket (string)

**Return Value**

A list of numbers read (list of floats, length=number+1)

## Programa de ejemplo para recibir una respuesta

- Continuar ejemplo
  - Abrir socket
  - Esperar a socket abierto
  - Enviar *string*
  - Enviar petición al Servidor
  - Leer tres *floats*
  - Esperar hasta recibir los tres *floats*
  - Guardar el valor de los *floats* recibido en variables *
    - x
    - y
    - rz
  - Cerrar socket




* El Servidor podría ser una cámara de visión

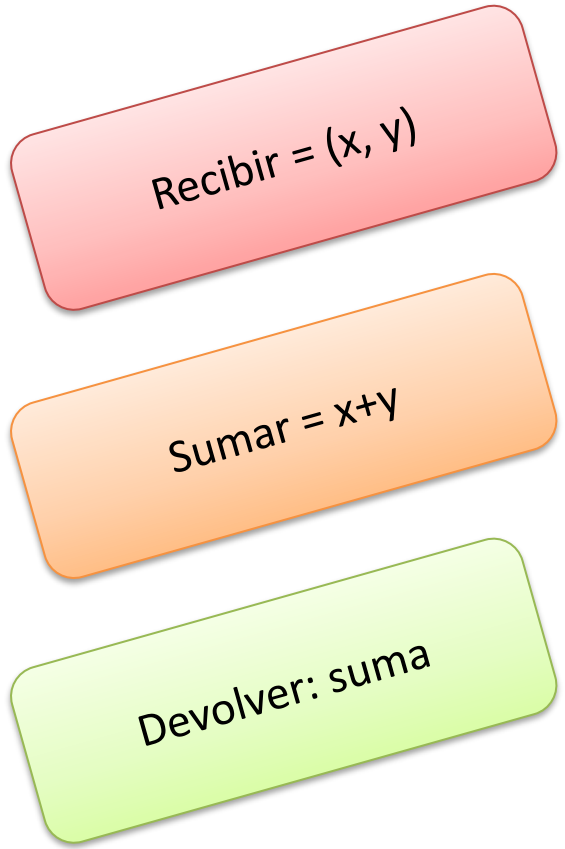
## Formatos de datos

- Formatos de datos disponibles
  - *String*
  - *Float*
  - *Integer*
  - *Byte*
  - *List*

- En el manual de script se describen todas las funciones para la comunicación pos sockets
  - Trabajar con las funciones set_var() y get_var() permite transmitir el nombre de la variable y el valor entero en un único mensaje

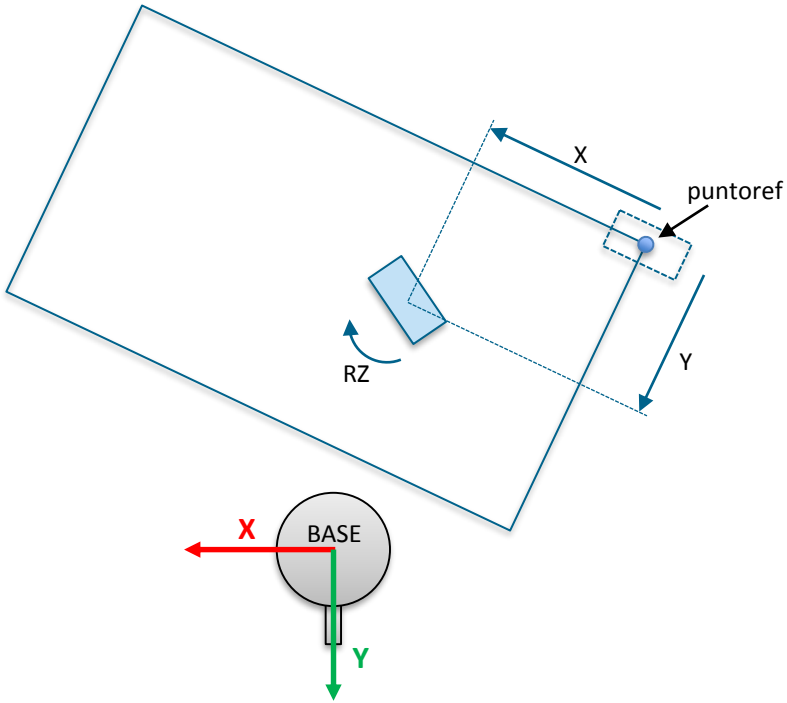
## Ejercicio práctico parte 1

- Escribir un programa para:
  - Establecer una conexión al servidor SocketTest
  - Recibir dos *floats* a través del socket
  - Sumar ambos valores
  - Devolver el resultado al Servidor
  - Esperar otro par de valores
- Ejemplo:
  - El Servidor envía: (5,2)
  - El programa devuelve: 7



## Ejercicio práctico parte 2

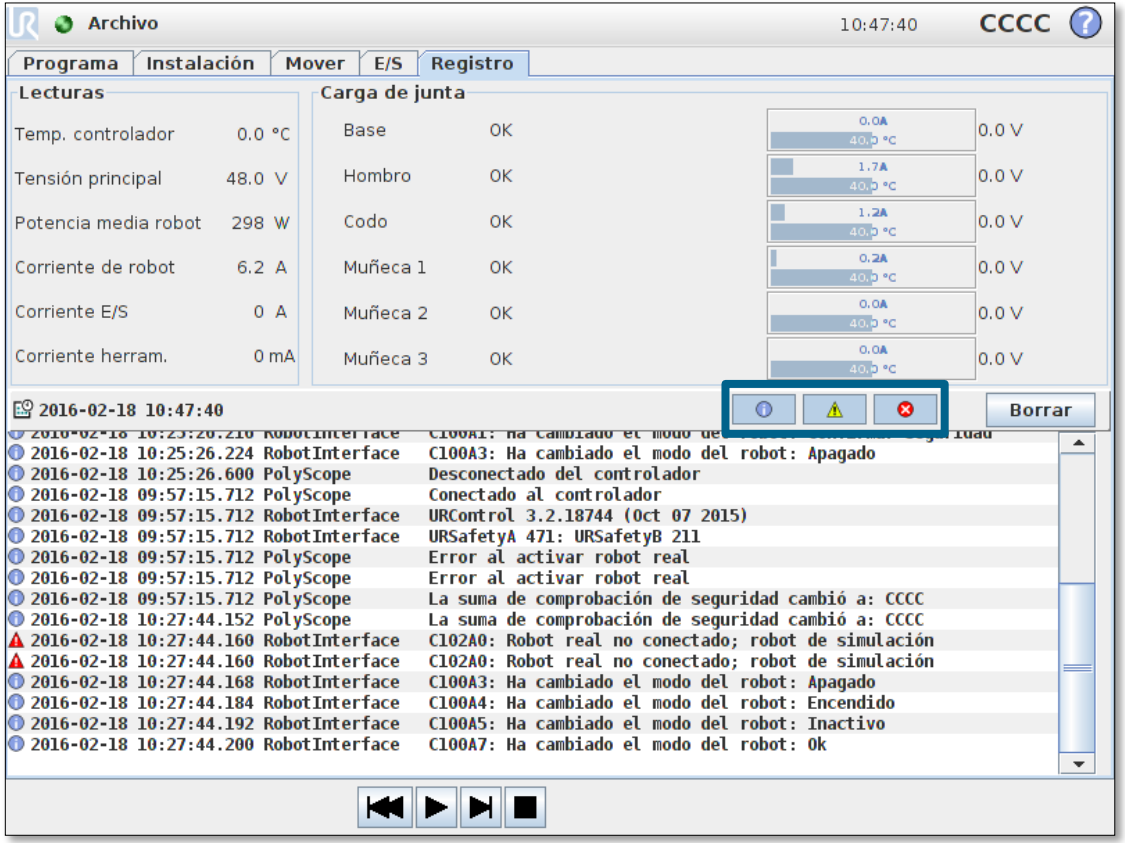
- Escribir un programa simulando recibir coordenadas desde un sistema de visión a través de un socket, y luego moverse a la posición recibida:
  - Enviar mensaje para pedir valores a través del socket y luego recibir los tres valores
  - Moverse a la posición, basarse en el ejemplo usando pose_trans() de la sección Variables




- Formato de datos:
  - Enviar *string* “Preparado” al Servidor
  - Leer tres *ascii floats*:
  - x
  - y
  - rz

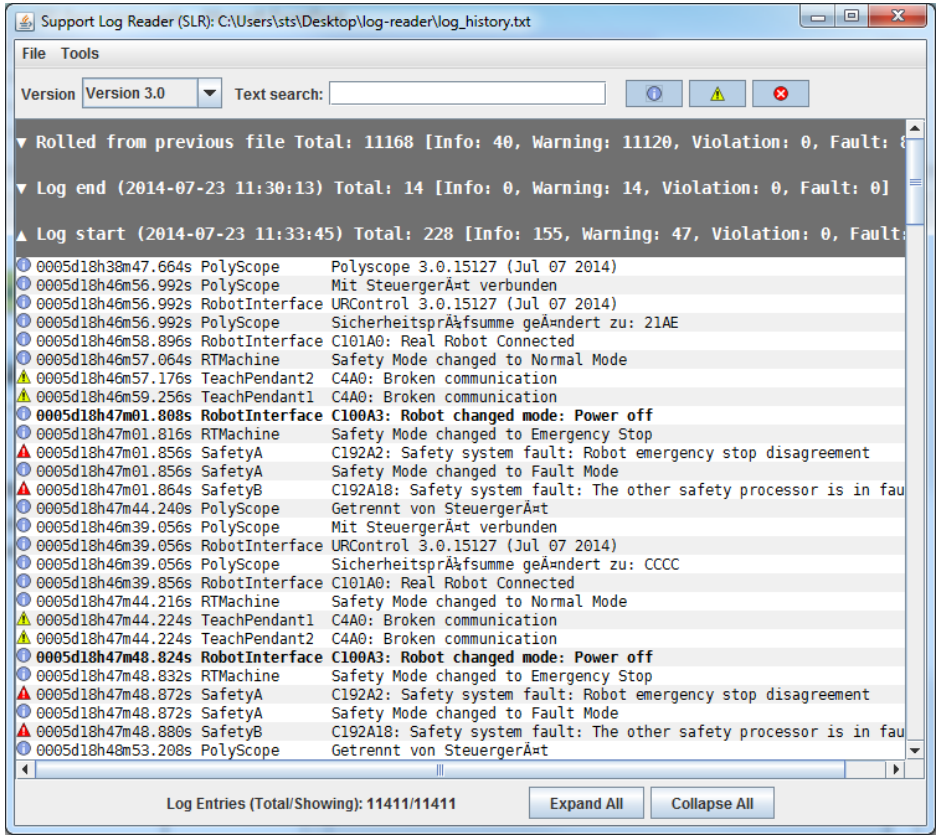
## Análisis de registro histórico

- Registro histórico
  - Contiene
    - Información
    - Avisos
    - Errores
  - Filtro con Mostrar/Ocultar
  - Obtener el fichero *log* del robot usando *Magic file* o FTP
- Análisis del *log*
  - ¿Qué significa cada mensaje?
  - ¿Cómo mejorar la estructura del programa?



## *Support Log Reader*

- *Support Log Reader*
  - Leer ficheros *log*
  - Convertir lenguaje
  - Exportar a fichero csv
  - Filtro de mensajes
  - Soporta
    - Formato CB3
    - Formato CB2




- *Códigos de error y acciones correctivas en Manual de mantenimiento*

## Advertencia de fuerza de medida alta

- Código de error

  - *URControl C114A0: Advertencia de fuerza de medida alta*

- Causa
  - Error común en programas con aceleraciones elevadas o definición incorrecta de la carga
  - Este mensaje por si sólo no afecta a la ejecución del programa, pero puede llevar a:

    - *URControl C100A4: Ha cambiado el modo del robot: SECURITY STOPPED*
    - *URControl C113A0: Para de protección por limitación de fuerza*
  - Estos mensajes sí detienen la ejecución del programa


- Cómo solucionarlo
  - Comprueba los ajustes de carga
  - Reducir la velocidad/aceleración en el programa

## Elevada carga del procesador

- Código de error

  - *C116A216: Advertencia de parte en tiempo real*

- Causa
  - Mensaje de aviso común en programas que hacen un uso intensivo de la CPU
  - Este mensaje por si sólo no afecta a la ejecución del programa, pero puede llevar a:

    - *URControl C100A4: Ha cambiado el modo del robot: SECURITY STOPPED*
    - *R 0002d02h47m51.616s SafetySys C19A0: La unidad principal no tenía datos que enviar a las juntas*
    - *R 0002d02h47m51.624s SafetySys C29A1: Se detectó pérdida de paquete Ethernet entre el PC y el robot*
  - Estos mensajes sí detienen la ejecución del programa


- Cómo solucionarlo
  - Reducir el número de condiciones if, reemplazarlas mediante if … else if … else cuando sea posible
  - Añadir instrucciones de espera o sync() para liberar tiempo de CPU
  - Asegurarse de no tener un gran número de canales ModBus con elevada frecuencia de refresco

## Bucle infinito detectado

- Código de error
  - *Error de tiempo de ejecución: bucle infinito detectado en el programa*
- Causa
  - Ejecución de instrucciones una y otra vez sin tiempo de espera intermedio
  - El programa se detiene
- Cómo solucionarlo
  - Añadir “Esperar” con un pequeño tiempo en los bucles
  - Usar la función sync()

## Singularidades

- Código de error
  - *C154A20: Parada de protección: Posición en singularidad*

- Causa
  - Los problemas de singularidad ocurren al controlar el robot en el espacio cartesiano (movel/movep)
  - Se produce una singularidad cuando:
    - Ejes del robot son redundantes (más ejes que los necesarios pueden causar el mismo movimiento) o
    - El robot se encuentra en cierta configuración que requiere de velocidades extremadamente altas de juntas para mover el PCH a cierta velocidad nominal en el espacio cartesiano

- Cómo solucionarlo
  - Usar movej() en los movimientos cercanos a la singularidad
  - Si se requieren movimientos en el espacio cartesiano (lineales), ajustar las posiciones de paso
  - Modificar el layout de la aplicación y ajustar los puntos de paso
  - Modificar la herramienta del robot cambiando el ángulo de la dirección de la herramienta

## Límites de ejes

- Código de error
  - *C150A0: Parada de protección Posición cercana a los límites de junta*
- Causa
  - Indica que alguna de las juntas está cerca de su límite de operación
- Cómo solucionarlo
  - Abrir la pestaña Mover
  - Comprobar la posición de las juntas
  - Rotar 360º el eje que se encuentre al límite de rotación
  - Ajustar de nuevos los puntos de paso MoveJ

## Test

- Tiene 30 minutos para responder.
- Las preguntas pueden tener múltiples respuestas, así que revise todas las opciones cuidadosamente antes de responder.
- Indique la respuesta correcta mediante una marca y rellene los espacios en blanco con claridad cuando sea necesario.
- Puede usar cualquier material que tenga disponible. Evite hacer comentarios con los demás examinados.

- ¡Buena suerte!

## Resultados del Test

- 1. ¿Cómo se puede modificar la posición del centro de masas?
  - ☑ Solución: 3.0 – Usar la función URScript ***set_payload()*** || Solución: 3.1 – **Introducir los valores de Carga en la pantalla de configuración del PCH**

- 2. ¿Con qué unidades de medida se trabaja en el lenguaje script de UR?
  - ☑ Distancias : ***metros*** Ángulos : ***radianes***

- 3. ¿En qué número de puerto esta disponible el Servidor Dashboard?
  - ☑ a. 29999
  - ❑ b. 30000
  - ❑ c. 30001
  - ❑ d. ¿Qué es el Servidor Dashboard?
  - ❑ e. Ninguno de los anteriores

- 4. ¿Cómo conectaría un HMI al robot?
  - ❑ a. Usando el Servidor Dashboard en el puerto 29999
  - ❑ b. Usando el interface de Cliente secundario en el puerto 30002
  - ❑ c. No es posible trabajar sin la consola de programación
  - ☑ d. (a) o (b)
  - ❑ e. (b) y (c)

- 5. ¿Cuáles de las siguientes consideraciones son importantes al realizar un análisis de riesgos completo?
  - ❑ a. Grado de severidad del daño
  - ❑ b. Posibilidad de evitar el daño
  - ❑ c. Frecuencia y duración de la exposición al riesgo
  - ❑ d. Distancia del robot al vallado de seguridad y número de sensores utilizados
  - ☑ e. (a)(b) y (c)

## Resultados del Test

- 6. ¿Qué función URScript puede usarse para conocer el estado de una E/S desde un valor entero en un registro de entrada ModBus?
  - ❑ a. get_standard_digital_out()
  - ❑ b. integer_to_binary_list()
  - ❑ c. socket_send_byte()
  - ☑ d. force()
  - ❑ e. Ninguna de las anteriores

- 7. ¿A qué corresponde el tercer valor en una variable de tipo “pose”?
  - ❑ a. Rotación en Y
  - ❑ b. Rotación en Z
  - ☑ c. Posición en Z
  - ❑ d. Posición en X
  - ❑ e. Ninguno de los anteriores

- 8. ¿Qué función es usada como referencia en la función pose_add() de URScript?
  - ❑ a. La posición del primer argumento
  - ☑ b. La función Base
  - ❑ c. La posición del Segundo argumento
  - ❑ d. La función de teatro
  - ❑ e. Ninguno de los anteriores

- 9. Al crear una función de usuario, el origen de la misma se encuentra en:
  - ❑ a. La tercera posición al definir la función
  - ❑ b. A la mitad del primer y el segundo punto
  - ☑ c. La primera posición al definir la función
  - ❑ d. En la base del robot
  - ❑ e. Ninguno de los anteriores

## Resultados del Test

- 10. La diferencia entre una variable de instalación y una variable de programa es:
  - ❑ a. Una variable de instalación contiene toda la información acerca de dónde y cómo se instala el robot. Una variable de programa no
  - ❑ b. El valor de una variable de programa se guarda entre las ejecuciones del programa y tras reiniciar el robot. En una variable de instalación no
  - ☑ c. El valor de una variable de instalación se guarda entre las ejecuciones del programa y tras reiniciar el robot. En una variable de programa no
  - ❑ d. No hay ninguna diferencia
  - ❑ e. La única diferencia es que se escriben de manera diferente

- 11. Para usar la funciones URScript set_tcp() en un programa, primero es necesario:
  - ❑ a. Fijar la carga en 0.0kg en la pantalla de configuración de PCH
  - ❑ b. Crear una función “Línea” en la pantalla de funciones
  - ❑ c. Establecer como variable la función ““Base” en la pantalla de funciones
  - ❑ d. Mover el robot a la posición Origen
  - ☑ e. Ninguna de las anteriores

- 12. La longitud de un registro ModBus es de:
  - ❑ a. 4 bits
  - ❑ b. 1 byte
  - ☑ c. 16 bits
  - ❑ d. 32 bits
  - ❑ e. Ninguna de las anteriores

- 13. La contraseña necesaria para conectar al servidor FTP del robot es:
  - ❑ a. Admin
  - ❑ b. 123456
  - ☑ c. easybot
  - ❑ d. Terminator
  - ❑ e. ur

## Resultados del Test

- 14. ¿Cuántos interfaces de Cliente existen? (sin incluir el Servidor Dashboard)
  - ❑ a. 5
  - ☑ b. 3
  - ❑ c. 1
  - ❑ d. 18
  - ❑ e. 0

- 15. Al usar la función URScript socket_open(), cuando es necesario incluir el argumento socket_name?
  - ❑ a. Siempre
  - ☑ b. Cuando hayan diferentes sockets abiertos en un programa
  - ❑ c. Cuando el número de puerto sea mayor que 45000
  - ❑ d. Nunca
  - ❑ e. Cuando el socket se sienta solo

- 16. ¿Qué hacer si nos encontramos con mensajes de aviso “Fuerza medida excesiva”?
  - ❑ a. Comprobar los ajustes de PCH/Carga
  - ❑ b. Reducir la aceleración en el programa
  - ❑ c. Parar, echarnos al suelo y rodar
  - ❑ d. Reducir la longitud del programa
  - ☑ e. (a) y (b)

- 17. Al usar la función URScript socket_read_ascii_float(), el primer valor en la lista de retorno corresponde a:
  - ❑ a. El primer float enviado por el servidor
  - ❑ b. El último float enviado por el servidor
  - ❑ c. El nombre del servidor
  - ☑ d. El número de floats enviados por el servidor
  - ❑ e. Ninguno de las anteriores

## Resultados del Test

- 18. Dentro de un programa, si a una variable se le asigna un valor entero y posteriormente se le asigna una cadena de texto, ¿qué ocurrirá?
  - ❑ a. El programa se ejecutará sin problemas, el tipo de las variables es dinámico
  - ☑ b. Se producirá un error de tipo, ya que la variable se ha definido de tipo entero y se le intenta asignar un tipo cadena.
  - ❑ c. Se creará un nuevo subproceso automáticamente
  - ❑ d. Se creará una nueva variable que contenga el valor de la cadena asignado
  - ❑ e. La variable se convertirá al tipo Stringteger

- 19. ¿Cómo podemos enviar al robot una posición desde una cámara de visión?
  - ❑ a. Usando la comunicación por Ethernet socket
  - ❑ b. Usando E/S digitales
  - ❑ c. Usando registros ModBus
  - ❑ d. Todas las anteriores
  - ☑ e. (a) y (c)

- 20. ¿Qué función URscript se usa para calcular la distancia cartesiana entre dos posiciones?
  - ❑ a. pose_inv()
  - ❑ b. get_target_tcp_pose()
  - ❑ c. pose_sub()
  - ☑ d. pose_dist()
  - ❑ e. Todas las anteriores