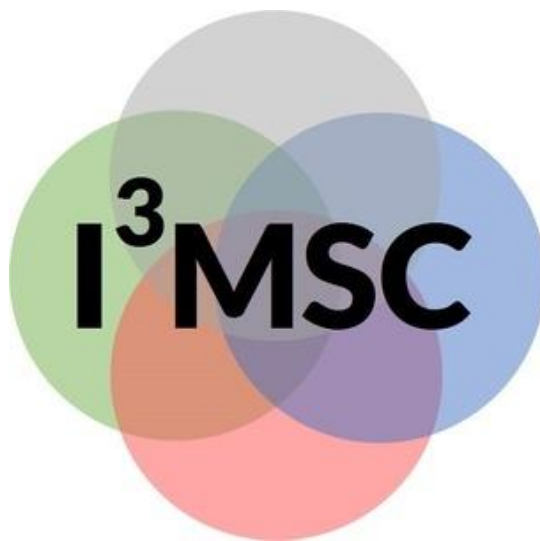


Documentación de Prácticas de Empresa 2021/2022

**Instituto Universitario de Investigación en
Ingeniería Mecatrónica y Sistemas
Ciberfísicos**



José Luis Cambil Calderón

Paolo Ruiz Stocchetti

4º GIERM UMA

Curso 2021/2022

INDICE

1. Introducción	1
2. Visual Components	2
2.1. Layout Básico	2
2.2. Configuraciones planteadas	3
2.2.1. Configuración 1.	3
2.2.2. Configuración 2.	4
2.3. Herramientas	5
2.4. Conexión TCP/IP con IRC5 (Real y RobotStudio). Sentido RS → VC.	10
2.5. Conexión TCP/IP con IRC5 (Real y RobotStudio). Sentido VC → RS.	15
3. RobotStudio	23
3.1. Conexión TCP/IP con Visual Components. Sentido RS → VC.	23
3.2. Conexión TCP/IP con Visual Components. Sentido VC → RS.	27
3.2.1. Tarea Main.	27
3.2.2. Tarea TCP.	33
4. Versión 2 del programa VC -> RS	37
4.1. Código de Visual Components	37
4.2. Código de RobotStudio (main)	43
4.3. Código de RobotStudio (TCP)	49

1. Introducción

Este documento tiene como intención recoger los avances y métodos empleados durante las prácticas curriculares realizadas en el I3MSC por José Luis Cambil y Paolo Ruiz en el curso 2021/2022, a fin de facilitar y acelerar el desarrollo y los avances realizados en relación con la preparación del segundo curso de Gemelos Digitales de la UMA. Estos son:

- Diseño y modelado de prototipos de célula robotizada.
- Implementación en Visual Components de herramientas.
- Desarrollo de un sistema que permite a Visual Components imitar en tiempo real el comportamiento del robot (RS -> VC).
- Desarrollo de un sistema que permite al usuario controlar el robot mediante rutinas creadas en Visual Components (VC -> RS).

Los apartados 1 y 2 dividen la mayor parte de la información en los dos programas empleados: Visual Components y RobotStudio; mientras que el tercero trata de una versión modificada de uno de los programas desarrollados, e incluye código de ambos programas.

2. Visual Components

2.1. Layout Básico

Como base para el desarrollo y diseño de distintas configuraciones o posibilidades de células robotizadas, hemos creado un proyecto de Visual Components que contiene los elementos básicos con los que vamos a trabajar y que busca imitar el espacio de trabajo que hay en el taller 7 de la Escuela de Ingenierías Industriales.

Concretamente tendremos dos mesas de dimensiones idénticas a las reales, con 4 robots IRB 120, colocados de la misma manera en la que están en el taller. A dichos robots se les ha añadido un controlador, que será útil en caso de querer hacer programas con procesos y usar los manipuladores; y las herramientas (pinzas y ventosas) que se han modelado (Para más información mirar el apartado *Herramientas*) junto con las señales y acciones para agarrar y liberar asignadas.

Este proyecto se llama "Layout Básico Herramientas.vcmx". Además, añadimos otro modelo que no incluye los controladores ni las herramientas, en caso de querer plantearlo de otra manera. Este proyecto se llama "Layout Básico.vcmx".

2.2. Configuraciones planteadas

Se han planteado varias configuraciones con diferentes comportamientos para tener ejemplos o primeras ideas de lo que se va a montar, además de para poder practicar con el software Visual Components y explorar varias de sus funcionalidades de cara a poder desarrollar el Gemelo Digital en un futuro con mayor soltura y velocidad.

Los modelos desarrollados se encuentran en el fichero comprimido adjunto.

2.2.1. Configuración 1.

Esta primera configuración (figura 1) consiste en añadir dos cintas en paralelo que atraviesen perpendicularmente las dos mesas de trabajo. Dichas cintas tendrán el mismo sentido y transportarán piezas que vienen desde un alimentador, estas piezas son cubos de diferentes colores. La idea es que una cámara clasifique las piezas según esos colores (dicho comportamiento se simula con un fichero python asociado a las cintas que detecta el tipo de piezas que tiene encima) y según si es de cierto color, el primer manipulador de cada cinta hará la acción de pick and place en un palé. Las piezas que sean de un segundo color pasarán hasta el segundo manipulador de la cinta para ser paletizadas en otro palé diferente. Además de las dos cintas y de la cámara, el montaje dispondrá de varios sensores acoplados a las cintas para detectar los objetos en determinadas posiciones y parar o activar la cinta.

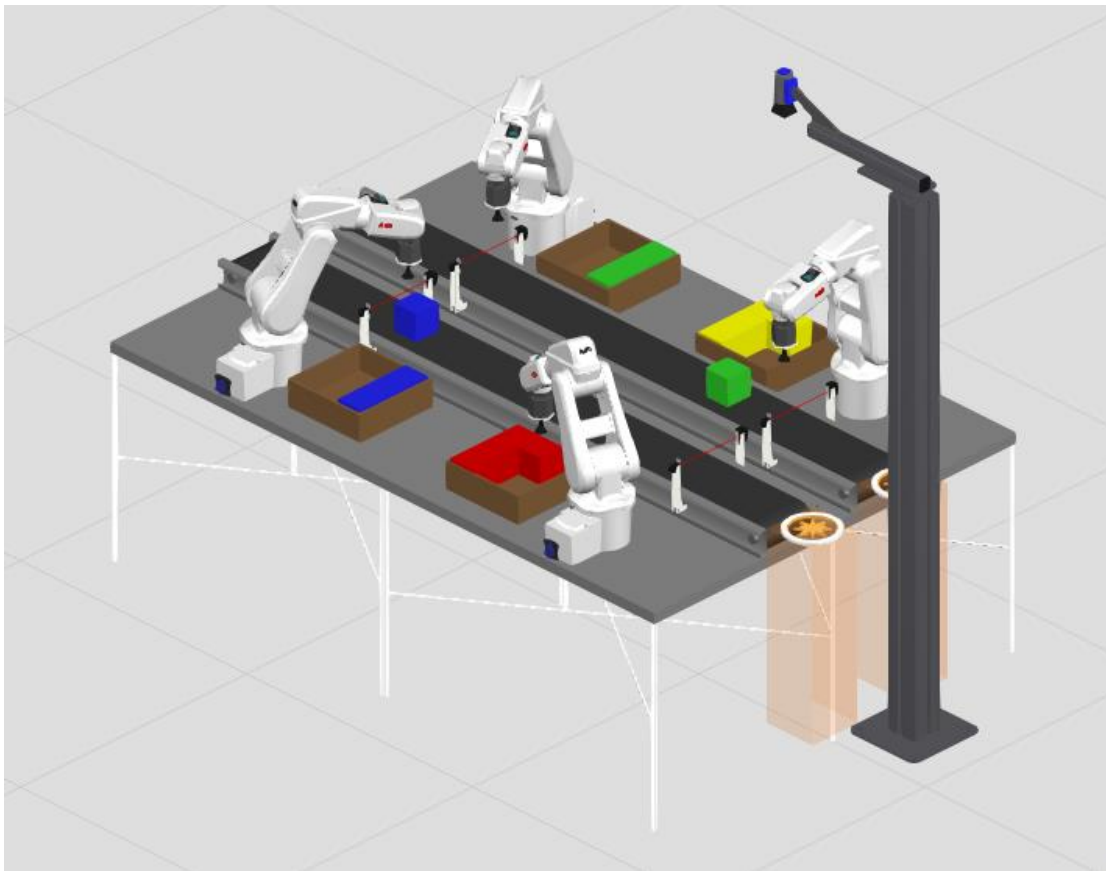


Figura 1 Configuración 1

2.2.2. Configuración 2.

Esta segunda configuración (figura 2) está formada por tres, donde se busca algo más cíclico donde cada robot tiene su propia tarea, de forma que todo sea más similar a un entorno industrial. Lo primero es la alimentación con un palé ya montado con 8 piezas (de las cuales 2 eran defectuosas), por tanto, la primera tarea es llevada a cabo por el primer robot y consiste en el despaletizado y en la colocación en la primera cinta transportadora. Al final de esa cinta, el segundo robot se encarga del transporte de las piezas entre la primera y la segunda cinta. En esta segunda cinta hay una cámara que evalúa las piezas (igual que en la configuración anterior). El tercer robot recibe el resultado de la cámara y en función de si es una pieza válida o no, descarta dicha pieza o pasarla a una tercera cinta. El cuarto robot se encarga de paletizar las piezas válidas. Igual que antes, se hacen uso de diferentes sensores en las cintas.

La idea de emplear tres cintas ya fue descartada por temas de presupuesto, por lo que consideramos alternativas como emplear un operador humano.

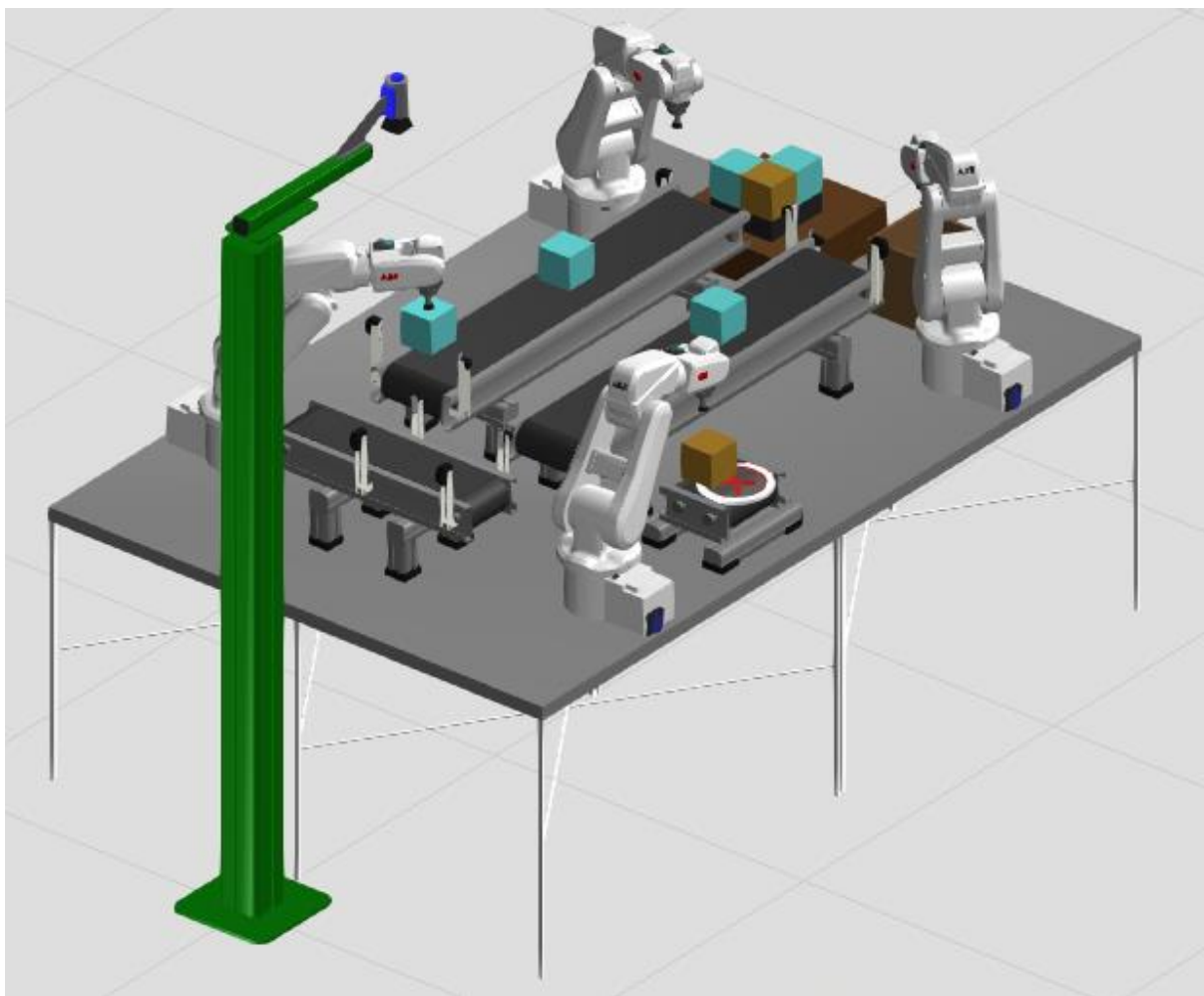


Figura 2 Configuración 2

2.3. Herramientas

Creamos otros 3 modelos de Visual Components para los tres tipos de herramientas instalados en los robots: una pinza y dos ventosas.

Para la pinza se ha usado la geometría dada y a partir de ahí se le ha dado el comportamiento de abrir y cerrar. Los movimientos de la pinza son de giro respecto al eje de anclaje de cada una de las piezas móviles y se ha establecido el estado abierto para la posición de reposo. Por lo tanto, se han creado los elementos necesarios (controlador servo, señales booleanas, etc.) para un correcto funcionamiento. Podemos ver en las figuras 3 y 4 los dos estados de la pinza.

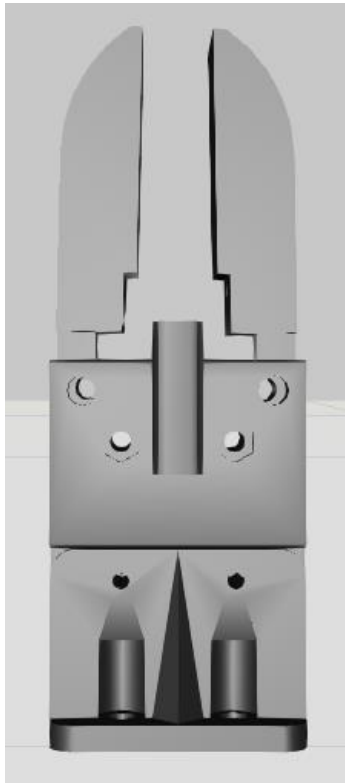


Figura 3 Pinza cerrada

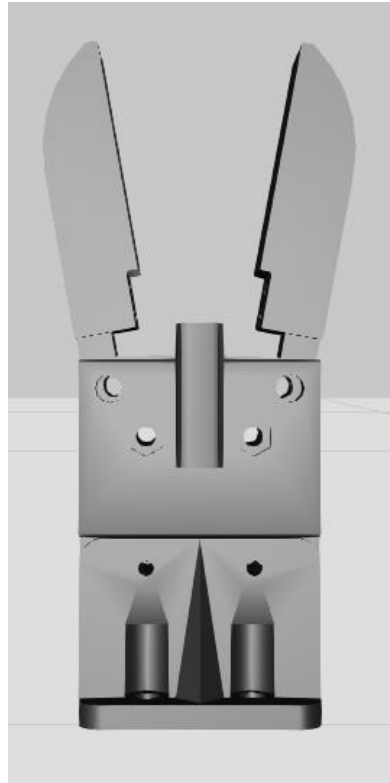


Figura 4 Pinza abierta

Una vez dado este comportamiento mecánico, es necesario darle el comportamiento correspondiente de herramienta. Usaremos el asistente desde la pestaña de modelado para ello, como vemos en la figura 5.

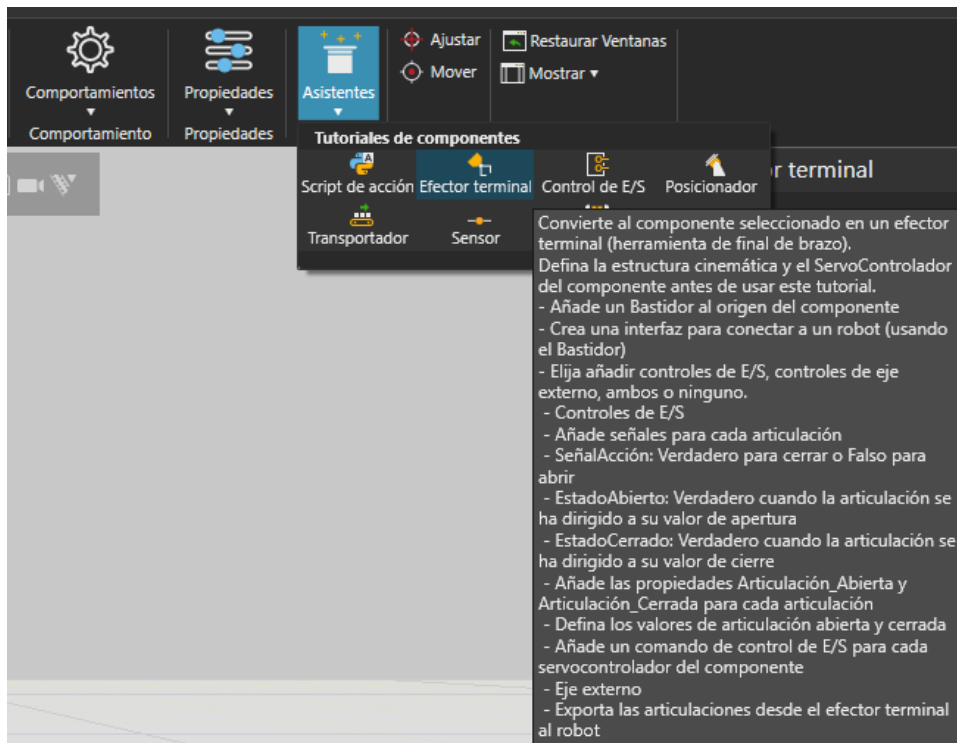


Figura 5 Asistente para modelado de herramienta

De esta manera, haremos que esta geometría (o cualquier otra que se quiera modelar) sea reconocida como herramienta. Al usar esta funcionalidad tendremos tres cambios principales en sus propiedades y comportamientos:

- Se podrá acoplar al último link de cualquier manipulador como herramienta de manera automática, mediante el uso del PnP, y se moverá junto al resto de la geometría.
- Se creará un sistema de referencia para acoplar dicha herramienta al manipulador (por defecto se sitúa en la base así que no hay porqué moverlo). Se creará otro sistema de referencia llamado TCP, que será donde en pasos posteriores, al darle la propiedad de agarrar una pieza, se sitúe el punto de agarre. Este sí que hay que situarlo en función de cada herramienta, sea una pinza o una ventosa.
- Si es una herramienta con movimiento de geometría (abierto, cerrado, por ejemplo), se crearán varias señales, para cambiar de estado entre esas posiciones y unos sensores para indicar la posición que tiene. Se puede elegir entre una señal que cambia la posición dependiendo de su estado, o dos señales y que cada una se encarga de controlar uno de los estados. Nosotros hemos planteado las dos soluciones.

En caso de no tener movimiento, como una ventosa, estas señales no se crearán. Aun así, recomendamos crear de manera manual una señal booleana a la que conectar posteriormente la salida 1 del manipulador (o cualquiera que se interprete como activadora de la herramienta), ya que Visual Components no permite activar manualmente las salidas que no estén conectadas.

Una vez realizado este paso ya podemos acoplarlo a cualquier manipulador como vemos en la figura 6.

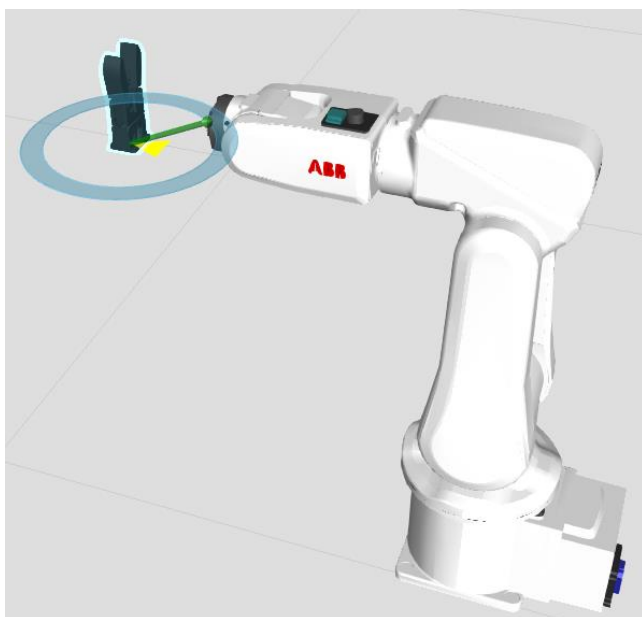


Figura 6 Acoplamiento de la herramienta a un robot

Para darle de manera fácil un comportamiento de agarre, desde la pestaña de inicio seleccionaremos al manipulador (todos tienen la misma propiedad) y elegiremos la configuración de acciones, donde añadiremos la acción de agarrar y liberar a la herramienta que acabamos de añadir (también sirve para herramientas por defecto de Visual Components como el *vacuum gripper*). Podemos verlo mejor en la figura 7. Se puede asociar a la señal que más convenga (nosotros hemos optado por emplear la salida 1).

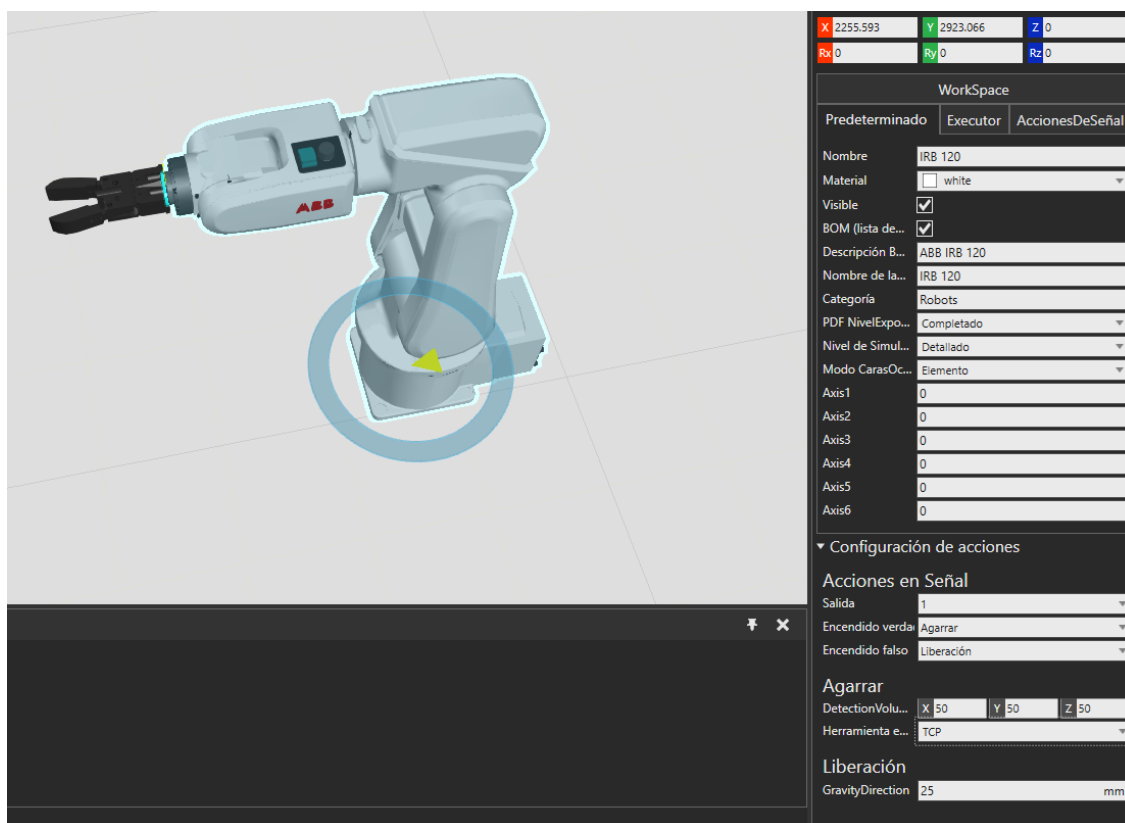


Figura 7 Configuración de acciones

Una vez modelada la pinza, pasamos a modelar las dos ventosas que hay en el layout real. Se nos han proporcionado también los modelos geométricos, los cuales hemos cargado y les hemos dado el comportamiento y propiedades de una herramienta (figuras 8 y 9).

Hemos ajustado el marco de referencia del montaje y el TCP. Para este caso no es necesario añadir ningún movimiento, como se había comentado anteriormente.

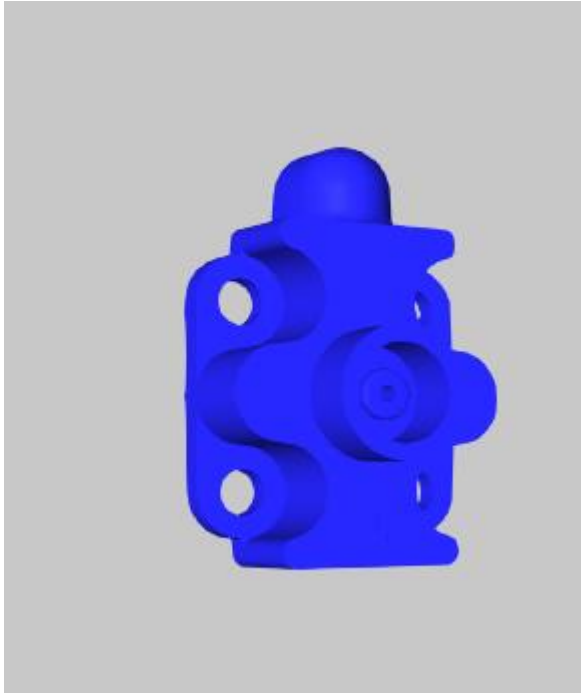


Figura 8 Modelo de ventosa 1

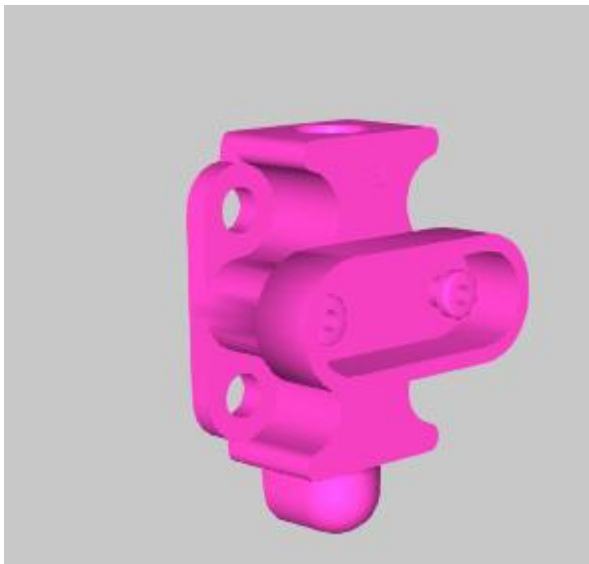


Figura 9 Modelo de ventosa 2

También es importante saber que esto solo le da la propiedad de agarrar los objetos, pero sin realizar la animación, por lo que la forma de acoplar ambas acciones (cerrar–agarrar objetos / abrir–liberar objetos) es la de elegir la entrada del robot al que asociamos la señal de la pinza que realiza la acción de cerrar, a la misma señal que acabamos de seleccionar para la acción de agarrar objetos.

Además, con ficheros de python, es fácil asociar estas acciones para que cuando se active una, se active la otra. En la ventosa es más fácil ya que al no tener movimiento en su geometría, solo es necesario añadir la acción de agarrar – liberar.

Debido a la animación de cerrar de la pinza, recomendamos dejar un delay de mínimo un segundo desde que se da la orden de activar la señal para que esta se complete.

También sugerimos introducir estos tres proyectos dentro de la carpeta “My Models”, que por defecto se encuentra en “Documentos/Visual Components/Version”. Como vemos en la figura 10. En caso de no hacerlo, VC no reconocerá los modelos y no los mostrará en el apartado “Mis modelos” del “eCatalogue”.

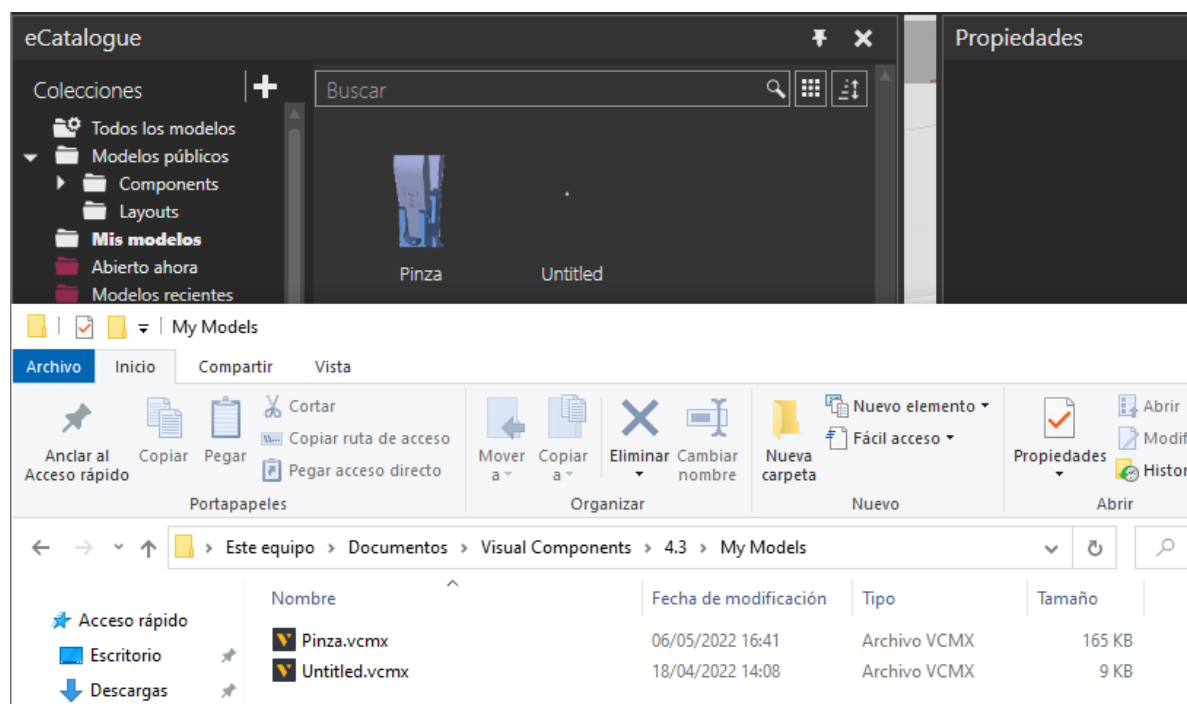


Figura 10 Guardado de nuestros modelos

2.4. Conexión TCP/IP con IRC5 (Real y RobotStudio). Sentido RS → VC.

Otra de las tareas realizadas fue la conexión en tiempo real entre el robot y Visual Components. Para lograrlo, empleamos el protocolo TCP/IP mediante sockets. Recomendamos hacer pruebas primero en RobotStudio antes de pasar el programa al robot.

Visual Components actúa como cliente en esta conexión, por lo que es muy importante adecuar la IP y puerto de entrada al del servidor (IP del robot o local en caso de RobotStudio). Estos parámetros se ajustan dentro del apartado Propiedades del Robots, en la pestaña Modelado.

El script de Python que permite realizar esta conexión debe incluir dentro del apartado Comportamientos (en Modelado) del modelo del IRB 120 (figura 11).

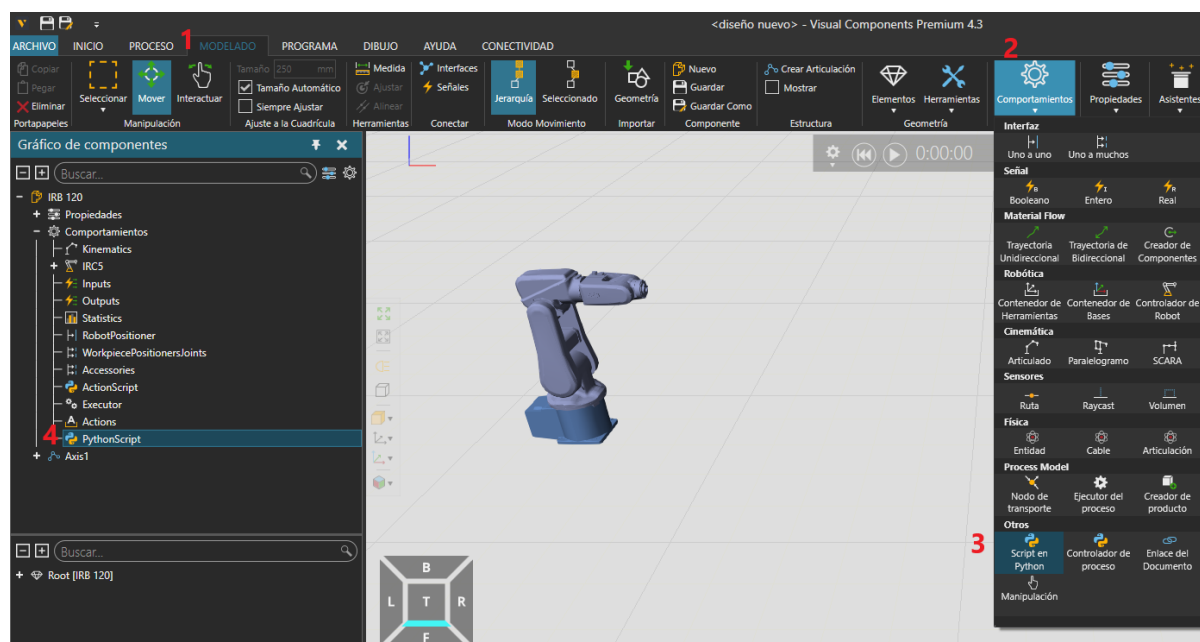


Figura 11 Adición de un nuevo script de Python en el modelo del robot

```
from vcScript import *
from vcHelpers.Robot import *
from vcHelpers.Robot2 import *
import socket
import time
import errno
import os

local_ip = '127.0.0.1'
robot_ip = '127.0.0.1'
recv_port = 30001
send_port = 30002
SIZE = 250

app = getApplication()
comp = getComponent()
robot = comp
conectado = False
err = "";
```

```

tool = False
robotnum = 1

def OnStart():
    #Executes before sim clock starts running when play is pressed
    global robot, exe, ctr, inmap, doStates, robotnum
    global local_ip, robot_ip, recv_port, send_port, my_socket, conectado

    #Connection vars
    local_ip = comp.getProperty('LocalIP').Value
    robot_ip = comp.getProperty('RobotIP').Value
    recv_port = comp.getProperty('RecvPort').Value
    send_port = comp.getProperty('SendPort').Value
    intentoConexion = 5

    while (intentoConexion > 0):
        print "ROBOT_%i: Intento de conexion n°: %i" %(robotnum, 6-
intentoConexion)
        try:
            my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            my_socket.settimeout(1)
            my_socket.connect((robot_ip, recv_port))
            print "ROBOT_%i: Éxito en la conexion" %robotnum
            break
        except socket.error, exc:
            print "ROBOT_%i: Error de conexion : %s" % robotnum, exc
            if (intentoConexion == 1):
                return
        finally:
            intentoConexion = intentoConexion - 1
    if robot:
        exe = robot.findBehavioursByType(VC_ROBOTEXECUTOR)[0]
        inmap = exe.DigitalInputSignals
        inmap.OnSignalTrigger = writeDI #Event that writes inputs from sim
to robot
        ctr = robot.findBehavioursByType(VC_ROBOTCONTROLLER)[0]

    else:
        exe = None
        ctr = None
        inmap = None
        doStates = {}
        conectado = True

def OnRun():
    #Executes after sim clock starts running
    global robot, exe, ctr, my_socket, conectado, err, tool, robotnum
    #Init receiving socket
    while conectado == True:
        print ('ROBOT_%i: Receiving packets from local IP {0}, port
{1}\n'.format(local_ip, recv_port)) %robotnum
        print ('ROBOT_%i: Sending packets to IP {0}, port
{1}\n'.format(robot_ip, send_port)) %robotnum

        if not robot:
            print 'ROBOT_%i: No robot associated to TCP link' %robotnum
            return

```

```

delay(0.001)
try:
    #Send dummy package so robot picks up IP
    my_socket.send('DUMMY')
except:
    pass

mt = ctr.createTarget()
mt.MotionType = VC_MOTIONTARGET_MT_JOINT
mt.UseJoints = True
jv = mt.JointValues

robot1=getRobot()
tool=robot1.SignalMapOut.getInternalPortSignal(1)

while True:
    delay(0.05)
    try:
        (data1, addr) = my_socket.recvfrom(SIZE)
        data = ""
        data = data1.split(" ")
        data = data[0]

        #Data from robot to sim (polling)
        if 'JOINTS' in data:
            readJoints(data, mt)
        elif 'DO' in data:
            readDO(data)
        #Estado actual de la herramienta
        estadotool = (tool.Value == True)*1
        #Mensaje con informacion de la herramienta
        infotool = "Tool=" + str(estadotool)
        #Envio del mensaje
        my_socket.sendto(infotool, (robot_ip, send_port))
        #Poll interval
        delay(0.01)

    except socket.error as error:
        delay(0.01)

        if error.errno == 10053:
            print("ROBOT_%i: Error "+str(error.errno)) %robotnum
            print("ROBOT_%i: Intentando reconectar...") %robotnum
            OnStart()
            break
        else:
            print(os.strerror(error.errno))

print('ROBOT_%i: Cerrando el programa') %robotnum

def readJoints(data, mt):
    #Parse joint values and set simulation joints
    global ctr
    data = data.split(':')
    if len(data) < 2:
        return

```

```

jv = mt.JointValues
data = data[1]
data = data.split(',')
for i in range(len(data)):
    val = float(data[i])
    jv[i] = float(val)
mt.JointValues = jv
ctr.moveImmediate(mt)

def readDO(data):
    #Parse string data and set simulation output if state has changed
    global exe, doStates

    data = data.split(':')
    if len(data) < 2:
        return
    try:
        port = int(data[0][2:])
        value = data[1] == '1'
    except:
        return

    old_value = not value
    if data[0] in doStates.keys():
        old_value = doStates[data[0]]

    if old_value != value:
        exe.DigitalOutputSignals.output(port, value)
        doStates[data[0]] = value

def writeDI(map, port, value):
    #Simulation input state changed, send data to robot
    global robot_ip, send_port, my_socket

    if not inmap:
        return

    #Data from sim to robot (event-based)
    value_as_int = 0
    if value:
        value_as_int = 1
    data = 'DI%i:%i' % (port, value_as_int)
    try:
        my_socket.sendto(data, (robot_ip, send_port))
    except socket.error as e:
        my_socket = socket.socket()
        my_socket.connect((robot_ip, recv_port))

```

Su función consiste en conectarse al servidor creado por el controlador IRC5 y copiar en tiempo real la posición del robot en el de Visual Components. Además, replica el estado de la herramienta y permite controlar la herramienta real mediante su activación en Visual Components. Para ello es importante que el modelo del robot tenga conectada su salida 1 a la señal de activación de la herramienta conectada.

Es importante destacar que, para que funcione correctamente, se deben incluir en el apartado “Propiedades”, dentro de la pestaña “Modelado” del robot, 4 variables (figura 12):

- Dos de tipo string con las IPs del robot y del programa (local):
 - o “LocalIP” = “127.0.0.1”.
 - o “RobotIP” = “172.16.78.XXX” (Depende del robot).
- Dos de tipo integer con los puertos de entrada y salida:
 - o “RecvPort” = 8007.
 - o “SendPort” = 8000.

Recomendamos modificar únicamente “RobotIP”, en función del robot con el que se quiera realizar la conexión.

Por otro lado, al comienzo del código, hay una variable llamada “robotnum”. Esta sirve para reconocer fácilmente a que robot corresponden los mensajes publicados en la consola, por lo que recomendamos asignarle uno distinto a cada uno de los colocados en el montaje.

Está basado en un script de conexión UDP publicado por el usuario “keke” en el foro de Visual Components¹, por lo que contiene código destinado a otras funciones, las cuáles descartamos temporalmente con el objetivo de focalizarnos en depurar la conexión y corregir errores.

Hemos logrado que el proceso de conexión sea bastante estable y que sea posible la reconexión sin necesidad de reiniciar el servidor del robot constantemente (cosa que antes era necesaria). Aun así, sigue sin ser perfecta y no contempla todas las situaciones adversas que puedan ocurrir, por lo que requiere depuración y tratamiento de los errores, pero sirve como una buena base.

CONSEJO: Durante la depuración de este programa, es bastante posible que Visual Components se bloquee, aclarándose la ventana del programa. Generalmente, se soluciona rápidamente presionando el botón de cerrar y luego a “intentar restaurar el programa” o, en su defecto, “esperar a que el programa responda”. Esto nos evita, en la mayoría de los casos, tener que cerrar y volver a abrir el programa.

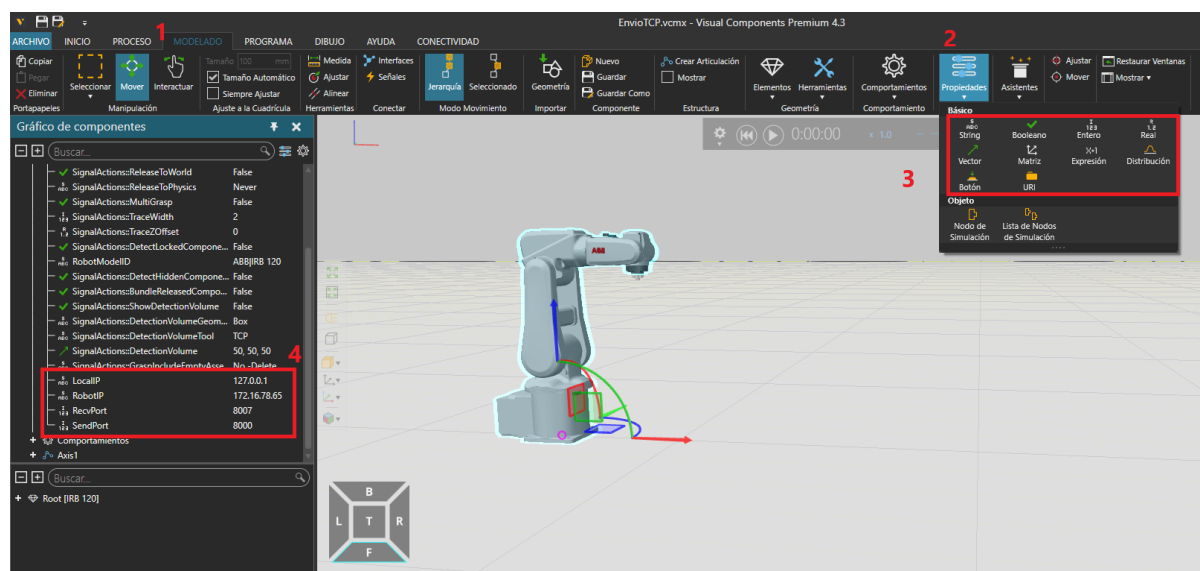


Figura 12 Variables necesarias para el funcionamiento del programa

¹ <https://forum.visualcomponents.com/t/how-to-connect-visual-components-and-robotstudio-for-simulation-commissioning/1270/3>

2.5. Conexión TCP/IP con IRC5 (Real y RobotStudio). Sentido VC → RS.

Una vez logrado el correcto funcionamiento del programa anterior, optamos por desarrollar uno en sentido contrario, es decir, que permitiese controlar el robot real desde Visual Components.

Para ello decidimos crear un sistema que enviase las instrucciones de una en una y que no permitiese a Visual Components continuar con el siguiente movimiento hasta recibir las confirmaciones pertinentes del robot. De esta manera logramos que ambos sistemas mantengan un ritmo similar sin forzar excesivamente al robot.

Este programa se basa en el anterior en lo relativo a la conexión TCP y los gestores de errores, pero está desarrollado desde cero en términos de funcionalidad. Se basa en la obtención de los distintos objetos de la librería de Visual Components que modelan y controlan los diferentes aspectos del programa (Procesos, movimientos, señales, ...) y en su lectura y modificación.

Al igual que el anterior, no está completado y todavía tiene errores y muchas modificaciones posibles, pero consideramos que puede resultar de gran utilidad y que es a su vez un buen ejemplo de como realizar ciertas tareas avanzadas en este programa mediante Python.

El código es el siguiente:

```
from vcScript import *
from vcHelpers.Robot import *
from vcHelpers.Robot2 import *
import socket
import time
import errno
import os
import vcMatrix

local_ip = '127.0.0.1'
robot_ip = '127.0.0.1'
recv_port = 30001
send_port = 30002
SIZE = 250

app = getApplication()
comp = getComponent()
robot = comp
conectado = False
err = "";
robotnum = 1;

def OnStart():
    #Executes before sim clock starts running when play is pressed
    global robot, exe
    global local_ip, robot_ip, recv_port, send_port, my_socket, conectado
    global SIZE, robotnum

    conectado = False
    #Connection vars
    local_ip = comp.getProperty('LocalIP').Value
    robot_ip = comp.getProperty('RobotIP').Value
    recv_port = comp.getProperty('RecvPort').Value
    send_port = comp.getProperty('SendPort').Value
    intentoConexion = 5
```

```

while (intentoConexion > 0):
    print "ROBOT_%i: Intento de conexion n°: %i" %(robotnum,6-
intentoConexion)
    try:
        my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        my_socket.settimeout(1)
        my_socket.connect((robot_ip,recv_port))
        print "ROBOT_%i: Éxito en la conexion" %(robotnum)
        break
    except socket.error, exc:
        print "ROBOT_%i: Error de conexion : %s" %(robotnum,exc)
        if (intentoConexion == 1):
            return
    finally:
        intentoConexion = intentoConexion - 1
if robot:
    exe = robot.findBehavioursByType(VC_ROBOTEXECUTOR)[0]
else:
    exe = None

conectado = True

def OnRun():
    #Executes after sim clock starts running
    global robot, exe, my_socket, conectado, err, robotnum

    #Init receiving socket
    if not conectado:
        print 'ROBOT_%i: No conectado' %(robotnum)
        print 'ROBOT_%i: Cerrando el programa' %(robotnum)
        return
    if not robot:
        print 'ROBOT_%i: No robot associated to TCP link' %(robotnum)
        print 'ROBOT_%i: Cerrando el programa' %(robotnum)
        return

    print ('ROBOT_%i: Receiving packets from local IP {0}, port
{1}\n'.format(local_ip, recv_port)) %(robotnum)
    print ('ROBOT_%i: Sending packets to IP {0}, port
{1}\n'.format(robot_ip, send_port)) %(robotnum)
    delay(0.001)
    try:
        #Send dummy package so robot picks up IP
        my_socket.send('DUMMY ')#, (robot_ip, send_port))
    except:
        pass

    delay(5)
    program = exe.Program
    mainroutine = program.MainRoutine

    EjecucionRutina(mainroutine)

    print('ROBOT_%i: Cerrando el programa') %(robotnum)

```

```

def ParamPos(posMatrix):
    # Funcion encargada de formatear matrices de posicion en strings
    # Parametros de posicion
    P = posMatrix.P
    X = P.X
    Y = P.Y
    Z = P.Z
    # Parametros de orientacion
    cuaternio = posMatrix.getQuaternion()
    QS = cuaternio.X
    Qi = cuaternio.Y
    Qj = cuaternio.Z
    Qk = cuaternio.W
    # Creacion del mensaje
    punto = "X"+"{: .2f}".format(X)
    punto = punto + "Y"+"{: .2f}".format(Y)
    punto = punto + "Z"+"{: .2f}".format(Z)
    punto = punto + "QS"+"{: .2f}".format(QS)
    punto = punto + "Qi"+"{: .2f}".format(Qi)
    punto = punto + "Qj"+"{: .2f}".format(Qj)
    punto = punto + "Qk"+"{: .2f}".format(Qk)

    return punto

def EnvioRobot(data):
    # Funcion encargada de enviar datos al robot
    global robot_ip, send_port
    global my_socket
    data = data + " "
    #print(data)
    my_socket.sendto(data, (robot_ip, send_port))
    pass

def Confirmacion():
    # Funcion encargada de realizar la confirmacion con el robot
    contador = 1
    while True:
        if (contador%100)==0:
            EnvioRobot("CNF")
        try:
            (data, addr) = my_socket.recvfrom(SIZE)
            #print(data)
            continuar = (data == "AccionRealizada")
            if continuar:
                break
        except:
            pass
        delay(0.5)
        contador = contador + 1
    return

def EjecucionRutina(rutina):
    # Funcion encargada de interpretar las rutinas programadas y controlar
    el robot
    global exe, robotnum

    instrucciones = rutina.Statements

```

```

nInstrucciones = len(instrucciones)

# En caso de que la rutina posea variables, las almacenamos para el
correcto
# funcionamiento de las instrucciones while, if,...
try:
    propiedades = rutina.Properties
    for i in range(1,len(propiedades)):
        var = propiedades[i]
        globals()[var.Name] = var.Value
except:
    pass

# Lectura de cada instruccion de la rutina y actuacion acorde
try:
    for i in range(0,nInstrucciones):
        #if inicio == 0 or robotOK == 1:
            instruccion = instrucciones[i]
            tipo = instruccion.Type
            #print(tipo)
            if (tipo == "PtpMotion" or tipo == "LinMotion"):    #Movimiento
                #Obtencion de datos de Wobj
                base = instruccion.Base
                wobj = base.PositionMatrix

                #Obtencion de datos de Posicion Objetivo
                posicion = instruccion.Positions[0]
                posicion = posicion.PositionInReference

                #Creacion de mensajes con la informacion
                wobjData = "WOB"+ParamPos(wobj)
                posicionData = "POS"+ParamPos(posicion)

                if (tipo == "PtpMotion"):
                    movData = "MOVM1" #MoveJ
                else:
                    movData = "MOVM0" #MoveL

                #Envio de mensajes
                EnvioRobot(movData)
                Confirmacion()
                EnvioRobot(wobjData)
                Confirmacion()
                EnvioRobot(posicionData)
                Confirmacion()

            elif (tipo == "SetBin"):                                #Señal
                valor = instruccion.OutputValue
                puerto = instruccion.OutputPort
                #print(instruccion.OutputPort)
                if (puerto == 1):
                    #Si el puerto de la señal es el 1 (herramienta), se indica al
robot
                    sigData = "SIG"+str(valor*1)
                    #print(sigData)
                    EnvioRobot(sigData)

```

```

        elif (tipo == "Call"):                                #Llamada a
subrutina                                                    subrutina
            subrutina = instruccion.Routine
            #print(subrutina)
            EjecucionRutina(subrutina)

        elif (tipo == "Delay"):                                #Retraso
            delay(instruccion.Delay)

        elif (tipo == "While"):                                #Bucle While
            condicion = instruccion.Condition
            scope = instruccion.Scope
            while eval(condicion):
                control = EjecucionRutina(scope)
                if control == "Break":
                    break
                elif control == "Continue":
                    continue

        elif (tipo == "IfElse"):                                #Instruccion
if/else
            condicion = instruccion.Condition
            if eval(condicion):
                EjecucionRutina(instruccion.ThenScope)
            else:
                EjecucionRutina(instruccion.ElseScope)

        elif (tipo == "Break" or tipo == "Continue"):          #Break o
continue (para bucles while)
            return tipo

        # Se ejecuta la instruccion en la simulacion
        exe.callStatement(instruccion,True)

        # Para aquellas instrucciones que se realicen de manera fisica en
        el robot,
        # esperamos una confirmacion
        if ((tipo == "SetBin" and puerto == 1) or tipo == "PtpMotion" or
tipo=="LinMotion"):
            #print "Ejecutando accion"
            Confirmacion()

        # En el caso del resto de señales (todas menos la de la
        herramienta), introducimos
        # un pequeño delay para mejorar el funcionamiento
        elif (tipo == "SetBin"):
            delay(1)

        #Poll interval
        delay(0.01)
        print "Tarea completada"
        return

        # En el caso de que se produzca un error de socket, se procede al
        reinicio del programa
        except socket.error as error:
            if error.errno == 10053:

```

```

print("ROBOT_%i: Error "+str(error.errno)) %(robotnum)
print("ROBOT_%i: Intentando reconectar...") %(robotnum)
OnStart()
else:
    print(os.strerror(error.errno))
    delay(0.01)
return

```

El funcionamiento del código, en resumidas cuentas, se basa en un proceso de conexión inicial, y uno de lectura e interpretación del programa creado para el robot (Figura 13).

Este último se realiza mediante una función a la que se proporciona la rutina main, y que analiza cada instrucción en orden, realizando distintos procesos en función del tipo de rutina que sea. Una vez hecho esto (y siempre que sea necesario), solicita una confirmación al robot para continuar con la siguiente instrucción.

Una explicación más detallada se puede obtener mediante la lectura de los comentarios situados en el código, los cuáles se ha pretendido que sean lo más clarificadores posible.

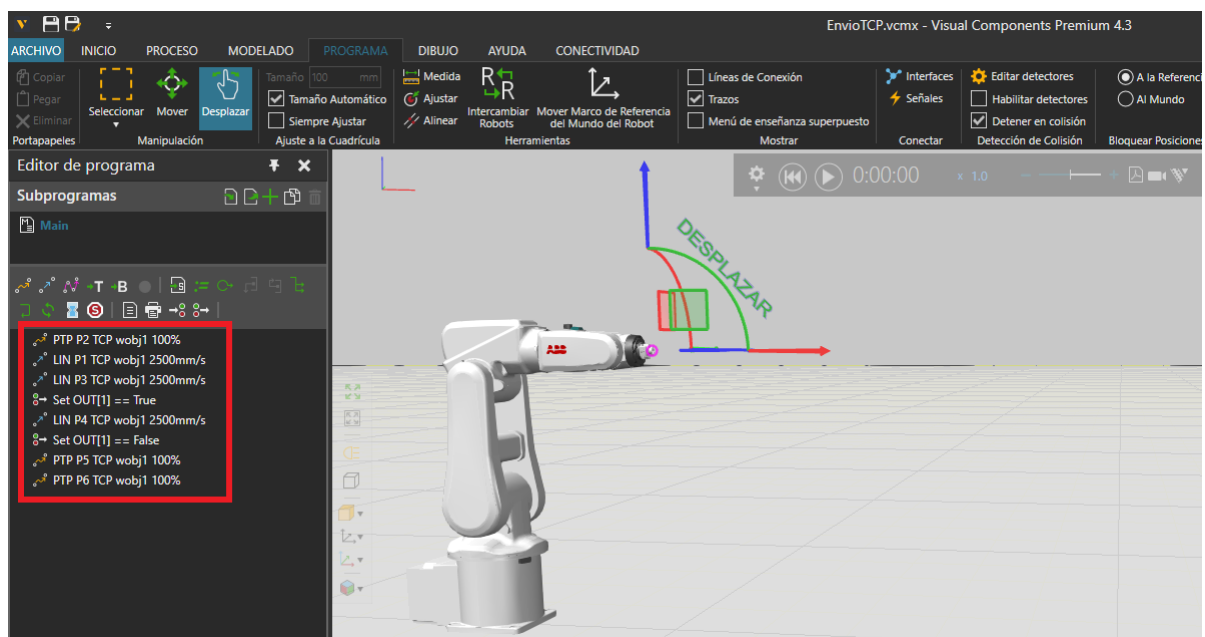


Figura 13 Configuración del programa

Es importante que los puntos que definen la trayectoria a seguir en el programa tengan definidos de manera correcta la herramienta y la base (work object) (Figura 14). Además, la base debe estar definida con respecto al robot (Figura 15).

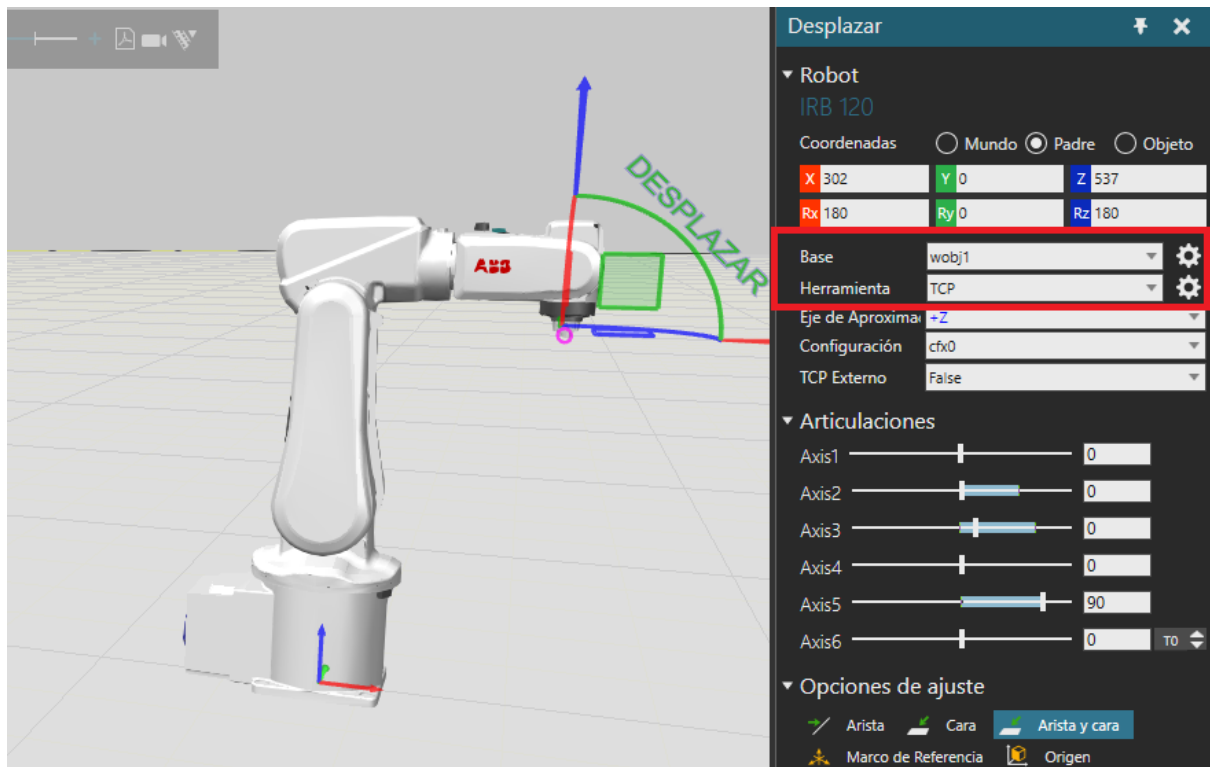


Figura 14 Definición de la base y la herramienta

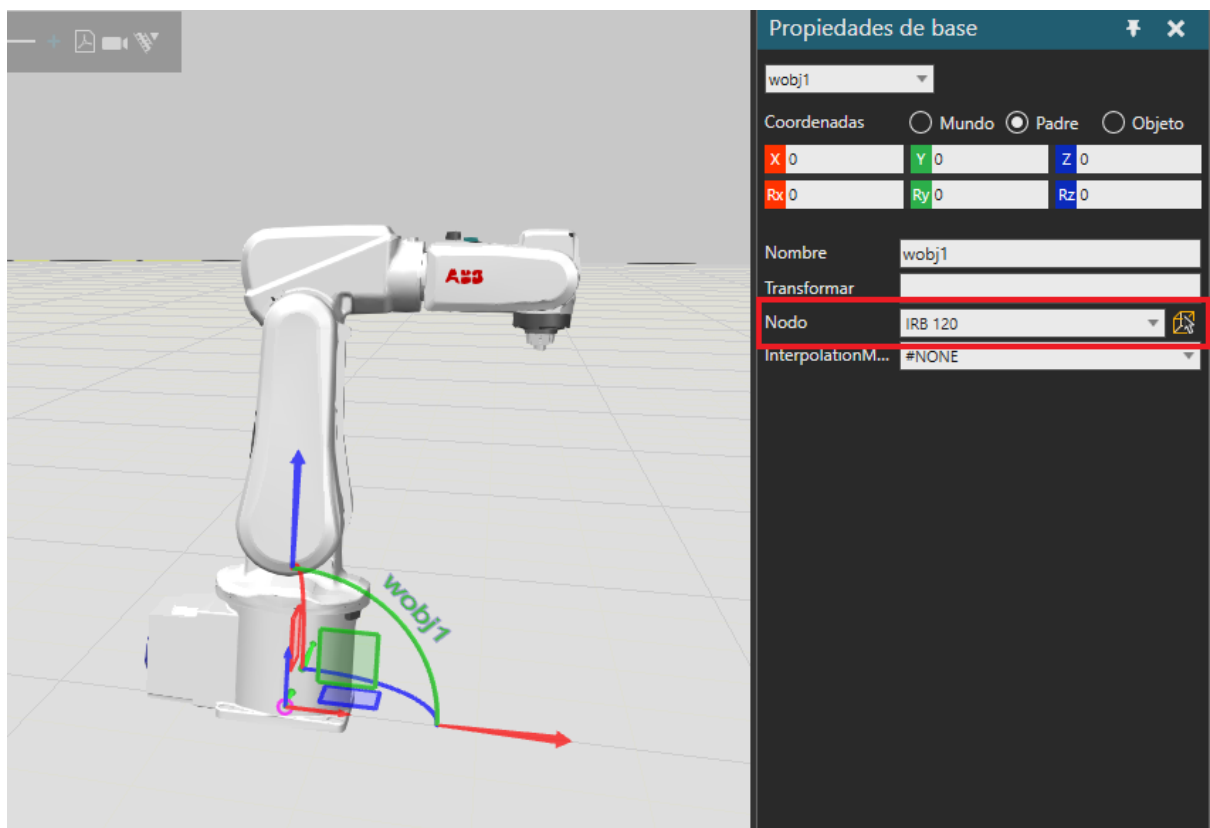


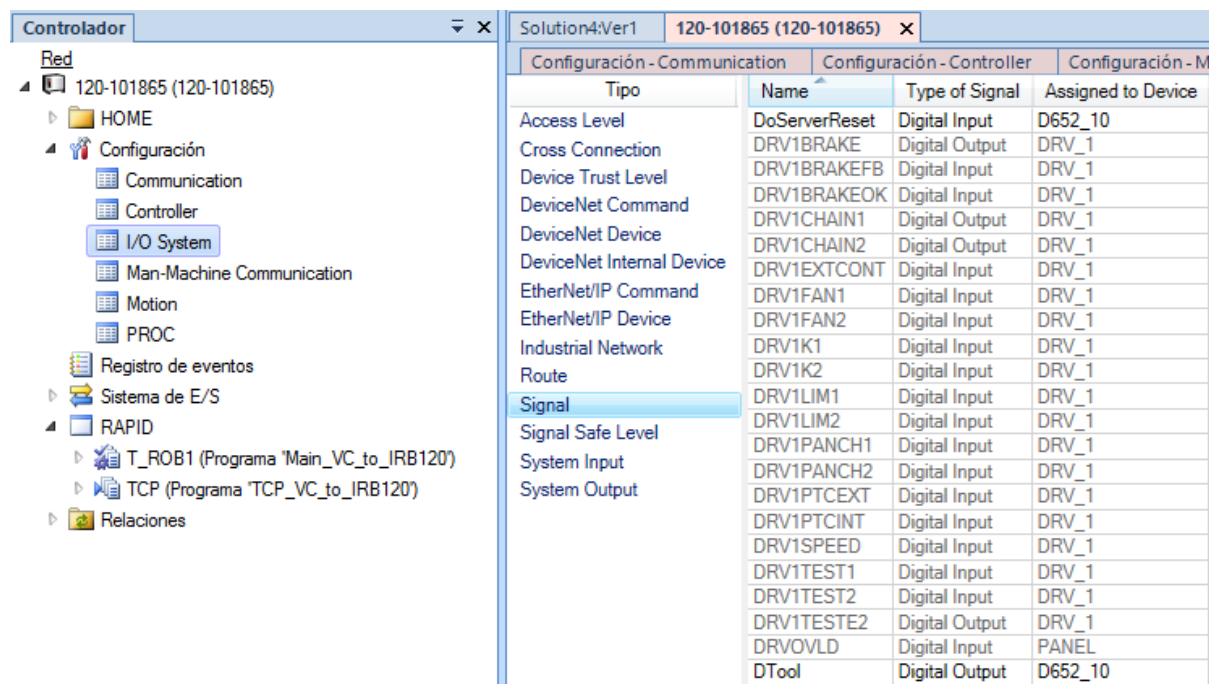
Figura 15 Cambio de referencia de la base

Como mejoras y ampliaciones posibles, proponemos:

- Análisis y respuesta ante otros errores de conexión.
- Mejora de la respuesta ante interrupciones de la conexión (actualmente se reinicia el proceso).
- Análisis y respuesta ante otros tipos de instrucción (rutinas).
- Envío de instrucciones al robot ante el cambio de valor de otras señales (a parte de la de activación de la pinza).
- Aumento de la cantidad de decimales de los mensajes de posición (en caso de ser necesaria una mayor precisión en los movimientos).
- Envío de más información para los movimientos: cambio de herramienta, velocidad de movimiento, configuración (cf1, cf6 y cfx), etc.

3. RobotStudio

Antes de comenzar con los programas desarrollados, es importante destacar que estos hacen referencia a las señales “DTool” y “DoServerReset”, las cuales corresponden con la salida 0 y la entrada 9, respectivamente. Es por ello que deben ser mapeadas correctamente para su correcto funcionamiento (Figura 16).



The screenshot shows the RobotStudio interface with the 'I/O System' configuration window open. The window displays a table mapping various signal types to specific names and devices. The 'Signal' type is selected in the left-hand menu.

Tipo	Name	Type of Signal	Assigned to Device
Access Level	DoServerReset	Digital Input	D652_10
Cross Connection	DRV1BRAKE	Digital Output	DRV_1
Device Trust Level	DRV1BRAKEFB	Digital Input	DRV_1
DeviceNet Command	DRV1BRAKEOK	Digital Input	DRV_1
DeviceNet Device	DRV1CHAIN1	Digital Output	DRV_1
DeviceNet Internal Device	DRV1CHAIN2	Digital Output	DRV_1
EtherNet/IP Command	DRV1EXTCONT	Digital Input	DRV_1
EtherNet/IP Device	DRV1FAN1	Digital Input	DRV_1
	DRV1FAN2	Digital Input	DRV_1
Industrial Network	DRV1K1	Digital Input	DRV_1
Route	DRV1K2	Digital Input	DRV_1
Signal	DRV1LIM1	Digital Input	DRV_1
	DRV1LIM2	Digital Input	DRV_1
Signal Safe Level	DRV1PANCH1	Digital Input	DRV_1
System Input	DRV1PANCH2	Digital Input	DRV_1
System Output	DRV1PTCEXT	Digital Input	DRV_1
	DRV1PTCINT	Digital Input	DRV_1
	DRV1SPEED	Digital Input	DRV_1
	DRV1TEST1	Digital Input	DRV_1
	DRV1TEST2	Digital Input	DRV_1
	DRV1TESTE2	Digital Output	DRV_1
	DRVOVLD	Digital Input	PANEL
	DTool	Digital Output	D652_10

Figura 16 Mapeo de señales en RobotStudio

La parte de los programas desarrollada en RobotStudio debe ejecutarse primero, ya que Visual Components debe conectarse al servidor creado por el robot.

3.1. Conexión TCP/IP con Visual Components. Sentido RS → VC.

La contraparte del programa comentado anteriormente es el script en RAPID de RobotStudio para crear el servidor TCP y publicar en él el estado de cada uno de los ejes del robot.

Lo principal es definir un puerto que sea igual al indicado en Visual Components, ya que no es necesario indicarle su propia IP (Con la ip de localhost es suficiente), y la del cliente la obtiene durante el proceso de conexión.

Para introducirlo dentro del controlador IRC5, una vez hecha la sincronización con RobotStudio, se deben configurar sus tareas para que contenga mínimo una que se ejecute constantemente en segundo plano (tipo “Semistatic”), dentro del apartado “Configuración/Controller/Task” (Figura 17). De esta forma, se puede controlar el movimiento del robot de manera normal mientras este realiza el proceso de conexión y envío de datos simultáneamente.

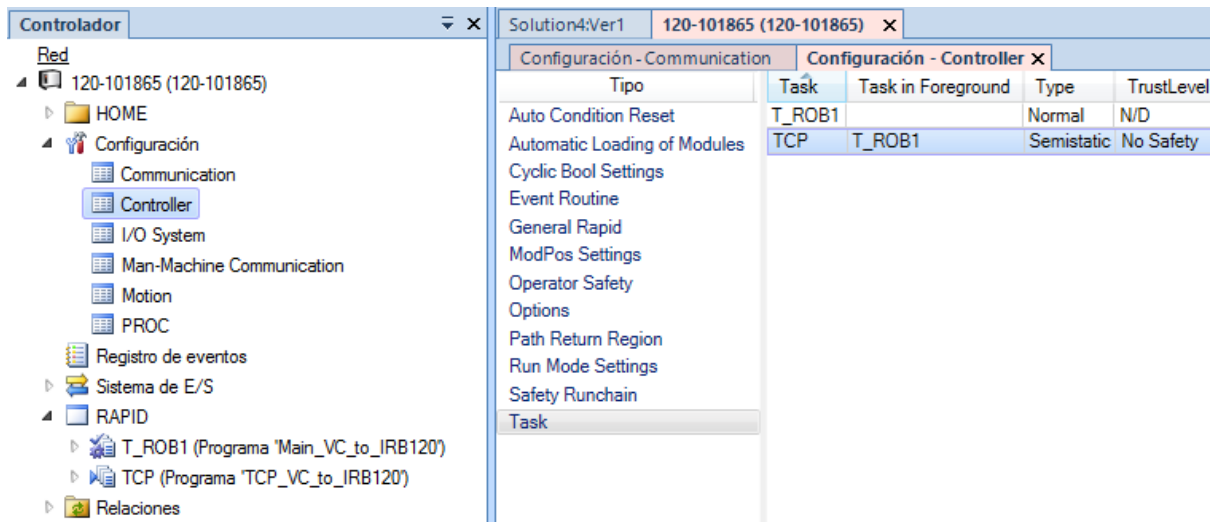


Figura 17 Configuración de tareas del robot

El código de la tarea en segundo plano es el siguiente:

```
MODULE Module1

VAR string data;
VAR socketdev server_socket;
VAR socketdev client_socket;

VAR bool listening := TRUE;

PERS string server_ip := "172.16.78.65";
PERS num send_port := 8007;
VAR socketstatus sockStat;

VAR bool conectado := FALSE;
VAR jointtarget jt;

VAR num ok;
CONST num SERVER_OK := 1;

VAR string recv_data := "";
VAR num tool := 0;
VAR num tool_old := 0;
VAR bool okey;

PROC main()
    !Conexion del socket
    conectado := FALSE;
    ServerCreateAndConnect server_ip, send_port;
    conectado := TRUE;

    !Loop del servidor
```

```

WHILE TRUE DO
    !Obtencion de las posiciones articulares
    GetJointValues;
    WaitTime 0.05;
    sockStat := SocketGetStatus(client_socket);
    IF sockStat = SOCKET_CONNECTED and data <> "" THEN
        !Envio de las posiciones articulares
        SocketSend client_socket \str := data;
        !Obtencion del estado de la herramienta
        GetToolState "DO1",DTool;
        !Envio del estado de la herramienta
        SocketSend client_socket \str := data;
        !Obtención de instruccion de herramienta desde Visual Components
        SocketReceive client_socket,\Str:=recv_data;
        IF StrPart(recv_data,1,4) = "Tool" THEN
            okey := StrToVal(StrPart(recv_data,6,1),tool);
            !Si la instruccion es diferente del estado actual de la herramienta, se cambia su
estado
            IF (tool <> tool_old) and (okey = TRUE) THEN
                TPWrite("Herramienta = "+NumToStr(tool,0));
                tool_old := tool;
                SetDO DTool,tool;
            ENDIF
        ENDIF
    ENDIF
ENDWHILE

!Gestor de errores
ERROR
    IF ERRNO=ERR_SOCK_CLOSED THEN
        TPWrite "SERVIDOR: El cliente ha cerrado la conexion.";
        conectado := FALSE;

        !Cierre del servidor
        SocketClose client_socket;
        SocketClose server_socket;

        !Reinicio del servidor
        ServerCreateAndConnect server_ip, send_port;
        conectado := TRUE;
        !reconnected := TRUE;
        ok := SERVER_OK;
        RETRY;
    ELSEIF ERRNO=ERR_SOCK_TIMEOUT THEN
        RETRY;
    ENDIF
ENDPROC

PROC ServerCreateAndConnect(string ip, num port)

```

!Proceso encargado de realizar la conexión entre Visual Components y el robot mediante TCP

VAR string clientIP;

SocketCreate server_socket;

SocketBind server_socket, ip, port;

SocketListen server_socket;

TPWrite "SERVIDOR: Esperando conexiones entrantes...";

WHILE SocketGetStatus(client_socket) <> SOCKET_CONNECTED DO

 SocketAccept server_socket, client_socket \ClientAddress:=clientIP

\Time:=WAIT_MAX;

 IF SocketGetStatus(client_socket) <> SOCKET_CONNECTED THEN

 TPWrite "SERVIDOR: Problema con conexión entrante.";

 TPWrite "Intentando reconectar.";

 ENDIF

 !Espera 0.5 segundos para intentar reconectar

 WaitTime 0.5;

ENDWHILE

TPWrite "SERVIDOR: Conectado a la IP " + clientIP;

ENDPROC

PROC GetJointValues()

!Proceso encargado de obtener las posiciones articulares y generar el mensaje

VAR jointtarget jt;

jt := CJointT();

data := "JOINTS:";

data := data + NumToStr(jt.robax.rax_1, 3);

data := data + "," + NumToStr(jt.robax.rax_2, 3);

data := data + "," + NumToStr(jt.robax.rax_3, 3);

data := data + "," + NumToStr(jt.robax.rax_4, 3);

data := data + "," + NumToStr(jt.robax.rax_5, 3);

data := data + "," + NumToStr(jt.robax.rax_6, 3);

data := data + " ";

ENDPROC

PROC GetToolState(string id, num value)

!Proceso encargado de obtener el estado de la herramienta y crear el mensaje

data := id + ":" + NumToStr(value, 0) + " ";

ENDPROC

ENDMODULE

3.2. Conexión TCP/IP con Visual Components. Sentido VC → RS.

En esta parte lo que se busca es que el código que se implemente en el robot sea capaz de obtener información procedente desde Visual Component y en función de la información que llegue realizará una acción u otra. Es decir, en este caso se tiene una rutina de ejecución en VC y lo que se va a buscar es que el robot IRB 120 copie las diferentes acciones de dicha rutina.

Para esta parte, al igual que en la anterior, emplearemos la tarea que se ejecuta en segundo plano para establecer la conexión TCP con el cliente de Visual Components, que presenta ciertas modificaciones respecto a la versión utilizada para la conexión en sentido RS → VC; pero además nos hará falta añadir un programa en el módulo principal del robot, que se ejecuta en primer plano. Esto lo hemos hecho con el objetivo de separar en dos partes bien diferenciadas la parte de conexión y recepción y envío de información, y la de interpretación de los datos y actuación sobre la posición y el estado del robot.

IMPORTANTE: el programa puede fallar si no se colocan los punteros de ambas tareas al principio antes de ejecutarlos.

MEJORA POSIBLE: los mensajes enviados mediante TCP a menudo se encadenan, por lo que nuestro programa puede tener problemas para identificarlos. Por ejemplo, hemos colocado un delay en el código de Visual Components entre el mensaje “DUMMY “ para conexión y los relativos a las instrucciones, ya que en algunos casos el mensaje reconocido por el robot era “DUMMY MOV1” (por ejemplo). Una mejora posible es añadir cierta lógica a la tarea TCP del robot para que pueda separar adecuadamente los mensajes por los espacios situados entre ellos y escoger el adecuado.

3.2.1. Tarea Main.

El programa desarrollado para esta tarea se encarga de procesar los mensajes que se reciben por TCP para poder copiar lo que se esté haciendo desde VC.

Es importante establecer una conexión entre esta tarea principal y la tarea secundaria de TCP. Para ello hacemos uso de variables comunes para ambas tareas. Estas variables son del tipo persistente: PERS, y es necesario que estén definidas en ambos programas e inicializadas únicamente en la tarea donde se irán modificando.

Esta tarea se encarga de acceder al mensaje que se recibe desde el servidor de TCP y procesarlo. Primero compara si dicho mensaje es el mismo que le había llegado anteriormente (con una variable auxiliar). En caso de que sea un mensaje nuevo, pasa a clasificarlo. Obtendrá los tres primeros caracteres del mensaje, los cuales conforman un identificador del tipo de instrucción:

- MOV: instrucción de movimiento.
- SGN: modificación del valor de una señal, como la de la ventosa o la pinza.
- CFR: es necesario reenviar el mensaje de confirmación.

En caso de ser un mensaje con identificador tipo MOV, el programa procederá a la escucha del resto de datos: el work object con un mensaje de identificador WOB y la posición con respecto al work object con identificador POS. Una vez recibidos estos tres mensajes, uno de tipo SGN o uno de tipo CFR, realiza la acción asociada.

El mensaje MOV tiene formato MOV1/0. En caso de ser un 0 nos indicará que es un movimiento lineal, si es un 1 será un movimiento tipo joint.

El formato del mensaje WOB es "WOBX__Y__Z__Qi__QJ__Qk__Qs__ ". De forma que recibirá el valor de la posición y de la orientación en cuaternios, teniendo que desglosar dicho mensaje y guardar los valores en las propiedades del wob del robot.

Para el mensaje de POS el formato es el mismo y el procedimiento también, solo cambia el identificador del mensaje.

Con estos tres datos ya podemos realizar el movimiento, fijando de antemano la velocidad de movimiento y valores como la configuración de las articulaciones.

Para la señal, se repite el procedimiento del programa anterior. Se ha diseñado para activar o desactivar la señal del efector final, pero se puede ampliar para cuando haya más señales sencillamente variando la respuesta dependiendo del número que acompaña al identificador.

IMPORTANTE: cómo se puede leer en los comentarios, durante el desarrollo de este programa observamos que el programa no avanzaba correctamente al siguiente paso cuando llegaba un nuevo mensaje si no situábamos un TPWrite o TPErase en el bucle de lectura de la variable "recv_data". Desconocemos el motivo de este comportamiento.

El código de la tarea es el siguiente:

```
MODULE Module1
  VAR robtarget Objetivo:=
[[0,0,0],[0,0,0,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  TASK PERS wobjdata
obj_trab:=[FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

  PERS string recv_data;
  VAR string msg:="";
  VAR string old_msg;

  VAR string ident:="";

  VAR num inicio;
  VAR num final;
  VAR num total;
  VAR num string_len:=0;
  VAR bool okey;

  VAR num tool := 0;
  VAR num tool_old := 0;

  VAR num tipo_mov:=0;

  PERS bool confirmacion :=FALSE;
  PERS bool confirmacion2;
  PERS num control :=0;

  PROC main()
    !Inicializacion de variables
    !IMPORTANTE: Iniciar ambas tareas con el puntero en main cada vez que se use el
programa
```

```

confirmacion := FALSE;
old_msg := "";
WaitTime 2;
WHILE TRUE DO
    msg := recv_data;

    !TPwrite "Main";
    !TPWrite old_msg;
    TPErase; !Muy importante añadir un TPWrite o un TPErase, en caso de no incluirlo
no se ejecutará.
    WaitTime(0.1);

    !Si llega un mensaje nuevo de mas de tres caracteres:
    IF (msg <> old_msg) and (StrLen(msg)>3) THEN

        TPWrite msg;
        WaitTime(0.01);

        !Extraemos el identificador, el cual define el tipo de instruccion
        ident := StrPart(msg,1,3);

        IF ident = "MOV" THEN      !Instruccion de movimiento

            string_len := StrLen(msg);
            total := string_len-7-1;

            !Se almacena el tipo de movimiento (MoveJ o MoveL) en la variable
tipo_mov
            okey := StrToVal(StrPart(msg,5,1),tipo_mov);

            !Se confirma al programa que ha llegado el mensaje
            Proc_confirmacion;
            !Se inicia el proceso "Movimiento"
            Movimiento;

        ELSEIF ident = "SIG" THEN    !Instruccion de señal
            !De momento solo sirve para activar o desactivar la herramienta, pero es
            !facilmente ampliable para su uso con otras salidas

            !Se extrae del mensaje el valor al que se debe poner la salida de la herramienta
            !y, en caso de ser distinto al anterior, se cambia su valor
            okey := StrToVal(StrPart(msg,4,1),tool);
            IF (tool <> tool_old) and (okey = TRUE) THEN
                TPWrite("Herramienta = "+NumToStr(tool,0));
                WaitTime(0.01);
                tool_old := tool;
                SetDO DTool,tool;
            ENDIF

```

```

        !Se confirma al programa que se ha realizado la accion pertinente
        Proc_confirmacion;

        ELSEIF ident = "CNF" THEN      !Programa a la espera de un mensaje de
confirmacion
        !Se envia un mensaje de confirmacion
        Proc_confirmacion;
        ENDIF
        !Se guarda el ultimo mensaje recibido como tal
        old_msg:=msg;
    ENDIF
ENDWHILE
ENDPROC

PROC Movimiento()
    VAR num control :=0;    !Variable que controla el bucle while

    WHILE control = 0 DO
        msg := recv_data;
        WaitTime(0.1);

        !En el momento en el que se reciba una nueva instruccion, se comprueba su tipo y
se
        !actua de manera acorde.

        !Se asume que los mensajes recibidos llegan en forma de secuencia "MOV - WOB
- POS"
        !Para mejorar el funcionamiento del programa deberia ampliarse para cuando se
produzcan
        !anomalias y se incumpla esta condicion

        IF msg <> old_msg THEN
            ident := StrPart(msg,1,3);
            TPWrite ident;
            WaitTime(0.01);
            IF ident = "WOB" THEN
                !Cuando se recibe un mensaje con identificado WOB, se guardan sus valores
                !en el objeto de trabajo "obj_trab"
                inicio := StrMatch(msg,1,"X")+1;
                final := StrMatch(msg,1,"Y");
                total := final-inicio;
                okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.trans.x);

                inicio := StrMatch(msg,1,"Y")+1;
                final := StrMatch(msg,1,"Z");
                total := final-inicio;
                okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.trans.y);

                inicio := StrMatch(msg,1,"Z")+1;

```



```

final := StrMatch(msg,1,"QS");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.trans.z);

```

```

inicio := StrMatch(msg,1,"QS")+2;
final := StrMatch(msg,1,"Qi");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.rot.q1);

```

```

inicio := StrMatch(msg,1,"Qi")+2;
final := StrMatch(msg,1,"Qj");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.rot.q2);

```

```

inicio := StrMatch(msg,1,"Qj")+2;
final := StrMatch(msg,1,"Qk");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.rot.q3);

```

```

inicio := StrMatch(msg,1,"Qk")+2;
string_len := StrLen(msg);
total := string_len-inicio-1;
okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.rot.q4);

```

!Se confirma al programa que se ha realizado la accion pertinente
Proc_confirmacion;

ELSEIF ident = "POS" THEN

!Cuando se recibe un mensaje de tipo POS, se guardan sus valores en la
!variable de posicion "Objetivo"

```

inicio := StrMatch(msg,1,"X")+1;
final := StrMatch(msg,1,"Y");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.trans.x);
!TPWrite StrPart(msg,inicio,total);

```

```

inicio := StrMatch(msg,1,"Y")+1;
final := StrMatch(msg,1,"Z");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.trans.y);
!TPWrite StrPart(msg,inicio,total);

```

```

inicio := StrMatch(msg,1,"Z")+1;
final := StrMatch(msg,1,"QS");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.trans.z);
!TPWrite StrPart(msg,inicio,total);

```

```

inicio := StrMatch(msg,1,"QS")+2;
final := StrMatch(msg,1,"Qi");

```

```

total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.rot.q1);
!TPWrite StrPart(msg,inicio,total);

inicio := StrMatch(msg,1,"Qi")+2;
final := StrMatch(msg,1,"Qj");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.rot.q2);
!TPWrite StrPart(msg,inicio,total);

inicio := StrMatch(msg,1,"Qj")+2;
final := StrMatch(msg,1,"Qk");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.rot.q3);
!TPWrite StrPart(msg,inicio,total);

inicio := StrMatch(msg,1,"Qk")+2;
string_len := StrLen(msg);
total := string_len-inicio-1;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.rot.q4);
!TPWrite StrPart(msg,inicio,total);

!Se actualiza la variable "control" para salir del bucle
control:=1;

!Se confirma al programa que se ha realizado la accion pertinente
Proc_confirmacion;
ENDIF
!Se guarda el ultimo mensaje recibido como tal
old_msg := msg;
ENDIF
ENDWHILE

!Se manda la instruccion al robot de moverse en función del tipo de
movimiento(MOV),
!objeto de trabajo (WOB) y posicion (POS)

!Una forma de ampliar el programa seria añadir funcionalidades como el cambio de
herramienta,
!la configuracion (cf1,cfx,...) o la velocidad de movimiento

IF tipo_mov=0 THEN
MoveL Objetivo,v50,fine,TVentosa \WObj:=obj_trab;
ELSE
MoveJ Objetivo,v50,fine,TVentosa \WObj:=obj_trab;
ENDIF

!Se confirma al programa que se ha realizado la accion pertinente
Proc_confirmacion;
ENDPROC

```

```

PROC Proc_confirmacion()
    !Este proceso indica a la tarea de segundo plano que debe enviar una confirmacion al
    programa
    !a traves de la variable "confirmacion".
    !Una vez sea enviada, la tarea de segundo plano indica a esta que se ha realizado a
    través de la
    !variable "confirmacion2".
    !Este comportamiento es posible gracias al empleo de variables tipo PERS.

    confirmacion := TRUE;
    WaitTime(0.1);
    WHILE confirmacion = TRUE DO
        IF confirmacion2 = TRUE THEN
            confirmacion:=FALSE;
            TPWrite "Confirmacion completada";
            WaitTime(0.5);
        ENDIF
    ENDWHILE
ENDPROC
ENDMODULE

```

3.2.2. Tarea TCP.

Como se ha comentado anteriormente, esta tarea se encarga de realizar la conexión y mandar y recibir datos a través de los sockets TCP. Es importante que se configure para funcionar en segundo plano (Semistatic).

IMPORTANTE: Hemos notado que imprimir por pantalla (TPWrite) en ambas tareas puede provocar que el programa se estanque y no funcione correctamente, por lo que recomendamos evitar su uso en esta tarea.

El código de la tarea es el siguiente:

```

MODULE Module1

    VAR string data;
    VAR socketdev server_socket;
    VAR socketdev client_socket;
    VAR string recibir:="";

    PERS string server_ip := "172.16.78.65";
    PERS num send_port := 8007;
    VAR socketstatus sockStat;

    VAR bool conectado := FALSE;
    VAR jointtarget jt;

    VAR num ok;
    CONST num SERVER_BAD_MSG := 0;
    CONST num SERVER_OK := 1;

```

```

PERS string recv_data:="";
VAR num tool := 0;
VAR num tool_old := 0;
VAR intnum siglint;
VAR intnum conf_interrup;
PERS bool confirmacion;
PERS bool confirmacion2:=FALSE;

PROC main()

    !Conexion del socket e inicializacion de variables
    conectado := FALSE;
    recv_data := "";
    ServerCreateAndConnect server_ip, send_port;
    conectado := TRUE;
    confirmacion2 := FALSE;

    CONNECT siglint WITH reset_routine;
    !Interrupcion asociada a la activacion de la señal de entrada DoServerReset (input 9),
    !encargada de reiniciar los sockets
    ISignalDI DoServerReset,1,siglint;

    CONNECT conf_interrup WITH conf_routine;
    !Interrupcion asociada a la puesta a True de la variable persistente confirmacion,
    !encargada de iniciar el proceso de envio de confirmacion al programa
    IPers confirmacion,conf_interrup;

    WHILE TRUE DO
        sockStat := SocketGetStatus(client_socket);
        IF sockStat = SOCKET_CONNECTED THEN
            !Si la conexion con el cliente es correcta, se procede a escuchar lo que este envie
            SocketReceive client_socket,\Str:=recibir \Time:=30;
            IF recibir <> "DUMMY " THEN
                !Se guardan los datos recibidos en la variable recv_data, siempre que no sean
                !la palabra "DUMMY ", empleada para establecer la conexion cliente-servidor
                recv_data := recibir;
            ENDIF
        ENDIF
    ENDWHILE

    !Gestor de errores
    ERROR
    IF ERRNO=ERR_SOCK_CLOSED THEN
        TPWrite "SERVIDOR: El cliente ha cerrado la conexion.";
        conectado := FALSE;
        WaitTime(0.01);

        !Cierre del servidor
        SocketClose client_socket;

```

```

SocketClose server_socket;

!Reinicio del servidor
ServerCreateAndConnect server_ip, send_port;
conectado := TRUE;
ok := SERVER_OK;
RETRY;
ELSEIF ERRNO=ERR_SOCK_TIMEOUT THEN
  TPWrite "SERVIDOR: Se ha sobrepasado el tiempo máximo de espera";
  conectado := FALSE;
  WaitTime(0.01);

!Cierre del servidor
SocketClose client_socket;
SocketClose server_socket;

!Reinicio del servidor
ServerCreateAndConnect server_ip, send_port;
conectado := TRUE;
!reconnected := TRUE;
ok := SERVER_OK;
RETRY;
ENDIF
recv_data := " ";
ENDPROC

PROC ServerCreateAndConnect(string ip, num port)
!Funcion encargada de crear el servidor y establecer la conexion con el cliente
VAR string clientIP;

SocketCreate server_socket;
SocketBind server_socket, ip, port;
SocketListen server_socket;
TPWrite "SERVIDOR: Esperando conexiones entrantes...";
WaitTime(0.01);
WHILE SocketGetStatus(client_socket) <> SOCKET_CONNECTED DO
  SocketAccept server_socket, client_socket \ClientAddress:=clientIP
\Time:=WAIT_MAX;
  IF SocketGetStatus(client_socket) <> SOCKET_CONNECTED THEN
    TPWrite "SERVIDOR: Problema con conexión entrante.";
    WaitTime(0.01);
    TPWrite "Intentando reconectar.";
    WaitTime(0.01);
  ENDIF
  !Espera 0.5 segundos para intentar reconectar
  WaitTime 0.5;
ENDWHILE
!TPWrite "SERVIDOR: Conectado a la IP " + clientIP;
ENDPROC

```

```

TRAP reset_routine
    !Rutina encargada de reiniciar el servidor mediante la activacion y desactivacion
    !de la señal "DoServerReset", asignada a la entrada 9
    TPWrite "SERVIDOR: Reinicio de Servidor solicitado";
    WaitTime(0.01);
    WaitDI DoServerReset, 0;

    conectado := FALSE;

    !Cierre del servidor
    SocketClose client_socket;
    SocketClose server_socket;

    !Reinicio del servidor
    ServerCreateAndConnect server_ip, send_port;
    conectado := TRUE;
    !reconnected := TRUE;
    ok := SERVER_OK;

    TPWrite "SERVIDOR: Reinicio completado";
    WaitTime(0.01);
    ERROR
    RAISE;
ENDTRAP

TRAP conf_routine
    !Rutina encargada de enviar la confirmacion al programa
    IF confirmacion = TRUE THEN
        SocketSend client_socket \str := "AccionRealizada";
        WaitTime(0.05);
        confirmacion2 := TRUE;
        WHILE confirmacion = TRUE DO
            WaitTime(0.01);
        ENDWHILE
    ENDIF
    confirmacion2 := FALSE;
ENDTRAP
ENDMODULE

```

4. Versión 2 del programa VC -> RS

Este último apartado hemos decidido dedicarlo a una versión algo más avanzada del programa de control del robot mediante procesos generados en Visual Components. El motivo por el cuál tiene su propia sección en lugar de sustituir a la primera versión es que no se ha podido comprobar en profundidad su funcionalidad y comportamiento ante errores.

El principal objetivo buscado en esta modificación es evitar los errores derivados de la asunción de que los mensajes siempre llegan correctamente, separados y en orden.

Las modificaciones llevadas a cabo son:

- Introducción del mensaje "RTY" tras un tiempo sin confirmación que provoca que Visual Components reintente el envío de una instrucción y RobotStudio se prepare para ello.
- Empleo de un algoritmo que permita separar los mensajes recibidos por los espacios (" ") y quedarse con el último de ellos. Esto es necesario debido a que, en ocasiones, los mensajes recibidos por TCP se encadenan uno detrás de otro en un mismo string, lo que impide al programa identificar correctamente el significado de las instrucciones recibidas.

4.1. Código de Visual Components

```
from vcScript import *
from vcHelpers.Robot import *
from vcHelpers.Robot2 import *
import socket
import time
import errno
import os
import vcMatrix

local_ip = '127.0.0.1'
robot_ip = '127.0.0.1'
recv_port = 30001
send_port = 30002
SIZE = 250

app = getApplication()
comp = getComponent()
robot = comp
conectado = False
err = "";
robotnum = 1;

def OnStart():
    #Executes before sim clock starts running when play is pressed
    global robot, exe
    global local_ip, robot_ip, recv_port, send_port, my_socket, conectado
    global SIZE, robotnum

    conectado = False
    #Connection vars
    local_ip = comp.getProperty('LocalIP').Value
    robot_ip = comp.getProperty('RobotIP').Value
```

```

recv_port = comp.getProperty('RecvPort').Value
send_port = comp.getProperty('SendPort').Value
intentoConexion = 5

while (intentoConexion > 0):
    print "ROBOT_%i: Intento de conexion n°: %i" %(robotnum, 6-
intentoConexion)
    try:
        my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        my_socket.settimeout(1)
        my_socket.connect((robot_ip, recv_port))
        print "ROBOT_%i: Éxito en la conexion" %(robotnum)
        break
    except socket.error, exc:
        print "ROBOT_%i: Error de conexion : %s" %(robotnum, exc)
        if (intentoConexion == 1):
            return
        finally:
            intentoConexion = intentoConexion - 1
if robot:
    exe = robot.findBehavioursByType(VC_ROBOTEXECUTOR)[0]
else:
    exe = None

conectado = True

def OnRun():
    #Executes after sim clock starts running
    global robot, exe, my_socket, conectado, err, robotnum

    #Init receiving socket
    if not conectado:
        print 'ROBOT_%i: No conectado' %(robotnum)
        print 'ROBOT_%i: Cerrando el programa' %(robotnum)
        return
    if not robot:
        print 'ROBOT_%i: No robot associated to TCP link' %(robotnum)
        print 'ROBOT_%i: Cerrando el programa' %(robotnum)
        return

    print ('ROBOT_%i: Receiving packets from local IP {0}, port
{1}\n'.format(local_ip, recv_port)) %(robotnum)
    print ('ROBOT_%i: Sending packets to IP {0}, port
{1}\n'.format(robot_ip, send_port)) %(robotnum)
    delay(0.001)
    try:
        #Send dummy package so robot picks up IP
        my_socket.send('DUMMY ')#, (robot_ip, send_port))
    except:
        pass

    #delay(5)
    program = exe.Program
    mainroutine = program.MainRoutine

    EjecucionRutina(mainroutine)

```



```

    print('ROBOT_%i: Cerrando el programa') %(robotnum)

def ParamPos(posMatrix):
    # Funcion encargada de formatear matrices de posicion en strings
    # Parametros de posicion
    P = posMatrix.P
    X = P.X
    Y = P.Y
    Z = P.Z
    # Parametros de orientacion
    cuaternio = posMatrix.getQuaternion()
    QS = cuaternio.X
    Qi = cuaternio.Y
    Qj = cuaternio.Z
    Qk = cuaternio.W
    # Creacion del mensaje
    punto = "X"+"{: .2f}".format(X)
    punto = punto + "Y"+"{: .2f}".format(Y)
    punto = punto + "Z"+"{: .2f}".format(Z)
    punto = punto + "QS"+"{: .2f}".format(QS)
    punto = punto + "Qi"+"{: .2f}".format(Qi)
    punto = punto + "Qj"+"{: .2f}".format(Qj)
    punto = punto + "Qk"+"{: .2f}".format(Qk)

    return punto

def EnvioRobot(data):
    # Funcion encargada de enviar datos al robot
    global robot_ip, send_port
    global my_socket
    data = data + " "
    #print(data)
    my_socket.sendto(data, (robot_ip, send_port))
    pass

def Confirmacion():
    # Funcion encargada de realizar la confirmacion con el robot
    contador = 1
    while True:
        if (contador%10)==0:
            EnvioRobot("RTY")
            print("RTY")
            return False
        elif (contador%5)==0:
            EnvioRobot("CNF")
        try:
            (data, addr) = my_socket.recvfrom(SIZE)
            #print(data)
            continuar = (data == "AccionRealizada")
            if continuar:
                break
        except:
            pass
        delay(0.5)
        contador = contador + 1
    return True

```

```

def EjecucionRutina(rutina):
    # Funcion encargada de interpretar las rutinas programadas y controlar
    # el robot
    global exe, robotnum

    instrucciones = rutina.Statements
    nInstrucciones = len(instrucciones)

    # En caso de que la rutina posea variables, las almacenamos para el
    # correcto
    # funcionamiento de las instrucciones while, if,...
    try:
        propiedades = rutina.Properties
        for i in range(1, len(propiedades)):
            var = propiedades[i]
            globals()[var.Name] = var.Value
    except:
        pass

    # Lectura de cada instruccion de la rutina y actuacion acorde
    i = 0
    for i in range(i, nInstrucciones):
        # Bucle while que permite reintentar una instruccion si falla
        # (mediante continue)
        while True:
            try:
                instruccion = instrucciones[i]
                tipo = instruccion.Type
                #print(tipo)
                if (tipo == "PtpMotion" or tipo == "LinMotion"): #Movimiento
                    #Obtencion de datos de Wobj
                    base = instruccion.Base
                    wobj = base.PositionMatrix

                    #Obtencion de datos de Posicion Objetivo
                    posicion = instruccion.Positions[0]
                    posicion = posicion.PositionInReference

                    #Creacion de mensajes con la informacion
                    wobjData = "WOB"+ParamPos(wobj)
                    posicionData = "POS"+ParamPos(posicion)

                    if (tipo == "PtpMotion"):
                        movData = "MOV1" #MoveJ
                    else:
                        movData = "MOV0" #MoveL
                    #print(tipo)
                    #Envio de mensajes
                    EnvioRobot(movData)
                    if not(Confirmacion()): # Si no se realiza la confirmacion, se
reintenta
                        delay(1)
                        continue
                    EnvioRobot(wobjData)
                    if not(Confirmacion()):
                        delay(1)

```

```

        continue
    EnvioRobot (posicionData)
    if not (Confirmacion()):
        delay(1)
        continue
    elif (tipo == "SetBin"):
        valor = instruccion.OutputValue
        puerto = instruccion.OutputPort
        #print(instruccion.OutputPort)
        if (puerto == 1):
            #Si el puerto de la señal es el 1 (herramienta), se indica al
robot
            sigData = "SIG"+str(valor*1)
            #print(sigData)
            EnvioRobot(sigData)

    elif (tipo == "Call"):
        subrutina = instruccion.Routine
        #print(subrutina)
        EjecucionRutina(subrutina)

    elif (tipo == "Delay"):
        delay(instruccion.Delay)

    elif (tipo == "While"):
        condicion = instruccion.Condition
        scope = instruccion.Scope
        while eval(condicion):
            control = EjecucionRutina(scope)
            if control == "Break":
                break
            elif control == "Continue":
                continue

    elif (tipo == "IfElse"):
        condicion = instruccion.Condition
        if eval(condicion):
            EjecucionRutina(instruccion.ThenScope)
        else:
            EjecucionRutina(instruccion.ElseScope)

    elif (tipo == "Break" or tipo == "Continue"):
        return tipo

    # Se ejecuta la instruccion en la simulacion
    exe.callStatement(instruccion,True)

    # Para aquellas instrucciones que se realicen de manera fisica en
    el robot,
    # esperamos una confirmacion
    if ((tipo == "SetBin" and puerto == 1) or tipo == "PtpMotion" or
    tipo=="LinMotion"):
        #print "Ejecutando accion"
        if not (Confirmacion()):

```

```

        delay(1)
        continue

    # En el caso del resto de señales (todas menos la de la
herramienta), introducimos
    # un pequeño delay para mejorar el funcionamiento
    elif (tipo == "SetBin"):
        delay(1)

    # En el caso de que se produzca un error de socket, se procede al
reinicio del programa
    except socket.error as error:
        if error.errno == 10053:
            print("ROBOT_%i: Error "+str(error.errno)) %(robotnum)
            print("ROBOT_%i: Intentando reconectar...") %(robotnum)
            OnStart() # reconexion
            continue # se reintenta la rutina
        else:
            print(os.strerror(error.errno))
            print("Cancelando operacion")
            return # A falta de un proceso de correccion de errores más
depurado, cerramos el programa
        delay(0.01)
        break

#Poll interval
delay(0.01)
print "Tarea completada"
return

```

4.2. Código de RobotStudio (main)

```
MODULE Module1
  VAR robtarget Objetivo:=
[[0,0,0],[0,0,0,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
  TASK PERS wobjdata
obj_trab:=[FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

  PERS string recv_data;
  VAR string msg:="";
  VAR string old_msg;

  VAR string ident:="";

  VAR num inicio;
  VAR num final;
  VAR num total;
  VAR num string_len:=0;
  VAR bool okey;

  VAR num tool := 0;
  VAR num tool_old := 0;

  VAR num tipo_mov:=0;

  PERS bool confirmacion :=FALSE;
  PERS bool confirmacion2;
  PERS num control :=0;

  PROC main()
    !Inicializacion de variables
    !IMPORTANTE: Iniciar ambas tareas con el puntero en main cada vez que se use el
programa
    confirmacion := FALSE;
    old_msg := "";
    WaitTime 2;
    WHILE TRUE DO
      msg := recv_data;

      !TPwrite "Main";
      !TPWrite old_msg;
      TPErase; !Muy importante añadir un TPWrite o un TPErase, en caso de no incluirlo
no se ejecutará.
      WaitTime(0.1);

      !Si llega un mensaje nuevo de mas de tres caracteres:
      IF (msg <> old_msg) and (StrLen(msg)>3) THEN

        TPWrite msg;
        WaitTime(0.01);
```

```

!Extraemos el identificador, el cual define el tipo de instruccion
ident := StrPart(msg,1,3);

IF ident = "MOV" THEN      !Instruccion de movimiento

    string_len := StrLen(msg);
    total := string_len-7-1;

    !Se almacena el tipo de movimiento (MoveJ o MoveL) en la variable
tipo_mov
    okey := StrToVal(StrPart(msg,5,1),tipo_mov);

    !Se confirma al programa que ha llegado el mensaje
    Proc_confirmacion;
    !Se inicia el proceso "Movimiento"
    Movimiento;

ELSEIF ident = "SIG" THEN    !Instruccion de señal
    !De momento solo sirve para activar o desactivar la herramienta, pero es
    !facilmente ampliable para su uso con otras salidas

    !Se extrae del mensaje el valor al que se debe poner la salida de la herramienta
    !y, en caso de ser distinto al anterior, se cambia su valor
    okey := StrToVal(StrPart(msg,4,1),tool);
    IF (tool <> tool_old) and (okey = TRUE) THEN
        TPWrite("Herramienta = "+NumToStr(tool,0));
        WaitTime(0.01);
        tool_old := tool;
        SetDO DTool,tool;
    ENDIF

    !Se confirma al programa que se ha realizado la accion pertinente
    Proc_confirmacion;

ELSEIF ident = "CNF" THEN    !Programa a la espera de un mensaje de
confirmacion
    !Se envia un mensaje de confirmacion
    Proc_confirmacion;
ENDIF
!Se guarda el ultimo mensaje recibido como tal
old_msg:=msg;
ENDIF
ENDWHILE
ENDPROC

PROC Movimiento()
    VAR num control :=0;    !Variable que controla el bucle while

```

```

WHILE control = 0 DO
    msg := recv_data;
    WaitTime(0.1);

    !En el momento en el que se reciba una nueva instruccion, se comprueba su tipo y
se
    !actua de manera acorde.

    !Se asume que los mensajes recibidos llegan en forma de secuencia "MOV - WOB
- POS"
    !Para mejorar el funcionamiento del programa deberia ampliarse para cuando se
produzcan
    !anomalias y se incumpla esta condicion

IF msg <> old_msg THEN
    ident := StrPart(msg,1,3);
    TPWrite ident;
    WaitTime(0.01);
    IF ident = "RTY" THEN
        !Mensaje de retry, indica que Visual Components va a reintentar el envio de la
!instrucción desde el principio
        RETURN;
    ELSEIF ident = "WOB" THEN
        !Cuando se recibe un mensaje con identificado WOB, se guardan sus valores
!en el objeto de trabajo "obj_trab"
        inicio := StrMatch(msg,1,"X")+1;
        final := StrMatch(msg,1,"Y");
        total := final-inicio;
        okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.trans.x);

        inicio := StrMatch(msg,1,"Y")+1;
        final := StrMatch(msg,1,"Z");
        total := final-inicio;
        okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.trans.y);

        inicio := StrMatch(msg,1,"Z")+1;
        final := StrMatch(msg,1,"QS");
        total := final-inicio;
        okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.trans.z);

        inicio := StrMatch(msg,1,"QS")+2;
        final := StrMatch(msg,1,"Qi");
        total := final-inicio;
        okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.rot.q1);

        inicio := StrMatch(msg,1,"Qi")+2;
        final := StrMatch(msg,1,"Qj");
        total := final-inicio;
        okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.rot.q2);

```

```

inicio := StrMatch(msg,1,"Qj")+2;
final := StrMatch(msg,1,"Qk");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.rot.q3);

```

```

inicio := StrMatch(msg,1,"Qk")+2;
string_len := StrLen(msg);
total := string_len-inicio-1;
okey := StrToVal(StrPart(msg,inicio,total),obj_trab.uframe.rot.q4);

```

!Se confirma al programa que se ha realizado la accion pertinente
Proc_confirmacion;

ELSEIF ident = "POS" THEN

!Cuando se recibe un mensaje de tipo POS, se guardan sus valores en la
!variable de posicion "Objetivo"

```

inicio := StrMatch(msg,1,"X")+1;
final := StrMatch(msg,1,"Y");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.trans.x);
!TPWrite StrPart(msg,inicio,total);

```

```

inicio := StrMatch(msg,1,"Y")+1;
final := StrMatch(msg,1,"Z");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.trans.y);
!TPWrite StrPart(msg,inicio,total);

```

```

inicio := StrMatch(msg,1,"Z")+1;
final := StrMatch(msg,1,"QS");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.trans.z);
!TPWrite StrPart(msg,inicio,total);

```

```

inicio := StrMatch(msg,1,"QS")+2;
final := StrMatch(msg,1,"Qi");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.rot.q1);
!TPWrite StrPart(msg,inicio,total);

```

```

inicio := StrMatch(msg,1,"Qi")+2;
final := StrMatch(msg,1,"Qj");
total := final-inicio;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.rot.q2);
!TPWrite StrPart(msg,inicio,total);

```

```

inicio := StrMatch(msg,1,"Qj")+2;
final := StrMatch(msg,1,"Qk");
total := final-inicio;

```



```

okey := StrToVal(StrPart(msg,inicio,total),Objetivo.rot.q3);
!TPWrite StrPart(msg,inicio,total);

inicio := StrMatch(msg,1,"Qk")+2;
string_len := StrLen(msg);
total := string_len-inicio-1;
okey := StrToVal(StrPart(msg,inicio,total),Objetivo.rot.q4);
!TPWrite StrPart(msg,inicio,total);

!Se actualiza la variable "control" para salir del bucle
control:=1;

!Se confirma al programa que se ha realizado la accion pertinente
Proc_confirmacion;
ENDIF
!Se guarda el ultimo mensaje recibido como tal
old_msg := msg;
ENDIF
ENDWHILE

!Se manda la instruccion al robot de moverse en función del tipo de
movimiento(MOV),
!objeto de trabajo (WOB) y posicion (POS)

!Una forma de ampliar el programa seria añadir funcionalidades como el cambio de
herramienta,
!la configuracion (cf1,cfx,...) o la velocidad de movimiento

IF tipo_mov=0 THEN
MoveL Objetivo,v50,fine,TVentosa \WObj:=obj_trab;
ELSE
MoveJ Objetivo,v50,fine,TVentosa \WObj:=obj_trab;
ENDIF

!Se confirma al programa que se ha realizado la accion pertinente
Proc_confirmacion;
ENDPROC

PROC Proc_confirmacion()
!Este proceso indica a la tarea de segundo plano que debe enviar una confirmacion al
programa
!a traves de la variable "confirmacion".
!Una vez sea enviada, la tarea de segundo plano indica a esta que se ha realizado a
través de la
!variable "confirmacion2".
!Este comportamiento es posible gracias al empleo de variables tipo PERS.

confirmacion := TRUE;
WaitTime(0.1);
WHILE confirmacion = TRUE DO

```

```
IF confirmacion2 = TRUE THEN
    confirmacion:=FALSE;
    TPWrite "Confirmacion completada";
    WaitTime(0.5);
ENDIF
ENDWHILE
ENDPROC
ENDMODULE
```

4.3. Código de RobotStudio (TCP)

```
MODULE Module1

VAR string data;
VAR socketdev server_socket;
VAR socketdev client_socket;
VAR string recibir:="";

PERS string server_ip := "172.16.78.65";
PERS num send_port := 8007;
VAR socketstatus sockStat;

VAR bool conectado := FALSE;
VAR jointtarget jt;

VAR num ok;
CONST num SERVER_BAD_MSG := 0;
CONST num SERVER_OK := 1;

PERS string recv_data:="POSX266.69Y-231.67Z365.78QS0.00Qi0.00Qj1.00Qk0.00 ";
VAR num tool := 0;
VAR num tool_old := 0;
VAR num pos := 0;
VAR num long := 1;

VAR intnum siglint;
VAR intnum conf_interrup;
PERS bool confirmacion;
PERS bool confirmacion2:=FALSE;

PROC main()

    !Conexion del socket e inicializacion de variables
    conectado := FALSE;
    recv_data := "";
    ServerCreateAndConnect server_ip, send_port;
    conectado := TRUE;
    confirmacion2 := FALSE;

    CONNECT siglint WITH reset_routine;
    !Interrupcion asociada a la activacion de la señal de entrada DoServerReset (input 9),
    !encargada de reiniciar los sockets
    ISignalDI DoServerReset,1,siglint;

    CONNECT conf_interrup WITH conf_routine;
    !Interrupcion asociada a la puesta a True de la variable persistente confirmacion,
    !encargada de iniciar el proceso de envio de confirmacion al programa
    IPers confirmacion,conf_interrup;
```

```

WHILE TRUE DO
    sockStat := SocketGetStatus(client_socket);
    IF sockStat = SOCKET_CONNECTED THEN
        !Si la conexion con el cliente es correcta, se procede a escuchar lo que este envie
        SocketReceive client_socket,\Str:=recibir \Time:=30;
        long := StrLen(recibir);
        pos := StrFind(recibir,1," ");

        !Si los mensajes se acumulan, se separan por los espacios colocados entre ellos y
se
        !toma el último
        WHILE pos <> long DO
            pos := StrFind(recibir,1," ");
            long := long - pos;
            recibir := StrPart(recibir,pos+1,long);
        ENDWHILE

        IF recibir <> "DUMMY " THEN
            !Se guardan los datos recibidos en la variable recv_data, siempre que no sean
            !la palabra "DUMMY ", empleada para establecer la conexion cliente-servidor
            recv_data := recibir;
        ENDIF
    ENDIF
ENDWHILE

!Gestor de errores
ERROR
IF ERRNO=ERR_SOCK_CLOSED THEN
    TPWrite "SERVIDOR: El cliente ha cerrado la conexion.";
    conectado := FALSE;
    WaitTime(0.01);

    !Cierre del servidor
    SocketClose client_socket;
    SocketClose server_socket;

    !Reinicio del servidor
    ServerCreateAndConnect server_ip, send_port;
    conectado := TRUE;
    ok := SERVER_OK;
    RETRY;
ELSEIF ERRNO=ERR_SOCK_TIMEOUT THEN
    TPWrite "SERVIDOR: Se ha sobrepasado el tiempo máximo de espera";
    conectado := FALSE;
    WaitTime(0.01);

    !Cierre del servidor
    SocketClose client_socket;
    SocketClose server_socket;

```

```

    !Reinicio del servidor
    ServerCreateAndConnect server_ip, send_port;
    conectado := TRUE;
    !reconnected := TRUE;
    ok := SERVER_OK;
    RETRY;
ENDIF
recv_data := " ";
ENDPROC

PROC ServerCreateAndConnect(string ip, num port)
    !Funcion encargada de crear el servidor y establecer la conexion con el cliente
    VAR string clientIP;

    SocketCreate server_socket;
    SocketBind server_socket, ip, port;
    SocketListen server_socket;
    TPWrite "SERVIDOR: Esperando conexiones entrantes...";
    WaitTime(0.01);
    WHILE SocketGetStatus(client_socket) <> SOCKET_CONNECTED DO
        SocketAccept server_socket, client_socket \ClientAddress:=clientIP
\Time:=WAIT_MAX;
        IF SocketGetStatus(client_socket) <> SOCKET_CONNECTED THEN
            TPWrite "SERVIDOR: Problema con conexión entrante.";
            WaitTime(0.01);
            TPWrite "Intentando reconectar.";
            WaitTime(0.01);
        ENDIF
        !Espera 0.5 segundos para intentar reconectar
        WaitTime 0.5;
    ENDWHILE
    !TPWrite "SERVIDOR: Conectado a la IP " + clientIP;
ENDPROC

TRAP reset_routine
    !Rutina encargada de reiniciar el servidor mediante la activacion y desactivacion
    !de la señal "DoServerReset", asignada a la entrada 9
    TPWrite "SERVIDOR: Reinicio de Servidor solicitado";
    WaitTime(0.01);
    WaitDI DoServerReset, 0;

    conectado := FALSE;

    !Cierre del servidor
    SocketClose client_socket;
    SocketClose server_socket;

    !Reinicio del servidor
    ServerCreateAndConnect server_ip, send_port;
    conectado := TRUE;

```

```

!reconnected := TRUE;
ok := SERVER_OK;

TPWrite "SERVIDOR: Reinicio completado";
WaitTime(0.01);
ERROR
  RAISE;
ENDTRAP

TRAP conf_routine
  !Rutina encargada de enviar la confirmacion al programa
  IF confirmacion = TRUE THEN
    SocketSend client_socket \str := "AccionRealizada";
    WaitTime(0.05);
    confirmacion2 := TRUE;
    WHILE confirmacion = TRUE DO
      WaitTime(0.01);
    ENDWHILE
  ENDIF
  confirmacion2 := FALSE;
ENDTRAP
ENDMODULE

```

