Trabajo de Fin de Máster

Máster Universitario en Ingeniería Electrónica, Robótica y Automática

Conexión de RobotStudio y Arduino mediante TCP/IP para la recolección y envío de datos de posicionamiento de cinta transportadora.

Autor: Mauricio Hinojosa Rea

Tutores: Dr. Luis Fernando Castaño Castaño

Dr. David Muñoz de la Peña Sequedo

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Máster
Máster Universitario en Ingeniería Electrónica, Robótica y Automática

Conexión de RobotStudio y Arduino mediante TCP/IP para la recolección y envío de datos de posicionamiento de cinta transportadora.

Autor:
Mauricio Hinojosa Rea

Tutores:
Dr. Luis Fernando Castaño Castaño
Dr. David Muñoz de la Peña Sequedo

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Máster: Conexión de RobotStudio y Arduino mediante TCP/IP para la recolección y envío de datos de posicionamiento de cinta transportadora.

Autor: Mauricio Hinojosa Rea

Tutores: Luis Fernando Castaño Castaño

David Muñoz de la Peña Sequedo

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A Flor Alba y Jazmín.

Agradecimientos

Quiero agradecer a mis padres, mis hermanas y a la vida misma por permitirme pasar un año lleno de aprendizaje y enseñanzas, tanto en la escuela de la vida como en la Escuela de Ingenieros de Sevilla. Estar aquí, me ha permitido abrir la mente y los ojos de manera infinita, inconmesurable y real. Gracias a mis padres, Lijia y Freddy, por mantener su fe en mí, darme la vida y guiarme por el camino que me llevó hasta aquí y que de seguro me llevará a las estrellas. Michelle y Micaela, por vosotras daría mi existencia.

Doy gracias a David Muñoz y Fernando Castaño, mis tutores; ya que con su propuesta de TFM me mostré a mi mismo hasta donde puedo llegar de forma intelectual. Gracias por sus enseñanzas y tiempo invertido en este trabajo. Por ayudarme y por su gran trabajo en lo cotidiano.

Mauricio Hinojosa Rea

Sevilla, 2019

Resumen

Este proyecto, nace de la necesidad de comunicar dos equipos utililizados en el laboratorio del departamento de Ingeniería de Sistemas y Automática de la Escula Técnica Superior de Ingeniería de Sevilla.

Uno de los equipos es un brazo robótico del tipo IRB120 del fabricante ABB, y el otro equipo es una cinta transportadora automatizada de fabricación propia en el departamento que ocupa parte de la zona de trabajo del brazo robótico. La cinta transportadora está gobernada por un microcontrolador (Arduino) que permite posicionar de forma precisa (en coordenadas X e Y) piezas que debe manipular el Robot.

El problema se presenta cuando, por medio de las entradas y salidas que posee la controladora real IRC5 (controladora asociada al robot IRB120) no permite la recogida de datos por medio de interrupciones, que es justo lo que se requiere para procesar la información que gestiona el Arduino de la cinta (posición X e Y de la pieza).

Este trabajo tiene la finalidad de inspeccionar, de forma remota, el acceso a datos y manipulación de funciones de RobotStudio; así como el envío y recolección de datos desde la fuente remota; siendo en este caso Arduino. Y con la visión, a posteriori, de servir como base de software para el montaje e inclusión del robot real con la cinta tansportadora. Arduino es una tecnología probada y además; cuenta con un shield Etherner que permite la manipulación y creación de servidores y clientes ethernet, por lo que es posible colgar un servidor en una red, clave esencial para este trabajo. RobotStudio tiene la capacidad de emular códigos, tantas veces como se requiera, para la manipulación y movimiento del robot emulado (segunda clave esencial para el desarrollo de este trabajo).

El trabajo desarrollado en este documento muestra la creación de un servidor ethernet por medio de protocolos TCP/IP con Arduino, que también sirve de gestor de sensores y datos de posicionamiento X e Y para, según petición de RobotStudio, se proceda a realizar 3 funciones en la cinta asociada a Arduino: *leer* (permite a RobotStudio conocer los datos de posición de la pieza), *absoluto* (permite a Arduino mover la cinta según se disponga en RobotStudio) e *incremental* (las mismas funciones que para absoluto, en un ciclo de repetición conocida y enviada desde RobotStudio).

Los objetivos alcanzados en este proyecto han sido los siguientes:

1. Montaje de un servidor en Arduino que conecte el cliente de RobotStudio a esta red, mediante TCP/IP.
2. Desarrollo un protocolo de comunicaciones sencillo entre los equipos.
3. Obtención de los datos de las coordenadas X e Y de la pieza en la cinta y enviarla a RobotStudio desde Arduino.

Abstract

This project is born from the need to communicate two equipment used in the laboratory of the Department of Systems Engineering and Automatic of the E.S.I in the Sevilla University.

One of the equipment is a robotic arm of the type IRB120 of the manufacturer ABB, and the other equipment is an automated conveyor belt of own manufacture in the department that occupies part of the work area of the robotic arm. The conveyor belt is governed by a microcontroller (Arduino) that allows precise positioning (in X and Y coordinates) parts to be handled by the Robot.

The problem arises when, through the inputs and outputs of the actual CONTROLLER IRC5 (controller associated with the robot IRB120) it does not allow the collection of data by means of interruptions, which is just what is required to process the information that is manages the Arduino of the tape (X and Y position of the part).

This work is intended to remotely inspect data access and manipulate RobotStudio functions; as well as sending and collecting data from the remote source; being in this case Arduino. And with the vision is serve as a software base for the assembly and inclusion of the real robot with the tancarrier tape. Arduino is a proven technology and it has an Etherner shield that allows the manipulation and creation of ethernet servers and clients, so it is possible to hang a server on a network, essential key to this work. RobotStudio can emulate codes, as many times as required, for the manipulation and movement of the emulated robot (second essential key for the development of this work).

The work developed in this document shows the creation of an ethernet server by means of TCP/IP protocols with Arduino, which also serves as a sensor manager and X and Y positioning data to, at RobotStudio's request, perform 3 functions on the Arduino-associated tape: *read* (allows RobotStudio to know the part's position data), *absolute* (allows Arduino to move the tape as available in RobotStudio) and *incremental* (same functions that for absolute, in a repetition cycle known and sent from RobotStudio).

The objectives achieved in this project have been as follows:

1. Mounting a server in Arduino that connects the RobotStudio client to this network, using TCP/IP.
2. Develop a simple communication protocol between Arduino and RobotStudio.
3. Obtain the X and Y coordinate data of the piece on the belt and send it to RobotStudio from Arduino.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

ABB	Acrónimo de Ase Brown Boveri, es una empresa multinacional.
CA	Corriente Alterna
CC	Corriente Continua
CLK	Señal de reloj
cm	Centímetros
DATA	Señal de datos
g	Gramos
GND	Tierra o masa
KB	Kilo Byte
Km	Kilómetros
LCD	Pantalla de tecnología LCD (Liquid Cristal Display)
LSB	Bit menos significativo (Least Significant Bit)
MHz	Mega Hertzios
mm	Milímetros
PWM	Modulación por ancho de pulsos (Pulse width modulation)
USB	Bus universal en Serie (Universal Serial Bus)
V	Voltios
TCPIP	Grupo de protocolos de red que respaldan a Internet y que hacen posible la transferencia de datos entre redes de ordenadores.

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Posicionado de piezas

En las prácticas de laboratorio que se hacían por parte de los alumnos en los equipos docentes de Robótica del Departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de Sevilla se limitaba la capacidad de uso de dichos complementos robóticos al posicionar las piezas dentro del rango de trabajo de un robot de manera manual, irrumpiendo la zona en donde el robot desempeña sus tareas, lo que desencadenaba una variación muy grande en la precisión de la posición de la pieza a recoger y las faltas a la seguridad industrial que podrían suceder al momento de la manipulación manual del sistema. El proceso consistía en colocar la pieza manualmente y reprogramar el movimiento del robot hasta que se tenga un margen de precisión aceptable.

Ahora, con el sistema de posicionamiento y comunicaciones realizado en este trabajo, se ayudará al estudiante en las labores de obtención de las piezas, evitando la invasión de la zona de trabajo por el operario; también se hará un servidor de ethernet desde Arduino para envíar la información de las coordenadas a las que se colocará la pieza, para luego ser recogida por el robot:

- Posicionado de piezas en medidas de ejes X e Y que el operario solicite para interactúar con el robot.
- Crear el servidor en Arduino y conectarlo con el cliente creado en RobotStudio para el intercambio de funciones y datos de coordenadas.
- Montar el servidor en Arduino y conoectar el cliente de RobotStudio a esta red, mediante TCP/IP.
- Desarrollar un protocolo de comunicaciones sencillo para la comunicación.
- Obtener los datos de las coordenas X e Y de la pieza en la cinta y enviarla a RobotStudio desde Arduino.

1.2 Robot ARB120

Robot industrial multiusos de 25 kg de peso. Este robot puede soportar cargas de hasta 3 kg (4 kg en posición vertical de la muñeca) con un alcance de 580 mm. Se trata del robot más pequeño de *ABB*, de gran capacidad de producción frente a una baja inversión [1]. Este robot posee una estructura abierta especialmente adaptada para un uso flexible y presenta la posibilidad de comunicación con sistemas externos [1].

Figura 1: *ABB IRB120*

1.3 Arduino TCP/IP

El modelo TCP/IP posee un conjunto de guías generales de diseño e implementación de protocolos de red específicos para consentir que un equipo logre comunicarse en una red local o internet. TCP/IP provee conectividad de extremo a extremo definiendo como los datos incumbirían ser formateados, direccionados, transmitidos, enrutados y recibidos por el destinatario [3].

Figura 2: Arduino con Shield Ethernet

La calidad del modelo TCP/IP está basada en el modelo usado para acceder a Internet o a redes internas (Intranet) de cualquier tipo de computadoras. Arduino va a permitir conectarse a Internet o a una red interna mediante TCP/IP y poder realizar múltiples operaciones o usarse como pasarela para conectar a Internet dispositivos que no tienen esa capacidad. La implementación de la pila de protocolos de TCP/IP en Arduino se hace mediante un shield o HW adicional que nos da la capa de acceso a red (ethernet o WiFi), capa de internet (IP) y capa de transporte. La capa de aplicación deberemos implementarla dentro de Arduino ya sea directamente o mediante una librería [3]. En esta comunicación por estándar se envían hasta 1024 bytes, tanto en RX como TX

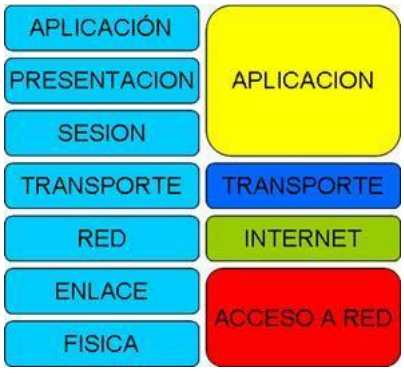


Figura 3: Protocolo OSI simplificado para TCP/IP

1.4 Aplicación

Este trabajo se enfoca en formalizar un posicionador de piezas, que incorporará, por un lado, la cinta transportadora que dispone de un motor de corriente continua, un encoder incremental, un calibrador digital ubicado a lo ancho del arreglo y un sensor del tipo fotoeléctrico. Por otra parte, la electrónica de este sistema, que establecerá las órdenes dictadas a la cinta transportadora usando un microcontrolador Arduino y una placa ethernet, éste a su vez facilitará las tareas para realizar la interfaz hombre/máquina, la lectura de respuesta de los elementos presentes en el proyecto, y también servirá para la comunicación del sistema (cinta-Arduino) con el robot. El control del posicionador de piezas en la zona de trabajo de dos robots se muestra en la Figura 4 [4].

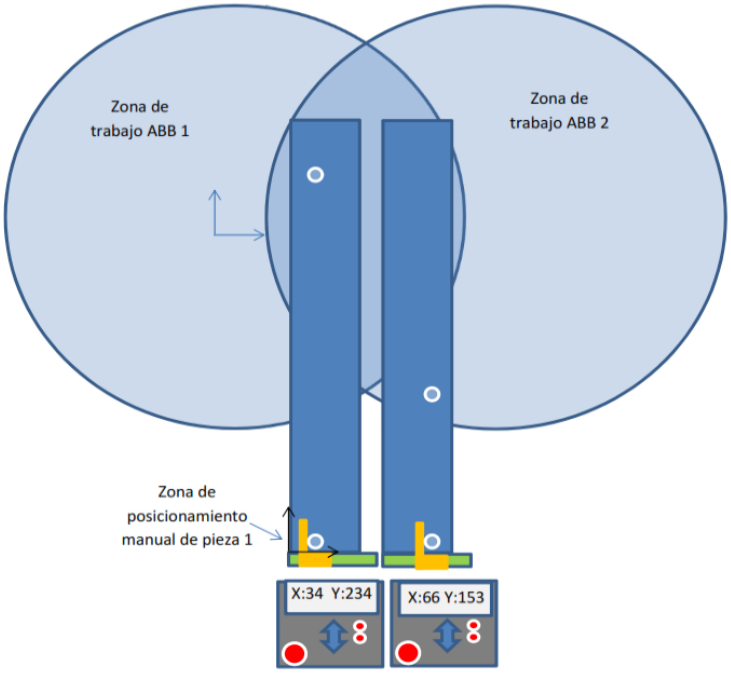


Figura 4: Croquis de los posicionadores y zonas de trabajo [3]

Añadido a esto, el sistema a desarrollar facilita la opción de funcionar cuando el microcontrolador Arduino haya caído, quemado o retirado del sistema, teniendo en cuenta que sí el Arduino no esta presente se podrá hacer un cambio a local para su uso en general y se podrá actúar de modo manual, provocando que la cinta transportadora avance, se detenga o retroceda mediante la acción de botones [3].

1.5 Definición de Funciones y Errores

En este apartado se especifican las **funciones** que deberán estar presentes y ser ejecutadas y reconocidas tanto en Arduino como en RobotStudio, por medio del socket ethernet creado en Arduino. Se defenirán de forma global y luego, en cada sección correspondiente, serán definidas según el lenguaje que se requiera. Lo mismo sucederá para los errores, que se desarrollarán para ser avisos de que algo falla en la conexión o en la cinta.

- **Leer:** se usará el caracter 'l' para definir la dunción de leer. Arduino envíará las coordenadas de la pieza a RobotStudio. Codigo 11 para afirmar la correcta ejecución de la función
- **Absoluto:** Al caracter 'a' enviado en la trama de datos se le aumentará una distancia, que deberá ser envíada desde RobotStudio a Arduino y será una función de movimiento para la cinta. Codigo 12.
- **Relativo**: El caracter 'r' tendrá como parámetros añadidos una distancia y el número de pasos a realizar. Codigo 13.

La lista de **errores** tendrá los siguientes identificadores, tanto en Arduino como en RobotStudio:

- Codigo 1: El caracter enviado desde RobotStudio no coincide con ninguna función; se deberá envíar un caracter correcto
- Codigo 2: Se ha cambiado (en Arduino) de Remoto a Local, y por tanto la cinta se detendrá.
- Codigo 3: Encoder no funciona. Revisar función de la cinta y el sensor encoder.
- Codigo 4: Valor de movimiento negativo (en retroceso para la cinta) es demasiado grande. Modificar el valor a envíar en RobotStudio.

Todos estos valores numéricos serán enviados desde Arduino y se podrán apreciar en RobotStudio y en el Serial de Arduino.

2 DESCRIPCIÓN DEL HARDWARE

Estimando el alcanze del proyecto, se establecen tres etapas, el control con Arduino (y que a la vez sea el servidor de la red Ethernet), el switch LAN o red interna de los laboratorios, y RobotStudio o El IRB120 que servirán para recoger la pieza (y que servirán como cliente). Por lo tanto, es necesario establecer un descriptivo general de el hardware, que permitirá entender el proceso de comunicaciones y de accionamiento del robot y de la cinta asociada al Arduino.

2.1 Arduino Mega 2560

Arduino basado en microcontrolador el ATmega2560 (Figura 5), posee 54 pinesdigitales de entrada / salida (de los cuales 15 se pueden usar como salidas PWM), 16 entradas analógicas, 4UART (puertos serie de hardware), un oscilador de cristal de 16 MHz, una conexión USB, un conector dealimentación, un encabezado ICSP, y un botón de reinicio. Para poder utilizar el mismo es necesarioconectarlo mediante cable USB a una computadora, se puede alimentar de voltaje al mismo mediante unadaptador de CA a CC o con una batería [3].

Figura 5: Arduino Mega 2560 [4]

La table de especificaciones muestra los pines, funciones especiales, comunicaciones, interrupciones y otras caracteristicas físicas y de hardware de este dispositivo.

Tabla 1: Especificaciones Técnicas de Arduino Mega 2560

Microcontrolador	ATmega2560
Tensión de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límite)	6-20V
Pines de E / S digitales	54 (de los cuales 15 proporcionan salida de PWM)
Clavijas de entrada analógica	dieciséis

Corriente DC por Pin E / S	20 mA
Corriente DC para 3.3V Pin	50 mA
Memoria flash	256 KB de los cuales 8 KB utilizados por el gestor de arranque

SRAM	8 KB
EEPROM	4 KB
Velocidad de reloj	16 MHz
LED_BUILTIN	13
Longitud	101.52 mm
Anchura	53.3 mm
Peso	37 g

2.1.1 Alimentación

Arduino Mega 2560 tiene la posibilidad de ser alimentado mediante la conexión USB que puede ser proporcionada por una computadora por poner un ejemplo o una fuente de alimentación externa. La fuente de poder se selecciona automáticamente [3].

A continuación, se detallan los pines de alimentación de la tarjeta:

- VIN: Entrada de voltaje de la placa cuando se realiza una alimentación externa.
- 5V: Este pin produce una tensión de 5 V regulados.
- 3V3: Fuente de voltaje con una tensión de 3.3V la misma que es generada por el regulador con el que cuenta la placa, el consumo máximo de corriente es de 50 mA.
- GND: Pines que posee la placa para tener conexión a tierra.
- IOREF: Pin de la placa encargado de proporcionar la referencia de tensión con la que trabaja el microcontrolador.

2.1.2 Memoria

El ATmega2560 tiene 256KB de memoria flash para almacenar código, tiene 8 KB de memoria SRAM y 4KB de EEPROM. Para el gestor de arranque son utilizados 8KB de memoria [4].

2.1.3 Entradas y Salidas

Cada uno de los 54 pines digitales del Mega se puede utilizar como entrada o salida, usando las funciones pinMode ( ), digitalWrite ( ) y digitalRead ( ). Operan a una tensión equivalente a los cinco voltios. Cada pin puede proporcionar o recibir 20 mA teniendo en cuenta las condiciones de funcionamiento recomendadas y tiene una resistencia interna de pull-up (desconectada por defecto) de 20-50kohmios. Para evitar daños permanentes en el microcontrolador no se debe exceder de un máximo en corriente igual a 40mA [3].

Además, algunos pines tienen funciones especializadas:

- Serie: 0 (RX) y 1 (TX); Serial 1: 19 (RX) y 18 (TX); Serial 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX). Se usa para recibir (RX) y transmitir (TX) datos en serie TTL. Los pines 0 y 1 también tienen conexión a los pines correspondientes del chip ATmega16U2 USB a TTL.
- Interrupciones externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3) y 21 (interrupción 2). Estos pines se pueden configurar para activar una interrupción en un nivel bajo LOW (0V), flancos de subida o bajada, o un cambio en el nivel.
- PWM: 2 a 13 y 44 a 46. Proporcionan salida PWM (Pulse Wave Modulation) de 8 bits con la función analogWrite ( ).
- SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Estos pines permiten comunicación SPI utilizando la

biblioteca SPI . Los pines SPI también se fraccionan en el encabezado del ICSP, que es físicamente compatible con el Arduino / Genuino Uno y los antiguos Duemilanove y Diecimila Arduino.

- LED: 13. Hay un LED integrado conectado al pin digital 13. Cuando el pin tiene un valor ALTO o lo que es lo mismo un valor HIGH (5V), el LED se encenderá, cuando el pin está BAJO es decir con un valor de LOW (0V), el LED se aparará.
- TWI: 20 (SDA) y 21 (SCL). Soporte de comunicación TWI utilizando la biblioteca Wire. Se debe tener en cuenta que estos pines no están en la misma ubicación que los pines TWI en las antiguas placas Duemilanove o Diecimila Arduino.

2.1.4 Comunicación

El ATmega2560 tiene cuatro UART de hardware para comunicación TTL (5V). Un ATmega16U2 en la placa canaliza uno de estos a través de USB y suministra un puerto virtual para el software en la computadora (las máquinas Windows requerirán un archivo .inf, pero las máquinas OSX y Linux reconocen la placa de forma automática). El software Arduino (IDE) incluye un monitor serie que posibilita el envío de datos de texto simples desde y hacia la placa de Arduino. Los LED RX y TX de la placa parpadearán cuando los datos se transmitan a través del ATmega8U2 / ATmega16U2 chip y la conexión USB al ordenador (pero no para comunicación serial en los pines 0 y 1) [3].

2.2 Hardware Ethernet en Arduino

El Arduino ethernet shield permite conectar un Arduino a una red ethernet. Es la parte física que implementa la pila de protocolos TCP/IP. Está basada en el chip ethernet Wiznet W5100. El Wiznet W5100 provee de una pila de red IP capaz de soportar TCP y UDP. Soporta hasta cuatro conexiones de sockets simultáneas. Usa la librería Ethernet para leer y escribir los flujos de datos que pasan por el puerto ethernet y permite escribir sketchs (con el IDE correspondiente) que se conecte a internet usando esta shield [2]. Para este trabajo se usará para colgar un servidor ethernet local, para el intercambio de datos (Figura 6) [3].

La shield contiene varios LEDs para información:

- ON: indica que la placa y la shield están alimentadas
- LINK: indica la presencia de un enlace de red y parpadea cuando la shield envía o recibe datos
- 100M: indica la presencia de una conexión de red de 100 Mb/s (de forma opuesta a una de 10Mb/s)
- RX: parpadea cuando el shield recibe datos
- TX: parpadea cuando el shield envía datos

Puntos clave del Ethernet Shield:

- Opera a 5V suministrados desde la placa de Arduino
- El controlador ethernet es el W5100 con 16K de buffer interno. No consume memoria.
- El shield se comunica con el microcontrolador por el bus SPI, por lo tanto, para usarlo siempre debemos incluir la libreria SPI.h: http://arduino.cc/en/Reference/SPI
- Soporta hasta 4 conexiones simultáneas
- Usar la librería Ethernet para manejar el shield: http://arduino.cc/en/Reference/Ethernet
- El shield dispone de un lector de tarjetas micro-SD que puede ser usado para guardar ficheros y servirlos sobre la red. Para ello es necesaria la librería SD: http://arduino.cc/en/Reference/SD
- Al trabajar con la SD, el pin 4 es usado como SS.
- Arduino usa los pines digitales 10, 11, 12, y 13 (SPI) para comunicarse con el W5100 en la ethernet shi

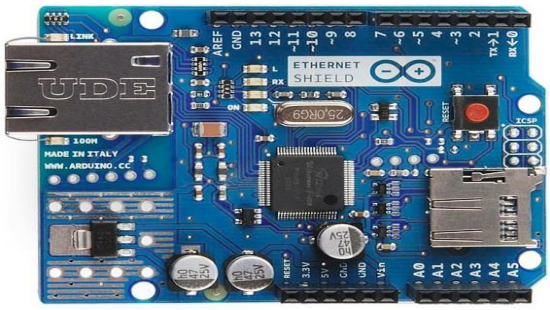


Figura 6: Shield Ethernet compatible con Arduino Mega [2]

2.3 Motor de Corriente Continua y Driver L298N para motores DC

Para este Proyecto se usan motores de 24 voltios DC, sin ninguna caracteristica en especial y bastante gnéricos, para su control se usarán puentes o shields L298D que permitirán invertir la polaridad de flujo de corriente y con esto la dirección y sentido de movimiento de la cinta acomplada al control de Arduino. El Puente H (como también se le conoce a L298D) también separa la parte de potencia de la parte electronica y de control a 5 voltios (Figura 7) [3].

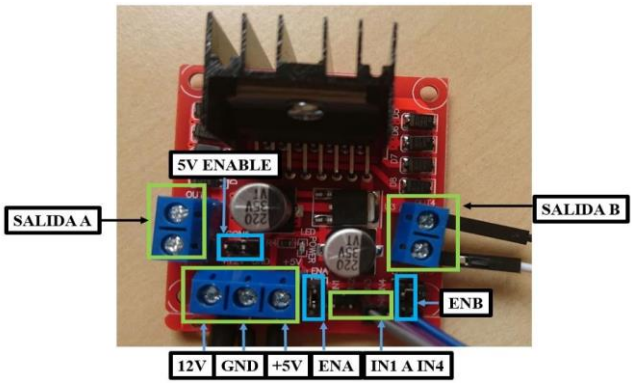


Figura 7: Pines de conexión del módulo L298N [3]

- **Salida A:** Compuesta por un OUT1 y OUT2, con pin de habilitación correspondiente a ENA.
- **Salida B:** Compuesta por OUT3 y OUT4, con pin de habilitación correspondiente a ENB.

  Los pines marcados como IN1, IN2, IN3 e IN4 los cuales realizan el control de dicho módulo.

2.3.1 Conexionado para alimentación del módulo L298N

Se tiene la opción de poder conectar a la tarjeta del módulo mediante dos maneras diferentes ya que el integrado LM7805 posibilita esta opción [3]. Para el trabajo lo usaremos de la forma con entrada lógica 5 Vdc (Figura 8) [3].

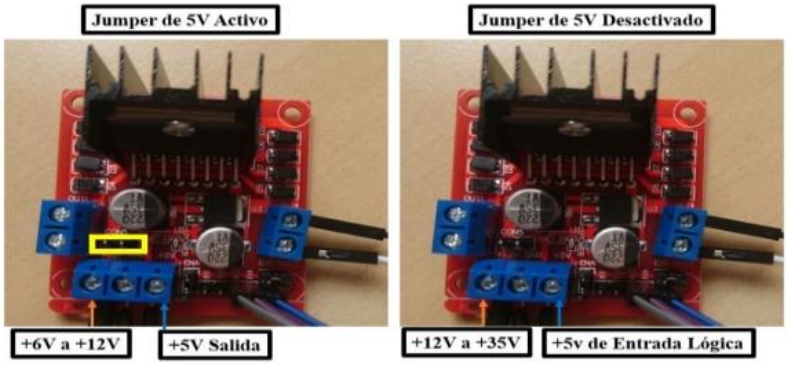


Figura 8: Alimentación de entradas y salidas de L298D [3]

2.4 Encoder Rotativo Incremental

Para realizar las mediciones con el Arduino para el ***eje Y***, se usa el enconder rotativo incremental LPD3806-600BM con una resolución de 1200 puntos por Vuelta. El coder giratorio tiene dos salidas de onda cuadrada (A y B) con desfase de 90 grados una de la otra. Siempre que el impulso de la señal A esté en el flanco descendente el impulso de la señal B será leída. Como muestra en la Figura 9, cuando el encoder gira en sentido horario, el pulso de B es siempre positivo. El pulso de B es negativo cuando el encoder gira en sentido antihorario. Teniendo conectadas estas dos salidas a un microcontrolador es posible determinar la dirección de giro. Las dos salidas representan el movimiento del disco del encoder como un tren de impulsos modulado en cuadratura [3]. El muestreo en Arduino se logra mediante interrupciones externas, que se habilitan cuando A o B se ponen en alto [3].

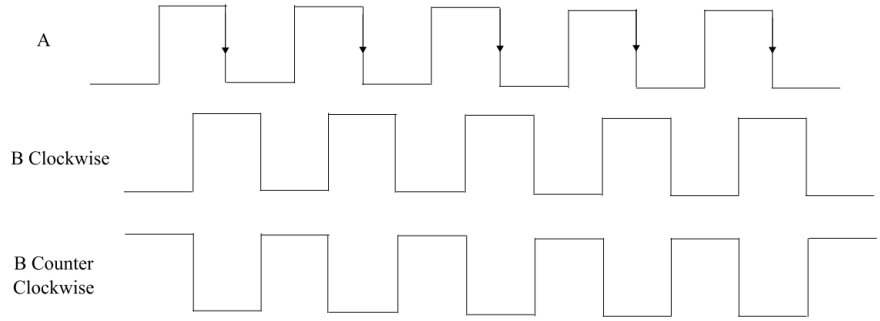


Figura 9: Salidas A y B del encoder incremental [3]

2.4.1 Especificaciones técnicas encoder incremental

El encoder rotativo de serie LPD3806 600BM G5 24C tiene las siguientes especificaciones que se indican a continuación:

- 600 pulsos/ revolución para una sola fase. La salida de dos fases conduce a 2400 pulsos/revolución.
- Velocidad mecánica máxima: 5000 revoluciones/minuto.
- Respuesta de frecuencia: 0-30KHz.

2.4.2 Cables de Conexiones de encoder incremental

El encoder cuenta con cuatro cables los cuales están diferenciados por el color que tienen cada uno de ellos.

- Rojo: Alimentación de 5-24 VDC
- Negro: GND o tierra.
- Verde: Fase A.
- Blanco: Fase B.

2.5 Calibrador Digital

El calibrador servirá para tomar la coordenada de la pieza a colocar a lo ancho de la cinta transportadora, este instrumento tiene un indicador LCD que va mostrando el cambio de la medición cuando se mueve a través de la regleta metálica que conforma su estructura. La herramienta de medición posee una cubierta de plástica que es donde se encuentra el LCD, removiendo la parte superior de esta cubierta se accede a cuatro pines los mismos que indican la alimentación de 1.5VDC, señal de reloj (CLK), señal de datos y por último un pin el cual servirá para realizar la conexión a tierra (Figura 10) [3].

Figura 10: Calibrador digital y pines de señales a utilizar [3]

Los procesos de solución para obtener los datos se encuentran en el trabajo anterior; bastará con añadir la librería y habilitar el puerto de comunicación por interrupción.

2.6 Banda transportadora con elementos integrados

La banda transportadora de la cual se hace uso para la elaboración de este proyecto la que se muestra en la Figura 11. Tiene acoplada a su estructura el motor de 24VDC, el calibrador digital, el encoder incremental y el sensor fotoeléctrico [3].

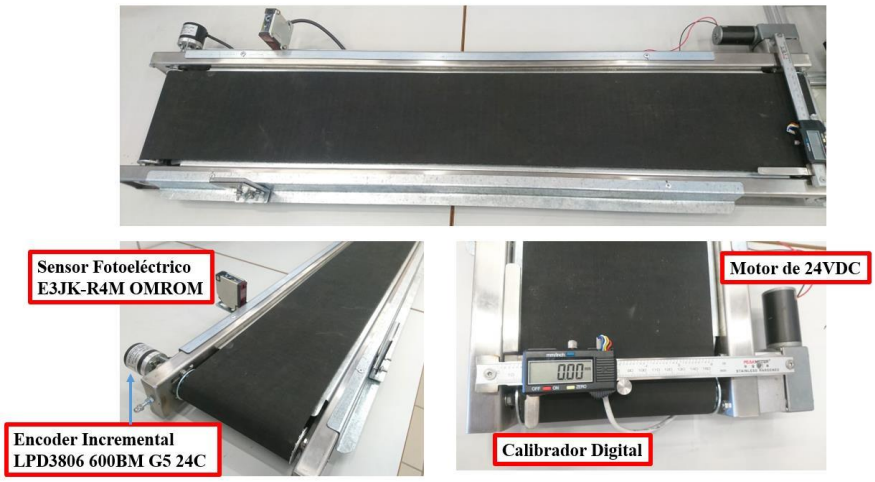


Figura 11: Estructura de la banda transportadora [3]

2.7 IRC5C Compact Controller

El IRC5 es el punto de referencia de la industria robótica en tecnología de controladores de robots. Además del control de movimiento único de ABB, aporta flexibilidad, seguridad, modularidad, interfaces de aplicación, control multirobot y puerto de suplén de herramienta de PC. El IRC5 viene en diferentes variantes para proporcionar una solución rentable y optimizada para cada necesidad [5]. Esta controladora está montada en los laboratorios y permitirá el acceso remoto a internet o a la red de los laboratorios (Figura 12).

Figura 12: IRC5C [5]

2.7.1 RemoteService

La finalidad de la caja de RemoteServicees actúar como puente entre el controlador del robot y un servidor remoto. La conexión entre la caja de Services y el servidor remoto se realiza a través de la tecnología inalámbrica GPRS e Internet. A través del puerto de la consola y el puerto Ethernet se almacena, interpreta y filtra la información proveniente del robot para obtener una valiosa información de servicio en la aplicación RemoteService (Figura 13) [6].

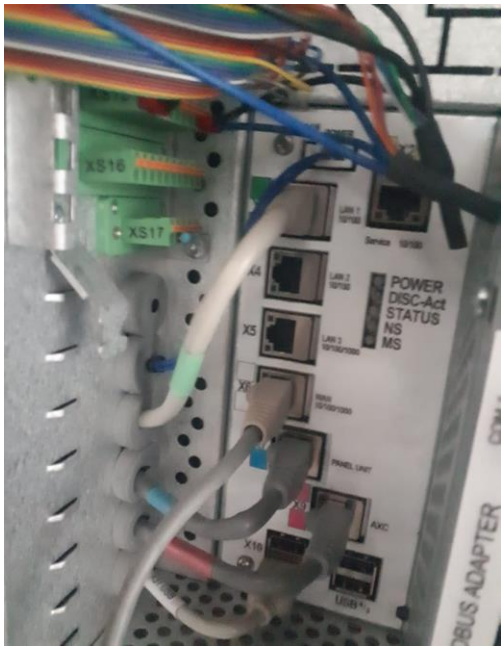


Figura 13: Puertos de conexión, entradas y salidas de IRC5C [6]

3 ESTACIÓN DE ROBOTSTUDIO

En este capítulo se va describir el proceso de creación de la estación virtual de *RobotStudio*, software de *ABB* el cual se puede obtener de su página web principal. La versión descargable está bastante limitada en funciones y para activar todas las funciones hay que tener una licencia, como la que posee la *US* [7]. La creación de la herramienta tiene que estar dotada de cierta inteligencia y se la desarrollará en la misma plataforma. La versión de RobotStudio es la 6.08.

3.1 Creación de la herramienta

Para este apartado es necesario crear las herramientas en el mismo sketch o plataforma, añadiendo los objetos necesarios para formar la herramienta, unirlas adecuadamente y montarlas. Una vez hecho esto, se necesita saber exactamente cual es su posición y orientacíón al unirlo a *IRB120* [7].

3.1.1 Montaje de la herramienta en *RobotStudio*

La herramienta fue creada previamente, en el mismo portal. La Figura 14 muestra el botón de importación de biblioteca. La Figura 15 es la herramienta "pinza.rslib" .

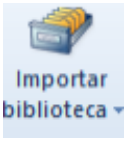


Figura 14: Boton de importación de herramienta

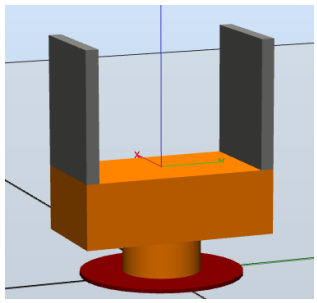


Figura 15: Herramienta Pinza

3.1.2 Añadir la herramienta al robot

Para añadir la herramienta a la junta de la muñeca del robot, simplemente se arrastra la herramienta hacia el robot en la estacion de herramientas lateral del programa. Se acepta y automáticamente se añade al robot (Figura 16).

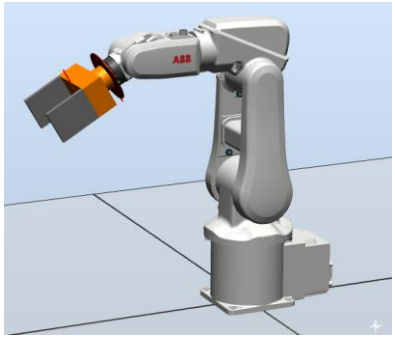


Figura 16: Herramienta añadida al Robot

3.2 Crontrol en RAPID

En este apartado se hará mensión al control creado en RAPID para el envío y recepción de datos por la red; así como de mavimientos básicos asociados a las coordenadas obtenidas por Arduino. Hay que mensionar que el programa no crea un servidor interno, ya que al estar RobotStudio montado en el landhost de la computadora es imposible tratar de crear un servidor que pueda soportar clientes externos y que pasen al la IP del landhost (127.0.0.1). Mas, el proceso inverso: crear un servidor externo y conectar RobotStudio a este (servidor remoto) es posible. El lenguaje de programación se llama RAPID.

Para el control de la cinta, se han creado 3 funciones que son: leer, absoluto y relativo. Estás 3 funciones serán explicadas en más detalle en siguientes subsecciones.

3.2.1 Creación de variables

La creación de variables (Figura 17) es muy parecida a Python o C. Cabe mensionar que las constantes del tipo *robtarget* son puntos de posiciones para trayectorias en RobotStudio, que tienen que ser definidas en la pantalla del robot y la estructura de datos, en general, es la siguiente:

```
VAR num n;
VAR string posx;
VAR string posy;
VAR string cadena;
VAR num posxint;
VAR num posyint;
VAR num estad:=0;
VAR bool ok;
VAR bool okposx;
VAR bool okposy;
VAR bool okack;
VAR num i:=1;
VAR rawbytes receive_string;
!recibe un paquete de datos en string
VAR string ack;
CONST robtarget Target_pieza:=[[436.2339,144,180.9312],[0.70711,0,0,0.70711],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_10:=[[436.2339,144.422,389.7914],[0.70711,0,0,0.70711],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_20:=[[-101.0329,144.422,389.7915],[0.70711,0,0,0.70711],[1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_30:=[[-101.0329,144.422,173.8547],[0.70711,0,0,0.70711],[1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Figura 17: Variables de RAPID

3.2.2 Funcion “Main”

La función M*ain* permite que el puntero de inicio se coloque en primer lugar en esta función, aquí se empiezan a leer todas las referencias, funciones y procedimientos de Rapid. Se pueden utilizar operaciones, llamadas a funciones y comparaciones lógicas, así como cualquier tipo de ciclo o bucle (Figura 18).

```
PROC main()

    !Añada aquí su código
    SocketClose my_socket;
    abrircomunicacion;
    WaitTime 1;
    !  leer;
    !  absoluto 5;  !!el valor hace referencia a la distancia que se va a recorrer por la cinta en x
    ! relativo 4, 3; !!el primer valor hace referencia a la distancia que se va a recorrer por la cinta en x
    !!el segundo hace referencia al numero de saltos que se va a recorrer por la cinta en x
    absoluto 5;
    ! leer;
    ! WHILE ok=TRUE DO

    ! ENDWHILE
    !TPWrite "Instruction SocketClose has been executed";
ENDPROC
```

Figura 18: Función Main

3.2.3 "Abrir comunicación"

Esta es la función que permite al robot crear un socket del tipo TCP/IP, y luego conectarlo a una dirección IP (192.168.1.200), que en este caso sería la dirección asignada al Arduino, de antemano conocida, con un puerto dedicado a esta comunicación (4012). Como se muestra, no es necesario es pecificarle una dirección IP propia al robot ya que él no esta creando el servidor (Figura 19).

```
PROC abrircomunicacion()

    SocketCreate my_socket;
    !crea el socket
    SocketConnect my_socket,"192.168.1.200",4012;
    !lo enlace a la direccion y al puerto correspondiente al arduino

ENDPROC
```

Figura 19: Función Abrir Comunicación

3.2.4 Función "Leer"

Esta función permite leer las coordenadas X e Y de la pieza, ambas coordenadas son enviadas desde Arduino. Primero se envía al socket el caracter *'l'* que Arduino interpretará como un caracter, con la correspondiente function de envíar a RobotStudio 24 caracteres ASCII (8 caracteres ASCII para el estado de la cinta, 8 para la coordena X y 8 para la coordenada Y). Se espera por estos bits durante un tiempo maximo, que es definido por RobotStudio, y se procede a desempaquetar dichos datos. Tercero, se asigna estos valores a las variables *ack*, *porx* y *posy*. Por último, se procede a convertir estos datos a valores numéricos (RobotStudio no distingue entre enteros y flotantes). Luego se hace la llamada a la función *Estado* (Figura 20).

```
PROC leer()
    ! ClearRawBytes receive_string;
    WaitTime 1;
    SocketSend my_socket,\Str:="l";
    !escribe en la red y lo envia a arduino (donde será leido)
    WaitTime 0.1;
    !espera un tiempo

    SocketReceive my_socket\RawData:=receive_string,\Time:=WAIT_MAX;
    WaitTime 0.01;
    !espera un tiempo
    UnpackRawBytes receive_string,1,ack\ASCII:=8;
    !8 es el espacio en caracteres que forman el numero
    UnpackRawBytes receive_string,9,posx\ASCII:=8;
    UnpackRawBytes receive_string,17,posy\ASCII:=8;
    !UnpackRawBytes receive_string, 1, posy \ASCII:=24;
    !ok:=StrToVal(ack,float);
    okposx:=StrToVal(posx,posxint);
    okposy:=StrToVal(posy,posyint);
    okack:=StrToVal(ack,estad);
    ClearRawBytes receive_string;
    estado;
ENDPROC
```

Figura 20: Función Leer

3.2.5 Funcion "Absoluto"

Esta función tiene el mismo proceso que la anterior, se envía un caracter que le dice a Arduino que se va a realizar un movimiento absoluto, luego se espera a la respuesta y se actúa segun lo dictaminado en la recepción de datos. Hay que recorder que se entregan desde Arduino 24 caracteres ASCII.

Esta función tiene una caracteristica añadida a la del apartado anterior, básicamente se solicita que el operario envié un valor numérico, el cuál le indicará a Arduino cuantos centímetros en ***Y*** (Figura 4) deberá moverse el brazo para alcanzar la pieza. Cuando Arduino alcanza el objetivo en centímetros se en envía la respuesta (Figura 21).

```
PROC absoluto(num distancia)
    !ClearRawBytes receive_string;
    WaitTime 1;
    cadena:=NumToStr(distancia,1);
    !cadena:="a"+cadena;
    SocketSend my_socket,\Str:="a"+cadena;
    !escribe en la red y lo envia a arduino (donde será leido)
    WaitTime 0.1;
    !espera un tiempo

    SocketReceive my_socket\RawData:=receive_string,\Time:=WAIT_MAX;
    WaitTime 0.01;
    !espera un tiempo
    UnpackRawBytes receive_string,1,ack\ASCII:=8;
    !8 es el espacio en caracteres que forman el numero
    UnpackRawBytes receive_string,9,posx\ASCII:=8;
    UnpackRawBytes receive_string,17,posy\ASCII:=8;
    okposx:=StrToVal(posx,posxint);
    okposy:=StrToVal(posy,posyint);
    okack:=StrToVal(ack,estad);
    ClearRawBytes receive_string;
    estado;
ENDPROC
```

Figura 21: Función Absoluto + valor Y para Arduino

3.2.6 Funcion “Relativo”

Esta función tiene el mismo sumario anterior, se envía un caracter que le dice a Arduino que se va a realizar un movimiento relativo, luego se espera a la respuesta y se actúa segun lo dictaminado en la recepción de datos. Hay que recorder que se entregan desde Arduino 24 caracteres ASCII (Figura 22).

Esta función tiene una caracteristica añadida a la del apartado anterior, se solicita que el operario envié un valor numérico. Cuando Arduino alcanza el objetivo en centímetros se envía la respuesta.

Se añade el número de pasos como un requisito para la función; por lo tanto, el operario deberá colocar ambos números, tanto el de movimiento en centímetros como el de pasos.

Se recibe como *ack*, tanto para ésta como para las dos funciones anteriores un número positivo, que permite decidir si las funciones antes enviadas se han cumplido o no y que en apartados posteriores será explicado con mayor detalle.

```
PROC relativo(num distancia,num pasos)
    !num para ingresar una constante; var num para ingresar una variable
    !ClearRawBytes receive_string;
    WaitTime 1;
    SocketSend my_socket,\Str:="r"+NumToStr(distancia,0)+"N"+NumToStr(pasos,0);
    !escribe en la red y lo envia a arduino (donde será leido)
    WaitTime 0.1;
    !espera un tiempo

    SocketReceive my_socket\RawData:=receive_string,\Time:=WAIT_MAX;
    WaitTime 0.01;
    !espera un tiempo
    UnpackRawBytes receive_string,1,ack\ASCII:=8;
    !8 es el espacio en caracteres que forman el numero
    UnpackRawBytes receive_string,9,posx\ASCII:=8;
    UnpackRawBytes receive_string,17,posy\ASCII:=8;
    okposx:=StrToVal(posx,posxint);
    okposy:=StrToVal(posy,posyint);
    okack:=StrToVal(ack,estad);
    ClearRawBytes receive_string;
    estado;
ENDPROC
```

Figura 22: Función Absoluto + valor Y + número de incrementos para Arduino

3.2.7 Función “Estado”

Aquí se interpreta el *ack* que es convertido en las funciones anteriores a un valor num con nombre *estad,* recibido desde Arduino (los primeros 8 caracteres ASCII) y permite conocer el estado en el que se encuentran la cinta y Arduino. 11, 12 y 13 corresponden a los estados realizados correctamente. 1, 2, 3 y 4 corresponden a errores producidos en el microcontrolador y el sistema (Figura 23).

```
PROC estado()

    !!Comunicación Efectuada correctamente

    IF estad=11 THEN
        TPWrite "Instruction leer have been done. Estado:"\Num:=estad;
        movimientorobot;
    ENDIF
    IF estad=12 THEN
        TPWrite "Instruction absoluto have been done.Estado:"\Num:=estad;
        movimientorobot;
    ENDIF
    IF estad=13 THEN
        TPWrite "Instruction relativo have been done.Estado:"\Num:=estad;
        movimientorobot;
    ENDIF

    !!listado de errores que se podrian enviar desde arduino
    IF estad=1 THEN
        TPWrite "No se envió un comando conocido por arduino.Error:"\Num:=estad;
        ErrWrite "RobotStudio error","No se envió un comando conocido por arduino."\RL2:="Escribir comando correcto";
        Stop;
    ENDIF
    IF estad=2 THEN
        TPWrite "Se cambio de remoto a local en la cinta. Error:"\Num:=estad;
        ErrWrite "Arduino error","Se cambio de remoto a local en la cinta."\RL2:="No cambiar durante ejecución";
        Stop;
    ENDIF
    IF estad=3 THEN
        TPWrite "Cinta no se mueve. Encoder dejo de funcionar. Error:"\Num:=estad;
        ErrWrite "Arduino error","Cinta no se mueve."\RL2:="Encoder dejo de funcionar.";

        Stop;
    ENDIF
    IF estad=4 THEN
        TPWrite "Valor absoluto negativo demasiado grande. Erro:"\Num:=estad;
        ErrWrite "Arduino error","Valor absoluto negativo demasiado grande."\RL2:="Enviar un valor menor o positivo.";

        Stop;
    ENDIF
ENDPROC
```

Figura 23: Función Estado

3.2.8 Función “Cerrar Comunicación”

Esta función permite cerrar el socket creado y por tanto cerrar las comunicaciones. Es especialmente efectivo si se necesitan cerrar el puerto o la conexión (Figura 24).

```
PROC cerrarcomunicacion()
    SocketClose my_socket;
    !cierra el socket
ENDPROC
```

Figura 24: Función Cerrar Comunicación

3.2.9 Función “Movimiento Robot”

Esta función realizará movimientos básicos con referencia a puntos de trayectoria, ya que el objetivo de este trabajo era permitir realizar la comunicación y envío de datos desde y hacia el mismo con el objetivo de control de la cinta. Para movimientos o funciones con más complejidad se muestra la Figura 25 como base para un trabajo posterior. Estos movimientos están dependientes de las coordenadas de X como de Y que arriban desde Arduino, partiendo desde un punto de referencia inicial.; dichos movimientos se efectúan si el *estado* o *ack* recibido es correcto.

```
PROC movimientorobot()

    Movel offs(Target_pieza,-posxint*5,posyint*5,0),v500,fine,t_pinza\WObj:=wobj0;
    !Movej offs(Target_10,-posxint*2,80-posxint*2,0),v500,z100,t_pinza\WObj:=wobj0;
    Movej Target_20,v500,fine,t_pinza\WObj:=wobj0;
    Movej Target_30,v500,fine,t_pinza\WObj:=wobj0;
    Movej Target_30,v500,fine,t_pinza\WObj:=wobj0;
    Movej Target_20,v500,fine,t_pinza\WObj:=wobj0;
    Movej Target_10,v500,fine,t_pinza\WObj:=wobj0;
  ! Movel offs(Target_pieza,-posxint*2,50-posxint*2,0),v500,z100,t_pinza\WObj:=wobj0;
    WaitTime 1;
ENDPROC
```

Figura 25: Función Movimiento Robot

3.3 Flexpendant

Parte notable de RobotStudio, es la simulación de uno de verdad y se conecta internamente al robot; y se puede utilizar tanto para visualizar código como para controlar directamente las articulaciones y realizar operaciones mediante código RAPID (Figura 26).

Figura 26: *FlexPendant*

3.3.1 Manejo básico

Para usar la controladora hemos de saber previamente que dispone de diferentes opciones de tipo de ejecución. Estas son *Auto*, *Manual* y *veloc máxima manual* (Figura 27).

Figura 27: Botonería para control [2]

En la controladora virtual tambien contamos de estos modos. Para tener acceso a ellos debemos ir a la pestaña de *Controlador* y seleccionar el icono de *Panel de control*. Luego podremos seleccionar el modo que se desee [7].

Figura 28: Panel de control RobotStudio [2]

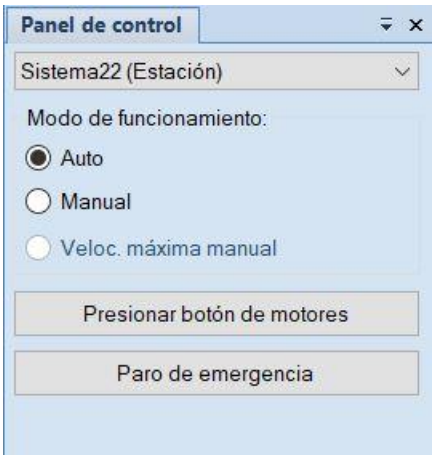


Figura 29: Pesaña Panel de control *RobotStudio* [2]

De vuelta al menu de la *Flexpendant* logramos ejecutar movimientos, hacienda uso de la opción *Jogging*. Al abrirse aparece una ventana donde tenemos dirección a las opciones de movimiento [7].

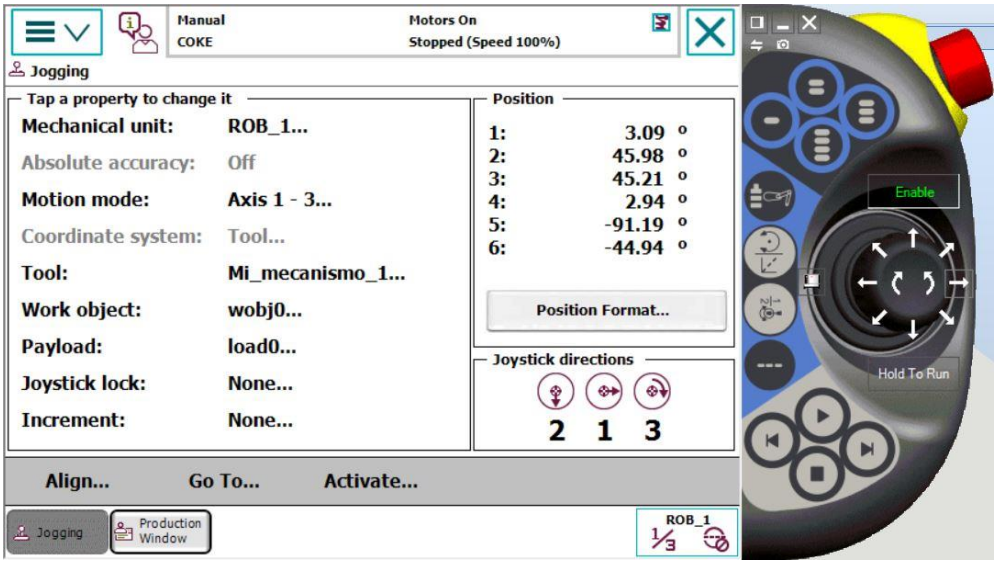


Figura 30: Ventana Jogging *FlexPendant RobotStudio* [2]

Se puede visualizar difrentes opciones. Las más relevantes en este y anteriores proyectos han sido la de *Tool* y *WorkObject*. Con la primera podemos decir que herramienta tenemos puesta en el robot y con la segunda respecto a que sistemas de coordenadas, o *WorkObject* como se nombra en *RobotStudio*. Si clicásemos sobre ellos nos aparecerían las diferestes herramientas y *WorkObjets*, deben haber sido creados previamente, se verá en apartados siguientes [7].

Al seleccionar *Motion Mode,* podemos elegir el tipo de movimiento que vamos a hacer. En el ejemplo de la ventana estaríamos controlando los ejes 1, 2 y 3 y es por ello que en el apartado position nos aparecen los grados a los que están los ejes del robot. Otra opción sería mover al robot de forma lineal. En este caso *Position* nos da las coordenadas en la que se encuentra el *TCP* de la herramienta en coordenadas cartesianas y respecto al *WorkObject* elegido ademas de sus correspondiente cuaterniones [7].

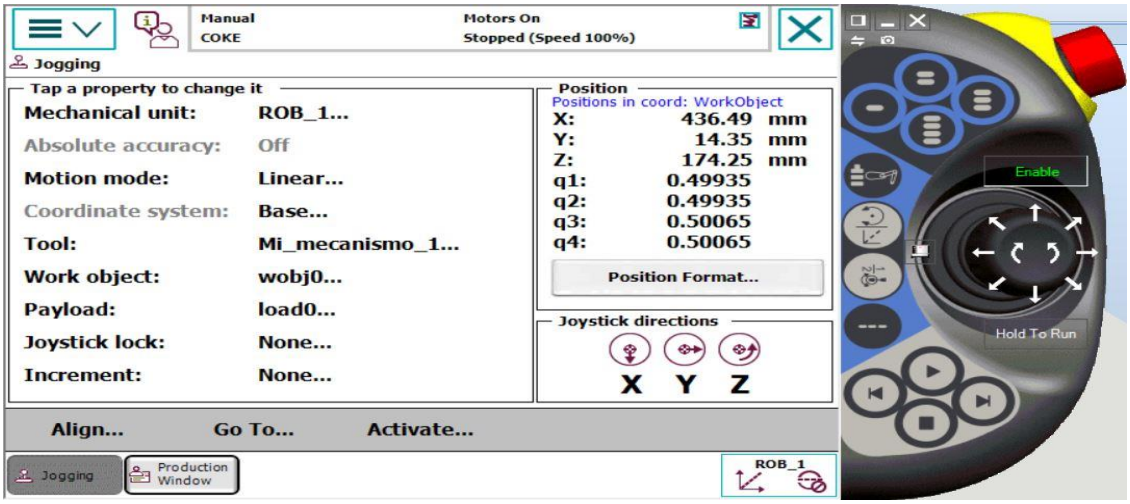


Figura 31: Ventana *Jogging Motion mode linear RobotStudio* [2]

Motion Mode también nos da la opción de hacer movimientos de reorientación sobre el *TCP* que hayamos definido [7].

Un paso que debemos dar antes de modificar la posición manualmente es habilitar la opción *Enable*. La controladora real tiene un botón en la parte trasera que nos permite habilitar el movimiento si la pulsamos (además recordar que debemos haber seleccionado *Manual* en el panel de control) y si liberamos el botón deja al robot muerto, sin poder hacer movimientos [7]. Haciendo uso del *joystick* podemos mover el robotcomo se desee tomando en cuenta sus limitaciones. Podemos de igual manera elegir la velocidad con el botón de velocidad [7].

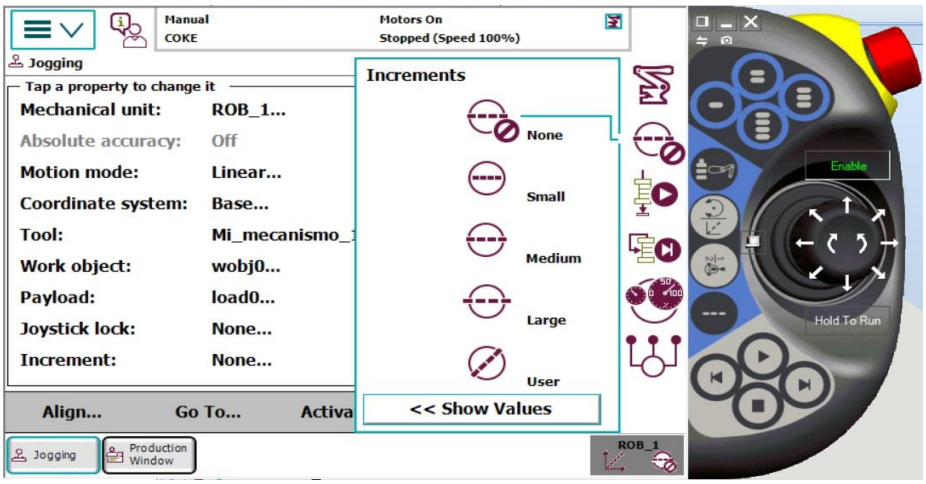


Figura 32: Menú cambio de velocidades *FlexPendant RobotStudio* [2]

4 DESARROLLO DE LA PROGRAMACIÓN EN ARDUINO

Una vez teminada la Sección de RobotStudio, es necesario programar las acciones de Arduino 2560, y ya que se tienen varias partes y varios sensores, se han hecho programas que prueban cada una de las fases de dasarrollo del proyecto del lado de Arduino. Estas partes son: El teclado matricial, el encoder incremental, la Shield de Ethernet, la pantalla LCD, control de motores, lectura de la señal obtenida en el calibrador digital, y que, varios de estos puntos fueron resueltos en trabajos anteriores.

4.1 Matriz de pulsadores

Los teclados de matriz son muy utilizados en los proyectos de electrónica debido a su versatilidad, pues permiten acomodar de una completa interfaz de entrada consumiendo un número mínimo de puertos de E/S. Para este trabajo se han utilizado varios caracteres, que serán especificados en secciones subsiguientes (Figura 33).

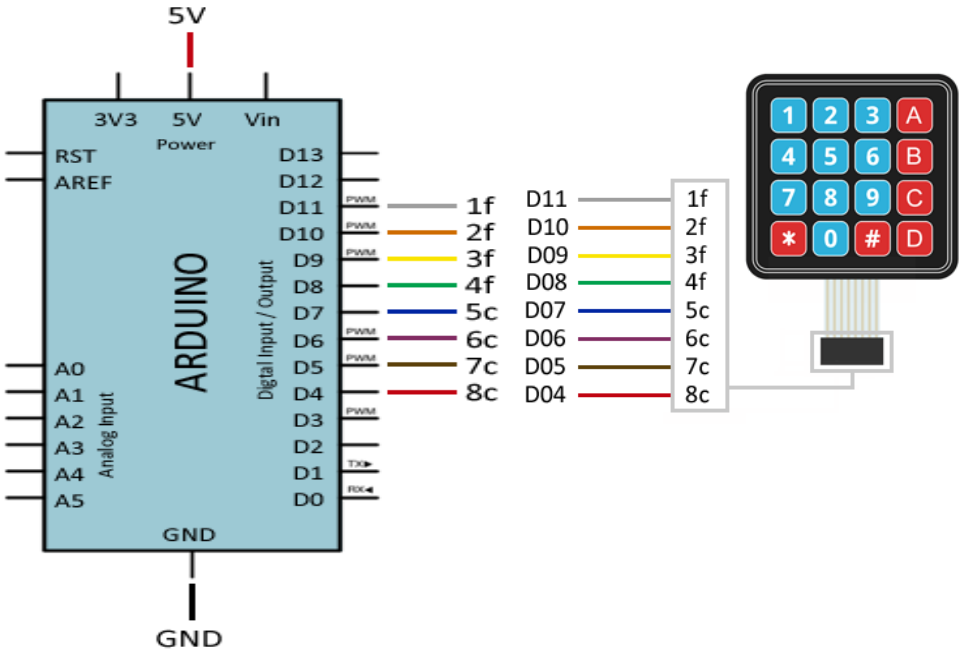


Figura 33: Matriz de pulsadores con Arduino

```
#include <Keypad.h>

const byte rowsCount = 4;
const byte columsCount = 4;

char keys[rowsCount][columsCount] = {
   { '1','2','3', 'A' },
   { '4','5','6', 'B' },
   { '7','8','9', 'C' },
   { '#','0','*', 'D' }
};

const byte rowPins[rowsCount] = { 23, 25, 27, 29 };
const byte columnPins[columsCount] = { 31, 33, 35, 37 };

Keypad keypad = Keypad(makeKeymap(keys), rowPins, columnPins, rowsCount,
columsCount);

void setup() {
   Serial.begin(9600);
}

void loop() {
   char key = keypad.getKey();

   if (key) {
      Serial.println(key);
   }
}
```

4.2 Control con Puente L298D

El módulo controlador de motores L298N H-bridge permite controlar la velocidad y la dirección de dos motores de corriente continua o un motor paso a paso de una forma muy sencilla, gracias a los 2 los dos H-bridge que posee.

El rango de tensiones en el que trabaja este módulo va desde 3V hasta 35V, y una intensidad de hasta 2A. A la hora de alimentarlo hay que tener en cuenta que la electrónica del módulo consume unos 3V, así que los motores reciben 3V menos que la tensión con la que alimentemos el módulo (Figura 34).

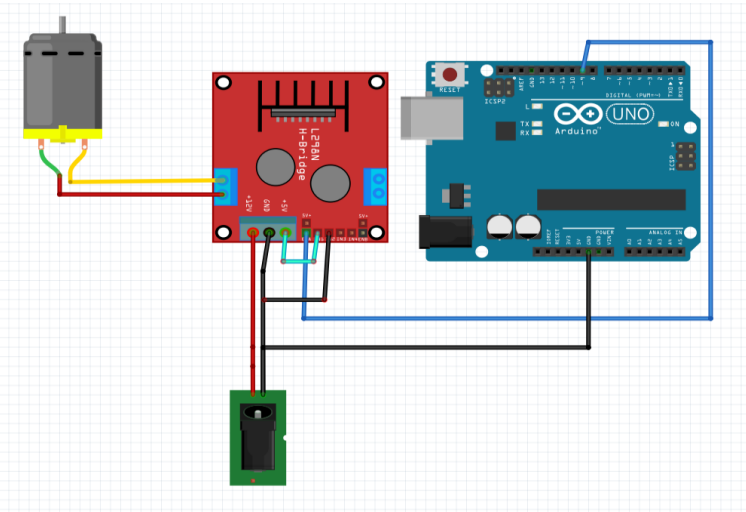


Figura 34: Puente H con Arduino

```
// constants won't change. Used here to set a pin number:
const int enaMotor =  14;// the number of the LED pin
const int motorizq =  15;// the number of the LED pin
const int motorder =  16;// the number of the LED pin
void setup() {
  // set the digital pin as output:
  pinMode(enaMotor, OUTPUT);
  pinMode(motorizq, OUTPUT);
  pinMode(motorder, OUTPUT);
}
void loop() {
  // here is where you'd put code that needs to be running all the
time.

  // check to see if it's time to blink the LED; that is, if the
difference
  // between the current time and last time you blinked the LED is
bigger than
  // the interval at which you want to blink the LED.

    digitalWrite(enaMotor, HIGH);
    digitalWrite(motorizq, HIGH);
    digitalWrite(motorder, LOW);
    delay(5000);
    digitalWrite(enaMotor, HIGH);
    digitalWrite(motorizq, LOW);
    digitalWrite(motorder, HIGH);
     delay(5000);
}
```

4.3 Puente H, encoder incremental y matriz de pulsadores

Se decidió trabajar con la matriz de pulsadores, ya que ocupan menos espacio y no hay que usar resistencias. La librería dedicada se encarga de obtener los pulsadores y eliminar falsos positivos. A demás se quiso manipular con el teclado la dirección y parada del motor conecado al Puente H. El programa descrito a continuación pone en acción en un sentido con el carácter *A*, con *B* en sentido contrario y con el caracter *3* para detener el motor. Con los mismos puertos descritos en las Figuras 33 Y 24, respectivamente.

```
#include <Keypad.h>

const byte rowsCount = 4;
const byte columsCount = 4;

char keys[rowsCount][columsCount] = {
  { '1', '2', '3', 'A' },
  { '4', '5', '6', 'B' },
  { '7', '8', '9', 'C' },
  { '#', '0', '*', 'D' }
};

const byte rowPins[rowsCount] = { 23, 25, 27, 29 };
const byte columnPins[columsCount] = { 31, 33, 35, 37 };

Keypad keypad = Keypad(makeKeymap(keys), rowPins, columnPins,
rowsCount, columsCount);


///variables para el encoder
float temp, counter = 0; //This variable will increase or decrease
depending on the rotation of encoder

// constants won't change. Used here to set a pin number:
const int enaMotor =  14;// the number of the LED pin
const int motorizq =  15;// the number of the LED pin
const int motorder =  16;// the number of the LED pin

void setup() {
  Serial.begin (9600);
  //pines para encoder
  pinMode(2, INPUT_PULLUP); // internal pullup input pin 2
  pinMode(3, INPUT_PULLUP); // internalเป็น pullup input pin 3
  //Setting up interrupt

  //interrupciones para encoder
  //A rising pulse from encodenren activated ai0(). AttachInterrupt 0
is DigitalPin nr 2 on moust Arduino.
  attachInterrupt(0, ai0, RISING);
  //B rising pulse from encodenren activated ai1(). AttachInterrupt 1
is DigitalPin nr 3 on moust Arduino.
  attachInterrupt(1, ai1, RISING);

  pinMode(enaMotor, OUTPUT);
  pinMode(motorizq, OUTPUT);
  pinMode(motorder, OUTPUT);
}
```

```
double vuelta = 0;
void loop() {
  // Send the value of counter
  // if ( counter != temp ) {
  char key = keypad.getKey();
  if(key == 'A'){
   vuelta=mapfloat(counter,0,1200,0,1);
   Serial.println (vuelta);
  motordelante();}
  if(key == '3'){
    vuelta=mapfloat(counter,0,1200,0,1);
   Serial.println (vuelta);
  motorparo();}
  if(key == 'B'){
    vuelta=mapfloat(counter,0,1200,0,1);
   Serial.println (vuelta);
  motoratras();}
}

float mapfloat(float val, float in_min, float in_max, float out_min,
float out_max) {
  return (val - in_min) * (out_max - out_min) / (in_max - in_min) +
out_min;
}
void ai0() {
  // ai0 is activated if DigitalPin nr 2 is going from LOW to HIGH
  // Check pin 3 to determine the direction
  if (digitalRead(3) == LOW) {
    counter++;
  } else {
    counter--;
  }

}
void ai1() {
  // ai0 is activated if DigitalPin nr 3 is going from LOW to HIGH
  // Check with pin 2 to determine the direction
  if (digitalRead(2) == LOW) {
    counter--;
  } else {
    counter++;

  }
}
void motordelante() {
  digitalWrite(enaMotor, HIGH);
  digitalWrite(motorizq, HIGH);
  digitalWrite(motorder, LOW);
}
void motoratras() {
  digitalWrite(enaMotor, HIGH);
  digitalWrite(motorizq, LOW);
  digitalWrite(motorder, HIGH);
}
void motorparo() {
  digitalWrite(enaMotor, LOW);
  digitalWrite(motorizq, LOW);
  digitalWrite(motorder, LOW);
}
```

4.4 Creación de un servidor IP con un puerto determinado.

En esta sección se probó creando un servidor en la dirección 192.168.1.200:4012 desde Arduino para que con RobotStudio se pudieran comunicar. Se crea el socket, luego se espera a que haya un servidor, se lee una instrucción y se procede a devolver dos valores. El programa es relativamente sencillo y cumplió con la función de ser el pilar más fundamental en el que sostendría este trabajo a largo plazo. Cada línea esta comentada con la acción que se realiza. Desde este punto, se puede notar que había ya que hacer cierto protocolo de comunicaciones sencillo para que ambos se pudieran entender, esto se puede ver desde la línea 76, función: corregircadena. El cliente se crea en RobotStudio y se indicá más adelante en la sección de RobotStudio.

```
#include <SPI.h>
#include <Ethernet.h>
unsigned char bufCAN[15];
unsigned int len2 = 0;
byte MAC[]=
{
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00 //direccion Mac generica (puede
ser cualquier otra mac
                                     // esta mac tiene que ser
diferente de la mac de todos los dispositivos en
                                     //la red
};

byte IP[]=
{
  192,168,1,200                    //Ip asiganda a la placa (aqui se
creará el socket)
};

EthernetServer server(4012);



void setup()
{
  Ethernet.begin(MAC, IP);
  server.begin();
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port
only
  }
  Serial.println("Ethernet WebServer Example");

}
char c;
void loop()
{
   int sensorValue = analogRead(A0);
   float voltage = sensorValue * (500.0 / 1023.0);
  comunicartcpip(voltage);
  delay(10);
}
```

```
void comunicartcpip(float x){
   EthernetClient client = server.available();
  int len=client.available(); //length de la cadena entrante
    if (len > 80) len = 80; //80bytes maximo de entrada en
buffer(solo referencia)
    int i =0; //paracontar caracteres
  while (client.connected()&& i<len) { //mientras cliente este
conectado y i<len
  if (client) //comprueba que cliente este abierto
  {
   c = client.read(); //lee el buffer
   Serial.print(c); //imprime en puerto serie como referencia
  }
  delay(10);  //tiempo de espera hasta que caracteres terminen de ser
leidos
  i++;
  }
  if (client.connected())
  {
   float z=-296.5;
   String a = String(z,2);
   String b= corregircadena(x);
   client.print(a);
   client.println(b);
   Serial.println(a);
   Serial.println(b);
   client.flush();
  delay(1);
  }
}


void paracomunicacion(){
EthernetClient client = server.available();
client.stop(); //cierra el socket
}


String corregircadena(float x){
  String result;
  if(x>=0){ //para x>0
    if(x>=0 && x<10){ //se añaden 0s a la cadena para enviar
       result= "0000"+String(x,2);
    }
    else{ if(x>=10 && x<100){
    result= "000"+String(x,2);
    }
    else{ if(x>=100 && x<1000){
    result= "00"+String(x,2);
    }
    else{ if(x>=1000 && x<10000){
    result= "0"+String(x,2);
    }
    }
    }
    }
  }
```

```
  if(x<0){//para x<0
    x=x*-1;
    if(x>=0 && x<10){
      result= "-000"+String(x,2); //se añaden 0 y el signo a la
cadena
    }
   else{if(x>=10 && x<100){
    result= "-00"+String(x,2);
    }
   else{ if(x>=100 && x<1000){
    result= "-0"+String(x,2);
    }
   else{if(x>=1000 && x<10000){
    result= '-'+String(x,2);
    }
    }
   }
   }
  }
  return result;
}
```

4.5 Funciones dependientes de la información enviadas desde RobotStudio

Una vez probado el servidor en Arduino, y que la información se recibe y envía correctamente, hacia y desde el Arduino a RobotStudio, se proceden a probar funciones: *leer, absoluto y relativo;* que serán interpretadas en base a un caracter enviado desde RobotStudio: *'l'* para leer, *'a'* para absoluto y *'r'* para relativo. El desarrollo está basado en el programa anterior y muestra en el serial la función que se haya enviado. Este apartado se desarrollo según los parámetros establecidos en el apartado 1.5 del primer capítulo.

```
#include <SPI.h>
#include <Ethernet.h>
//_______ VARIABLES DE TCP_IP
char bufCom[40]; //buffer de comunicaciones
unsigned int len2 = 0; //longitud de los caracteres a obtener de
robotstudio
byte MAC[] =
{
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00 //direccion Mac generica (puede
ser cualquier otra mac
  // esta mac tiene que ser diferente de la mac de todos los
dispositivos en
  //la red
};

byte IP[] =
{
  192, 168, 1, 200                  //Ip asiganda a la placa (aqui se
creará el socket)
};
//_____________________________________
```

```
//_____________________________________

//__________VARIABLES PARA ENVIO DE DATOS
//-----
int estado;
//____comando leer
float posx = 35.2;
float posy = -25;
//-------------

EthernetServer server(4012);

void setup() {
  Ethernet.begin(MAC, IP);
  server.begin();
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port
only
  }
  Serial.println("Ethernet WebServer Example");
  pinMode(13, OUTPUT);
  delay(10);
}

void loop() {
  int sensorValue = analogRead(A0);
  float voltage = sensorValue * (500.0 / 1023.0);
  comunicartcpip();
  delay(11);

}
int len = 0;
void comunicartcpip() {

  EthernetClient client = server.available();
  len  = client.available(); //length de la cadena entrante
  if (len > 80) {
    len = 80;
  } //80bytes maximo de entrada en buffer(solo referencia)
  int i = 0; //paracontar caracteres
  while (i < len) { //mientras cliente este conectado y i<len
    if (client && i < len) //comprueba que cliente este abierto
    {
      bufCom[i] = client.read(); //lee el buffer
      Serial.println(bufCom[i]); //imprime en puerto serie como
referencia
      // Serial.println(i);
    }
    //   delay(1);  //tiempo de espera hasta que caracteres terminen
de ser leidos
    i++;
    // delay(20);

  }
```

```
//client.flush();
  if (client.connected())  {
    funcioncinta(bufCom);
    String a = corregircadena(estado);
    String b = corregircadena(posx);
    String c = corregircadena(posy);
    String d = a + b + c;
    while (estado == 0) {
      ;
    }
    client.println(d);
    Serial.println(d);

    client.flush();

  }
  //  client.stop(); //cierra el socket
}

void funcioncinta(char bufCm[40]) { //se verifica que funcion debe
tener la cinta dependiendo de lo que
  //el cliente solicite
  String cadena;
  switch (bufCm[0]) {
    case 'l':  //funcion leer
      estado = 0;
      //Serial.println(funcion);
      digitalWrite(13, !digitalRead(13));
      // posx=-35.5;
      // posy=23;
      estado = 11;
      break;
    case 'a': //funcion movimiento absoluto
      estado = 0;
      cadena = "";
      //Serial.println(funcion);
      for (int i = 1; i < len; i++) {
        cadena = cadena + bufCm[i];
      }

      delay(5000);
      Serial.println(cadena);

      estado = 12;
      break;
    case 'r' : //funcion movimiento relativo
      estado = 0;
      cadena = "";
      //Serial.println(funcion);
      for (int j = 1; j < len; j++) {
        cadena = cadena + bufCm[j];
      }
```

```
delay(5000);
      Serial.println(cadena);

      estado = 13;
      break;
    default:
      estado = 1; //ningun caso coincide
      Serial.println("El caracter enviado desde RobotStudio no
coincide con ninguna funcion");
      Serial.println("Enviar: l (leer) a (absoluto) o r (relativo)
");
      break;
  }
  bufCm[0] = 'x';
}



String corregircadena(float x) { //corrigue y añade 0 al número para
enviar al cliente
  String result;
  if (x >= 0) { //para x>0
    if (x >= 0 && x < 10) { //se añaden 0s a la cadena para enviar
      result = "0000" + String(x, 2);
    }
    else {
      if (x >= 10 && x < 100) {
        result = "000" + String(x, 2);
      }
      else {
        if (x >= 100 && x < 1000) {
          result = "00" + String(x, 2);
        }
        else {
          if (x >= 1000 && x < 10000) {
            result = "0" + String(x, 2);
          }
        }
      }
    }
  }
  if (x < 0) { //para x<0
    x = x * -1;
    if (x >= 0 && x < 10) {
      result = "-000" + String(x, 2); //se añaden 0 y el signo a la
cadena
    }
    else {
      if (x >= 10 && x < 100) {
        result = "-00" + String(x, 2);
      }
      else {
        if (x >= 100 && x < 1000) {
          result = "-0" + String(x, 2);
        }
```

```
        else {
          if (x >= 1000 && x < 10000) {
            result = '-' + String(x, 2);
          }
        }
      }
    }
  }
  return result;
}
```

4.6 Version final del programa desarrollado en Arduino

Todos estos programas podrán ser cargados directamente a Arduino para las pruebas correspondientes, cada valor y función desarrollada cumple un objetivo único y sólo para este tabajo. Y para finalizar, el desarrollo mismo se creo la versión final que toma en cuenta: la matriz de pulsadores, la comunicación serial, el encoder, la comunicación ethernet, la definición de funciones y la creación de un buffer para la lectura y envío de datos Figura 37.

En la siguiente Figura (35), se muestra el esquema físico de conexión de ATMega328, microcontrolador de arduino Mega al shield de ethernet. El montaje Ethernet-Arduino se muestra en la Figura 36. (Esquema adjunto en los anexos).

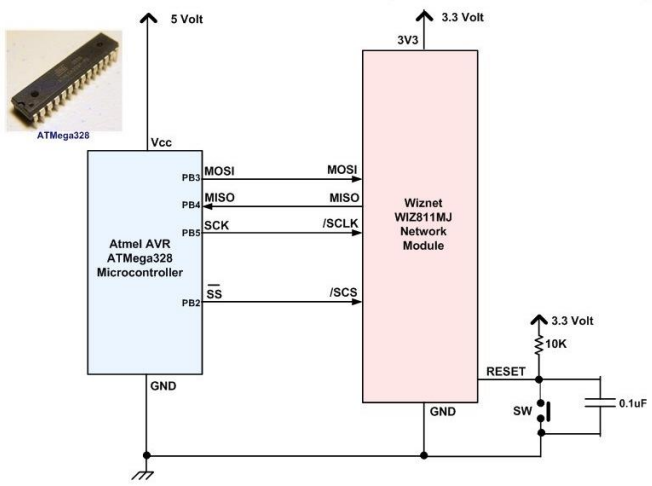


Figura 35: Esquema de conexión Arduino-Ethernet

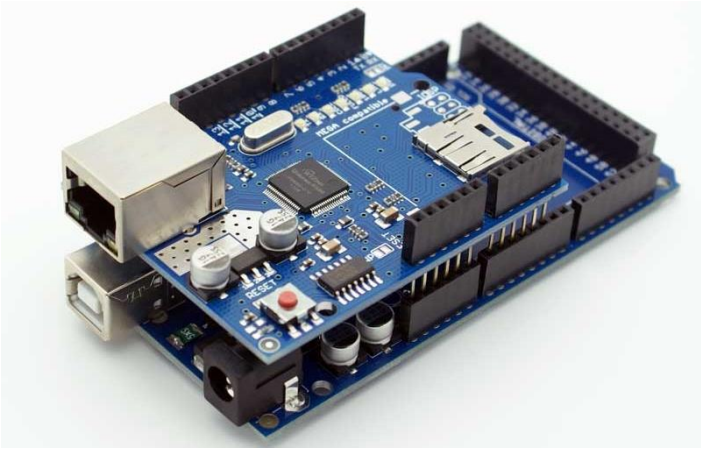


Figura 36: Montaje Arduino-Ethernet

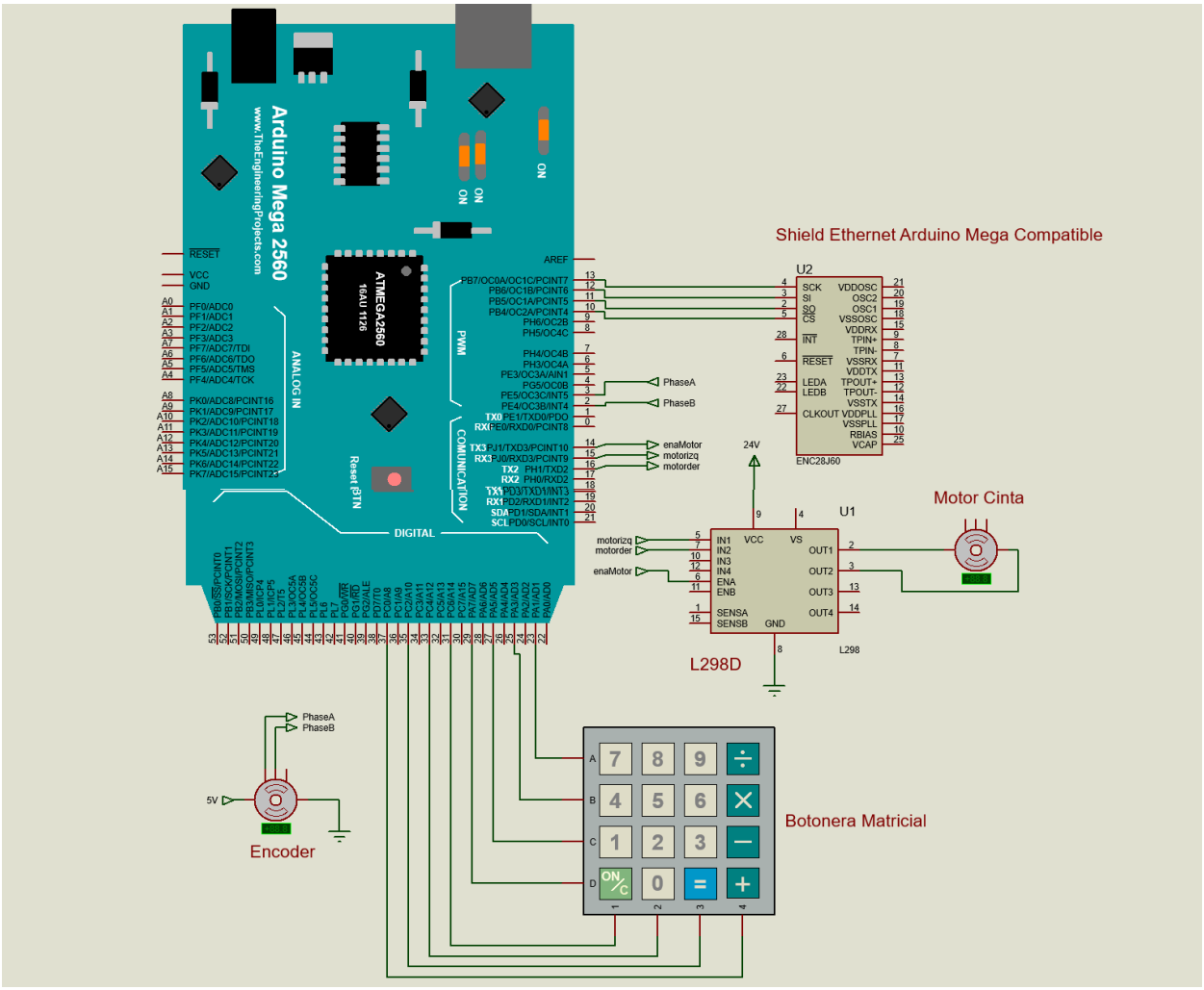


Figura 37: Conexión final, esquema cableado de Arduino y banda transportadora

```
#include <SPI.h> //libreria comunicacion ethernet
#include <Ethernet.h> //libreria ethernet
#include <Keypad.h> //libreria para el teclado matricial

//_______ VARIABLES para crear servidor TCP_IP
char bufCom[40]; //buffer de comunicaciones
unsigned int len2 = 0; //longitud de los caracteres a obtener de
robotstudio
byte MAC[] =
{
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00 //direccion Mac generica (puede
ser cualquier otra mac
  // esta mac tiene que ser diferente de la mac de todos los
dispositivos en
  //la red
};

byte IP[] =
{
  192, 168, 1, 200                  //Ip asiganda a la placa (aqui se
creará el socket)
};
//-------------------------------------------------------
```

```
//__________VARIABLES PARA ENVIO DE DATOS por tcpip
//-----
int estado;
//____comando leer
float posx = 0;
float posy = 0;
float posxaux = 0; //auxiliar para operaciones
//-------------------------------------------------------

//__________VARIABLES PARA tratamiento del teclado matricial
//-----


//          A= local
//          B=remoto
//          C=motor delante (local)
//          D=motor atras (local)
//          *=motro paro (local)
//          3=sensor
//
//
//
const byte rowsCount = 4;
const byte columsCount = 4;

char keys[rowsCount][columsCount] = {
  { '1', '2', '3', 'A' },
  { '4', '5', '6', 'B' },
  { '7', '8', '9', 'C' },
  { '#', '0', '*', 'D' }
};

const byte rowPins[rowsCount] = { 23, 25, 27, 29 };
const byte columnPins[columsCount] = { 31, 33, 35, 37 };

Keypad keypad = Keypad(makeKeymap(keys), rowPins, columnPins,
rowsCount, columsCount);

//-------------------------------------------------------

//__________VARIABLES PARA el encoder rotacional
//-----
float temp, counter = 0; //This variable will increase or decrease
depending on the rotation of encoder

// constants won't change. Used here to set a pin number:
const int enaMotor =  14;// the number of the LED pin
const int motorizq =  15;// the number of the LED pin
const int motorder =  16;// the number of the LED pin

//-------------------------------------------------------


//Aqui se inicia el servidor en el puerto específico
EthernetServer server(4012);
//-------------------------------------------------------
```

```
void setup() {
  //pines para encoder
  pinMode(2, INPUT_PULLUP); // internal pullup input pin 2
  pinMode(3, INPUT_PULLUP); // internalเป็น pullup input pin 3
  //interrupciones para encoder
  //A rising pulse from encodenren activated ai0(). AttachInterrupt 0
is DigitalPin nr 2 on moust Arduino.
  attachInterrupt(0, ai0, RISING);
  //B rising pulse from encodenren activated ai1(). AttachInterrupt 1
is DigitalPin nr 3 on moust Arduino.
  attachInterrupt(1, ai1, RISING);
  //--------------------------------------------------------

  //Estados de pines para Encoder
  pinMode(enaMotor, OUTPUT);
  pinMode(motorizq, OUTPUT);
  pinMode(motorder, OUTPUT);
  digitalWrite(enaMotor, LOW);
  //--------------------------------------------------------

  //inicializa la funcion ethernet
  Ethernet.begin(MAC, IP); //Inicia la funcion Ethernet con la mac
  server.begin(); //inicia el servidor
  Serial.begin(9600); //inicia comunicacion serial
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port
only
  }
  Serial.println("Espera");
  pinMode(13, OUTPUT); //Cambia de estado cuando se llama la funcion
Leer desde RbotStudio
  delay(100); // tiempo de espera hasta que arduino despierte
  //--------------------------------------------------------

  //inicia con la cinta parada
  motorparo();
  delay(2000);
  Serial.println ("Cinta inicia detenida");
}

int cinta = 0; //variable auxiliar para saber el estado de la cinta
int error = 0; //variable indica que se ha cambiado de remoto a local
mientras se comunica con Robot Studio
char key;  //para decidir si local o remoto 'A' para local

void loop() {
  key = keypad.getKey(); //para decidir si local o remoto 'A' para
local
local:;
  if (key == 'A') { //local
local2:;
   motorparo();
    cinta = 1;
    Serial.println("local");
    char key2 = '*'; //key2 variable asociada al movimiento de la
cinta
```

```
while (key2 != 'B') { //mientras no se cambie a remoto
      key2 = keypad.getKey(); //obtiene la direccion del movimiento
de la cinta
      if (key2 == 'B') { //sale del ciclo si == 'B'
        key = 'B';
        goto remoto;//goto para salir del ciclo while
      }
      if (key2 == 'C') { //movimiento para adelante
        posx = mapfloat(counter, 0, 1200, 0, 1);
        Serial.println (posx);
        motordelante();
      }

      if (key2 == 'D') { //movimiento en reverza
        posx = mapfloat(counter, 0, 1200, 0, 1);
        Serial.println (posx);

        motoratras();
      }
      if (key2 == '*') {
        posx = mapfloat(counter, 0, 1200, 0, 1);
        Serial.println (posx);
        motorparo();
      }
    }
    key2 = '*';
    motorparo();
  }
remoto:;
  if (key == 'B') { ////para decidir si local o remoto 'B' para
remoto
    motorparo();
    cinta = 2;
    Serial.println("remoto");
    char key2 = '*'; // Key2 variable asociada al movimiento de la
cinta
    while (key2 != 'A') {
      key2 = keypad.getKey(); //obtiene la direccion del movimiento
de la cinta
      if (key2 == 'A') { //sale del ciclo si == 'B'
        key = 'A';
        error = 2;
        comunicartcpip();
        if (error == 2) {
          error = 0;
          goto local2;//goto para salir del ciclo while
        }
      }
      posx = mapfloat(counter, 0, 1200, 0, 1);
      comunicartcpip();
      if (error == 2) {
        error = 0;

        goto local2;//goto para salir del ciclo while
      }
```

```
    }
    key2 = '*';
  }

  // comunicartcpip();
  delay(11);

}
int len = 0;
void comunicartcpip() {

  EthernetClient client = server.available();
  client.flush();

  len  = client.available(); //length de la cadena entrante
  if (len > 80) {
    len = 80;
  } //80bytes maximo de entrada en buffer(solo referencia)
  int i = 0; //paracontar caracteres
  while (i < len) { //mientras cliente este conectado y i<len
    if (client && i < len) //comprueba que cliente este abierto
    {
      bufCom[i] = client.read(); //lee el buffer
      // Serial.println(bufCom[i]); //imprime en puerto serie como
referencia
      //  Serial.println (bufCom[i]);
    }
    i++;
  }
  delay(10);  //tiempo de espera hasta que caracteres terminen de ser
leidos


  //client.flush();
  if (client.connected())  {
    if (keypad.getKey() == 'A') {
      error = 2;
      Serial.println("se cambio de remoto a local se estaba rec");
    }
    funcioncinta(bufCom); //determina la funcion a realizar enviada
desde RobotStudio
    String d = cadenaenviar();
    while (estado == 0) {
      ;
    }
    client.println(d);
    Serial.println(d);

    client.flush();

  }
  //  client.stop(); //cierra el socket
}
```

```
void funcioncinta(char bufCm[40]) { //se verifica que funcion debe
tener la cinta dependiendo de lo que
  //el cliente solicite
  String cadenadistancia;
  String pasos;
  if (bufCm[0] == 'a') {
    estado = 0;
    cadenadistancia = "";
    //Serial.println(funcion);
    for (int i = 1; i < len; i++) {

      cadenadistancia = cadenadistancia + bufCm[i];
    }

    posx = mapfloat(counter, 0, 1200, 0, 1); //posx
    float valorposicionx = cadenadistancia.toFloat(); //transforma el
dato del movimiento en un flotante para mover la cinta
    posxaux = valorposicionx + posx; //para comparar valores
    long tiempoespera = millis() + 10000; //TIEMPO DE ESPERA
    long tiempoactual = millis();   //TIEMPO QUE TRANSCURRIRA

    if (valorposicionx > 0) { //si valor posicion para ser movido es
positivo
      motordelante(); //mover motor adelante
      while (posx - posxaux <= 0) {
        if (keypad.getKey() == 'A') {
          error = 2;
          motorparo();
          break;
        }
        tiempoactual = millis(); //tiempo por si no se ejecuta la
orden
        float posxnew;
        posx = mapfloat(counter, 0, 1200, 0, 1);
        if (tiempoactual == tiempoespera - 10000) {
          posxnew = posx; //se asigna el valor de posx a posxnew en
el tiempo 0
        }
        if (tiempoactual >= tiempoespera) { //espera los 10 segundos
          if (posxnew == posx) { //si no hay cambios en la posicion
en x
            error = 3; //se asigna el codigo de error para enviar
            motorparo();
            break;
          }
        }
        delay(1);
      }
    }
    if (valorposicionx < 0) {
      motoratras();
      if (keypad.getKey() == 'A') {
        error = 2;
        motorparo();
```

```
goto finproceso;
      }
      if (valorposicionx + posx < 0) { //si el valor es negativo
entonces esta fuera de rango
        error = 4; //codigo de error
        motorparo();
        goto finproceso;
      }
      while (posx - posxaux >= 0) {
        tiempoactual = millis();
        float posxnew;
        posx = mapfloat(counter, 0, 1200, 0, 1);
        if (tiempoactual == tiempoespera - 10000) {
          posxnew = posx;//se asigna el valor de posx a posxnew en el
tiempo 0
        }
        if (tiempoactual >= tiempoespera) {
          if (posxnew == posx) {//si no hay cambios en la posicion en
x
            error = 3; //se asigna el codigo de error para enviar
            motorparo();
            break;
          }
        }
        delay(1);
      }
    }
    motorparo();
    delay(1);
    estado = 12;
  }

  //CASO Leer enviado desde Robotstudio
  else if (bufCm[0] == 'l') { //funcion leer
    estado = 0;
    //Serial.println(funcion);
    digitalWrite(13, !digitalRead(13));
    // posx=-35.5;
    // posy=23;
    estado = 11;
  }

  else if (bufCm[0] == 'r') { //funcion movimiento relativo
    estado = 0;
    cadenadistancia = "";
    //Serial.println(funcion);
    int j = 0;
    for (j = 1; j < len; j++) {
      if (bufCm[j] == 'N') {
        goto findistancia;
      }
      cadenadistancia = cadenadistancia + bufCm[j];
    }
findistancia:;
```

```
  for (int k = j + 1; k < len; k++) {
      pasos = pasos + bufCm[k];
    }

    int cont = 0;
    int pasosfloat = pasos.toInt();
    while (cont < pasosfloat) {
      posx = mapfloat(counter, 0, 1200, 0, 1); //posx
      float valorposicionx = cadenadistancia.toFloat(); //transforma
el dato del movimiento en un flotante para mover la cinta
      posxaux = valorposicionx + posx; //para comparar valores
      long tiempoespera = millis() + 10000; //TIEMPO DE ESPERA
      long tiempoactual = millis();   //TIEMPO QUE TRANSCURRIRA

      if (valorposicionx > 0) { //si valor posicion para ser movido
es positivo
        motordelante(); //mover motor adelante
        while (posx - posxaux <= 0) {
          if (keypad.getKey() == 'A') {
            error = 2;
            motorparo();
            goto finproceso;
          }
          tiempoactual = millis(); //tiempo por si no se ejecuta la
orden
          float posxnew;
          posx = mapfloat(counter, 0, 1200, 0, 1);
          if (tiempoactual == tiempoespera - 10000) {
            posxnew = posx; //se asigna el valor de posx a posxnew en
el tiempo 0
          }
          if (tiempoactual >= tiempoespera) { //espera los 10
segundos
            if (posxnew == posx) { //si no hay cambios en la posicion
en x
              error = 3; //se asigna el codigo de error para enviar
              motorparo();
              posxnew=0;
              goto finproceso;
            }

          }
          delay(1);
        }
      }
       if (valorposicionx < 0) {
        motoratras();
        if (keypad.getKey() == 'A') {
          error = 2;
          motorparo();
          goto finproceso;
        }
```

```
        if (valorposicionx + posx < 0) { //si el valor es negativo
entonces esta fuera de rango
          error = 4; //codigo de error
          motorparo();
          goto finproceso;
        }
        while (posx - posxaux >= 0) {
          tiempoactual = millis();
          float posxnew;
          posx = mapfloat(counter, 0, 1200, 0, 1);
          if (tiempoactual == tiempoespera - 10000) {
            posxnew = posx;
          }
          if (tiempoactual >= tiempoespera) {
            if (posxnew == posx) {
              error = 3;
              motorparo();
              goto finproceso;
            }
          }
          delay(1);

        }
      }
      motorparo();
      if(error==3)goto finproceso;
      Serial.println(error);
      cont++;
      delay(1000);
    }
    estado = 13;
  }
  if (estado != 11 && estado != 12 && estado != 13) {
    estado = 1;
    Serial.println("El caracter enviado desde RobotStudio no coincide
con ninguna funcion");
    Serial.println("Enviar: l (leer) a (absoluto) o r (relativo) ");
  }

finproceso:;
  bufCm[0] = '+'; //reiniciar funcion en el buffer
  if (error == 2) {
    estado = 2;
    Serial.println("Se cambio de remoto a local 2");
  }
  if (error == 3) {
    estado = 3;
    Serial.println("Encoder no funciona 3");
  }

  if (error == 4) {
    estado = 4;
    Serial.println("Valor absoluto negativo demasiado grande 4");
  }
}
```

```
String cadenaenviar() { //funcion para ingresar todos los datos:
estado de la cinta
  // posx y pos y para ser enviados por tcpip
  String a = corregircadena(estado); //estado
  String c = corregircadena(posx); //posy
  String b = corregircadena(posy); //pos x
  String d = a + b + c; //estado, posy, pos x
  return d;
}
String corregircadena(float x) { //corrigue y añade 0 al número para
enviar al cliente
  String result;
  if (x >= 0) { //para x>0
    if (x >= 0 && x < 10) { //se añaden 0s a la cadena para enviar
      result = "0000" + String(x, 2);
    }
    else {
      if (x >= 10 && x < 100) {
        result = "000" + String(x, 2);
      }
      else {
        if (x >= 100 && x < 1000) {
          result = "00" + String(x, 2);
        }
        else {
          if (x >= 1000 && x < 10000) {
            result = "0" + String(x, 2);
          }
        }
      }
    }
  }
  if (x < 0) { //para x<0
    x = x * -1;
    if (x >= 0 && x < 10) {
      result = "-000" + String(x, 2); //se añaden 0 y el signo a la
cadena
    }
    else {
      if (x >= 10 && x < 100) {
        result = "-00" + String(x, 2);
      }
      else {
        if (x >= 100 && x < 1000) {
          result = "-0" + String(x, 2);
        }
        else {
          if (x >= 1000 && x < 10000) {
            result = '-' + String(x, 2);
          }
        }
      }
    }
  }
  return result;
}
```

```
void motordelante() { //motor avanza
  digitalWrite(enaMotor, HIGH);
  digitalWrite(motorizq, HIGH);
  digitalWrite(motorder, LOW);
  Serial.println("Motor avanza");
}
void motoratras() { //motor retrocede
  digitalWrite(enaMotor, HIGH);
  digitalWrite(motorizq, LOW);
  digitalWrite(motorder, HIGH);
  Serial.println("Motor retrocede");
}
void motorparo() { //motor para
  digitalWrite(enaMotor, LOW);
  digitalWrite(motorizq, LOW);
  digitalWrite(motorder, LOW);
  Serial.println("Paro motor");
}


//-----------Interrupcion para incrementar el contador del encoder
void ai0() {
  // ai0 is activated if DigitalPin nr 2 is going from LOW to HIGH
  // Check pin 3 to determine the direction
  if (digitalRead(3) == LOW) {
    counter++;
  } else {
    counter--;
  }

}

//-----------Interrupcion para decrementar el contador del encoder
void ai1() {
  // ai0 is activated if DigitalPin nr 3 is going from LOW to HIGH
  // Check with pin 2 to determine the direction
  if (digitalRead(2) == LOW) {
    counter--;
  } else {
    counter++;

  }
}

//-----------Funcion map para el conteo de giros del encoder
float mapfloat(float val, float in_min, float in_max, float out_min,
float out_max) {
  return (val - in_min) * (out_max - out_min) / (in_max - in_min) +
out_min;
}
```

5 DESARROLLO DE LA PROGRAMACIÓN EN ROBOTSTUDIO

Establecidas las pruebas del capitulo anterior y su respectivo desarrollo, se fueron probando y creando los codigos de RobotStudio que permitieron comprobar la comunicación y el envío de datos; así como la de las funciones y la corrección de errores. Todo esto permitió realizar versiones de prueba de código en RAPID para sostener al cliente ethernet y también de ser el encargado de mandar las funciones solicitadas a la cinta (Arduino).

5.1 Conexión a un servidor IP Montado en arduino.

Este apartado corresponde a la creación del puerto IP en la sección 4.4 del capítulo anterior; y tiene que ver con la conexión básica e intercambio de datos entre servidor y cliente, el servidor montado en Arduino y el Cliente siendo creado y conectado desde RobotStudio. En esta sección se crea el puerto simplemente llamando a la función *SocketConect* de RAPID más el puerto IP que fue creado en Arduino, hay que colocar cualquiera de los 3 caracteres en la salida para que el programa no de ningún error. Para probar basta copiar el código tal como se muestra a continuación.

```
MODULE Module1
VAR socketdev my_socket;
VAR rawbytes data;
VAR rawbytes raw_data;
VAR byte nuevo:=0;
VAR num n;
VAR string n2;
var num float;
var num float2;
VAR byte data_buffer;
VAR bool ok:=true;
VAR num i:=1;
VAR rawbytes receive_string;
VAR string string1;

    PROC main()
        !Añada aquí su código
        !SocketCreate server;
    SocketCreate my_socket; !crea el socket
    SocketConnect my_socket, "192.168.1.200", 4012; !lo enlace a la
direccion y al puerto correspondiente al arduino
    WHILE ok=TRUE DO
    SocketSend my_socket,\Str:="a23"; !escribe en la red y lo envia a
arduino (donde será leido)
    WaitTime 0.5; !espera un tiempo
    SocketReceive my_socket \RawData :=
receive_string,\Time:=WAIT_MAX;

    UnpackRawBytes receive_string, 1, string1 \ASCII:=7; !7 es el
espacio en caracteres que forman el número
    UnpackRawBytes receive_string, 8, n2 \ASCII:=8;
    ok:=StrToVal(string1,float);
    ok:=StrToVal(n2,float2);
```

```
    ENDWHILE
     !TPWrite "Instruction SocketClose has been executed";


   !UnpackRawBytes data,1,nuevo,\ASCII:=15;
    !!UnpackRawBytes data,2,nuevo,\ASCII:=1;
    !UnpackRawBytes data,3,nuevo1,\ASCII:=1;
   ! UnpackRawBytes data,4,nuevo2,\ASCII:=1;

    SocketClose my_socket; !cierra el socket


    ENDPROC
ENDMODULE
```

5.2 Funciones enviadas desde RobotStudio.

En este apartado se consideran las funciones especificadas en el apartado 1.5. Se crean funciones según los caracteres aceptados en Arduino, cada función se ejecuta una vez, primero leer, luego absoluto y relativo; para todas estas funciones se añade un valor que será leído en Arduino, y luego se recibirá tres valores, un ack, pasx y posy. Esta sección está relacionada con la sección 4.5 del capítulo anterior.

```
MODULE Module1
 VAR socketdev my_socket;

VAR num n;
VAR string posx;
VAR string posy;
VAR num posxint;
VAR num posyint;
VAR num estad;
VAR bool ok;
VAR bool okposx;
VAR bool okposy;
VAR bool okack;
VAR num i:=1;
VAR rawbytes receive_string; !recibe un paquete de datos en string
VAR string ack;

    PROC main()
        !Añada aquí su código
        abrircomunicacion;
        leer;
        WaitTime 1;
        absoluto;
        WaitTime 1;
        relativo;
  ! WHILE ok=TRUE DO

    ! ENDWHILE
     !TPWrite "Instruction SocketClose has been executed";
    ENDPROC
    PROC abrircomunicacion()
```

```
    SocketCreate my_socket; !crea el socket
    SocketConnect my_socket, "192.168.1.200", 4012; !lo enlace a la
direccion y al puerto correspondiente al arduino

    ENDPROC
    PROC leer()
   ! ClearRawBytes receive_string;
    WaitTime 1;
    SocketSend my_socket,\Str:="l230"; !escribe en la red y lo envia a
arduino (donde será leido)
    WaitTime 0.1; !espera un tiempo

    SocketReceive my_socket \RawData := receive_string,\Time:=WAIT_MAX;
     WaitTime 0.1; !espera un tiempo
    UnpackRawBytes receive_string, 1, ack \ASCII:=8; !8 es el espacio
en caracteres que forman el número
    UnpackRawBytes receive_string, 9, posx \ASCII:=8;
    UnpackRawBytes receive_string, 17, posy \ASCII:=8;
    !UnpackRawBytes receive_string, 1, posy \ASCII:=24;
    !ok:=StrToVal(ack,float);
    okposx:=StrToVal(posx,posxint);
    okposy:=StrToVal(posy,posyint);
    okack:=StrToVal(ack,estad);
    ClearRawBytes receive_string;
    ENDPROC
    PROC absoluto()
    !ClearRawBytes receive_string;
    WaitTime 1;
    SocketSend my_socket,\Str:="a899"; !escribe en la red y lo envia a
arduino (donde será leido)
    WaitTime 0.1; !espera un tiempo

    SocketReceive my_socket \RawData := receive_string,\Time:=WAIT_MAX;
     WaitTime 0.1; !espera un tiempo
    UnpackRawBytes receive_string, 1, ack \ASCII:=8; !8 es el espacio
en caracteres que forman el número
    UnpackRawBytes receive_string, 9, posx \ASCII:=8;
    UnpackRawBytes receive_string, 17, posy \ASCII:=8;
    okposx:=StrToVal(posx,posxint);
    okposy:=StrToVal(posy,posyint);
    okack:=StrToVal(ack,estad);
    ClearRawBytes receive_string;
    ENDPROC
    PROC relativo()
    !ClearRawBytes receive_string;
    WaitTime 1;
    SocketSend my_socket,\Str:="r19"; !escribe en la red y lo envia a
arduino (donde será leido)
    WaitTime 0.1; !espera un tiempo

    SocketReceive my_socket \RawData := receive_string,\Time:=WAIT_MAX;
     WaitTime 0.1; !espera un tiempo
    UnpackRawBytes receive_string, 1, ack \ASCII:=8; !8 es el espacio
en caracteres que forman el número
    UnpackRawBytes receive_string, 9, posx \ASCII:=8;
    UnpackRawBytes receive_string, 17, posy \ASCII:=8;
    okposx:=StrToVal(posx,posxint);
    okposy:=StrToVal(posy,posyint);
    okack:=StrToVal(ack,estad);
```

```
    ClearRawBytes receive_string;
    ENDPROC

    PROC estado()

    ENDPROC
    PROC cerrarcomunicacion()
    SocketClose my_socket; !cierra el socket
    ENDPROC

ENDMODULE
```

5.3 Version final del programa desarrollado en Arduino

Al igual que en la versión final para Arduino, se ha creado un código en base a los anteriores y se han añadido funciones exteriores al programa principal o Main; a demás, se han complementado los errores y movimientos asociados a la posición X e Y que se señalan en el primer capítulo; por lo que, ahora, el robot simulado en RobotStudio se mueve dependiendo de la posición que se envie desde Arduino. Cada una de las secciones del código están explicadas en la sección 3.2 del Capítulo 3. Esta sección esta relacionada con la sección 4.6.

```
MODULE Module1
    VAR socketdev my_socket;

    VAR num n;
    VAR string posx;
    VAR string posy;
    VAR string cadena;
    VAR num posxint;
    VAR num posyint;
    VAR num estad:=0;
    VAR bool ok;
    VAR bool okposx;
    VAR bool okposy;
    VAR bool okack;
    VAR num i:=1;
    VAR rawbytes receive_string;
    !recibe un paquete de datos en string
    VAR string ack;
    CONST robtarget
Target_pieza:=[[436.2339,144,180.9312],[0.70711,0,0,0.70711],[0,-1,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget
Target_10:=[[436.2339,144.422,389.7914],[0.70711,0,0,0.70711],[0,-1,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_20:=[[-
101.0329,144.422,389.7915],[0.70711,0,0,0.70711],[1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_30:=[[-
101.0329,144.422,173.8547],[0.70711,0,0,0.70711],[1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    PROC main()

        !Añada aquí su código
        SocketClose my_socket;
        abrircomunicacion;
        WaitTime 1;
        !  leer;
        !  absoluto 5;  !!el valor hace referencia a la distancia que
se va a recorrer por la cinta en x
        ! relativo 4, 3; !!el primer valor hace referencia a la
distancia que se va a recorrer por la cinta en x
        !!el segundo hace referencia al número de saltos que se va a
recorrer por la cinta en x
        !absoluto 5;
         leer;
        ! WHILE ok=TRUE DO

        ! ENDWHILE
        !TPWrite "Instruction SocketClose has been executed";
    ENDPROC

    PROC abrircomunicacion()

        SocketCreate my_socket;
        !crea el socket
        SocketConnect my_socket,"192.168.1.200",4012;
        !lo enlace a la direccion y al puerto correspondiente al
arduino

    ENDPROC


    PROC leer()
        ! ClearRawBytes receive_string;
        WaitTime 1;
        SocketSend my_socket,\Str:="1";
        !escribe en la red y lo envia a arduino (donde será leido)
        WaitTime 0.1;
        !espera un tiempo

        SocketReceive
my_socket\RawData:=receive_string,\Time:=WAIT_MAX;
        WaitTime 0.01;
        !espera un tiempo
        UnpackRawBytes receive_string,1,ack\ASCII:=8;
        !8 es el espacio en caracteres que forman el número
        UnpackRawBytes receive_string,9,posx\ASCII:=8;
        UnpackRawBytes receive_string,17,posy\ASCII:=8;
        !UnpackRawBytes receive_string, 1, posy \ASCII:=24;
        !ok:=StrToVal(ack,float);
        okposx:=StrToVal(posx,posxint);
        okposy:=StrToVal(posy,posyint);
        okack:=StrToVal(ack,estad);
        ClearRawBytes receive_string;
        estado;
    ENDPROC
```

```
    PROC absoluto(num distancia)
        !ClearRawBytes receive_string;
        WaitTime 1;
        cadena:=NumToStr(distancia,1);
        !cadena:="a"+cadena;
        SocketSend my_socket,\Str:="a"+cadena;
        !escribe en la red y lo envia a arduino (donde será leido)
        WaitTime 0.1;
        !espera un tiempo



        SocketReceive
my_socket\RawData:=receive_string,\Time:=WAIT_MAX;
        WaitTime 0.01;
        !espera un tiempo
        UnpackRawBytes receive_string,1,ack\ASCII:=8;
        !8 es el espacio en caracteres que forman el número
        UnpackRawBytes receive_string,9,posx\ASCII:=8;
        UnpackRawBytes receive_string,17,posy\ASCII:=8;
        okposx:=StrToVal(posx,posxint);
        okposy:=StrToVal(posy,posyint);
        okack:=StrToVal(ack,estad);
        ClearRawBytes receive_string;
        estado;
    ENDPROC



    PROC relativo(num distancia,num pasos)
        !num para ingresar una constante; var num para ingresar una
variable
        !ClearRawBytes receive_string;
        WaitTime 1;
        SocketSend
my_socket,\Str:="r"+NumToStr(distancia,0)+"N"+NumToStr(pasos,0);
        !escribe en la red y lo envia a arduino (donde será leido)
        WaitTime 0.1;
        !espera un tiempo

        SocketReceive
my_socket\RawData:=receive_string,\Time:=WAIT_MAX;
        WaitTime 0.01;
        !espera un tiempo
        UnpackRawBytes receive_string,1,ack\ASCII:=8;
        !8 es el espacio en caracteres que forman el número
        UnpackRawBytes receive_string,9,posx\ASCII:=8;
        UnpackRawBytes receive_string,17,posy\ASCII:=8;
        okposx:=StrToVal(posx,posxint);
        okposy:=StrToVal(posy,posyint);
        okack:=StrToVal(ack,estad);
        ClearRawBytes receive_string;
        estado;
    ENDPROC
```

```
    PROC estado()

        !!Comunicación Efectuada correctamente

        IF estad=11 THEN
            TPWrite "Instruction leer have been done.
Estado:"\Num:=estad;
            movimientorobot;
        ENDIF
        IF estad=12 THEN
            TPWrite "Instruction absoluto have been
done.Estado:"\Num:=estad;
            movimientorobot;
        ENDIF
        IF estad=13 THEN
            TPWrite "Instruction relativo have been
done.Estado:"\Num:=estad;
            movimientorobot;
        ENDIF

        !!listado de errores que se podrian enviar desde arduino
        IF estad=1 THEN
            TPWrite "No se envió un comando conocido por
arduino.Error:"\Num:=estad;
            ErrWrite "RobotStudio error","No se envió un comando
conocido por arduino."\RL2:="Escribir comando correcto";
            Stop;
        ENDIF
        IF estad=2 THEN
            TPWrite "Se cambio de remoto a local en la cinta.
Error:"\Num:=estad;
            ErrWrite "Arduino error","Se cambio de remoto a local en la
cinta."\RL2:="No cambiar durante ejecución";
            Stop;
        ENDIF
        IF estad=3 THEN
            TPWrite "Cinta no se mueve. Encoder dejo de funcionar.
Error:"\Num:=estad;
            ErrWrite "Arduino error","Cinta no se mueve."\RL2:="Encoder
dejo de funcionar.";

            Stop;
        ENDIF
        IF estad=4 THEN
            TPWrite "Valor absoluto negativo demasiado grande.
Erro:"\Num:=estad;
            ErrWrite "Arduino error","Valor absoluto negativo demasiado
grande."\RL2:="Enviar un valor menor o positivo.";

            Stop;
        ENDIF
    ENDPROC

    PROC cerrarcomunicacion()
        SocketClose my_socket;
        !cierra el socket
    ENDPROC
```

```
    PROC movimientorobot()
        IF estad=11 THEN
        Movel offs(Target_pieza,-
posxint*5,posyint*5,0),v500,fine,t_pinza\WObj:=wobj0;
        !Movej offs(Target_10,-posxint*2,80-
posxint*2,0),v500,z100,t_pinza\WObj:=wobj0;
        Movej Target_20,v500,fine,t_pinza\WObj:=wobj0;
        Movej Target_30,v500,fine,t_pinza\WObj:=wobj0;
        Movej Target_30,v500,fine,t_pinza\WObj:=wobj0;
        Movej Target_20,v500,fine,t_pinza\WObj:=wobj0;
        Movej Target_10,v500,fine,t_pinza\WObj:=wobj0;
      ! Movel offs(Target_pieza,-posxint*2,50-
posxint*2,0),v500,z100,t_pinza\WObj:=wobj0;
        ENDIF
    ENDPROC


ENDMODULE
```

5.4 Tramas de datos enviadas y recibidas desde y hacia RobotStudio

Las tramas de datos según la función que se invoque en el programa serán 3 y serán enviadas dentro del protocolo TCP/IP Ethernet.

Para el proyecto es fundamental el entendimiento de las tramas de datos que se envían.

- Leer:

Figura 38: Trama de datos envíada desde RobotStudio para la Función *Leer*

- Absoluto:

Figura 39: Trama de datos envíada desde RobotStudio para la Función *Absoluto*

- Relativo:

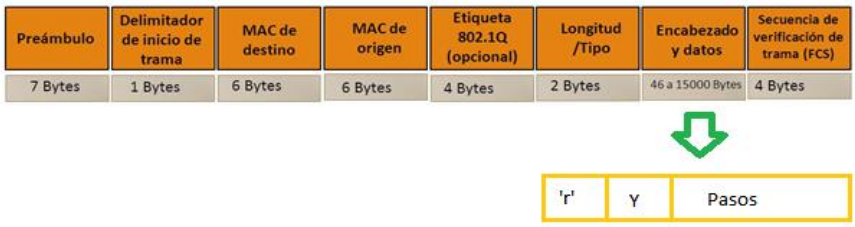


Figura 40: Trama de datos envíada desde RobotStudio para la Función *Relativo*

Ahora, cada función que se envía tiene una respuesta desde Arduino:

- Leer:

Figura 41: Trama de datos envíada desde Arduino para la Función *Leer*

- Absoluto:

Figura 42: Trama de datos envíada desde Arduino para la Función *Absoluto*

- Relativo:

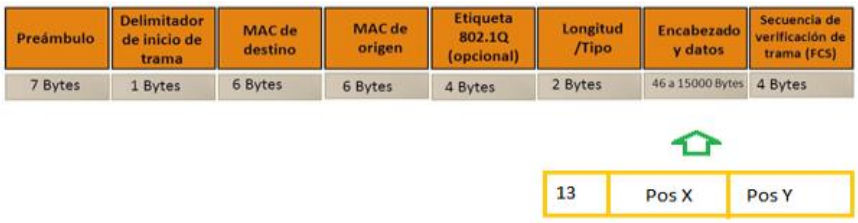


Figura 43: Trama de datos envíada desde Arduino para la Función *Relativo*

5.5 Tramas de errores enviadas y recibidas desde y hacia RobotStudio

Arduino envía 4 codigos alfanuméricos hacia RobotStudio siendo: 1, 2, 3 y 4. Según la Sección 1.5 del primer Capítulo, cada error corresponde a algún fallo determinado previamente.

Figura 44: Trama de errores envíada desde Arduino

6 SIMULACIÓN

Entrando en RobotStudio en la pestaña de "*Simulación*" encontramos la siguiente botonera:

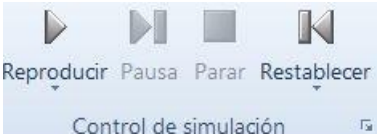


Figura 45: Botonera simulaciones *RobotStudio*

Con esta botonera se puede ejecutar el código de *RAPID* en la estación actúal y en el caso de que se desee grabar la simulación, se selecciona la opción de reproducir, grabar en visor [7].

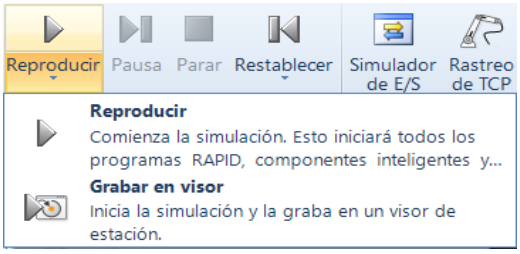


Figura 46: Opciones reproducción

Con la opción *Pausa* estando en reproducción, podemos ir pausando continuamente la simulación para ver lo que está pasando en el sistema siguiendo el puntero en RAPID [7].

Cuando se esta realizando pruebas, lo normal es que queramos volver a una situación de partida cada vez que se desee y no sea necesario volver a colocar todos los componentes o accione nuevamente: *Restablecer* permite al programador guardar una situación en la estación de forma que pueda volver cuantas veces quiera a ella sin problema. E incluso también se puede tener guardadas diferentes posiciones lo cual también es de gran utilidad. Pulsando en las opciones de *Restablecer* se puede guardar el estado actúal [7].

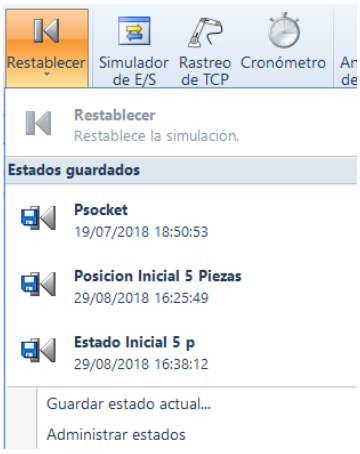


Figura 47: Opciones restablecer *RobotStudio*

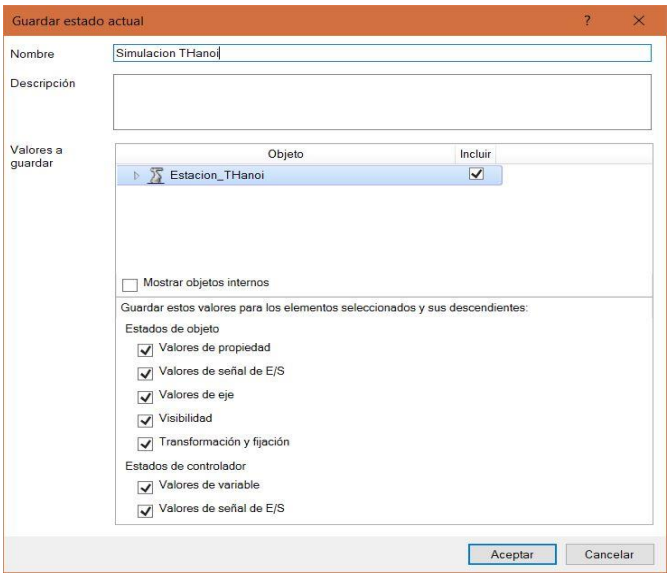


Figura 48: Ventana guardar estado actual *RobotStudio*

6.1 Muestra de datos e impresión por serial desde Arduino

Para realizar la conexión mediante puerto serie únicamente es necesario conectar nuestra placa Arduino empleando el mismo puerto que empleamos para programarlo. A continuación, abrimos el IDE Standard de Arduino y hacemos click en el «Monitor Serial» como se indica en la imagen.

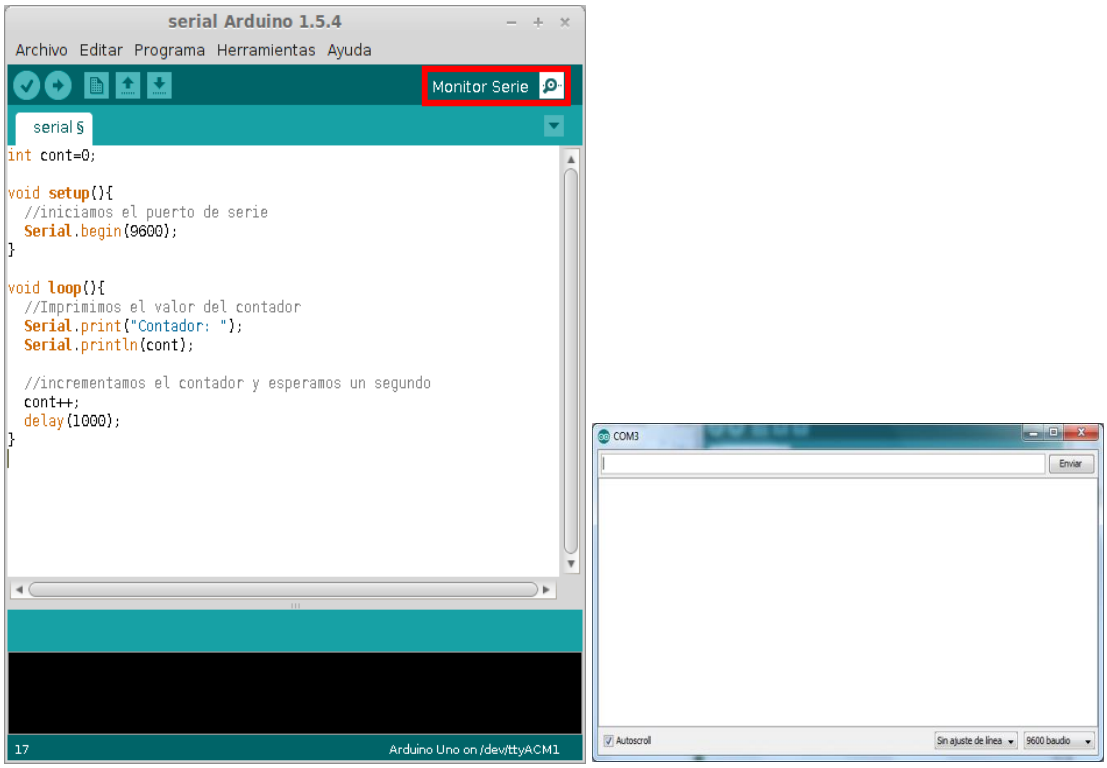


Figura 49: Monitor Serie en Arduino

7 RESULTADOS

Estimadas las necesidades, el alcance y las herramientas necesarias; así como la programación, tanto para Aruino como para RobotStudio, se procede a mostrar los resultados en pantalla para cada uno de los programas mencionados; en esta sección se separá por modos de Funcionamiento (*Local* o *Remoto*) y luego por funciones de tareas (*Leer*, *Absoluto* y *Relativo*), esto permitirá entender el funcionamiento y salida por pantalla de las secciones 5.3 (RobotStudio) y 4.6 (Arduino). Permitirá valorar la efectividad con la que las funciones realizan su tarea y la relación en ambos programas.

Cada componente electrónico se ilustra en la Figura 50 y permite, en forma general, entender el funcionamiento de la cinta; el componente más importante y que simula la cinta es la matriz de leds conectado al puente H, esté permite visualizar de forma práctica la función que se esta procesando por Arduino; para los siguientes apartados será de vital importancia. Al encenderse, Arduino parte de un estado de reposo, apagando el motor y seteando a 0 las coordenadas X e Y de la cinta.

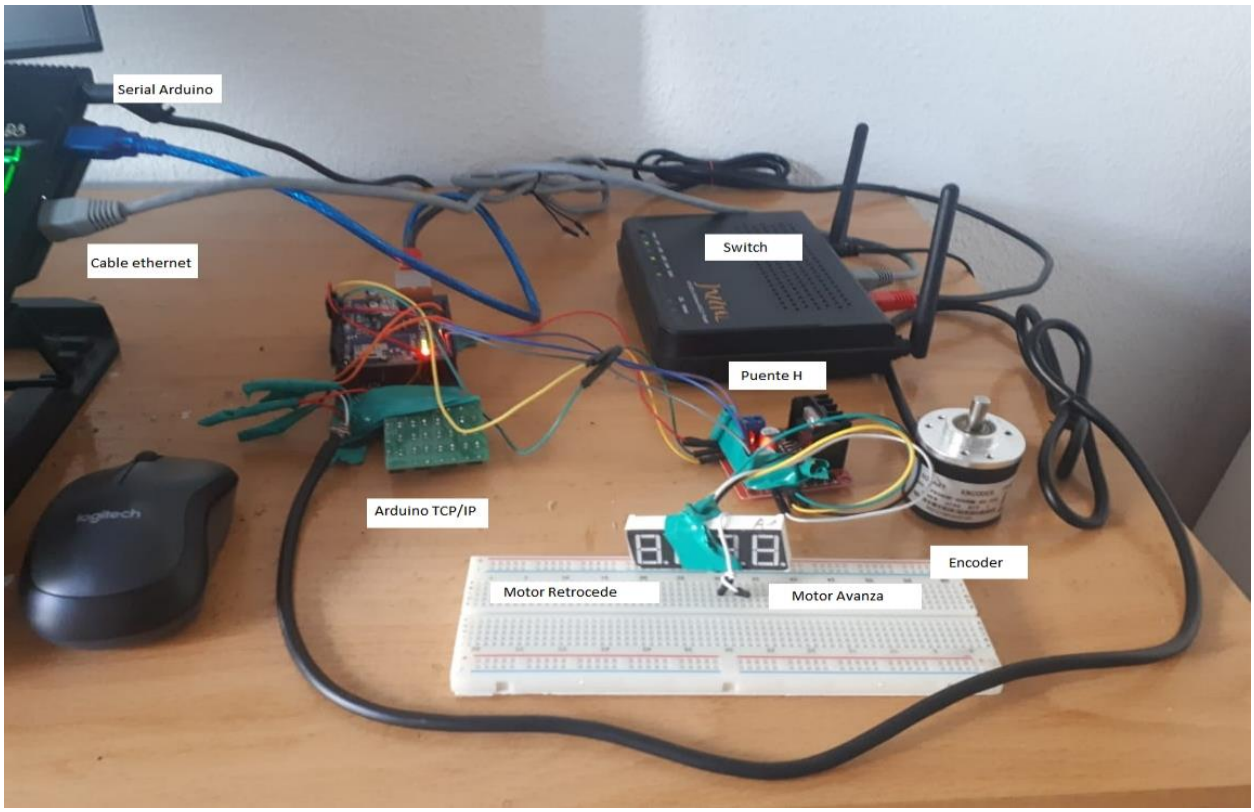


Figura 50: Esquema general de los componentes electrónicos

7.1 Función Local en Arduino

La función *Local* permite mantener el control de la cinta de modo manual. De este modo permite 3 funciones: *motor adelante* (la cinta avanza), *motor atrás* (la cinta retrocede) y *motor paro* (el motor se detiene y la cinta se para). Claramente, si el motor realiza cualquier función por medio del driver motor o puente H, la cinta hará exactamente lo mismo; existe una relación entre el motor y la cinta

7.1.1 Motor Adelante (Cinta avanza)

Para esta función se muestra la salida por *serial* de Arduino (Figura 51); así como la salida correspondiente y su efecto en los componentes electrónicos reales: se verá como se activa el motor. La función requiere la activación manual por parte del operario del botón correspondiente. Para dejar de avanzar hay que apretar el botón de detener avance.

7.1.1.1 Salida Serial

Para efectos didácticos, se presenta la explicación de cada línea mostrada en la salida serial de Arduino (Figura 51):

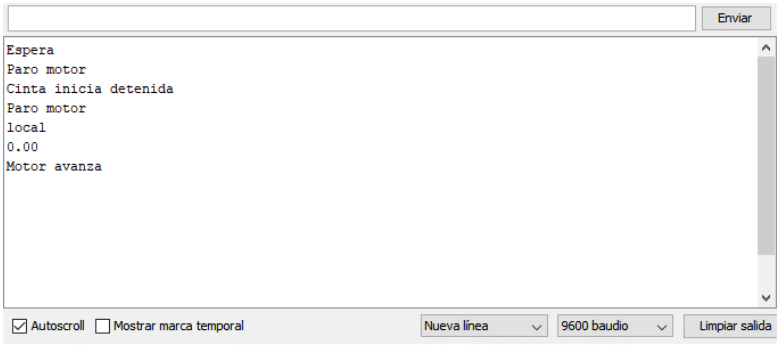


Figura 51: Salida serial para función: Motor Adelante

- Espera: Arduino espera 1 segundo hasta que todos sus niveles de voltaje al encenderse sean los correctos
- Paro Motor: Por cualquier eventualidad empieza con el motor de la cinta detenido
- Cinta inicia detenida: Mensaje de ratificación
- Paro Motor: Cuando ingresamos al modo Local, este primero apaga el motor
- Local: Modo local, permite al operario la manipulación del sistema.
- 0.00: Valor numérico (float x.xx) que se muestra desde Arduino como coordenada Y de la cinta
- Motor Avanza: Se clicó el botón que habilita el avance del motor

7.1.1.2 Cinta

Como se puede apreciar en la Figura 52, el motor se activa en la dirección solicitada. Hasta que no se envie el comando de detener la cinta (por medio del botón), el motor continuará permanentemente encendido.

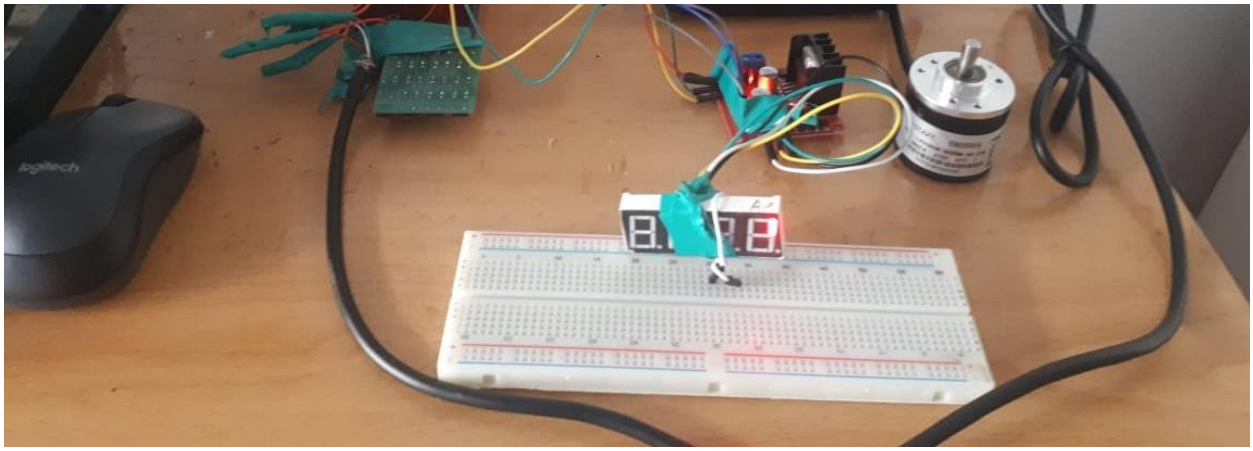


Figura 52: Salida en la cinta para función: Motor Adelante

7.1.2 Motor Atrás (Cinta retrocede)

En esta función la salida por *serial* de Arduino (Figura 53); así como la salida correspondiente y su efecto en los componentes electrónicos reales: el motor se encenderá. La función requiere la activación manual por parte del operario del botón correspondiente. Para dejar de retroceder hay que apretar el botón de detener avance.

7.1.2.1 Salida Serial

La salida serial muestra los mismos resultados con respecto a la sección 7.1.1.1, con excepción de la última línea que corresponde a la función *motor atrás* (motor retrocede en la salida). Para el resto de los parámetros, lo mencionado en la sección 7.1.1.1 son los esencialmente lo mismo.

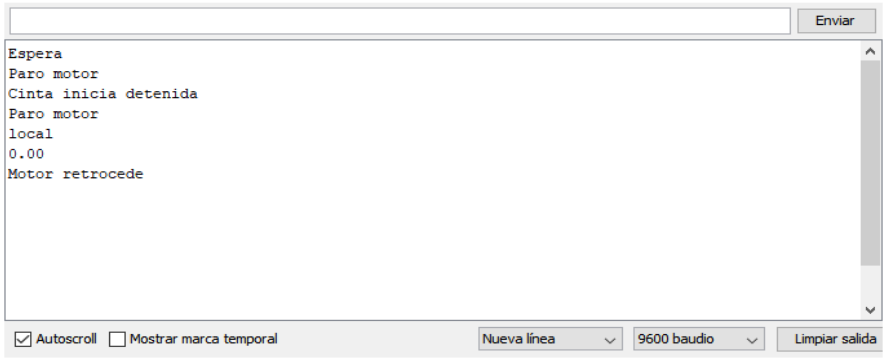


Figura 53: Salida serial para función: Motor Atrás

7.1.2.2 Cinta

Como se puede apreciar en la Figura 54, el motor se activa en la dirección solicitada. Hasta que no se envie el comando de detener la cinta, el motor continuará permanentemente encendido.

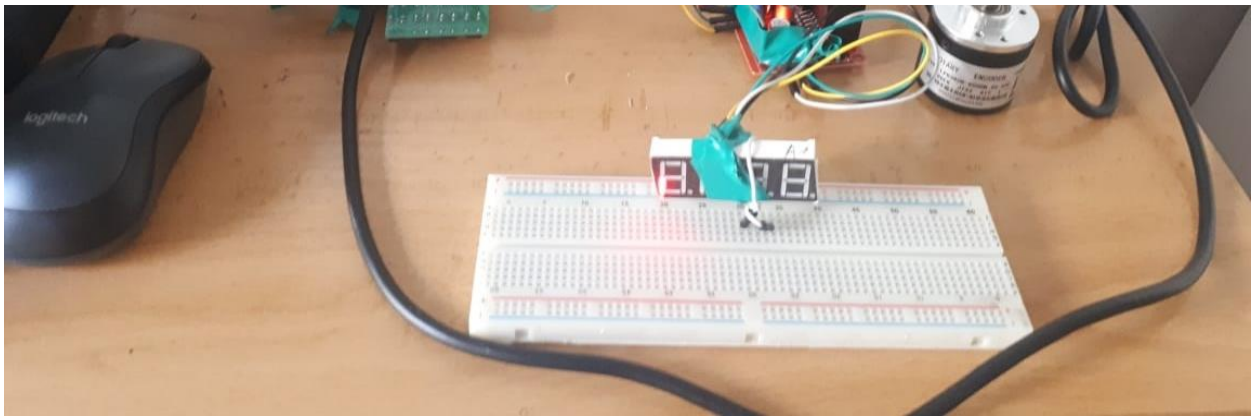


Figura 54: Salida en la cinta para función: Motor Adelante

7.1.3 Motor Paro (Cinta se detiene)

Para esta función la salida por *serial* de Arduino se muestra en la Figura 55; así como la salida correspondiente y su efecto en los componentes electrónicos reales. La función requiere la activación manual por parte del operario del botón correspondiente. Para dejar de retroceder hay que apretar el botón de detener avance.

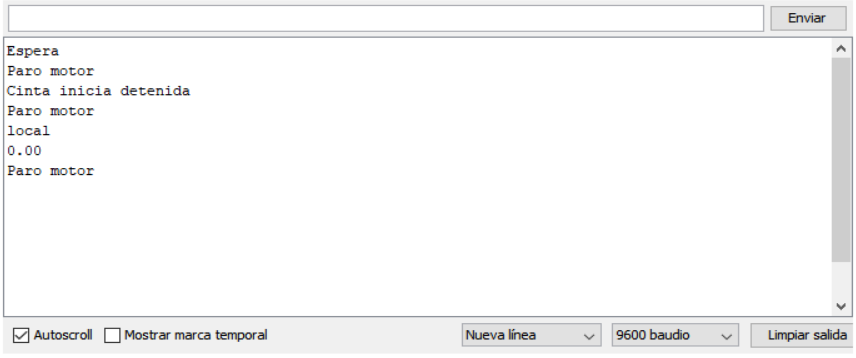


Figura 55: Salida serial para función: Motor Paro

7.1.3.1 Cinta

Como se puede apreciar en la Figura 56, el motor se activa en la dirección solicitada. Hasta que no se envie el comando de detener la cinta, el motor continuará permanentemente encendido.

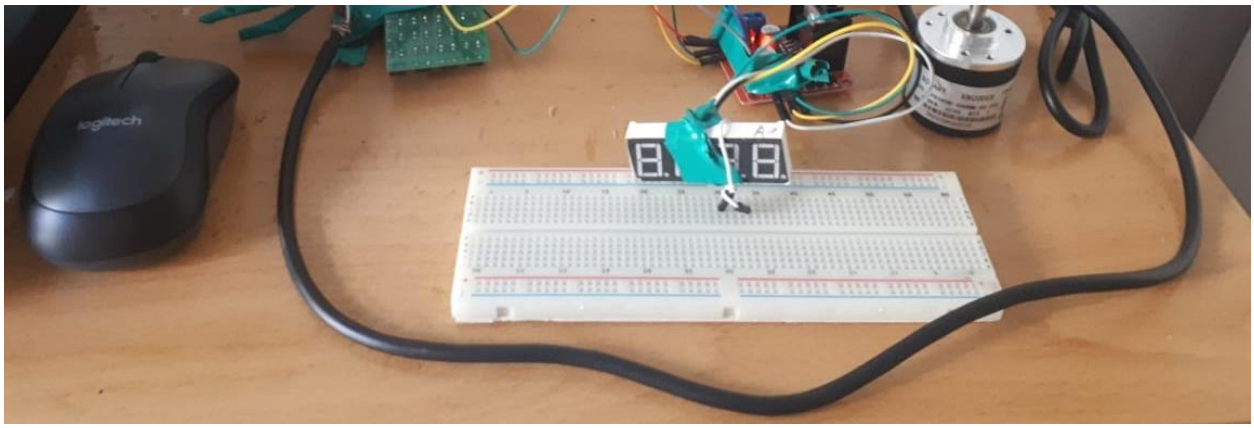


Figura 56: Salida en la cinta para función: Motor Paro

7.2 Función Remoto en Arduino desde RobotStudio

La función *Remoto* permite el control de la cinta desde RobotStudio. Este modo permite 3 funciones: *leer, absoluto* y *relativo*. Para entrar en este modo desde Arduino hay que presionar el botón de remoto (botón correspondiente al caracter 'B' y que en la cinta real será reemplazado por un selector en conjunto con el botón de *local*). La cinta realizará la función solicitada desde RobotStudio.

7.2.1 Función *Leer*

Esta función, envíada desde RobotStudio, permite leer las coordenadas X e Y de la cinta en la posición actúal. Se puede envíar este comando tantas veces como sea necesario. Se crean observaciones de las variables asociadas al estado de la cinta, posición X y posición Y; ambas enviadas desde Arduino.

7.2.1.1 RobotStudio

Como se ha indicado en apartados y capítulos anteriores, basta llamar en la función Main a la función leer; empezando desde Arduino y poniéndolo en modo Remoto, luego se empieza con la simulación. RobotStudio realiza todas las operaciones solicitadas y envía la función leer (basta con quitar el signo de comentario colocado delante de la función dentro de Main, como se muestra en la Figura 57.

```
PROC main()

    !Añada aquí su código
    SocketClose my_socket;
    abrircomunicacion;
    WaitTime 1;
    !  leer;
    !  absoluto 5;  !!el valor hace referencia a la distancia que se va a recorrer por la cinta en x
    ! relativo 4, 3; !!el primer valor hace referencia a la distancia que se va a recorrer por la cinta en x
    !!el segundo hace referencia al numero de saltos que se va a recorrer por la cinta en x
    !absoluto 5;
    leer;
    ! WHILE ok=TRUE DO

    ! ENDWHILE
    !TPWrite "Instruction SocketClose has been executed";
ENDPROC
```

Figura 57: Función Leer en RobotStudio

7.2.1.2 Arduino

Ahora bien, en Arduino se presentará la siguiente salida, la cual permite apreciar los valores que se envían hacia RobotStudio. Como la función no requiere salida hacia los motores; entonces, no se produce cambio alguno, pero claro; la salida serial muestra lo que esta pasando.

7.2.1.3 Salida Serial de Arduino

La salida serial muestra lo siguiente (Figura 58):

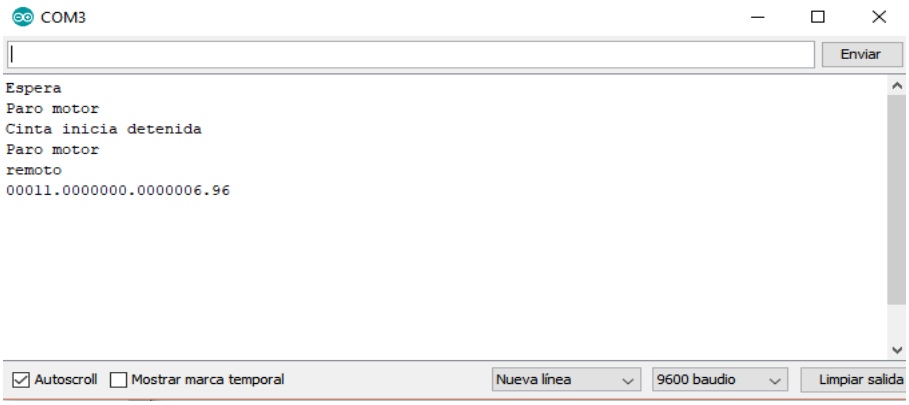


Figura 58: Función Leer en Arduino

La cadena de caracteres impresa al final, son 24 caracteres en ASCII y representa:

- Los primeros 8 caracteres presentan el valor en decimal de 00011.00 que como se definió en la sección 1.5 muestran que la ejecución de la función leer desde RobotStudio se ejecutó correctamente y se envió por TcpIp.
- Los siguientes 8 caracteres muestran el valor asociado a la posición X de la pieza (00000.00)
- Luego los caracteres (00006.96) muestran la coordenada Y de la cinta

7.2.1.4 Lectura de variables en RobotStudio

La lectura muestra las siguientes variables: *estad*, *posxint*, y *posyint*. La Figura 59 muestra la salida en pantalla de RobotStudio, luego de ser recibido desde Arduino, con los valores ya convertidos de ASCII a num.

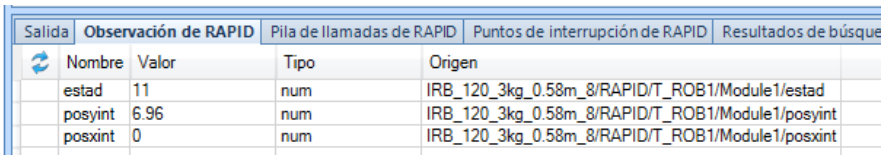


Figura 59: Variables recibidas en RobotStudio

7.2.2 Función *Absoluto*

La función *Absoluto*, envíada desde RobotStudio, permite leer las coordenadas X e Y de la cinta en la posición actúal de la pieza. Se puede envíar este comando tantas veces como sea necesario. Se crean observaciones de las variables asociadas al estado de la cinta, posición X y posición Y; ambas enviadas desde Arduino. Como se requiere la acción de la cinta, Arduino completará las acciones necesarias, y luego devolverá la cadena de caracteres correspondiente. Esta función requiere de un valor numérico (distancia en Y) para mover la cinta.

7.2.2.1 RobotStudio

Como se ha indicado previamente, basta llamar en la función Main a la función *absoluto*; empezando desde Arduino y poniéndolo en modo Remoto. RobotStudio realiza todas las operaciones solicitadas y envía la función (basta con quitar el signo de comentario colocado delante de la función dentro de Main, como se muestra en la

Figura 60; se requiere aumentar la cantidad en cm que se desea que la cinta avance.

```
PROC main()

    !Añada aquí su código
    SocketClose my_socket;
    abrircomunicacion;
    WaitTime 1;
    !  leer;
    !  absoluto 5;  !!el valor hace referencia a la distancia que se va a recorrer por la cinta en x
    ! relativo 4, 3; !!el primer valor hace referencia a la distancia que se va a recorrer por la cinta en x
    !!el segundo hace referencia al numero de saltos que se va a recorrer por la cinta en x
    absoluto 5;
    ! leer;
    ! WHILE ok=TRUE DO

    ! ENDWHILE
    !TPWrite "Instruction SocketClose has been executed";
ENDPROC
```

Figura 60: Función Absoluto en RobotStudio

7.2.2.2 Arduino

En Arduino se presentará la siguiente salida, la cual deja apreciar los valores que se envían hacia RobotStudio luego de cumplir con la condición de movimiento en cm recibida. Arduino espera alrededor de 10 segundos a la espera de que haya un cambio en la coordenada Y (esto para saber si la cinta se esta moviendo gracias al encoder incorporado); cuando se ha cumplido, se empaqueta la trama de datos correspondiente y se envía a RobotStudio.

7.2.2.3 Salida Serial de Arduino

La salida serial muestra lo siguiente (Figura 61):

Figura 61: Función Absoluto en Arduino

La cadena de caracteres impresa al final, son 24 caracteres en ASCII y representa:

- Los primeros 8 caracteres presentan el valor en decimal de 00012.00 que como se definió en la sección 1.5 muestran que la ejecución de la función absolut desde RobotStudio se ejecutó correctamente.
- Los siguientes 8 caracteres muestran el valor asociado a la posición X de la pieza (00000.00)
- Luego los caracteres (00005.00) muestran la coordenada Y de la cinta

7.2.2.4 Cinta

Arduino activará el motor de la cinta, y esperará a los 10 segundos para afirmar que la cinta se ha movido y empaquetarán los datos para envíarlos a RobotStudio. Durante el proceso y antes de envíar la cadena de caracteres correspondiente, Arduino revisará que la cantidad Y trasladada (por el motor), ya sea en sentido horario o antihorario (dependerá del signo con el que se ha enviado la distancia en RobotStudio), se haya completado correctamente.

La Figura 62, muestra la función absoluta ejecutándose con un valor de 5 cm (envíada desde RobotStudio). La Figura 63, muestra el proceso detenido luego de completarse correctamente los 5 cm (para lograr los 5 cm hay

que dar 5 vueltas en el encoder: cada vuelta corresponde a 1 cm).

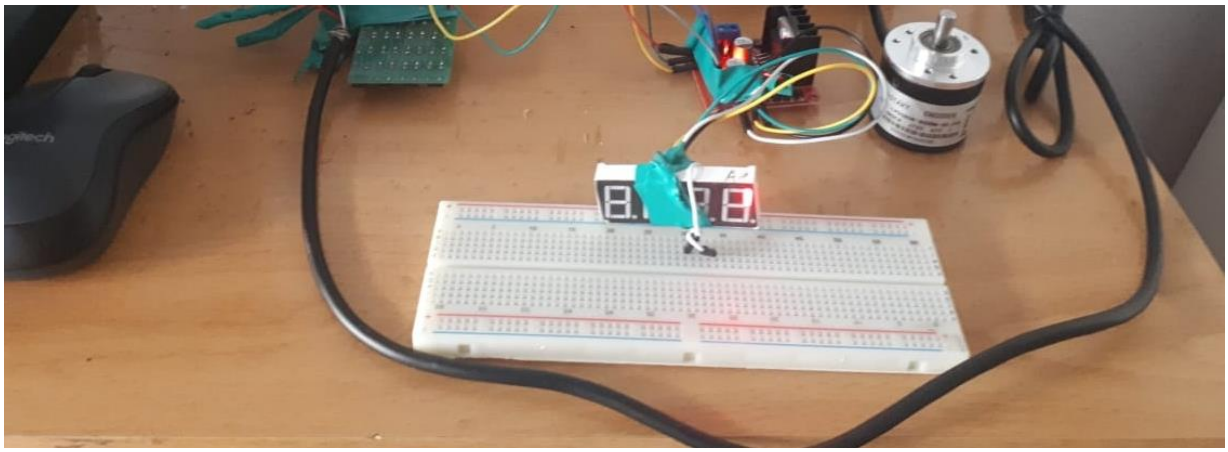


Figura 62: Función Absoluta ejecutándose.

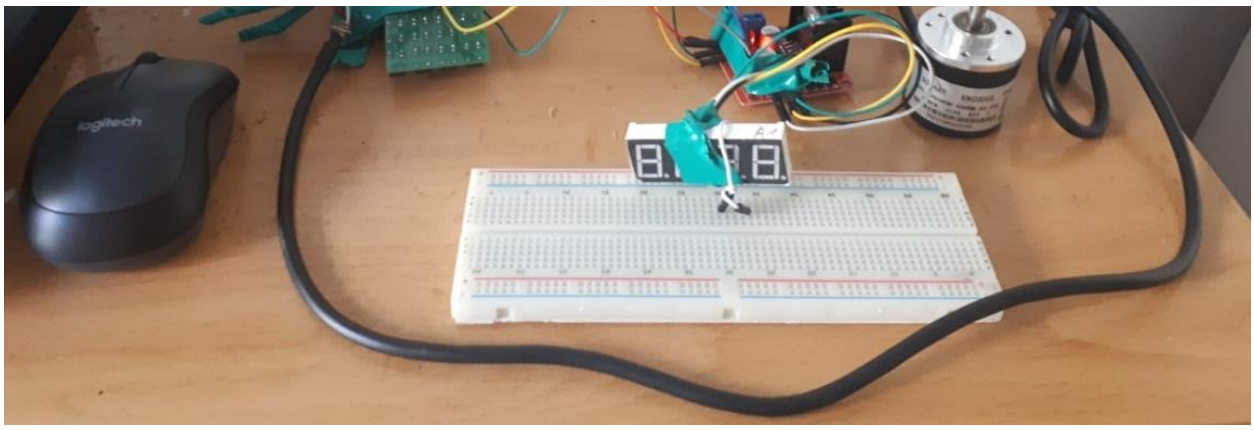


Figura 63: Cinta detenida luego de ejecutarse el comando Absoluto.

7.2.2.5 Lectura de variables en RobotStudio

La lectura muestra las siguientes variables: *estad*, *posxint*, y *posyint*. La Figura 64 muestra la salida en pantalla de RobotStudio, luego de ser recibido desde Arduino, con los valores ya convertidos de ASCII a num.

Salida | **Observación de RAPID** | Pila de llamadas de RAPID | Puntos de interrupción de RAPID | Resultados de búsqueda

Nombre	Valor	Tipo	Origen
estad	12	num	IRB_120_3kg_0.58m_8/RAPID/T_ROB1/Module1/estad
posyint	5	num	IRB_120_3kg_0.58m_8/RAPID/T_ROB1/Module1/posyint
posxint	0	num	IRB_120_3kg_0.58m_8/RAPID/T_ROB1/Module1/posxint

Figura 64: Variables recibidas en RobotStudio

7.2.2.6 IRB120

El robot (simulado), tomará los datos y dependiendo de sí se han ejecutado correctamente, y realizará movimientos simples según puntos de trayectorias propuestos. Si el valor recibido desde Arduino es positivo el brazo se moverá a la derecha de la pantalla; si es negativo se irá moviendo a la izquierda. Cada función (11, 12 y 13) ejecutadas correctamente provocarán el movimiento del robot. En apartados posteriores se realizará una comparativa en la localización espacial del brazo del robot al recoger la pieza para apreciar el movimiento producido.

7.2.3 Función *Relativo*

La función *Relativo*, envíada desde RobotStudio, permite leer las coordenadas X e Y de la cinta en la posición actúal de la pieza. Se crean observaciones de las variables asociadas al estado de la cinta, posición X y posición Y; ambas enviadas desde Arduino. Como se requiere la acción de la cinta, Arduino completará las acciones necesarias, y luego devolverá la cadena de caracteres correspondiente. Esta función requiere de un par de números para ser completada por la cinta (la distancia a recorrer por la cinta y el número de veces que se desea hacer este movimiento).

7.2.3.1 RobotStudio

Basta llamar en la función Main a la función *relativo*; empezando desde Arduino y poniéndolo en modo Remoto. RobotStudio realiza todas las operaciones solicitadas y envía la función (basta con quitar el signo de comentario colocado delante de la función dentro de Main, como se muestra en la Figura 65. Se requiere aumentar la cantidad en cm que se desea que la cinta avance y el numéro de ciclos que se va a repetir este proceso.

```
PROC main()

    !Añada aquí su código
    SocketClose my_socket;
    abrircomunicacion;
    WaitTime 1;
    !  leer;
    !  absoluto 5;  !!el valor hace referencia a la distancia que se va a recorrer por la cinta en x
    relativo 4, 3; !!el primer valor hace referencia a la distancia que se va a recorrer por la cinta en x
    !!el segundo hace referencia al numero de saltos que se va a recorrer por la cinta en x
    !absoluto 5;
    ! leer;
    ! WHILE ok=TRUE DO

    ! ENDWHILE
    !TPWrite "Instruction SocketClose has been executed";
ENDPROC
```

Figura 65: Función Absoluto en RobotStudio

7.2.3.2 Arduino

En Arduino se presentará la siguiente salida, la cual deja apreciar los valores que se envían hacia RobotStudio luego de cumplir con la condición de movimiento en cm recibida y el número de ciclos. Arduino espera alrededor de 10 segundos, en cada ciclo, a la espera de que haya un cambio en la coordenada Y (esto para saber si la cinta se esta moviendo gracias al encoder incorporado); cuando se ha cumplido, se empaqueta la trama de datos correspondiente y se envía a RobotStudio.

7.2.3.3 Salida Serial de Arduino

La salida serial muestra lo siguiente (Figura 66):

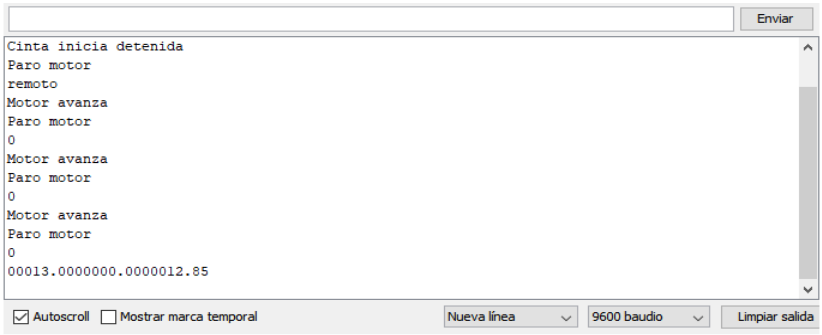


Figura 66: Función Absoluto en Arduino

- Los primeros 8 caracteres presentan el valor en decimal de 00013.00 (La función se completó correctamente.
- Como se puede apreciar la salida por serial es diferente a la de la función *absoluto,* ya que el motor tiene

que cumplir 3 ciclos de 4cm cada ciclo; lo que sumado son 12 cm (12.85 para el encoder que es una aproximación por el mapeo en los valores en Arduino).

7.2.3.4 Cinta

Arduino activará el motor de la cinta, y esperará a los 10 segundos, en cada ciclo, para afirmar que la cinta se ha movido lo que corresponda (entre ciclo y siglo se espera 1 segundo). Se empaquetarán los datos para envíarlos a RobotStudio. Durante el proceso y antes de envíar la cadena de caracteres correspondiente, Arduino revisará que la cantidad Y trasladada (por el motor), ya sea en sentido horario o antihorario (dependerá del signo con el que se ha enviado la distancia en RobotStudio), se haya completado correctamente.

La Figura 67, muestra la función realativa ejecutándose con un valor de 4 cm (envíada desde RobotStudio) con 3 ciclos. La Figura 68, muestra el proceso detenido luego de completarse correctamente los 4 cm (para lograr los 4 cm hay que dar 4 vueltas en el encoder: cada vuelta corresponde a 1 cm).

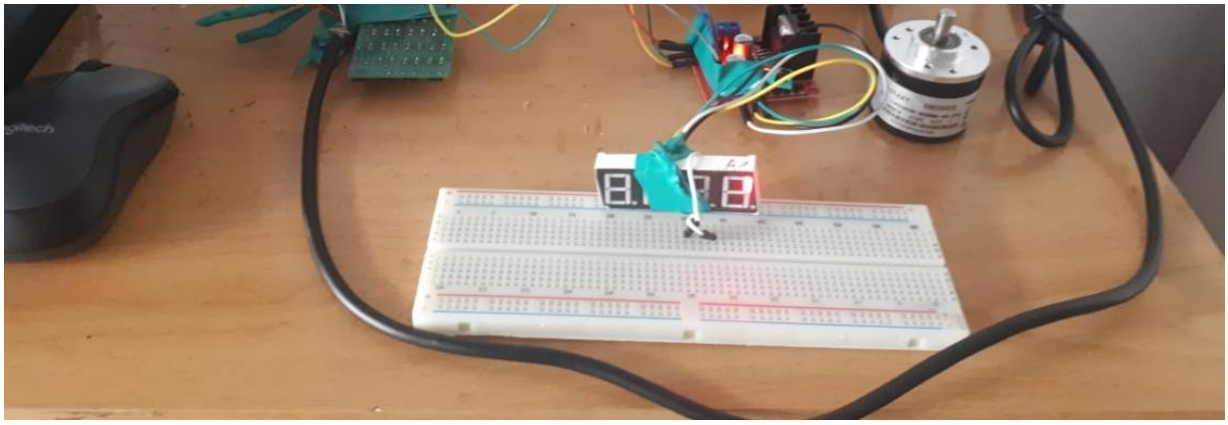


Figura 67: Función Relativa ejecutándose.

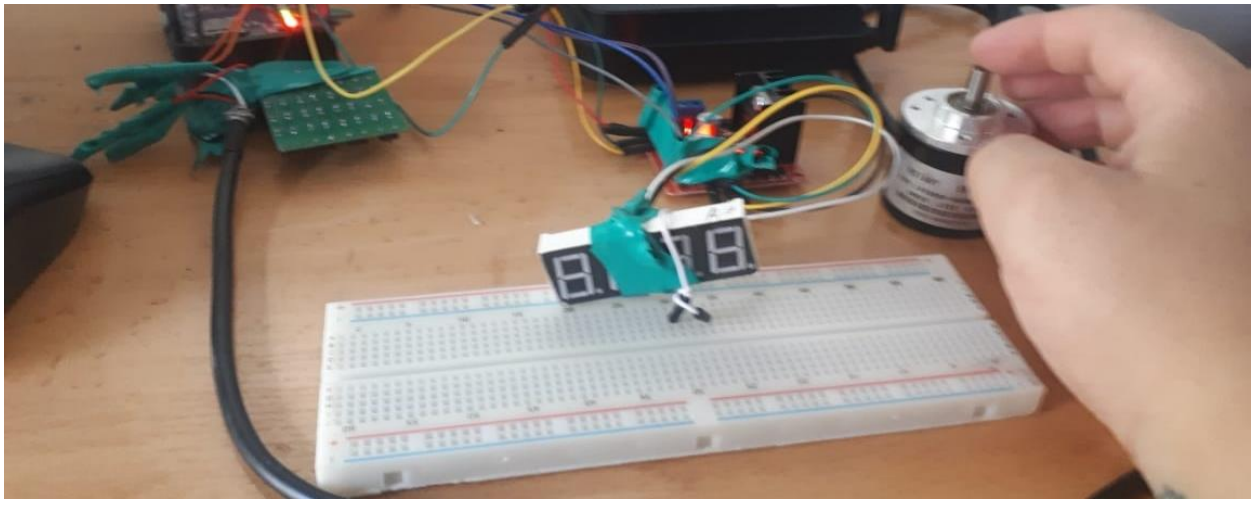
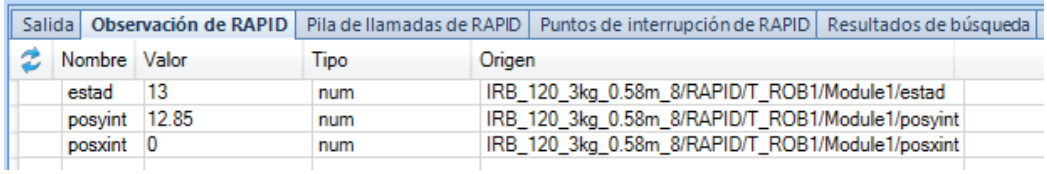


Figura 68: Cinta detenida luego de ejecutarse el comando Relativo.

7.2.3.5 Lectura de variables en RobotStudio

La lectura muestra las siguientes variables: *estad*, *posxint*, y *posyint*. La Figura 69 muestra la salida en pantalla de RobotStudio, luego de ser recibido desde Arduino, con los valores ya convertidos de ASCII a num.



Salida | **Observación de RAPID** | Pila de llamadas de RAPID | Puntos de interrupción de RAPID | Resultados de búsqueda

Nombre	Valor	Tipo	Origen
estad	13	num	IRB_120_3kg_0.58m_8/RAPID/T_ROB1/Module1/estad
posyint	12.85	num	IRB_120_3kg_0.58m_8/RAPID/T_ROB1/Module1/posyint
posxint	0	num	IRB_120_3kg_0.58m_8/RAPID/T_ROB1/Module1/posxint

Figura 69: Variables recibidas en RobotStudio

7.2.3.6 IRB120

Al igual que en el apartado 7.2.2.6 el robot (simulado), tomará los datos y dependiendo de sí se han ejecutado correctamente, y realizará movimientos según puntos de trayectorias propuestos. Si el valor recibido desde Arduino es positivo el brazo se moverá a la derecha de la pantalla; si es negativo se irá moviendo a la izquierda. Cada función (11, 12 y 13) ejecutadas correctamente provocarán el movimiento del robot. En apartados posteriores se realizará una comparativa en la localización espacial del brazo del robot al recoger la pieza para apreciar el movimiento del robot.

7.3 Errores

Como se propuso en el apartado 1.5, también se han considerado errores que pudieran suceder a lo largo del trabajo con la cinta; cada error tiene un código único que tiene un significado en particular, también se mostrará por pantalla en el serial de Arduino y como un mensaje de error en RobotStudio. Se pueden producir uno o mas de estos errores en el mismo ciclo de trabajo.

7.3.1 Error 01

Codigo 1: El caracter enviado desde RobotStudio no coincide con ninguna función; se deberá envíar un caracter correcto. Esto significa que RobotStudio no esta envíando ninguno de los 3 caracteres conocidos por Arduino; por lo que inevitablemente provocará este fallo. La Figura 70 muestra la salida en el serial de Arduino y la Figura 71, muestra el caracter enviado desde RobotStudio (también podrá ser visto en Flexpendant) que provoca el fallo. La Figura 72 muestra el error en la salida de RobotStudio.

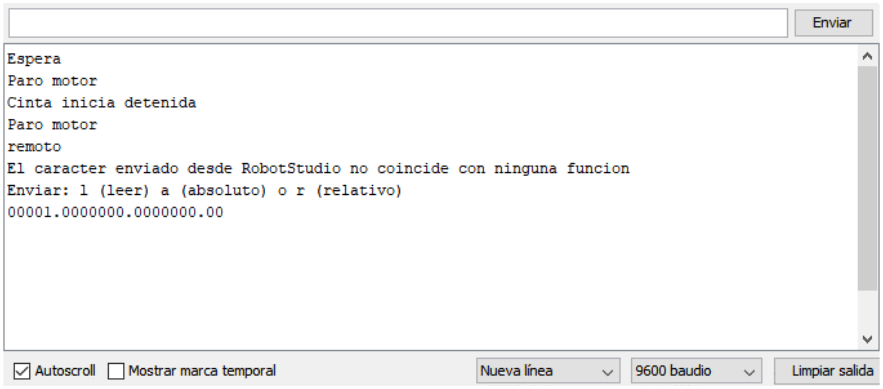


Figura 70: Salida del Serial de Arduino para Error 01

```
PROC relativo(num distancia,num pasos)
    !num para ingresar una constante; var num para ingresar una variable
    !ClearRawBytes receive_string;
    WaitTime 1;
    SocketSend my_socket,\Str:="y"+NumToStr(distancia,0)+"N"+NumToStr(pasos,0);
    !escribe en la red y lo envia a arduino (donde será leido)
```

Figura 71: Origen del fallo desde RobotStudio para Error 01

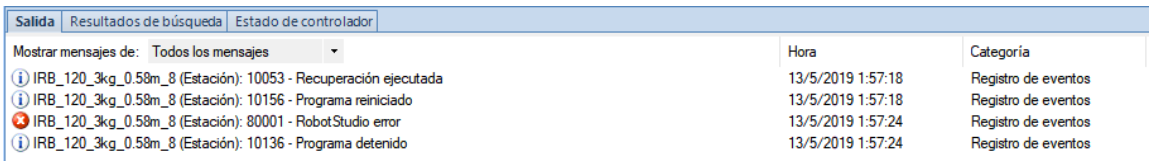


Figura 72: Salida RobotStudio para Error 01

7.3.2 Error 02

Codigo 2: Se ha cambiado (en Arduino) de Remoto a Local, y por tanto la cinta se detendrá. En este caso el código de error nos dice que el operario cambió de remoto a local sin esperar que el proceso que se estaba ejecutando en Arduino termine. La Figura 73 muestra la salida serial en Arduino y la Figura 74 muestra la Salida en RobotStudio.

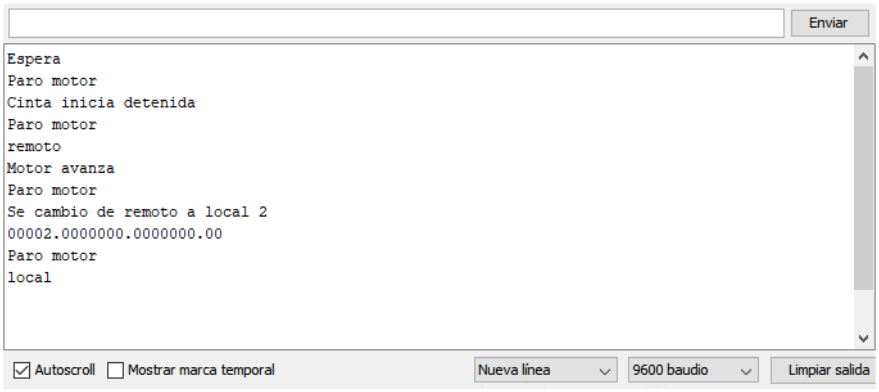


Figura 73: Salida del Serial de Arduino para Error 02

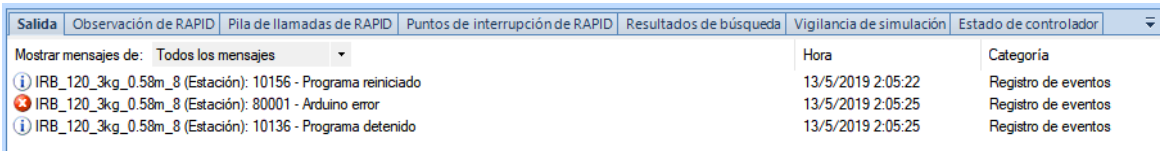


Figura 74: Salida RobotStudio para Error 02

7.3.3 Error 03

Codigo 3: Encoder no funciona. Revisar función de la cinta y el sensor encoder. Para este error, se toma en cuenta los 10 segundos en los que, si el puente H está activo, pero no se lee un incremento en la señal de salida del encoder que corresponde al eje Y, entonces se crea este error. Esto puede significar: que el motor esta desconectado o que el encoder lo está; por lo general se toma el caso de que el encoder se haya desconectado, como más frecuente; por lo tanto, la Figura 75 muestra la salida por serial para este error. La Figura 76 muestra la salida en RobotStudio.

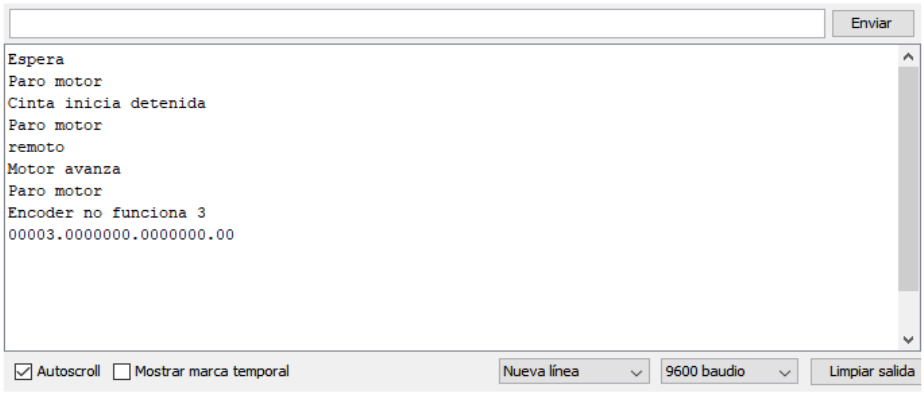


Figura 75: Salida del Serial de Arduino para Error 03

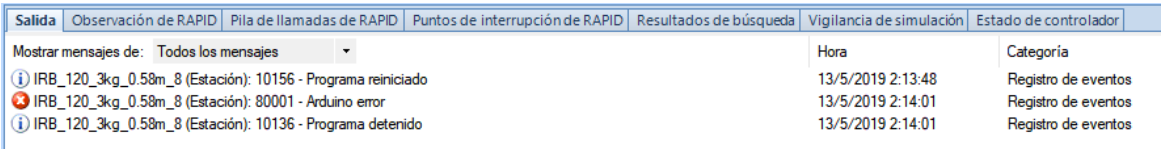


Figura 76: Salida RobotStudio para Error 03

7.3.4 Error 04

Codigo 4: Valor de movimiento negativo (en retroceso para la cinta) es demasiado grande. Modificar el valor a envíar en RobotStudio. Este error solo se da si la petición de movimiento de RobotStudio es un valor negativo superior a los cm en los que ya se haya movido la pieza o en su estado inicial. Por lo tanto, se suguieren dos alternativas: incrementar el avance de la cinta y luego retroceder o cambiar el valor negativo por uno mucho menor (en términos absolutos). La Figura 77 muestra el Serial y la Figura 78 muestra la salida en RobotStudio. La Figura 79 muestra el valor enviado.

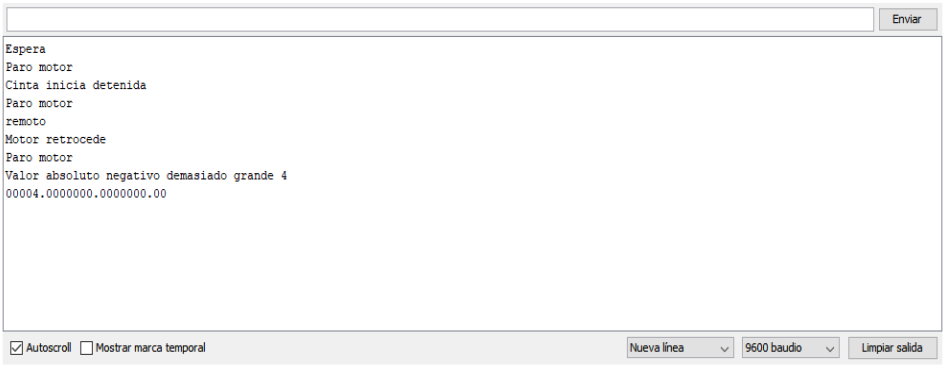


Figura 77: Salida del Serial de Arduino para Error 04

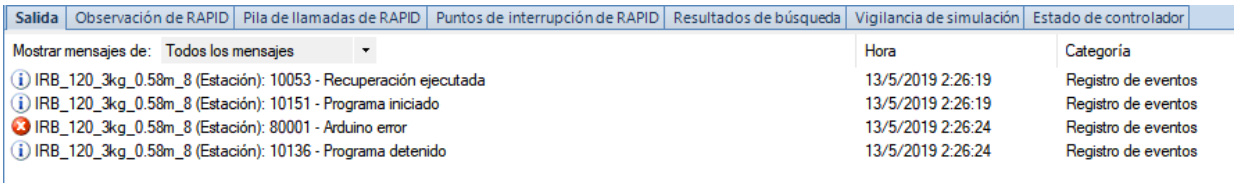


Figura 78: Salida RobotStudio para Error 04

```
PROC main()

    !Añada aquí su código
    SocketClose my_socket;
    abrircomunicacion;
    WaitTime 1;
    !  leer;
    !  absoluto 5;  !!el valor hace referencia a la distancia que se va a recorrer por la cinta en x
    ! relativo 4, 3; !!el primer valor hace referencia a la distancia que se va a recorrer por la cinta en x
    !!el segundo hace referencia al numero de saltos que se va a recorrer por la cinta en x
    absoluto -5;
```

Figura 79: Valor enviado desde RobotStudio para Error 04

7.4 Ejemplo de varias funciones y movimiento relativo del robot IRB120

Para el siguiente ejemplo y mostrar como afectan las posiciones de la cinta en RobotStudio, se han puesto las funciones mostradas en la Figura 80, que para efectos demostrativos no son necesarias más funciones, pero que, si el operario así lo desea podrá modificarlos y añadir cualquier función con valores distintos a estos.

```
absoluto 5;
leer;
relativo 4, 3;
absoluto -8;
absoluto 10;
```

Figura 80: Funciones enviadas para ejemplo ilustrativo

En Arduino se presentan las líneas en la siguiente salida de pantalla del Serial; y corresponde a las funciones mostradas en la Figura 81. La línea 8 muestra que el comando *absoluto 5* se realizó correctamente; la línea 9 muestra que el comando *leer* se completó; La línea 19 muestra que el comando *relativo 4- 3* se completó también. La líne 22 corresponde con el *absoluto -8* y la línea 25 con la de que el comando *absoluto 10* se realizó correctamente. El valor final en la línea 25 correspondiente al valor de 20, que realizando las operaciones correspondientes a cada movimiento: 5+3*4-8+10 dan como resultado un valor de 19 (que para Arduino serían 20 por el map antes mencionado). Lo que corrobora con éxito que las funciones y los valores enviados se cumplen a cabalidad y que por lo tanto el sistema cumple con los requisitos establecidos.

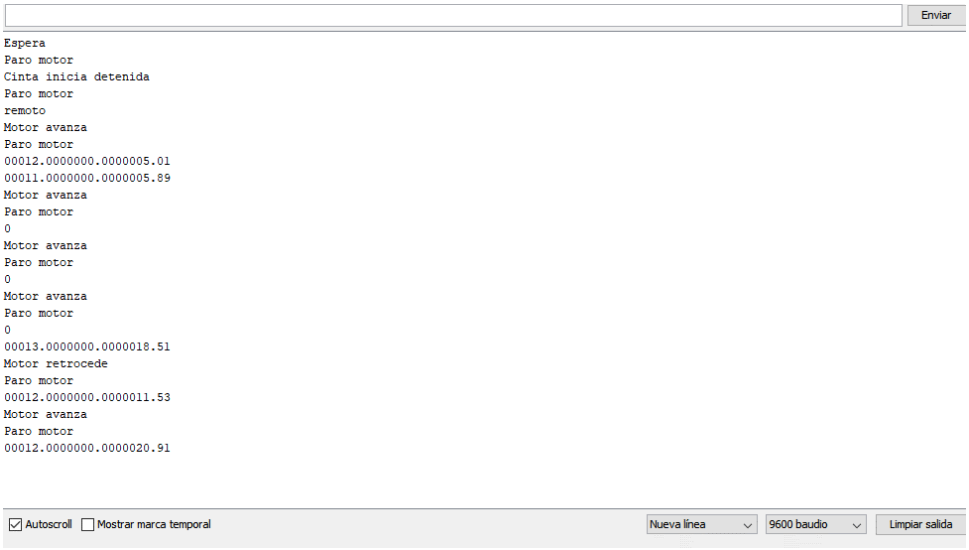


Figura 81: Seriel para ejemplo ilustrativo

La Figura 82, corresponde con el estado de reposo inicial del robot. La Figura 83 con el movimiento realizado después de la función *relativo 4, 3* y la Figura 84 después de recoger la pieza en *absoluto 10.* Se apreciará un ligero cambio en la posición de recogida de pieza en ambos.

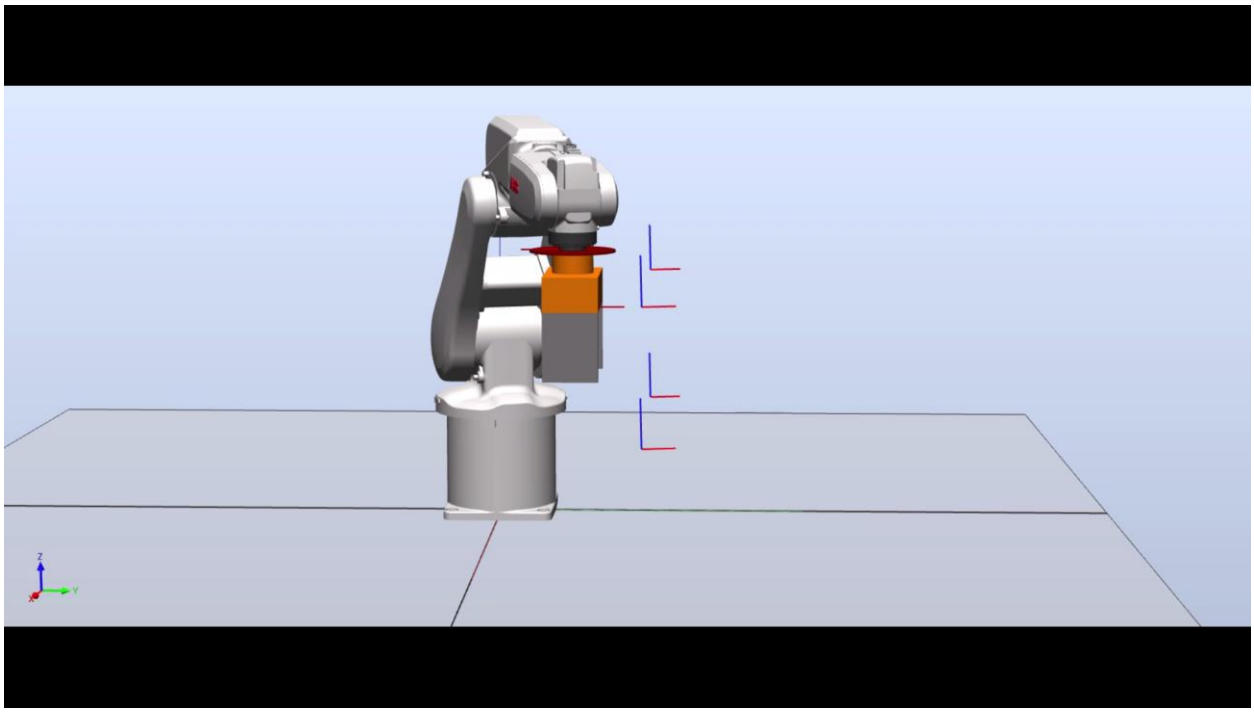


Figura 82: Estado en reposo del robot

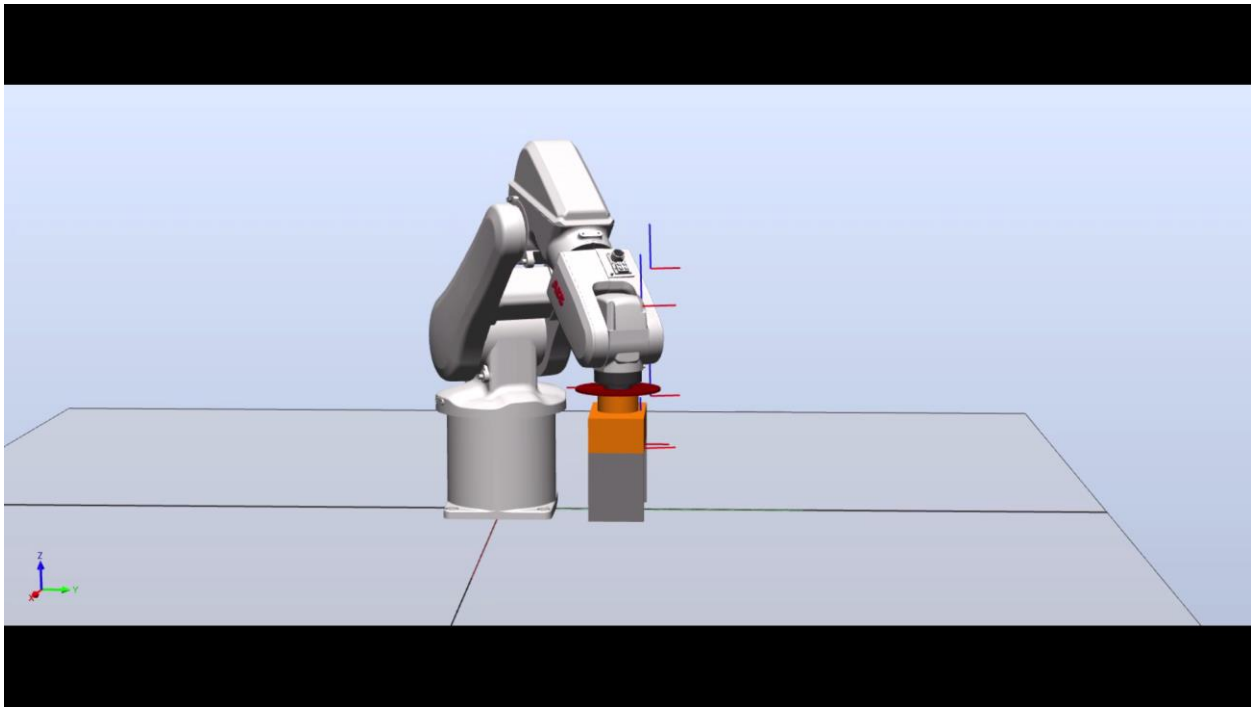


Figura 83: Robot después de recibir información de la función *relativo 4, 3*

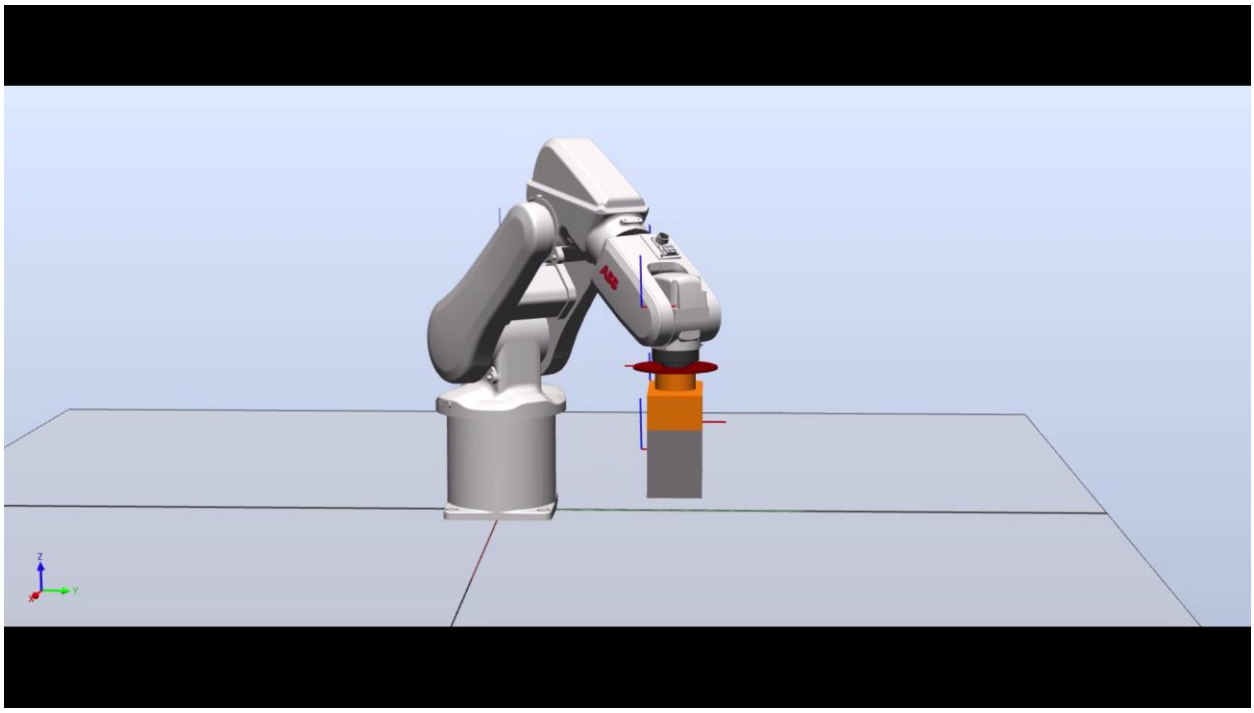


Figura 84: Robot después de recibir información de la función *absoluto 10*

Estos resultados, en forma de video se añaden como Anexos, en modo de demostración y verificación del trabajo realizado.

8 CONCLUSIONES

Tras lo conseguido en el tabajo se pueden obtener varias conclusiones.

Pero antes de mencionar estas conclusiones quisiera hacer incapíe en que, para realizar este trabajo, a parte de los conocimientos previos de mecatrónica que se tiene, es esencial aprender y desarrollar nuevas formas de ver el mundo y en especial adquirir nuevos conocimientos basados en protocolos como el TCP/IP o el lenguaje RAPID para RobotStudio, entre otras muchas cosas.

- Resulta bastante interesante el desarrollo de este proyecto ya que permite controlar, de forma remota, la operación en la cinta, sin la necesidad de estar en el lugar.
- Otro aspecto importante, es que, antes no se habían hecho proyectos parecidos y tratar de conectar de alguna forma a RobotStudio o los Robots ABB (en especial el IRB120) de forma remota con Arduino, es sumamente difícil; por lo que prácticamente este es un trabajo pionero y que permitirá a los estudiantes e investigadores adquirir mayor destreza en el manejo de RobotStudio y Arduino.
- Al estar ya conectado con Arduino, no sería necesario utilizar entradas o salidas físicas del robot para cumplir una tarea; bastaría con modificar el protocolo y expandir el buffer de comunicaciones para ingresar toda la información relevante que se le quiera pasar al IRB120 y hacerlo por medio del protocolo TCP/IP. Lo cuál expande la funcionalidad tanto de RobotStudio, como de Arduino. Las posibilidades son enormes.
- Uno de los puntos a destacar más importantes es que Arduino no goza de una estabilidad en cuanto a rendimiento y protocolos de comunicación, pero aún así el programa desarrollado es bastante efectivo

y eficaz. Se recomienda intentar con otro tipo de alternativas.

- Otra conclusión, quizá menos importante, es que no se puede crear un servidor en RobotStudio para enlazar dispositivos fuera del LocalHost; lo cuál se solucionaría portándolo de RobotStudio a la controladora real del robot. Sin embargo, si el dispositivo que se quiere conectar como cliente hacia RobotStudio esta en la misma maquina (127.0.0.1) no hay problema; como se ha visto en otros trabajos de master y de grado.

Todo esto en lo que concierne al trabajo desarrollado.

9 BIBLIOGRAFÍA

[1] ABB Asea Brown Boveri, «Abb.com,» [En línea]. Available: https://new.abb.com/products/robotics/es/robots-industriales/irb-120. [Último acceso: 05 07 2019].

[2] E. Crespo, «Aprendiendo Arduino,» Wordpress, 04 07 2016. [En línea]. Available: https://aprendiendoarduino.wordpress.com/category/tcpip/. [Último acceso: 05 07 2019].

[3] J. A. T. Herrera, Control de equipo de posicionado de piezas., Sevilla: Universidad de Sevilla, 2018.

[4] MCI Electronic, «Arduino.cl,» [En línea]. Available: http://arduino.cl/arduino-mega-2560/. [Último acceso: 05 07 2019].

[5] ABB, «abb.com/robotics,» 2019. [En línea]. Available: https://search-ext.abb.com/library/Download.aspx?DocumentID=ROB0295EN&LanguageCode=en&DocumentPartId=&Action=Launch. [Último acceso: 05 07 2019].

[6] ABB, «abb.com/controllerwithflextpendant,» 2019. [En línea]. Available:

] https://library.e.abb.com/public/2b5b950d68a0503cc1257c0c003cb703/3HAC041344-es.pdf. [Último acceso: 05 07 2019].

[7 Á. C. Sánchez, Diseño de una estación virtual e implantación en sistema real de un robot IRB120, Sevilla:
] Universidad de Sevilla, 2018.

10 ANEXO

10.1 Anexo A: Arduino

En la carpeta *"tcp_ip_1"* adjunta en el disco del proyecto se encuentran las carpetas con ficheros "*.ino" correspondientes.*

- *"tcp_ip_1"* corresponde al apartado 4.8 del documento.
- *"tcp_ip_2"* corresponde al apartado 4.9 del documento.
- *"tcp_ip_3"* corresponde al apartado 4.10 del documento.

10.2 Anexo B: RobotStudio

- Ene la carpeta *"robot_estudio"* adjunta también al disco se encuentran 2 archivos:
    - *"arduino_robotstudio.mod"* este archivo es el correspondiente a la versión final de Rapid.
    - *"estacion_comunicacion.rsstn"* este archivo corresponde a la estación y controladora.
- En la carpeta *"robot_estudio"* se encuentra la carpeta "*IRB_120_3kg_0.58m_8"*, que es la estación virtual:
- En la carpeta *"robot_estudio"* se encuentra la carpeta "*Estación comprimida"*, con un archivo de

nombre "*estacion_comunicacion.rspag"*; este es el archivo mas importante ya que permite abrir la estación y todos sus componentes y librerias sin la necesidad de añadir nada mas. La versión final se abrirá al ejecutar este archivo. *Archivo>>Compartir>>Unpack and Work*

- En la carpeta *"robot_estudio"* se encuentra la carpeta "*Archivos Rapid Para Pruebas"*, con los siguientes archivos:
    - *"Comunicacion_prueba1.txt",* que deberá ser abierta y copiada en la sección de rapid de la estación y corresponde a la sección 5.1 de este trabajo
    - *"funcionesdesderobotstudio.txt",* que al igual que la anterior deberá ser abierta y pegada en la sección de Rapid de RobotStudio y corresponde a la sección 5.2 de este trabajo
    - *"comunicacionprueba_robotstudio.txt"* y "*conversor.txt"* pueden ser útiles para futuros trabajos
- En la carpeta *"robot_estudio"* se encuentra la carpeta "*StationBackups"*, que son estaciones Backups que RobotStudio va creando a lo largo del trabajo.

10.3 Anexo B: Esquematico

Dentro de la carpeta "*Esquematico_proteus8.6*" adjunta también al disco se encuentran un archivo del tipo .pdsprj de nombre "EsquematicoFinal"

10.4 Anexo B: Esquematico

Dentro de la carpeta "*Videos*" adjunta en el disco se encuentran los archivos de video al ejemplo de la sección 7.4.

10.5 Anexo C: Especificaciones *IRB120*

Specification			
Variants	**Reach**	**Payload**	**Armload**
IRB 120-3/0.6	580 mm	3 kg (4kg)*	0.3 kg

Features	
Integrated signal supply	10 signals on wrist
Integrated air supply	4 air on wrist (5 bar)
Position repeatability	0.01 mm
Robot mounting	Any angle
Degree of protection	IP30
Controllers	IRC5 Compact / IRC5 Single cabinet

Movement			
Axis movements	**Working range**	**Maximum speed**	
		IRB 120	**IRB 120T**
Axis 1 Rotation	+165° to -165°	250 °/s	250 °/s
Axis 2 Arm	+110° to -110°	250 °/s	250 °/s
Axis 3 Arm	+70° to -110°	250 °/s	250 °/s
Axis 4 Wrist	+160° to -160°	320 °/s	420 °/s
Axis 5 Bend	+120° to -120°	320 °/s	590 °/s
Axis 6 Turn	+400° to -400°	420 °/s	600 °/s

Performance		
	IRB 120	**IRB 120T**
1 kg picking cycle		
25 x 300 x 25 mm	0.58 s	0.52 s
25 x 300 x 25 with 180° axis 6 reorientation	0.92 s	0.69 s
Acceleration time 0-1 m/s	0.07 s	0.07 s

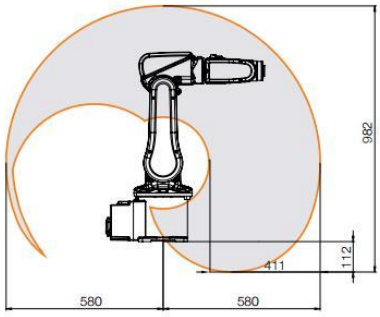
Electrical connections	
Supply voltage	200–600 V, 50/60 Hz
Rated power Transformer rating	3.0 kVA
Power consumption	0.25 kW

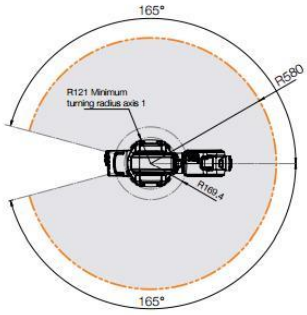
Physical	
Dimension robot base	180 x 180 mm
Dimension robot height	700 mm
Weight	25 kg

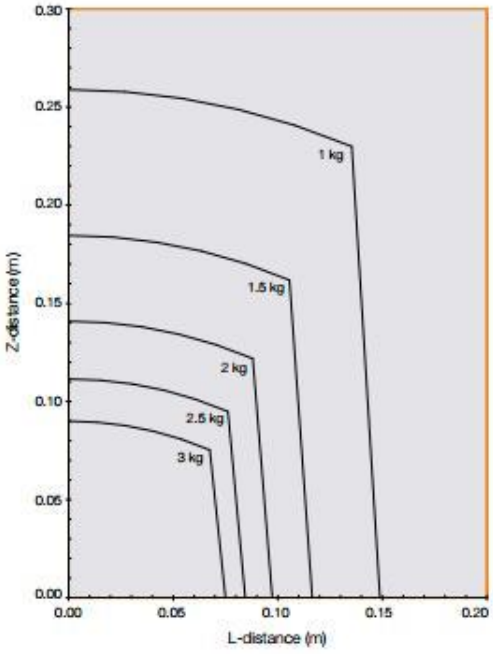
Environment	
Ambient temperature for Robot manipulator:	
During operation	+5°C (41°F) to +45°C (122°F)
Relative transportation and storage	-25°C (-13°F) to +55°C (131°F)
For short periods	up to +70°C (158°F)
Relative humidity	Max 95%
Options	Clean Room ISO class 5 (certified by IPA)**
Noise level	Max 70 dB (A)
Safety	Safety and emergency stops 2-channel safety circuits supervision 3-position enabling device
Emission	EMC/EMI-shielded

* With vertical wrist
** ISO class 4 can be reached under certain conditions
Data and dimensions may be changed without notice

Working range at wrist center & load diagram







10.6 Anexo D: Vínculos *RobotStudio*

- Curso de *RobotStudio* en *Youtube*

  https://www.youtube.com/watch?v=4l2FT-MdqZU&list=PL7fQXFqChkTe0snNKx67OHyaZxYfdlRBz

- Curso de *RobotStudio* en *Youtube*

  https://www.youtube.com/watch?v=oDnHaVMZjp4&list=PLVdvHpsfqw1bX194j7Slup6ri5se2668p

- Página principal *ABB*

  https://new.abb.com/es

- Foro *ABB*

  https://forums.robotstudio.com/

- Página desarrolladores *ABB*

  http://developercenter.robotstudio.com/