Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Desarrollo del software de control de un brazo robotizado mediante Python

Autor:

Poncelas Bodelón, Rubén

Tutor:

Sanz Angulo, Pedro

Departamento de Organización de Empresas y C e I.M.

Valladolid, agosto 2014

Dr. D. Pedro Sanz Angulo, profesor del Departamento de Organización de Empresas y Comercialización e Investigación de Mercados de la Escuela de Ingenierías Industriales de la Universidad de Valladolid.

Hace constar que el proyecto fin de carrera titulado “Desarrollo del software de control de un brazo robotizado mediante Python” que presenta Don Rubén Poncelas Bodelón para optar al título de Graduado en Ingeniería de Electrónica Industrial y Automática ha sido realizado en el Departamento de Organización de Empresas y C e I.M. de la E.I.I. de Valladolid bajo mi dirección, y constituye una valiosa aportación en el campo de ingeniería.

Y para que así conste a efectos de su presentación ante el tribunal correspondiente, firmo la presente.

En Valladolid, agosto de 2014.

Fdo. Dr. D. Pedro Sanz Angulo

Deseo dedicar el presente trabajo fin de grado a mi familia, en especial a mis padres por todo el apoyo que me han brindado, lo que me ha permitido llegar hasta aquí.

A mis amigos y compañeros de carrera, por todo lo que hemos vivido juntos en estos años.

Finalmente, me gustaría extender este agradecimiento a mi tutor Pedro Sanz Angulo, por la ayuda y colaboración prestada en su realización.

Resumen

Este proyecto persigue desarrollar un software para el control del brazo robótico industrial Mitsubishi Movemaster EX RV-M1, usando el lenguaje de programación Python.

Mediante la comunicación serie se va a transmitir una serie de órdenes y comandos, desde la interfaz de usuario, para su interpretación y ejecución por parte del robot. Por lo que la implementación de la conexión con el robot y la interfaz gráfica del usuario van a ser los pilares de nuestro proyecto.

Palabras clave

Software, robot, Python, robótica.

Índice del Trabajo Fin de Grado

Índice de figuras:

Índice de tablas:

INTRODUCCIÓN

Antecedentes

Debido a que los robots nos han permitido realizar tareas de forma más precisa, rápida y durante más tiempo que los humanos, sustituyéndolos en aquellas tareas repetitivas, monótonas o peligrosas, se ha pasado de una producción artesanal, produciendo pocos productos en mucho tiempo con un alto coste por unidad, a la fabricación de grandes lotes de productos, elaborados en poco tiempo y a un coste muy inferior, debido a la economía de escala.

A la vez que surgían los robots, también aparecía la necesidad de programarlos para la realización de estas tareas, surgiendo diferentes lenguajes de programación. Un lenguaje que está teniendo una notable difusión desde su creación allá por la década de los 90, es el lenguaje de programación Python, un lenguaje multiplataforma, de muy alto nivel, explícito y simple.

Desde sus orígenes, el lenguaje de programación Python ha ido evolucionando y ganando adeptos no solo entre los usuarios sino también entre grandes empresas. Esto unido a su sintaxis clara, fácil aprendizaje y potencial nos llevó a pensar en realizar un software de control para el robot Mitsubishi, perteneciente a la célula de fabricación del departamento de Organización de Empresas de la Universidad de Valladolid. Además, gracias a la modularidad del lenguaje, se podrían aprovechar muchas partes de este para los posibles futuros proyectos.

Motivación

Debido a las condiciones actuales de evolución tecnológica constante y en un marco de una mejora continua de los sistemas productivos, estamos en la necesidad de lograr una actualización constante que proporcione la máxima eficiencia en nuestros sistemas de producción. Partiendo de esta base, nos encontramos con la necesidad de actualizar el software de control del robot Mitsubishi MoveMasterEX y para ello utilizaremos el lenguaje de programación Python (frente a los lenguajes C++ y Java empleados anteriormente).

Esto nos va a permitir disponer de un software multiplataforma, con un código de alto nivel y con una sintaxis de fácil lectura. Además, realizaremos unas mejoras en el programa, como puede ser una gestión de los errores a la hora de ejecutar los comandos.

En cuanto al robot, disponemos del Mitsubishi MovemasterEX RV-M1 que consta de cinco grados de libertad con una capacidad de carga de 1 kg, una velocidad aceptable y una gran posibilidad de programación.

Objetivos

El presente proyecto persigue desarrollar un software de control basado en Python que permita realizar el control remoto del robot Mitsubishi MoveMasterEX modelo RV-M1 que se encuentra integrado en la célula de fabricación flexible del Laboratorio de Organización Industrial ubicada en la Escuela de Ingenierías Industriales sede Paseo del Cauce de la Universidad de Valladolid.

Este proyecto sigue la misma estructura que una serie de proyectos con similar propósito pero realizados en C++ y Java, manteniendo los aspectos más significativos respecto a las versiones anteriores y mejorando algún otro:

- Conexión serie con el robot mucho más sencilla, gracias a la librería pyserial de Python.
- Posibilidad de ver la lista de puertos serie disponibles para la conexión con el software pudiendo elegir entre los distintos puertos.
- Gestión de errores de tipo II (se explicará detalladamente qué es esto dentro de la memoria).
- Continuación con la flexibilidad y modularidad, debido a que Python también es un lenguaje multiplataforma basado en clases.
- En cuanto al software de control, intentaremos que sea lo más intuitivo, gráfico y sencillo y que todas las operaciones necesarias se puedan realizar sin tener que soltar la mano del ratón, lo que nos permitirá ganar en comodidad y rapidez.

Organización o estructura de la memoria:

Tras definir el objetivo de este proyecto, se exponen a continuación los aspectos más importantes que se van a tratar en esta memoria. En concreto, el trabajo fin de grado realizado durante se ha resumido en cinco capítulos.

En el *primer capítulo* realizaremos, en primer lugar, un estudio de la evolución de la robótica industrial hasta la actualidad, para contextualizar y ver la importancia de este campo. Ya en la segunda parte del capítulo, nos centraremos en el estudio de nuestro robot, viendo sus funcionalidades, formas de funcionamiento y características de programación.

Las comunicaciones son cruciales en el funcionamiento de cualquier robot y lo han sido para el desarrollo del presente trabajo. Por tanto, en el *segundo capítulo*, nos centraremos en la comunicación serie y el protocolo RS 232-C

debido a que ésta es la forma de conexión entre el robot y nuestro ordenador para su comunicación bidireccional.

Ya en el *tercer capítulo*, pondremos el foco sobre el lenguaje de programación Python, analizando sus características, ventajas y estilo de programación. Es la base de nuestro proyecto y en él veremos cómo se ha realizado la comunicación serie y la interfaz gráfica de nuestro software.

En el *capítulo cuarto*, explicaremos la interfaz gráfica creada para el control del robot. Es un capítulo muy visual en el que explicamos uno por uno cómo funciona cada botón de la interfaz gráfica y las pantallas que van surgiendo cuando se maneja el programa. Un tema como éste se hace imprescindible para completar la documentación ya que conforma el manual de usuario de nuestro software de control del robot.

El *estudio económico*, que se muestra en el quinto capítulo, debe ser común para cualquier proyecto sea del tipo que sea. En nuestro caso, se hace un estudio detallado desglosando el coste total en las diferentes partidas en las que se incurren, para finalmente hacer un estudio de viabilidad económica para un futuro empresarial de nuestro proyecto.

Tras las *conclusiones* más relevantes extraídas a lo largo del desarrollo del proyecto y las indicaciones de *posibles mejoras* y futuros proyectos a realizar, la memoria finaliza con la *bibliografía* empleada para la elaboración de este proyecto.

Capítulo 1:

El robot MITSUBISHI RV-M1

1.1 Robótica Industrial

Un **robot** es una máquina controlada por ordenador y programada para moverse, manipular objetos y realizar trabajos a la vez que interacciona con su entorno. Su objetivo principal es el de sustituir al ser humano en tareas repetitivas, difíciles, desagradables e incluso peligrosas de una forma más segura, rápida y precisa. Algunas definiciones aceptadas son las siguientes:

> *"Dispositivo multifuncional reprogramable diseñado para manipular y/o transportar material a través de movimientos programados para la realización de tareas variadas." (Robot Institute of America, 1979).*
>
> *"Dispositivo automático que realiza funciones normalmente adscritas a humanos o máquina con forma humana." (Webster Dictionary, 1974).*

Esta última definición, sin embargo, no es la más acertada, ya que un robot no tiene por qué tener forma humana.

Los robots exhiben tres elementos claves según las definiciones adoptadas:

- **Programabilidad**: lo que significa disponer de capacidades computacionales y de manipulación de símbolos (el robot es un computador).
- **Capacidad mecánica**: que lo capacita para realizar acciones en su entorno y no ser un mero procesador de datos (el robot es una máquina).
- **Flexibilidad**: puesto que el robot puede operar según un amplio rango de programas y manipular material de formas distintas.

Con todo, se puede considerar un robot como una máquina complementada con un ordenador o como un ordenador con dispositivos de entrada y salida sofisticados.

La idea más ampliamente aceptada de robot está asociada a la existencia de un dispositivo de control digital que, mediante la ejecución de un programa almacenado en memoria, va dirigiendo los movimientos de un brazo o sistema mecánico. El cambio de tarea a realizar se verifica ordenando el cambio de programa.

La **robótica** es la rama de la tecnología que estudia su diseño y construcción. En concreto la podemos definir como:

> *"La Robótica es una ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de inteligencia." (Enciclopedia Universal,1972)*

El origen del término robot, se encuentra en Karel Čapek, escritor checo que la acuñó en 1921 en su obra dramática *Rossum's Universal Robots* a partir de la palabra checa "*robota*", que significa servidumbre o trabajo forzado.

Por su parte, el término robótica fué acuñado por Isaac Asimov, también escritor, definiéndolo como la ciencia que estudia a los robots. Asimov formuló las *Tres Reyes de la Robótica*:

1. *Un robot no puede actuar contra un ser humano o, mediante la inacción, que un ser humano sufra daños.*

2. *Un robot debe obedecer las órdenes dadas por los seres humanos, salvo que estén en conflicto con la primera ley.*

3. *Un robot debe proteger su propia existencia, a no ser que esté en conflicto con las dos primeras leyes.*

Historia de la Robótica

Desde la época de los griegos se intentó crear dispositivos que tuvieran un movimiento sin fin, que no fuera controlado ni supervisado por personas. Ya avanzando en el tiempo, concretamente en los siglos XVII y XVIII en Europa, fueron construidos autómatas humanoides muy ingeniosos que tenían algunas características de robots, realizados con mecanismos de relojería.

Estas creaciones mecánicas de forma humana deben considerarse como invenciones aisladas que reflejan el genio de hombres que se anticiparon a su época.

Hubo otras invenciones mecánicas durante la revolución industrial, muchas de las cuales estaban dirigidas al sector de la producción textil. Entre ellas se puede citar la hiladora giratoria de Hargreaves (1770), la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwright (1785), el telar de Jacquard (1801), etc.

Los robots como ya los conocemos actualmente se dieron a conocer a finales de los años 50 y principio de los 60 debido a un nuevo desarrollo de la tecnología que se desarrolla con la invención de los transistores y circuitos integrados.

El primer robot industrial fue el llamado UNIMATE, diseñado por George Devol, instalándose en una cadena de montaje de General Motors. Este robot se puede observar en la Figura 1.1.

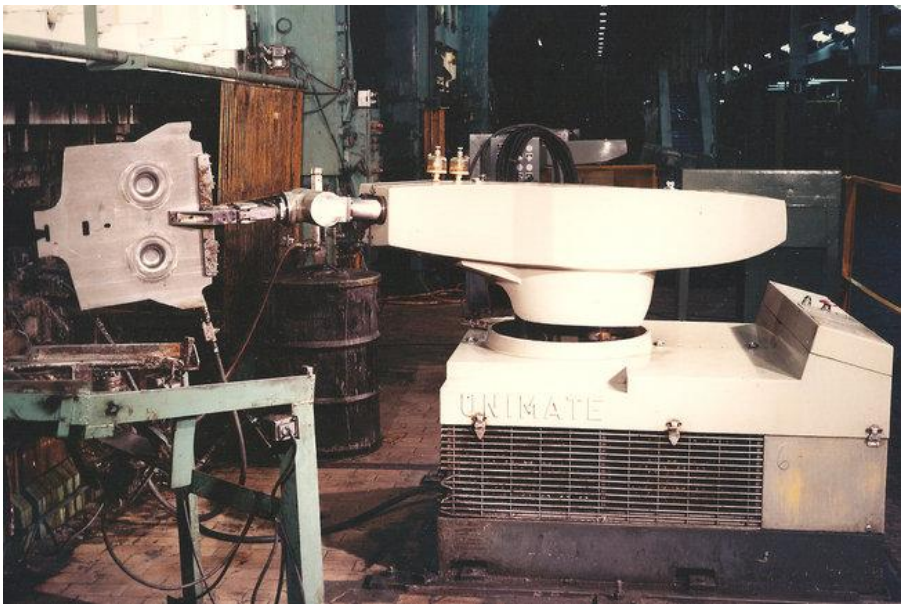


Figura 1.1 Primer Robot Industrial: El robot Unimate

Se destacan cinco hitos relevantes en el desarrollo de la Robótica Industrial, como fue el diseño en 1950 por parte del laboratorio nacional Arognne (EEUU) de manipuladores maestro-esclavo para manejar material radioactivo.

La compañía Unimation, fundada en 1956 por George Devol y Joseph Engelberger (absorbida posteriormente por Westinghouse) realiza los primeros proyectos de robots a principios de la década de los sesenta, instalando el primero en 1961 (Unimate, Figura 1.1). Posteriormente, en 1967, y debido a su éxito se instalarán más en una factoría de General Motors.

Ya en 1970 los laboratorios de la Universidad de Stanford y del MIT intentan la tarea de controlar un robot mediante un computador. A partir del 1975, la introducción del microprocesador abarató los costes de los robots a los que había que asociarles computadoras también muy caras.

Finalmente, en 1980,y gracias al fuerte impulso en la investigación por parte de las empresas fabricantes de robots, empresas auxiliares y diversos departamentos de Universidades de todo el mundo sobre la informática aplicada y la experimentación de los sensores, se produce un enorme desarrollo de la robótica.

La introducción de los microprocesadores desde los años 70 ha hecho posible que la tecnología de los robots haya sufrido grandes avances. Los modernos ordenadores han ofrecido un “cerebro” a los músculos de los robots mecánicos. Ha sido esta fusión de electrónica y mecánica la que ha

hecho posible al moderno robot siendo los japoneses los que han acuñado el término "mecatrónica" para describir esta fusión.

El año 1980 fue llamado "primer año de la era robótica" porque la producción de robots industriales aumentó ese año un 80 % respecto del año anterior. En base al rápido avance de la tecnología distinguimos varias generaciones de robots, tal y como se describe en la Tabla 1.1:

Tabla 1.1 Generaciones de robots

1ª generación: Manipuladores	Esta primera etapa se puede considerar desde la década de los 50, en donde las máquinas diseñadas contaban con un sistema de control relativamente sencillo de lazo abierto. Esto significa que no existe retroalimentación alguna por parte de algún sensor y realizan tareas previamente programadas que se ejecutan secuencialmente. Eran robots que realizaban tareas previamente programadas que sólo podían memorizar movimientos repetitivos.
2ª generación: Robots de aprendizaje	La segunda etapa se desarrolla hasta los años 80. Este tipo de robots son un poco más conscientes de su entorno en comparación con la anterior generación, disponiendo de sistemas de control de lazo cerrado en donde, por medio de sensores, adquieren información de su entorno y obtienen la capacidad de actuar o adaptarse según los datos analizados. También pueden aprender y memorizar la secuencia de movimientos deseados mediante el seguimiento de los movimientos de un operador humano.
3ª generación: Robots con control sensorial	Durante esta etapa, que tiene lugar durante los años 80 y 90, los robots cuentan con controladores (ordenadores) que, usando los datos o la información obtenida de los sensores, obtienen la habilidad de ejecutar las órdenes de un programa escrito en alguno de los lenguajes de programación que surgen a raíz de las necesidades de introducir las instrucciones deseadas en dichas máquinas.
4ª generación: Robots inteligentes	Se caracterizan por tener sensores mucho más sofisticados que mandan información al controlador y la analizan mediante estrategias completas de control. Se califican como "inteligentes" debido a sus nuevas tecnologías y estrategias. Se adaptan y aprenden de su entorno utilizando "redes neuronales" y otros métodos de análisis y obtención de datos para así mejorar el desempeño general del sistema en tiempo real.

Ya centrándonos en nuestro ámbito, podemos definir un **Robot Industrial** como una máquina que puede efectuar un número diverso de trabajos automáticamente mediante una programación informática previa.

Se caracteriza por tener una estructura en forma de brazo mediante el cual puede usar diferentes herramientas, situadas en el extremo del brazo. Además, es capaz de tomar decisiones en función de la información procedente del exterior.

El Robot industrial ha formado parte del progresivo desarrollo de la automatización industrial, y se ha visto favorecido notablemente por el avance de las técnicas de control por computadora, lo que ha contribuido de manera decisiva a la automatización en los procesos de fabricación de series de mediana y pequeña escala.

La fabricación en series pequeñas había quedado hasta ahora fuera del alcance de la automatización, debido a que requiere una modificación rápida de los equipos producción. El Robot, como manipulador reprogramable y multifuncional, puede trabajar de forma continua y con flexibilidad.

El cambio de herramienta o dispositivo especializado y la facilidad de variar el movimiento a realizar permiten que, al incorporar al Robot en el proceso productivo, sea posible y rentable la automatización en procesos que trabajan con series más reducidas y gamas más variadas de productos.

Sistemas de brazos mecánicos

Los robots más sofisticados en la ciencia, industria e investigación y desarrollo tienen al menos un brazo para sujetar, reorientar o mover objetos. Los brazos extienden el alcance de los robots y los hacen más parecidos a los humanos.

Los diseños hechos de brazos se pueden emplear como robots estacionarios del tipo de los utilizados en las fábricas, o se pueden colocar sobre un robot móvil como un apéndice. Cuando hablamos de brazos por lo general queremos decir sólo el mecanismo del brazo, excluida la mano.

Los brazos robóticos se clasifican por la forma del área que el extremo del brazo (donde se coloca la pinza) puede alcanzar. Esta área accesible se llama **envolvente de trabajo**. En beneficio de la simplicidad, la envolvente de trabajo no tiene en consideración el movimiento del cuerpo del robot sino sólo los mecanismos del brazo.

El **brazo robótico** es un manipulador multifuncional reprogramable diseñado para desplazar materiales, piezas, herramientas o dispositivos especiales, mediante movimientos variables programados y que simulan, se basan o amplían los movimientos básicos de un brazo humano.

En la Figura 1.2 podemos observar cómo el brazo mecánico imita al humano tanto en los movimientos básicos como en las articulaciones:

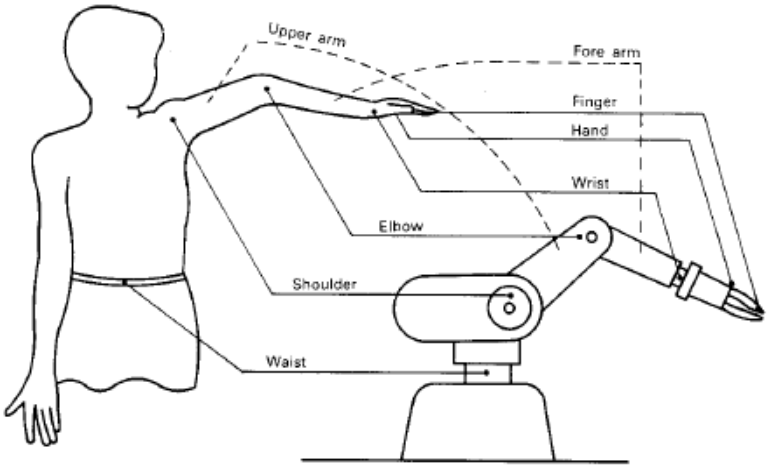


Figura 1.2 Correspondencia entre las articulaciones del brazo mecánico y el humano

El manipulador consta de un conjunto de herramientas interrelacionadas que permiten los movimientos del elemento terminal del brazo del Robot. Consta de una base para sujetarse a una plataforma rígida (como el suelo), un cuerpo donde se suele integrar la mayor parte del hardware interno que lo hará funcionar (circuitería, placas impresas, etc.), un brazo para permitir un gran movimiento en las 3 dimensiones y un antebrazo para hacer también movimientos en las 3 dimensiones aunque muy pequeños y de mucha precisión.

En un robot, los cuatro tipos más importantes de brazos robóticos son los de coordenadas angulares, coordenadas polares, coordenadas cilíndricas y coordenadas cartesianas o rectangulares.

- **Coordenadas de revolución o angular**: Los brazos con coordenadas de revolución se modelan a partir del brazo humano, de modo que tengan muchas de sus capacidades. Está formado por tres ejes rotacionales. Ejemplos de robots con este sistema, son el Robot Cincinnati Milacron o el Robot PUMA, robot que se puede ver en la Figura 1.3.

Figura 1.3 Robot PUMA

- **Coordenadas polares o esféricas**: La envolvente de trabajo del brazo de coordenadas polares tiene forma esférica, de ahí su nombre. El primer grado de libertad es la rotación de la base, siendo una especie de plataforma giratoria. Su segundo grado de libertad es la articulación del codo, que mueve el antebrazo arriba y abajo. El tercer grado de libertad se consigue variando el alcance del antebrazo, que se extiende o se retrae para llevar la pinza más o menos lejos del robot. Por tanto, nos encontramos ante un eje lineal (el tercero) y dos ejes rotacionales. Para más detalle, consultar la Figura 1.4.

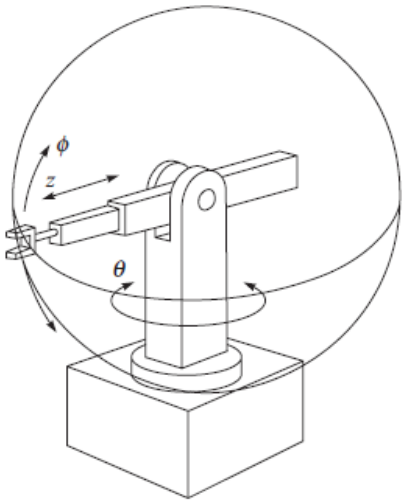


Figura 1.4 Robot de coordenadas esféricas

- **Coordenadas cilíndricas**: Su envolvente de trabajo se asemeja a un cilindro. La rotación del hombro se consigue mediante una base que gira, como en los brazos de coordenadas de revolución y de coordenadas polares. El antebrazo se fija a un mecanismo elevador y se mueve arriba y debajo de esta columna para agarrar objetos de varias alturas. Los robots que son de este tipo están constituidos por dos ejes lineales y un eje rotacional. En la Figura 1.5 se muestra un robot de este tipo.

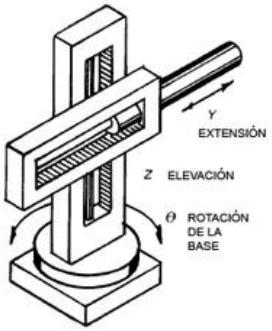


Figura 1.5 Robot de coordenadas cilíndricas

- **Coordenadas rectangulares o cartesianas**: La envolvente de trabajo del brazo de coordenadas cartesianas se parece a una caja. Es el brazo más diferente a un brazo humano y a los demás tipos de brazos robóticos; no tiene componentes giratorios, sus tres ejes principales son lineales y forman ángulos rectos unos respecto de los otros, tal y como observamos en la Figura 1.6.

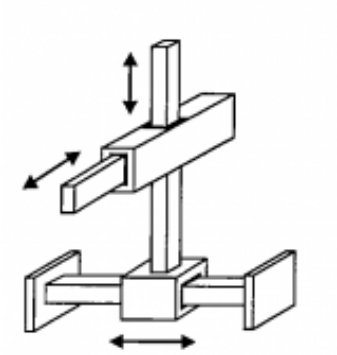


Figura 1.6 Robot de coordenadas cartesianas

En la Figura 1.7 resumimos estos cuatro tipos de tipos de robot según sus tipos de coordenadas:

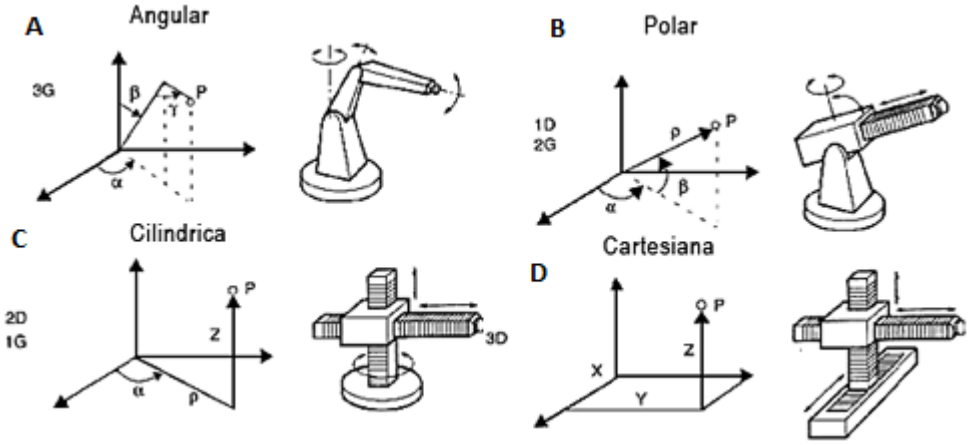


Figura 1.7 Tipos de robots según sus coordenadas

Los parámetros a tener en cuenta a la hora de diferenciar y clasificar a los robots son los siguientes:

- **Grados de libertad**: Es un parámetro fundamental para el uso que se le vaya a dar al robot. Un brazo mecánico puede admitir rotaciones, movimientos verticales, etc.
- **Alcance horizontal:** Mide la distancia (fija) de alcance horizontal entre la base del robot y el extremo del brazo.
- **Tamaño de la pinza:** Mide el ancho máximo de los "dedos" o mandíbulas cuando están completamente abiertas.

- **Peso manipulable:** Se suelen usar una serie de pesos pequeños para determinar cuál es el peso máximo aproximado que es capaz de manejar el brazo sin quedar bloqueado.
- **Área de trabajo barrida:** El área barrida por el brazo manipulador cuando trabaja. Puede ser de tres tipos: rectangular, esférica (semiesférica) y cilíndrica.
- **Técnicas de activación** Hay tres maneras en general de mover las articulaciones de un brazo robótico:
    - **Eléctrica**: La actuación eléctrica tiene que ver con el empleo de motores, electroimanes y otros dispositivos electromecánicos. Es la más sencilla y común de aplicar.
    - **Hidráulica**: La actuación hidráulica utiliza la presión de depósitos de aceite similares a los usados en equipos de movimiento de tierras y frenos de vehículos.
    - **Neumática**: La actuación neumática es análoga a la hidráulica, excepto que se emplea aire comprimido en lugar de aceite u otro fluido. Tanto los sistemas hidráulicos como los neumáticos proporcionan más potencia que los sistemas eléctricos, pero son más difíciles de usar.

1.2 El Robot MITSUBISHI MOVEMASTER EX RV-M1

El robot protagonista de nuestro proyecto es el robot Mitsubishi MoveMaster EX RV-M1 (Figura 1.8) .Es un brazo mecánico articulado con cinco grados de libertad. Va montado sobre un eje de deslizamiento o slider que le aporta otro grado de libertad aunque para este trabajo fin de grado, vamos a prescindir de él.

Su capacidad de carga es de 1,2 Kg. sin incluir el peso del efector final adaptado a él, que en nuestro caso es una mano motorizada que nos permitirá la sujeción de objetos, mayormente de forma cilíndrica.

El sistema que nos va a permitir gobernar el robot, y que se muestra en la Figura 1.9, se encuentra constituido por:

- Brazo articulado.
- Teaching box o consola de programación portátil.
- Controlador.
- Cables de conexión.
- Computador con software para establecer comunicación con el robot.

Figura 1.8 Robot Mitsubishi

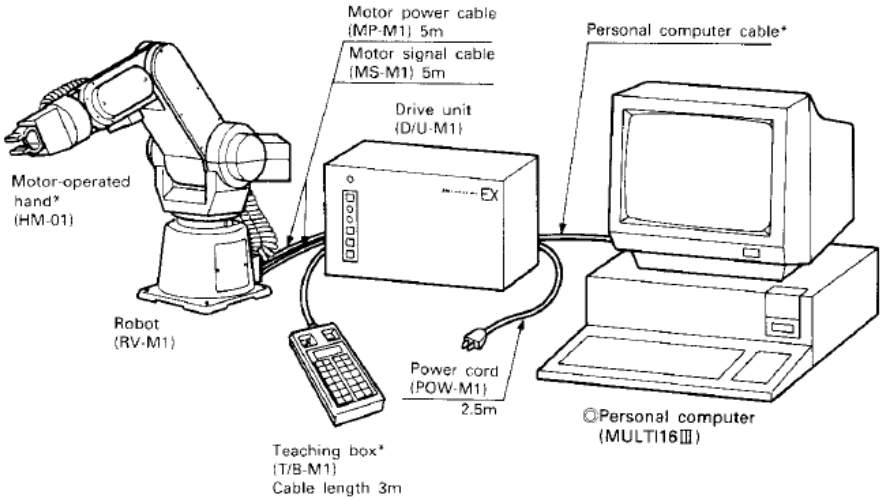


Figura 1.9 Componentes del sistema del robot Mitsubishi

El brazo cuenta con cinco articulaciones, siendo todas ellas rotacionales tal y como se pueden observar con más detalle en la Figura 1.10. Para ver con más detalle las articulaciones del robot, mostramos en la Tabla 1.2 un resumen de las articulaciones, ya que a la hora de mover el robot podremos mover individual o conjuntamente dichas articulaciones y tenemos que identificar cada articulación con su símbolo.

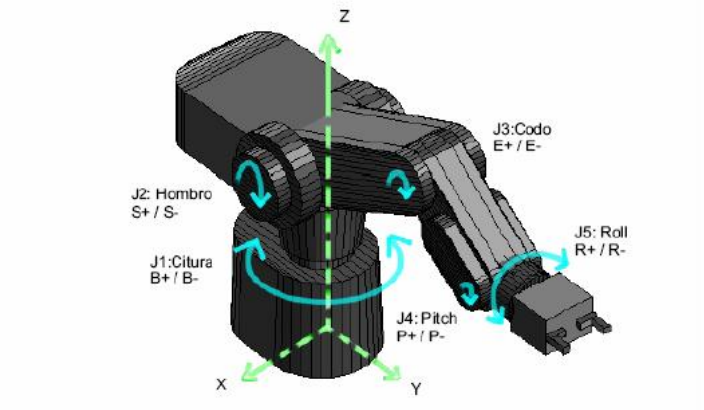


Figura 1.10 Articulaciones y grados de libertad del robot

Tabla 1.2 Articulaciones del robot

J1	Cintura (Waist).
J2	Hombro (Shoulder).
J3	Codo (Elbow).
J4	Inclinación de la muñeca (Pitch).
J5	Giro de la muñeca (Roll).

Las **dimensiones** totales del robot son las mostradas en la Figura 1.11:

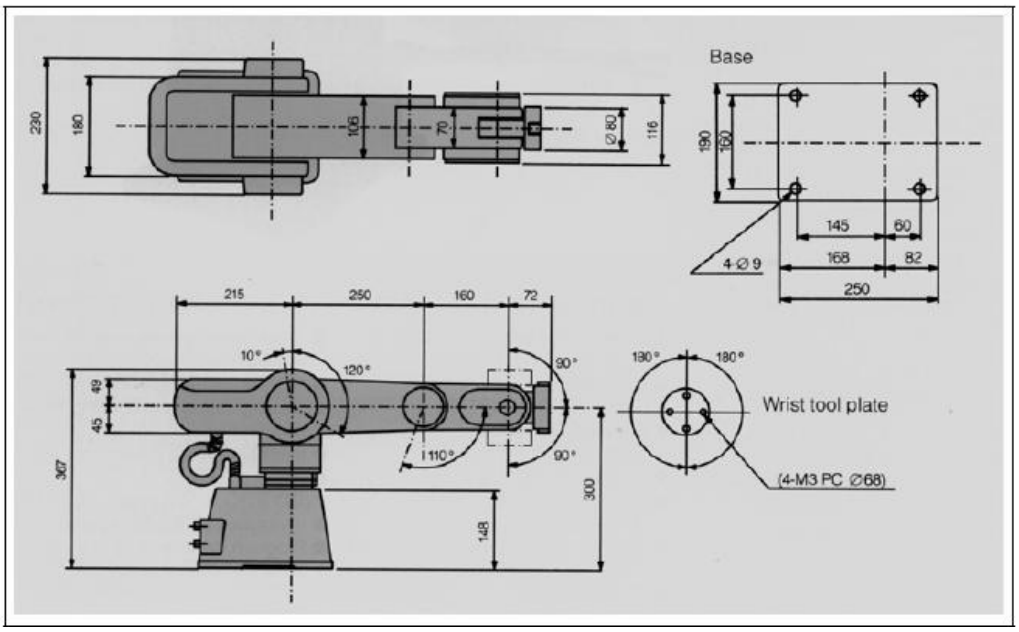
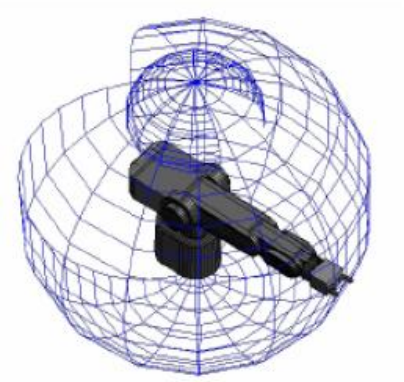


Figura 1.11 Dimensiones del robot

Cada articulación puede rotar de forma limitada, para ofrecer al robot la posibilidad de posicionarse en cualquier lugar de un espacio confinado al que se le denomina **volumen de trabajo** del robot. Las Figuras 1.12 Y 1.13 presentan los límites de movimiento para cada articulación y el volumen de trabajo del robot.



Límites de las articulaciones (grados)

J1	**-150º**	a	**150º**
J2	**-30º**	a	**100º**
J3	**0º**	a	**110º**
J4	**-90º**	a	**90º**
J5	**-180º**	a	**180º**

Figura 1.12 Volumen del trabajo del robot Mitsubishi y límites de las articulaciones

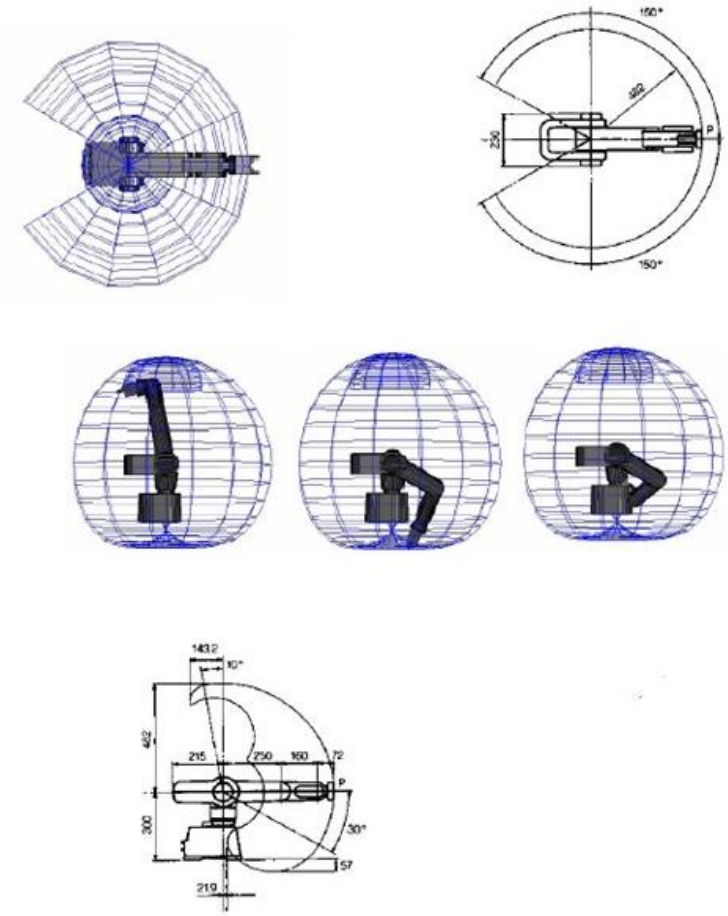


Figura 1.13 Volumen de trabajo robot

Como se observa en la Figura 1.13, la base del robot se encuentra DENTRO de su volumen de trabajo. La siguiente combinación de ángulos de movimiento por articulación, produce colisión del efector con la base del robot:

J1= (-150º a 150º), J2 = -30º, J3 = -110º, J4 = -90º, J5 = (-180º a 180º).

Según esto, el programa que maneje el robot deberá verificar en todo momento que las posiciones a las que se moverá el robot no produzcan colisiones con su base (o con objetos que se encuentren dentro de su volumen de trabajo).

El robot Mitsubishi MoveMaster permite realizar operaciones tanto en el sistema de ejes articulados como en el sistema de coordenadas cartesianas, como puede apreciarse en la Figuras 1.14 y Figura 1.15.

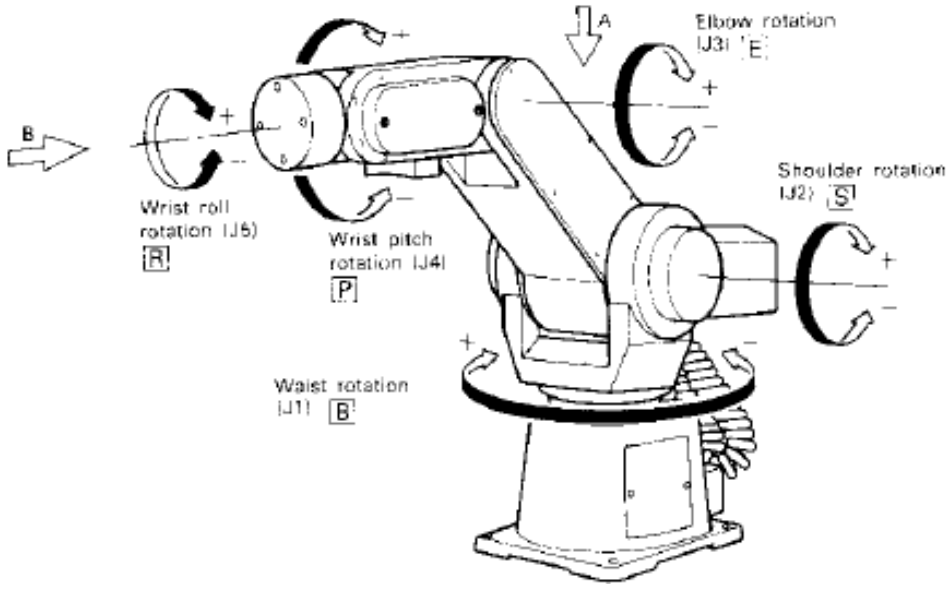


Figura 1.14 Movimiento en el sistema de ejes articulado

Tras ser encendido, el Mitsubishi siempre debe ir a orígenes (Home). Si durante la operación de llegada a orígenes hubiese riesgo de colisión con alguno de los equipos anexos al robot, deberá moverse manualmente los brazos articulados hasta una posición en la que dicho riesgo sea casi nulo.

Como consecuencia, conviene conocer los pasos necesarios para situar el robot a orígenes. Consta de 5 pasos, 3 desplazamientos y 2 rotaciones, tal y como observamos en la Tabla 1.3.

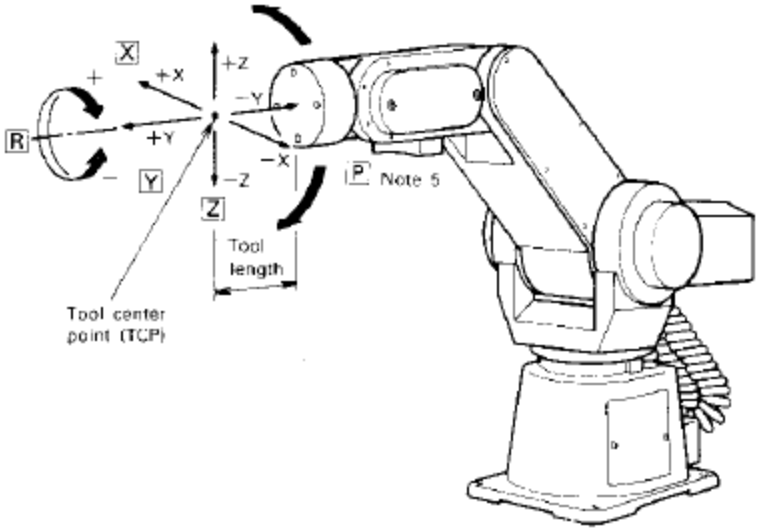


Figura 1.15 Movimiento en el sistema de coordenadas

Tabla 1.3 Pasos para ir a la posición Origen

Desplazamientos	Paso 1: El hombro (J2) se desplaza en sentido positivo.
	Paso 2: El codo (J3) se desplaza en sentido positivo.
	Paso 3: La muñeca (J4) se desplaza en sentido negativo.
Rotaciones	Paso 4: La cintura (J1) rota en sentido positivo.
	Paso 5: La muñeca (J5) rota en sentido positivo.

Encendido del sistema y controladora del robot

Si el manipulador robot se encuentra conectado a un transformador (110 a 220 V), se presiona el botón de encendido de este dispositivo. A continuación se presiona el botón de encendido del controlador del robot que se encuentra en la parte posterior del mismo.

Primero habrá que asegurarse de que el "*teaching box*" se encuentre encendido (ON) y llevar el robot a su origen mecánico "NEST" oprimiendo los botones <NST> y <ENT>.

En la Figura 1.16 se observa que el controlador cuenta en la parte frontal con una serie de botones e indicadores.

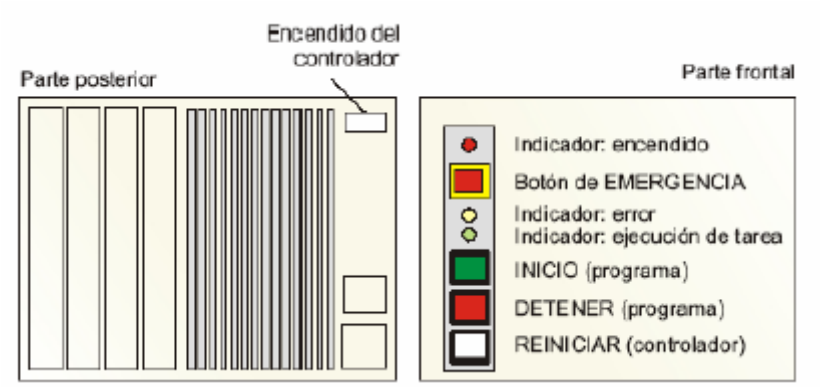


Figura 1.16 Partes posterior y frontal del controlador del robot

- **Indicador de encendido (POWER)**: LED que emite una luz amarilla si el controlador está encendido. Si hemos encendido el controlador y este LED no emite la luz, es probable que no llegue corriente al controlador. Habrá que revisar que el fusible de la parte posterior no se haya fundido, que las conexiones eléctricas no se hayan estropeado, etc.
- **Botón de emergencia (Emg. Stop)**: Interrumpe la ejecución de cualquier tarea en desarrollo, dando lugar a una alarma sonora intermitente que se suspende apagando el controlador. Además, al presionar este interruptor, el LED situado justamente por debajo de él (LED de error) comenzará a parpadear.
- **Indicador de error**: Se activa cuando se produce un error. Generalmente, se activa simultáneamente con una alarma sonora. Si se produce un *Error de Tipo I*, esta luz parpadea a intervalos de medio segundo y el sonido emitido es también intermitente, mientras que si el *Error* es de *Tipo II*, la luz brilla *consta*ntemente sin parpadear y el pitido es continuo.
- **Indicador de ejecución de tarea (Execute):** LED que emite un color verde mientras un comando está siendo ejecutado y se apaga cuando dicho comando ha sido completado. También luce mientras se está ejecutando el programa interno del controlador.
- **Botón de inicio (Start)**: Permite empezar la ejecución del programa almacenado en memoria, o lo reinicia si estaba suspendido.
- **Botón de parada (Stop)**: Suspende la ejecución del programa. Al presionar este botón, el robot completa la ejecución de la línea de comando actual antes de parar.
- **Botón para reiniciar el controlador (RESET)**: Se utiliza para eliminar un *Error de Tipo II*, ya que restablece el sistema tras una parada de emergencia, para reinicializar un programa suspendido (bien porque se produjo un *Error de Tipo II* o bien porque se presionó el interruptor de Parada de Emergencia). Tras presionar este botón, el controlador se sitúa en la primera línea del programa a la espera de que se presione el botón START. Por tanto, el botón RESET además apagará el LED indicador de error y suspenderá el sonido continuo de la alarma.

Al explicar el indicador de error, hemos hablado de Error de Tipo I y Error de Tipo II. El **Error tipo I** corresponde a errores de hardware aunque puede haber varias razones que motiven un error de este tipo: las más comunes son que el robot choque con un equipo u objeto externo o que el cable que transmite la corriente a la mano motorizada impide su desplazamiento. Para eliminar este tipo de error, la única solución consiste en apagar el controlador. Recordar que, al volver a encender el robot, debemos comprobar que el desplazamiento a orígenes esté libre de peligro de colisión.

El **Error tipo II** se corresponde a errores de software siendo las principales causas de su origen que el comando transmitido por el ordenador personal es erróneo; por ejemplo si hemos enviado un comando indefinido, el formato de transmisión es erróneo, o se ha producido un error durante la transmisión.

También puede suceder cuando el comando no puede ser ejecutado; esto se debe a que los parámetros exceden un rango o estamos ordenando al robot moverse a una posición no definida.

Aquí se engloba uno de los errores más comunes que se suele dar y que consiste en intentar mover el robot fuera de alguno de sus límites del volumen del trabajo. Tal y como se ha dicho, para solucionar un Error de Tipo II se debe presionar el botón RESET del controlador.

Para continuar con la descripción de la controladora, nos fijaremos en el panel lateral de la misma que se puede apreciar en la Figura 1.17.

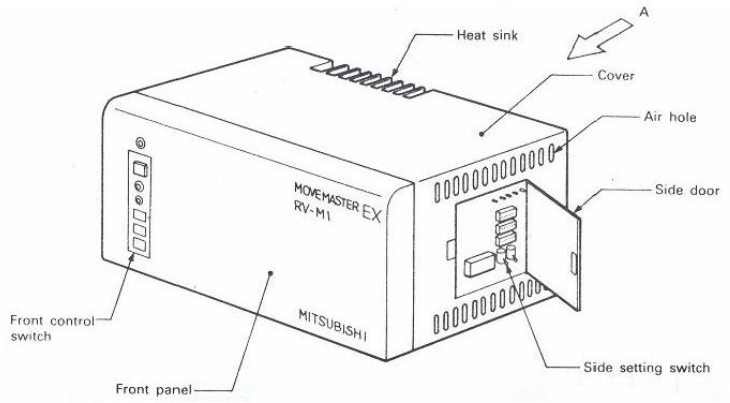


Figura 1.17 Panel lateral de la unidad controladora

Donde podemos distinguir los siguientes interruptores laterales (*Side setting switchs*):

> **Interruptor ST1:** Establece el modo de control. Si se encuentra en la posición superior, es el microprocesador del propio controlador quien posee el control. En cambio, si se encuentra en la posición inferior, será la computadora la encargada de dicho control. Por tanto, para usar el programa de control remoto, el interruptor ST1 deberá estar en la *posición inferior.*

Interruptor ST2: Si se encuentra en la posición superior, los datos de la EPROM son transferidos a la memoria RAM de la unidad controladora y si se encuentra en la posición inferior, los datos de la EPROM NO son transferidos a la RAM de la unidad controladora. En nuestro caso, al no disponer de memoria EPROM, el interruptor ST2 deberá estar en la *posición inferior.*

Justo encima de estos interruptores, se encuentran otros tres interruptores, cada uno de ellos formado por 8 bits o patillas:

El **Interruptor SW1:** Compuesta por 8 bits, numerados del 1 al 8 de izquierda a derecha con las funciones recogidas en la Tabla 1.4:

Tabla 1.4 Posiciones interruptor SW1

Bit 1	Selecciona la terminación para la transmisión de datos desde la conexión RS-232-C. Para la posición superior: CR + LF ("\r" + "\n") y para la posición inferior: CR ("\r"). Se pondrá en la posición inferior (a no ser que se esté usando MULTI16).
Bit 2	Indica si se revisa o no que los contenidos de la memoria RAM son retenidos cuando no está encendido; si se encuentran en la posición superior se revisa. Si en cambio se encuentra en la posición inferior, no se revisa. Se pondrá en la posición inferior si no se usa la batería.
Bit 3	Selecciona el tipo de tarjeta de entrada/salida usada; para la posición superior es del Tipo A16 o B16 y si está la posición inferior: es del Tipo A8 o B8.
Bit 4	Selecciona si se establecen, se cambiar o se borrar las referencias a las posiciones de los datos en un sistema de coordenadas cartesiano, en la posición superior se activa mientras que en la posición se desactiva. Activaremos esta opción cuando hayamos establecido un sistema de referencia en coordenadas cartesianas y queramos enviar los datos escritos en la EPROM a la RAM.
Bit 5	Selecciona entre el uso de los interruptores de la parte frontal del controlador o el de las señales externas para cargar el programa mientras la tarjeta de tipo A16 o B16 está siendo usada; la posición superior selecciona las señales externas. Mientras que la posición inferior señala los interruptores de la parte frontal del controlador. Si se usa una tarjeta A8 o B8 deberá ponerse en la posición inferior.
Bit 6	Nos permite desactivar la tecla ENT del "teaching box" para liberar los frenos del robot. Normalmente, este bit se encontrará en su posición inferior.
Bit 7	No se usa.
Bit 8	Permite activar o desactivar la alarma sonora. En la posición superior se emite un pitido cuando ocurre un error mientras que en la posición inferior no se emite sonido alguno al haber error.

-SW2: Establece el formato para la transmisión de datos asíncrona. Para ver de la función que tiene cada bit del SW2, remitimos al lector a una descripción la sección 2.7.

-SW3: Establece la tasa de baudios, es decir, la velocidad media de transferencia que va a utilizar el controlador. Para más información sobre este interruptor, remítase a la sección 2.7.

En la Figura 1.18 se muestra la posición que se ha adoptado para las patillas de cada interruptor descrito.

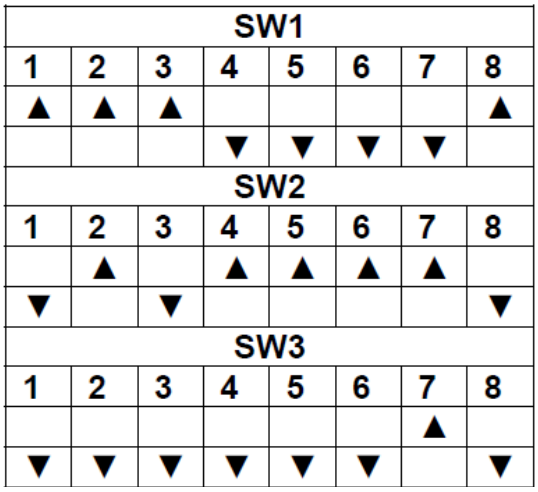


Figura 1.18 Estado de las patillas de los interruptores SW1, SW2 y SW3

Con esta combinación, los parámetros del robot son los siguientes:

- Secuencia de terminación: CR + LF (“\r” + “\n”).
- 4800 baudios.
- 7 bits de datos.
- Paridad par.
- 1 bit de parada.

Además de los anteriores elementos descritos, en la parte posterior de la controladora (Figura 1.19) encontramos otros elementos, como son el conector CENTRONICS (interfaz paralela que veremos más adelante), conector RS-232-C, conector para un equipo externo de entrada/salida; conector del “teaching box”, conector del cable a red, conector del cable de señal, terminal de entrada para la conexión de un equipo de parada de emergencia externo, toma a tierra, selector CC/CA, fusible, interruptor ON/OFF de la unidad y entrada CA.

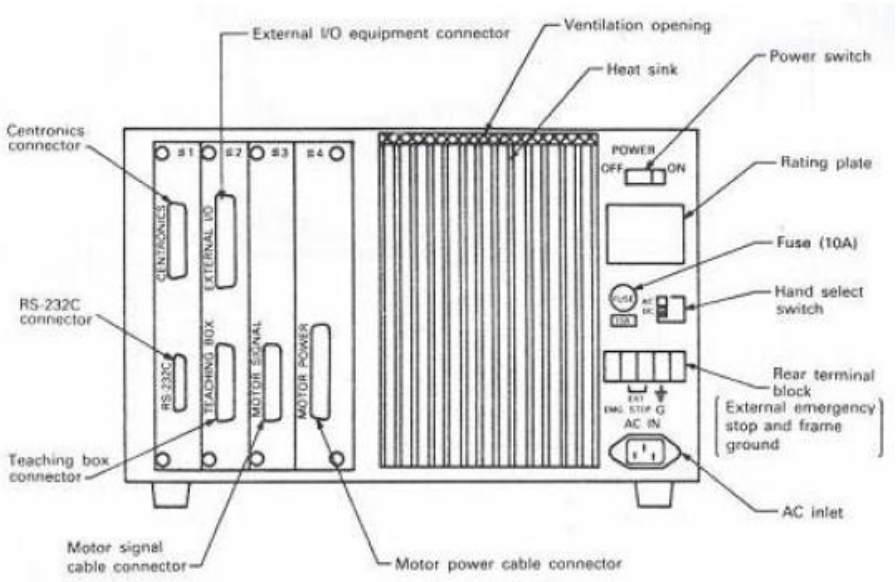


Figura 1.19 Parte posterior de la unidad controladora

En cuanto al "Teaching box", este dispositivo permite manejar manualmente el robot (Figura 1.20). Cuenta con un teclado cuyos botones presentan varias funciones según el modo configurado.

Figura 1.20 Cuadro de mando del "teaching box".

Obsérvese que en su parte inferior se encuentra un botón de parada de emergencia así como un interruptor ON/OFF. Este interruptor es el que activa/desactiva la consola de programación portátil o teaching box.

Si deseamos controlar el robot mediante el ordenador central o mientras el programa del robot se esté ejecutando, deberemos ponerlo en OFF. Si por el contrario queremos manejar el robot manualmente, lo pondremos en ON. Durante la realización de este proyecto deberá colocarse, según lo indicado, en OFF.

En la Tabla 1.5 se presentan los comandos más utilizados cuando se opera manualmente el robot. En ella se incluyen la secuencia de botones que se utiliza para habilitar cada una de las funciones del "teaching box".

Tabla 1.5 Funciones más utilizadas del "teaching box"

Secuencia de Botones	Función
INC	Movimiento del robot a la siguiente posición definida en memoria
DEC	Movimiento del robot a la anterior posición definida en memoria
P.S <núm.> ENT	Guarda la posición del robot asignándole un número
P.C <núm.> ENT	Borra La posición correspondiente al número indicado
NTS	Lleva el robot a su origen mecánico (inicialización)
ORG	Inicializa rotación por articulación(J1=J2=J3=J4=J4=0°)
MOV <núm.> ENT	Mueve a la posición asociada a dicho número
PTP	Activa la opción de movimiento en coordenadas cartesianas
XYZ	Activa la opción de movimiento en coordenadas cartesianas
X+ o X-	Mueve la mano paralela al eje X (coordenadas cartesianas)
Y+ o Y-	Mueve la mano paralela al eje Y (coordenadas cartesianas)
Z+ o Z-	Mueve la mano paralela al eje Z (coordenadas cartesianas)
B+ o B-	Movimiento de la articulación J1(cintura)
S+ o S-	Movimiento de la articulación J2(hombro)
E+ o E-	Movimiento de la articulación J3(codo)
P+ o P-	Movimiento de la articulación J4(pitch)
R+ o R-	Movimiento de la articulación J5 (roll)
O	Abrir pinza
C	Cerrar pinza

Programación del Robot

Existen 2 modos de control posibles para el Robot MoveMaster, que son:el modo de ordenador personal y el modo de controladora.

Para establecer el **modo de control a través del ordenador**, la lengüeta del interruptor ST1 debe estar en su posición INFERIOR. Este modo permite al ordenador ejecutar los comandos directamente, escribir y transferir un programa, y comenzar el programa transferido a la memoria RAM desde la controladora (Este modo corresponde al sistema de configuración centrado en el ordenador personal).

Este es el *modo utilizado en el presente proyecto* para generar y ejecutar los diferentes programas que el software de control podrá importar. Los programas redactados contendrán órdenes de movimiento y posición.

Existen tres fases distintas para establecer la programación del robot en este modo de control. Durante estas operaciones, hay que asegurarse de mantener el interruptor ON/OFF del "teaching box" apagado, OFF:

a) **Ejecución directa**: Se ejecutan directamente los comandos inteligentes del MoveMaster. Por ejemplo, si queremos mover el robot a un punto previamente almacenado (posición 1) – usando el comando "MO" (Move) – se enviará la cadena de caracteres: *"MO 1"* (Mover a la posición 1)
En código ASCII (los caracteres de bajo nivel son automáticamente convertidos en caracteres de alto nivel).
Esto corresponde a la sentencia:

LPRINT #1, "MO 1" para la interfaz CENTRONIC, y

PRINT #1, "MO 1" para la interfaz RS-232-C (el espacio puede ser omitido).

Los comandos son enviados secuencialmente y ejecutados de uno en uno, no formando un programa almacenado en la unidad controladora.

b) **Generación de un programa**: En esta fase, el ordenador personal genera un programa usando los comandos del MoveMaster. El programa es almacenado en la memoria RAM de la controladora.
Por ejemplo, para escribir un programa que realice la misma función que el mostrado anteriormente (moverse hasta la posición 1), se enviará la cadena de caracteres:

"10 MO 1"

Donde el número *"10"* representa el número de línea del programa, que identifica la orden de almacenaje en memoria, como se solía hacer en BASIC. El programa es entonces ejecutado en orden según los números de línea. El rango de números posibles que se pueden utilizar va desde el 1 hasta el 2048. Cualquier número de línea mayor que éste, provocará un error.

El equivalente CENTRONIC para la sentencia de arriba es:

LPRINT "10 MO 1"

Mientras que para el RS-232-C es:

PRINT #1, "10 MO 1" (el espacio puede ser omitido).

c) **Ejecución de un programa**: Por último, el programa almacenado en la memoria RAM de la controladora se ejecuta. El programa comienza cuando se envía el comando "RN" (que corresponde a "RUN", el comando de comienzo de BASIC).

El equivalente CENTRONIC para la sentencia de arriba es:

LPRINT "RN"

Mientras que para el RS-232-C es:

PRINT #1, "RN".

Por otro lado, para trabajar en el **modo de controladora:** la lengüeta del interruptor ST1, localizada en el interior de la puerta lateral de la controladora, deberá ponerse en su posición SUPERIOR. En este caso, se permite la ejecución del programa almacenado en la EPROM de la controladora (Este modo corresponde al sistema de configuración centrado en la controladora). De esta forma, los interruptores de la parte frontal de la unidad controladora (véase figura 1.16) pueden usarse para encender, parar o poner a cero (reset) el programa. Si se usara una tarjeta de tipo A16 ó B16, esta operación es posible por medio de señales externas. En este modo, cualquier comando enviado desde el ordenador es ignorado*. Esta forma de programación no va ser utilizada en este proyecto.*

Comandos de programación

Para programar el robot se utilizan diferentes comandos que se clasifican de la siguiente forma:

- Instrucciones de control de posición y movimiento.
- Comandos para estructurar los programas.
- Instrucciones de control de la mano (*gripper*).
- Comandos de control de entrada salida I/O.
- Instrucciones de lectura a través de RS-232.

Las instrucciones marcadas con asterisco (*) se ejecutan inmediatamente después de ser enviadas y no pueden ser incluidas en un programa en el que el número de línea que precede cada comando indique la secuencia que desarrollará el robot.

Control de posición y movimiento. Los comandos de control de posición y movimiento del robot, se muestran en la Tabla 1.6.

Tabla 1.6 Comandos para el control de posición y movimiento

DP	A partir de la posición actual, el robot se mueve a la anterior posición definida
DW	DW < *distancia en x* >, < *distancia en y* >, < *distancia en z* >. Conservando la orientación, el robot desplaza la mano desde el punto en el que se encuentra, hasta un nuevo punto a una distancia determinada por los parámetros anteriores, en los ejes X, Y y Z
HE	HE < *número de la posición*>. Guarda la posición actual asignándole el número suministrado como parámetro. Debe cumplirse que: 1 < *número de la posición <629*
HO	Establece la posición de referencia en el sistema de coordenadas cartesianas
IP	Lleva el robot a la siguiente posición definida.
MA	MA < *posición 1*>, < *posición 2*>, < *O ó C* >. Mueve el robot a la posición que se obtiene al sumar las componentes de las posiciones 1 y 2. Estas componentes son las coordenadas X, Y y Z, en las que se encuentra la mano y los ángulos correspondientes a *pitch* y *roll*, que determinan su orientación.
MC	MC < *posición 1* >, < *posición 2* >. El robot se mueve en forma continua entre la *posición 1* y *posición 2*, pasando a través de las posiciones intermedias que hayan sido declaradas.
MJ	MJ < *cintura* >, < *hombro* >, < *codo* >, < *pitch* >, < *roll* > *(*)* Mueve el robot por articulaciones. Cada parámetro se suministra en grados según los límites presentados anteriormente.
MO	MO< *posición* >, < *O ó C* >. Movimiento a la posición seleccionada con la mano abierta o cerrada (O ó C).
MP	MP < *coord. X* >, < *coord. Y* >, < *coord. Z* >, < *ángulo Pitch* >, < *ángulo roll* > *(*).* Mueve la mano del robot al punto dado por las coordenadas X, Y y Z, con orientación definida por los ángulos *pitch* y *roll.*
MS	MS < *posición* >, < *número de puntos intermedios* >, < *O ó C* > Genera movimiento desde la posición actual hasta la nueva posición pasando través de un número definido de puntos intermedios.
MT	MT < *posición* >, < *distancia* >, < *O ó C* > Movimiento en dirección de la herramienta a partir de la posición dada, a lo largo de la distancia definida.
NT	NT Lleva el robot a su origen mecánico.

OG	OG Lleva el robot a la posición de referencia en el sistema de coordenadas cartesianas establecido por el comando HO.
PA	PA < número de la paleta >, < número de columnas >, < número de filas > Define el número de puntos de red tanto en filas como en columnas para una determinada paleta o rejilla.
PC	PC < *posición 1* >, < *posición 2* > Borra las posiciones definidas en el intervalo comprendido entre la posición 1 y la posición 2.
PD	PD < *posición* >, <*coord. X* >, < *coord. Y* >, < *coord. Z* >, < *Pitch* >, < *Roll* > *(*)* Crea una posición en las coordenadas dadas y con la orientación definida por los ángulos *pitch* y *roll.*
PL	PL < *posición 1* >, < *posición 2* > Asigna las coordenadas y ángulos correspondientes a la *posición 2* en la *posición 1*, borrando su contenido en caso de que ésta se encontrara definida previamente.
PT	PT < número de paleta > Calcula las coordenadas de una red de puntos de una paleta e identifica las coordenadas de una posición con una paleta determinada.
PX	PX< *posición 1* >, < *posición 2* > Asigna la *posición 1* a la *posición 2* y viceversa.
SF	SF < *posición 1* >, < *posición 2* > Asigna a la *posición 2* la suma de las coordenadas y ángulos de las *posiciones 1* y *2*.
SP	SP < *nivel 0 a 9* >, < *H o L* > Define la velocidad de movimiento del robot y su aceleración, que puede ser alta (H) o baja (L).
TI	TI < *contador de 0 a 327ó7* > Espera un periodo de tiempo en segundos equivalente al valor del contador dividido entre 10.
TL	TL < *longitud* > Permite variar la longitud de la herramienta o efector final (la mano) utilizados por el robot, para que los cálculos de posición sean realizados en función de esta nueva dimensión.
CP	CP < *contador de 1 a 99* > Permite seleccionar un contador para comparar el valor acumulado con otro valor en una instrucción posterior.

DA	DA < *número de bit* > Deshabilita la interrupción correspondiente al bit de entrada que se encuentre habilitado previamente.
DC	DC < *contador* > Reduce el contador restando 1 al valor acumulado actual.
DL	DL < *número de línea 1* >, < *número de línea 2* > *(*)* Borra el contenido de la memoria de programa (2048 líneas), desde la *línea 1* hasta la *línea 2*.
EA	EA < + *ó* - > < *número de bit* > < *número de línea* > Permite la interrupción por una señal dada a través del bit específico del terminal de entrada externo, y especifica el número de línea al que el programa salta cuando ocurre la interrupción.
ED	ED Finaliza el programa.
EQ	EQ < *valor* >, < *número de línea* > Salta a la línea indicada por el parámetro *número de línea,* si *valor* es igual al contenido de un contador seleccionado previamente mediante la instrucción CP.
GS	GS < *número de línea* > Salto a una subrutina que inicia en la línea suministrada como parámetro.
GT	GT < *número de línea* > Salto a la línea indicada.
IC	IC < *contador* > Incrementa en 1 el valor acumulado en el contador indicado.
LG	LG < *valor* >, < *número de línea* > Salta a la línea indicada por el parámetro *número de línea,* si el parámetro *valor* es mayor que el contenido de un contador seleccionado previamente mediante la instrucción CP. El rango para el parámetro *valor* es: -32768 < *valor* < 32768.
NE	NE < *valor* >, < *número de línea* > Salta a la línea indicada por el parámetro *número de línea,* si el parámetro *valor* es diferente al contenido de un contador seleccionado previamente mediante la instrucción CP.
NW	NW Borra el programa y las posiciones que se encuentren almacenadas en memoria en el controlador del robot.
NX	NX Indica el final de un ciclo.

RC	RC < *número de repeticiones* > Indica el número de veces que debe repetirse el fragmento de programa consecutivo, cuyo final es la instrucción NX. El parámetro *número de repeticiones* puede tomar un valor comprendido entre 1 y 32767.

Control de la mano: Los comandos de control de la mano del robot, se muestran en la Tabla 1.7.

Tabla 1.7 Comandos para el control de la mano.

GC	GC Cerrar la mano.
GF	GF < *estado 0 ó 1* > Establece el valor de una bandera que permite mantener el estado (abierto o cerrado) de la mano durante la ejecución de varias instrucciones.
GO	GO Abrir la mano.
GP	GP < *fuerza inicial* >, < *fuerza de retención* >, < *tiempo de aplicación de fuerza inicial* > Gradúa parcialmente la presión de la mano al momento de cerrar y abrir la mano.

Control de Entrada y Salida (I/O) de Datos: Los comandos para el control de entrada y salida de datos del robot, se muestran en la Tabla 1.8.

Tabla 1.8 Comandos para la entrada y salida de datos.

ID	ID Captura directa de datos en el puerto de entrada.
IN	IN Captura de datos de forma sincronizada, considerando otras señales disponibles en el puerto de entrada-salida.
OB	OB <+ *ó* - >, < *número de bit* > Activa (+) o desactiva (-) uno de los bits de salida (0 a 15)
OD	OD < *dato* > Fija directamente un dato (1ó bits) en la salida.

OT	OT < *dato* > Genera datos de forma sincronizada, considerando otras señales disponibles en el puerto de entrada-salida.
TB	TB < + *ó* - >, < *número de bit* >, < *número de línea* > Verifica el estado de un bit de entrada cuyo número es suministrado por el segundo parámetro. Si éste se encuentra en el estado indicado por el primer parámetro, se produce un salto a la línea de programa definida por el tercer parámetro.

Control de comunicación a través de un puerto RS-232-C: Los comandos de control de comunicación, a través de un puerto RS-232-C del robot, se muestran en la Tabla 1.6.

Tabla 1.9 Comandos para el control de comunicación a través del puerto RS-232-C.

CR	CR < contador 1 a 99 > Lee el valor acumulado en el contador.
DR	DR Lee el contenido del registro interno.
ER	ER Lectura del tipo de error presente durante la ejecución de alguna instrucción.
LR	LR < número de línea > Lee el contenido de la línea de programa designada por el parámetro número de línea.
PR	PR < número de posición > Lee las coordenadas y ángulos correspondientes a la posición suministrada como parámetro.
WH	WH Lee las coordenadas y ángulos de orientación de la posición en la que se encuentra el robot.

Otros comandos: Finalmente, en la Tabla 1.10 mostramos otros posibles comandos del robot Mitsubishi.

Tabla 1.10 Otros comandos

RS	RS Puesta a cero del programa y la condición de error. Transfiere el programa y los datos de posición almacenados en la EPROM a la memoria RAM de la controladora.
WR	WR Escribe el programa generado y los datos de posición almacenados en la EPROM.
<cadena de caracteres>	*<cadena de caracteres>* Permite al programador escribir un comentario – contenido en *cadena de caracteres* -que no exceda de 7 caracteres.

Capítulo 2:

Comunicación serie

2.1 Introducción

Los ordenadores personales y los grandes sistemas automáticos necesitan un medio de comunicación con el exterior a través del cual poder intercambiar información y órdenes.

La comunicación en serie, que es de la que se va a tratar en este capítulo, es la utilizada en la comunicación con el robot Mitsubishi. Ha sido en gran medida determinante considerando que es la base para el correcto funcionamiento del programa y ha sido, con mucho, uno de los retos fundamentales que se ha tenido que superar para la realización del presente proyecto.

Por todo ello, se ha estimado conveniente la elaboración de un capítulo que introduzca el tema de las comunicaciones entre un PC y un dispositivo externo, en nuestro caso el ya citado robot.

Hablaremos del estándar de comunicación RS-232-C para posteriormente centrarnos en las particularidades propias del robot Mitsubishi en este área e indicar los pasos que se han tenido que dar para la comunicación.

Ya en el Capítulo 3 nos centraremos en las instrucciones Python implementadas para programar la correcta comunicación del PC con el robot (básicamente el uso de la librería pyserial).

Debido al estrecho vínculo que ha existido siempre entre puerto serie y módem, se suele utilizar este término para explicar la terminología de la comunicación serie. En nuestro proyecto no existen módems, por lo que nos estaremos refiriendo en todo momento al controlador remoto del robot Mitsubishi.

2.2 Comunicación serie

Las comunicaciones serie se utilizan para enviar datos a través de largas distancias, ya que las comunicaciones en paralelo exigen demasiado cableado para ser operativas. Los datos serie recibidos son convertidos posteriormente a paralelo para ser manejados por el bus del PC.

Los equipos de comunicaciones serie se pueden dividir entre:

- **Simplex:** Una comunicación serie simplex envía información en una sola dirección (por ejemplo, una emisora de radio comercial).
- **Half-duplex:** Significa que los datos pueden ser enviados en ambas direcciones entre dos sistemas, pero en una sola dirección al mismo tiempo.
- **Full-duplex:** En una transmisión full-duplex cada sistema puede enviar y recibir datos al mismo tiempo.

En cuanto a la comunicación serie, existen dos formas posibles: asíncrona y síncrona. En la forma de **transmisión asíncrona** se envían cadenas de bits no suficientemente largas de forma ininterrumpida. Es decir, los datos se transmiten enviándolos carácter a carácter. Normalmente, cada carácter tiene una longitud de 5 a 8 bits. La sincronización se debe mantener solamente durante cada nuevo carácter. Esta técnica se va a explicar con la ayuda de la Figura 2.1.

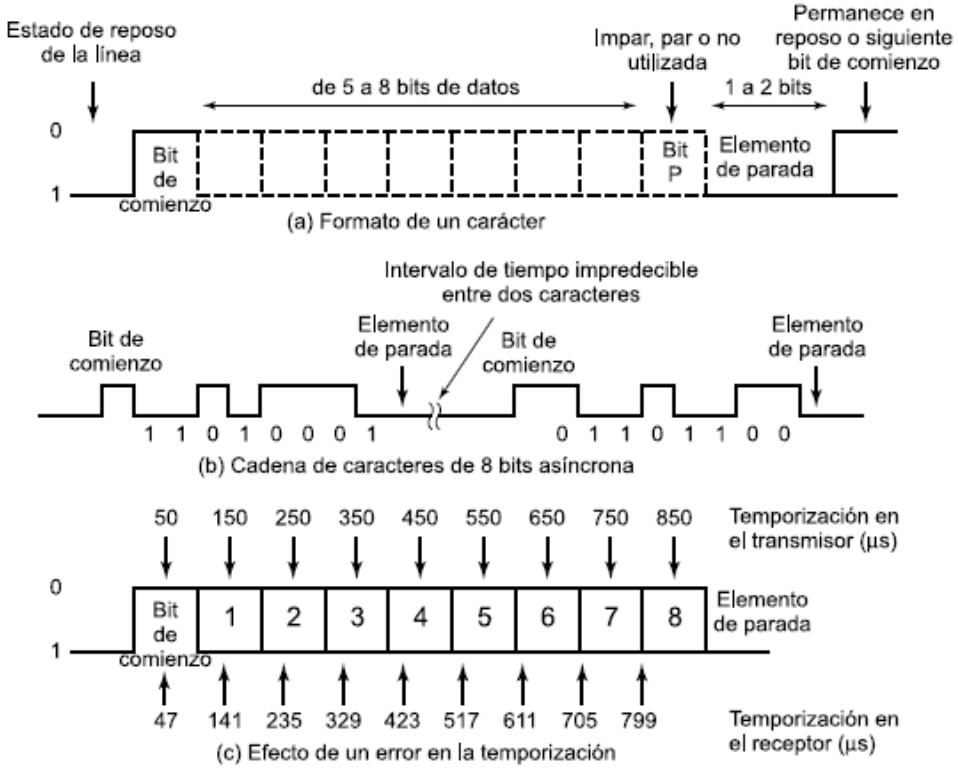


Figura 2.1 Transmisión asíncrona

Cuando no se transmite ningún carácter, la línea entre el emisor y el receptor estará en estado de *reposo*. La definición de *reposo* es equivalente al elemento de señalización correspondiente al 1 binario. Así, en la señalización habitual en la transmisión asíncrona, el estado de reposo correspondería con la presencia de una tensión negativa en la línea. El principio de cada carácter se indica mediante un *bit de comienzo* que corresponde al valor binario 0. A continuación se transmite el carácter, comenzando por el bit menos significativo, que tendrá entre cinco y ocho bits.

A modo de ejemplo a los bits de datos se les añade un bit de paridad, el cual ocupa, por tanto, la posición correspondiente al bit más significativo. El bit de paridad se determina en el emisor, de tal manera que el número de unos dentro del carácter, incluyendo el bit de paridad, sea par (paridad par) o impar (paridad impar), dependiendo del criterio que se elija. Este bit se usa en el receptor para la detección de errores.

Por último, está el denominado elemento de parada, que corresponde a un 1 binario. Se debe especificar la longitud mínima del elemento de parada, la cual normalmente es igual a 1, 1,5 o 2 veces la duración de un bit convencional. No se especifica un valor máximo. Debido a que el elemento de parada es igual que el estado de reposo, el transmisor seguirá transmitiendo la señal de parada hasta que se transmita el siguiente carácter.

Este esquema no es muy exigente en cuanto a los requisitos de temporización. Por ejemplo, usualmente los caracteres IRA se envían como unidades de 8 bits, incluyendo el bit de paridad. Si el receptor es un 5 por ciento más rápido, o más lento, que el emisor, el octavo muestreo estará desplazado un 45 por ciento, lo que significa que todavía es aceptable.

En la Figura 2.1c se muestra el efecto de un error de temporización lo suficientemente grande como para provocar un error en la recepción. En este ejemplo supondremos una velocidad de transmisión de 10.000 bits por segundo (10 Kbps); por tanto, se transmite un bit cada 0,1 milisegundos (ms), es decir, tiene una duración de 100s. Supongamos que el receptor está fuera de sincronismo un 6 por ciento, es decir, en 6s cada intervalo de duración de un bit.

Por tanto, el receptor muestrea el carácter de entrada cada 94s). Como se puede observar, la última muestra será errónea. Un error como el anterior en realidad dará lugar a dos errores. Primero, el último bit muestreado será incorrecto, y segundo, la cuenta de bits puede estar desalineada. Si el bit 7 es un 1 y el bit 8 es un 0, el bit 8 se puede interpretar erróneamente como un bit de comienzo.

Este tipo de error se denomina error de delimitación de trama, ya que a la unidad constituida por el carácter más el bit de comienzo y el elemento de parada se denomina trama. Se puede dar igualmente un error de delimitación de trama si el ruido hace que se detecte un bit de comienzo erróneamente durante el estado de reposo.

La transmisión asíncrona es sencilla y de bajo coste, si bien requiere 2 o 3 bits suplementarios por cada carácter. Por ejemplo, en un código de 8 bits sin bit de paridad y con un elemento de parada de duración 1 bit, de cada diez bits, dos no contendrán información ya que se dedicarán a la sincronización; por tanto, los bits suplementarios llegan a un 20 por ciento.

Por descontado que el porcentaje de bits suplementarios se podría reducir mediante la transmisión de bloques con más bits entre el bit de comienzo y el de parada. No obstante, como se muestra en la Figura 2.1c, cuanto mayor sea el bloque de bits, mayor será el error de temporización acumulativo. Para conseguir un mejor rendimiento en la sincronización se puede usar una estrategia diferente denominada transmisión síncrona.

Nuestro robot trabaja con una comunicación serie de transmisión de datos asíncrona.

Por su parte, en la transmisión síncrona, cada bloque de bits se transmite como una cadena estacionaria sin utilizar códigos de comienzo o parada. El bloque puede tener una longitud de muchos bits. Para prevenir la pérdida de sincronismo entre el emisor y el receptor, sus relojes se deberán sincronizar de alguna manera. Una posibilidad puede ser proporcionar la señal de reloj a través de una línea independiente.

Uno de los extremos (el receptor o el transmisor) enviará regularmente un pulso de corta duración. El otro extremo utilizará esta señal a modo de reloj. Esta técnica funciona bien a distancias cortas. Sin embargo, a distancias superiores, los pulsos de reloj pueden sufrir las mismas dificultades y defectos que las propias señales de datos, por lo que pueden aparecer errores de sincronización.

La otra alternativa consiste en incluir la información relativa a la sincronización en la propia señal de datos. En la transmisión síncrona se requiere además un nivel de sincronización adicional para que el receptor pueda determinar dónde está el comienzo y el final de cada bloque de datos. Para llevar a cabo esto, cada bloque comienza con un patrón de bits denominado preámbulo y, por lo general, también termina con un patrón de bits denominado final.

Además de los anteriores, se añaden otros bits que se utilizan en los procedimientos de control del enlace. Al conjunto de bits, o unidad de información formada por los datos más el preámbulo más los bits de final junto con la información de control se le denomina trama. El formato en particular de la trama dependerá del procedimiento de control del enlace que se utilice.

Normalmente, la trama comienza con un preámbulo de 8 bits llamado delimitador (flag). El mismo delimitador se utiliza igualmente como indicador del final de la trama. El receptor buscará la aparición del delimitador que determina el comienzo de la trama. Este delimitador estará seguido por algunos campos de control, el campo de datos (de longitud variable para la

mayoría de los protocolos), más campos de control y, por último, se repetirá el delimitador indicando el final de la trama.

Para los bloques de datos que sean suficientemente grandes, la transmisión síncrona es mucho más eficiente que la asíncrona. La transmisión asíncrona requiere un 20 por ciento, o más, de bits suplementarios. La información de control, el preámbulo y el final son normalmente menos de 100 bits.

En la Figura 2.2, vemos el formato de una trama síncrona:

Figura 2.2 Formato de una trama síncrona

2.3 Descripción del puerto serie y protocolo RS-232-C

El puerto serie RS-232-C, es la forma más común y sencilla usada para realizar transmisiones de datos entre ordenadores. El RS-232-C es un estándar que constituye la tercera revisión de la antigua norma RS-232, propuesta por la EIA (*Electronic Industries Asociation*), realizándose posteriormente una versión internacional por el CCITT, conocida como V.24.

Las diferencias entre ambas son mínimas, por lo que a veces se habla indistintamente de V.24 y de RS-232-C (incluso sin el sufijo "C"), refiriéndose siempre al mismo estándar.

Este estándar fue diseñado en los 60 para comunicar un equipo terminal de datos o **DTE** (*Data Terminal Equipment*, el PC en este caso) y un equipo de comunicación de datos o **DCE** (*Data Communication Equipment*, habitualmente un módem). Dependiendo de la velocidad de transmisión empleada, es posible tener cables de hasta 15 metros, aunque en la mayoría de los casos se recomienda una distancia menor para evitar riesgos.

Si se desea tener una comunicación bidireccional por un par de hilos y ésta consiste en una serie de bits de información, se requieren otras terminales que indiquen a la interfaz cuál de los aparatos interconectados transmite y cuál recibe, qué tipo de información es, cuándo el aparato receptor está listo para recibir, cuándo el transmisor está listo para transmitir, a qué velocidad va ser la comunicación, etc. Esto es lo que hace que el puerto serie tenga otras terminales que se usan para coordinar la comunicación entre los equipos.

Según esto, el estándar especifica 25 pines de señal, y que el conector de DTE debe ser macho y el conector de DCE hembra. Los conectores más usados son el **DB-25** macho, pero muchos de los 25 pines no son necesarios, por lo que en muchos PC modernos se utilizan los **DB-9** macho. Se pueden

localizar uno o más de estos conectores en el panel trasero del PC (al menos, deberíamos poder localizar uno de cada tipo). En el presente proyecto se ha utilizado el conector de 25 pines, DB-25. En la Figura 2.3 se pueden observar los conectores DB-9 y DB-25, tanto macho como hembra.

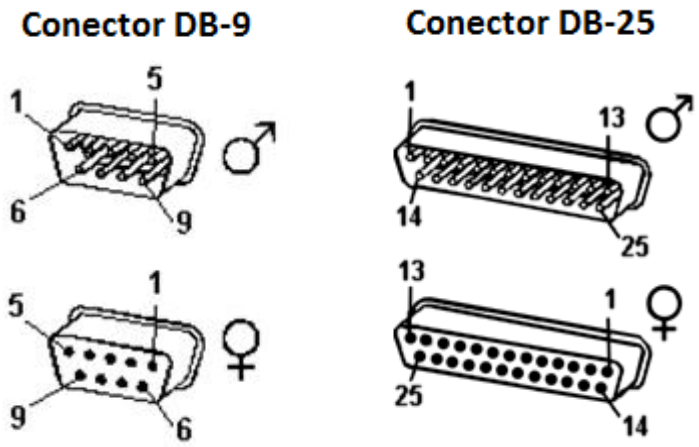


Figura 2.3 Pines de los conectores DB-9 y DB-25

Para entender con mayor profundidad el funcionamiento del puerto RS-232, es importante saber que éste trabaja entre +12 voltios y -12 voltios, de manera que un cero lógico es cuando el terminal esté entre +9 y +12 voltios, y un uno lógico cuando esté entre -9 y -12 voltios, por lo que en un puerto serie que no está transmitiendo mantiene la terminal de transmisión en un 1 lógico, es decir, entre -9 y -12 voltios.

2.4 Transferencia de datos serie

La comunicación de datos en un puerto serie se usa normalmente para efectuar comunicaciones asíncronas, es decir, sin tiempo preestablecido para iniciarse y como hemos visto, los datos llegan en ráfagas o paquetes de información; normalmente cada paquete es de 8 bits = 1 byte (el equivalente a un carácter en código ASCII). Algunos equipos envían carácter por carácter, mientras que otros guardan muchos caracteres en la memoria y cuando les toca enviarlos, los envían uno tras otro. El **número de bits** que se emplean para cada paquete, pueden ser 5, 6, 7 u 8.

Uno de los parámetros más importantes en la comunicación serie es la velocidad con la que los datos se transmiten, para el caso del RS-232, pueden transmitir de los 300 baudios (1 baudio = 1 bit/seg) hasta 115.200 baudios. La velocidad depende de los equipos conectados en el puerto serie y la calidad y longitud de los cables.

Hay dos tipos de paridad adicional que se usan y son:

- *Marca* (mark): El bit de paridad que se intercala siempre es un 1.
- *Espacio* (space): El bit de paridad que se intercala siempre es un 0.

En este momento, es necesario hacer una pequeña descripción para ayudar al lector a entender algo acerca de la paridad y los bits de arranque y parada.

La paridad fue incorporada a RS-232 porque las líneas de comunicaciones podían ser ruidosas, es decir, si se mandaba el código ASCII 0, que en hexadecimal es 0x30 (ó 00110000 en binario), a lo largo de su viaje podía encontrarse con acciones de campos magnéticos que podía hacer que alguno de sus bits cambiasen de valor.

Para evitarlo, en vez de mandar 8 bits como sería normal, se incluye un bit más a la cadena de bits enviada indicando si la suma total de bits enviados es par o impar pudiendo detectar el error si no coincide la paridad mandada, y ésa es la paridad.

Quizás el parámetro de más trascendencia es el bit de inicio, es decir, el bit que le indica al puerto receptor que va a llegar un byte de información. Los *bits de arranque* (start) y *parada* (stop) fueron añadidos al protocolo de comunicaciones serie para permitir a los receptores sincronizarse con respecto a los caracteres que están siendo enviados.

La paridad, al usar un solo bit, no permite la corrección del error, solamente la detección. La definición de la función que realiza cada señal se recoge en la normativa RS-232-C de comunicación en serie. Un resumen de la misma se muestra en la Tabla 2.1.

Cada pin puede ser de entrada o de salida, teniendo una función específica cada uno de ellos. Las señales más utilizadas se explican a continuación:

- DTR (o *Data-Terminal-Ready*): El PC indica al módem que está encendido y listo para enviar datos.
- DSR (o *Data-Set-Ready*): El módem indica al PC que está encendido y listo para transmitir o recibir datos.
- RTS (o *Request-To-Send*): El PC pone esta señal a 1 (ON) cuando tiene un carácter listo para ser enviado.
- CTS (o *Clear-To-Send*): El módem está preparado para transmitir datos. El ordenador empezará a enviar datos al módem.
- CD (o *Carrier-Detect*): El módem pone esta señal a 1 cuando ha detectado el ordenador.
- TxD: El módem recibe datos desde el PC.
- RxD: El módem transmite datos al PC.
- RI (o *Ring Indicator*): Permite al ordenador saber si el módem está realizando una llamada. Nosotros no la usaremos.

Tabla 2.1 Nomenclatura y función de cada pin en la normativa RS-232-C

Número de Pin		Señal	Descripción	E/S
DB-25	DB-9			
1	1	-	Tierra física (de protección)	-
2	3	TxD	Datos transmitidos	Salida
3	2	RxD	Datos recibidos	Entrada
4	7	RTS	Petición para enviar	Salida
5	8	CTS	Listo para enviar	Entrada
6	6	DSR	Paquete de datos preparado	Entrada
7	5	SG	Señal de referencia (Tierra lógica)	-
8	1	CD/DCD	Detector de portadora	Entrada
15	-	TxC(*)	Reloj de transmisión	Salida
17	-	RxC(*)	Reloj de recepción	Entrada
20	4	DTR	Terminal de datos preparada	Salida
22	9	RI	Indicador de llamada	Entrada
24	-	RTC(*)	Reloj de transmisión/recepción	Salida

(*)= Normalmente no conectado en el DB-25

Para controlar al puerto serie, la CPU emplea direcciones de puertos de E/S y líneas de interrupción (**IRQ**). Mediante los puertos de E/S se pueden intercambiar datos, mientras que las IRQ producen una interrupción para indicar a la CPU que ha ocurrido un evento (por ejemplo, que ha llegado un dato, o que ha cambiado el estado de algunas señales de entrada). La CPU debe responder a estas interrupciones lo más rápido posible, para que dé tiempo a recoger el dato antes de que el siguiente lo sobrescriba.

El circuito integrado que convierte los datos de paralelo a serie y viceversa se llama **UART** (*Universal Asynchronous Receiver-Transmitter*). La UART típica para un PC es el Intel 8251A. Este circuito integrado puede ser programado para realizar comunicaciones serie síncronas o asíncronas.

Ejemplo

Uno de los protocolos más utilizados suele ser el **8N1** (que significa, 8 bits de datos, sin paridad y con 1 *bit de Stop*). El receptor indica al emisor que puede enviarle datos activando la salida RTS. El emisor envía un bit de START (nivel alto) antes de los datos, y un bit de STOP (nivel bajo) al final de estos.

2.5 Comunicación serie con el robot Mitsubishi: unión robot-ordenador.

Una vez estudiadas las características y la misión de cada elemento de un puerto serie, vamos a detallar las peculiaridades de nuestra interfaz en el robot Mitsubishi.

La unidad controladora del Mitsubishi tiene disponible dos tipos de interfaces para la unión entre el MoveMaster y el ordenador personal: la interfaz paralela CENTRONICS y la serie RS-232-C.

Interfaz CENTRONICS Aunque este tema está dedicado a la comunicación serie, que es la que hemos configurado y utilizado, mencionamos aquí el hecho de que la controladora del MoveMaster admite una comunicación en paralelo con el estándar CENTRONICS, lo cual a su vez nos permite hacer una comparativa de las ventajas e inconvenientes que esta comunicación tiene frente a la serial.

El CENTRONICS es originalmente el estándar paralelo para impresoras establecido por la corporación *CENTRONICS*. La mayoría de las impresoras y los plotters XY en uso soportan este estándar. Aquí el ordenador personal envía simultáneamente (o sea, en paralelo) 8 bits, y las líneas de señal adecuadas controlan el flujo de datos. Aunque restringida a pequeñas distancias de 1 ó 2 metros, la transmisión en paralelo asegura una rápida velocidad de transmisión y no requiere configuraciones especiales

Así, teniendo en cuenta la facilidad de aplicación, el MoveMaster puede usar una interfaz como el de una impresora, lo que significa que la transferencia de datos sólo podrá ser en una dirección (desde el ordenador al robot). Además, alguno de los comandos inteligentes que usa la controladora (aquellos requeridos para la lectura de instrucciones por el robot, incluidos WH, PR y LR) no pueden ser usados. La sentencia de programación del robot usando esta interfaz se inicia con *LPRINT* en BASIC.

Interfaz RS-232-C: La interfaz RS-232-C era originalmente el estándar de comunicaciones usando las líneas telefónicas, como ya hemos visto. Sin embargo, también ha evolucionado mucho como estándar de transmisión de datos entre un ordenador y sus periféricos. Pero claro, como ahora los datos son enviados a lo largo de un único cable o canal, 1 bit al tiempo, el tiempo de transmisión es mucho más elevado, sobre todo si la velocidad media es baja. Y por supuesto, hay que asegurarse de que las configuraciones del robot y las de la computadora deben ajustarse, y no siempre puede hacerse con facilidad. Por el contrario, su capacidad de datos bidireccional permite a los ordenadores personales leer los datos internos del robot. Además se puede usar con un cable más largo – de 3 a 15 metros, y la comunicación sigue siendo posible cuando el puerto CENTRONICS está siendo ocupado por una impresora, por ejemplo.

Evidentemente, la característica bidireccional es una de las que priman a la hora de escoger un tipo de transmisión serie o paralelo. Lógicamente, nuestro proyecto se vería mermado en gran medida si el programa no recibiese continuamente datos relativos a la actividad y posición actual del robot. Las sentencias de programación en BASIC cuando se usa el RS-232-C se forman con los comandos *OPEN PRINT #* y *LINE INPUT #*.

2.6 Asignaciones de los pines del conector RS-232-C

Como ya hemos señalado, el puerto del robot Mitsubishi es un puerto serie estándar RS-232-C de 25 pines DB-25. La Tabla 2.2 nos muestra la señal que corresponde a cada uno de esos 25 pines que podíamos ver en la Figura 2.3.

Tabla 2.2 Asignación de pines en el conector RS-232-C del robot Mitsubishi.

Nº de pin	Señal	Nº de Pin	Señal
1	FG	14	No conectado
2	SD(TxD)	15	No conectado
3	RD(RxD)	16	No conectado
4	RS(RxD)	17	No conectado
5	CS(CTS	18	No conectado
6	DR (DSR)	19	No conectado
7	SG	20	ER (DTR)
8	No conectado	21	No conectado
9	No conectado	22	No conectado
10	No conectado	23	No conectado
11	No conectado	24	No conectado
12	No conectado	25	No conectado
13	No conectado		

Aunque ya hemos visto qué función se asigna a cada señal para la transmisión RS-232-C, se ha creído conveniente insistir en aquellas señales del estándar que implementa el robot Mitsubishi, y poder así particularizar dichas funciones en la transmisión de datos ordenador-controladora.

Para ello, incluimos en la Tabla 2.3 las funciones de cada señal en el conector RS-232-C del robot Mitsubishi, indicando el nombre de la señal, dirección (salida o entrada) así como su función concreta.

Tabla 2.3 Función de cada señal en el conector RS-232-C del robot Mitsubishi

Señal	Dirección	Función
FG	-	Tierra conectada al Terminal FG de la controladora
SD(TxD)	Salida	Proporciona las líneas sobre las cuales la controladora envía los datos al ordenador
RD(RxD)	Entrada	Proporciona las líneas sobre las cuales el ordenador envía los datos a la controladora
RS(RTS)	Salida	Deberá estar activada si el ordenador desea transmitir datos
CT(CTS)	Entrada	Señal usada para autorizar a la controladora a transmitir datos
DR(DSR)	Entrada	Indica que el ordenador está listo para transmitir y recibir datos
SG	-	Señal de referencia para datos y líneas de control (Tierra lógica)
ER(DTR)	Salida	Indica que la unidad controladora está lista para transmitir y recibir datos

2.7 Establecimiento de los parámetros del puerto RS-232-C.

Cuando usamos la interfaz RS-232-C para comunicar nuestro ordenador personal con el controlador del robot, debemos asegurarnos que los parámetros de comunicación del puerto (velocidad de transferencia, bit de paridad,...) deben ser los mismos. La comunicación no se logrará si existe cualquier discrepancia en este aspecto. Para establecer estos parámetros en la unidad controladora, se utilizan los interruptores localizados en el interior del panel lateral de la propia controladora, sobre todo los SW2 y SW3.

SW2: Formato de establecimiento en una transmisión asíncrona

Como ya se ha comentado, el formato de transferencia asíncrona se establece mediante el interruptor SW2 de la unidad controladora. Las posiciones de este interruptor determinan el número de bits de parada, el número de bits de datos, el bit de paridad y el factor de velocidad de transferencia. La siguiente gráfica aclara el uso de cada bit SW2 (un 0 significa posición baja y un 1 posición alta):

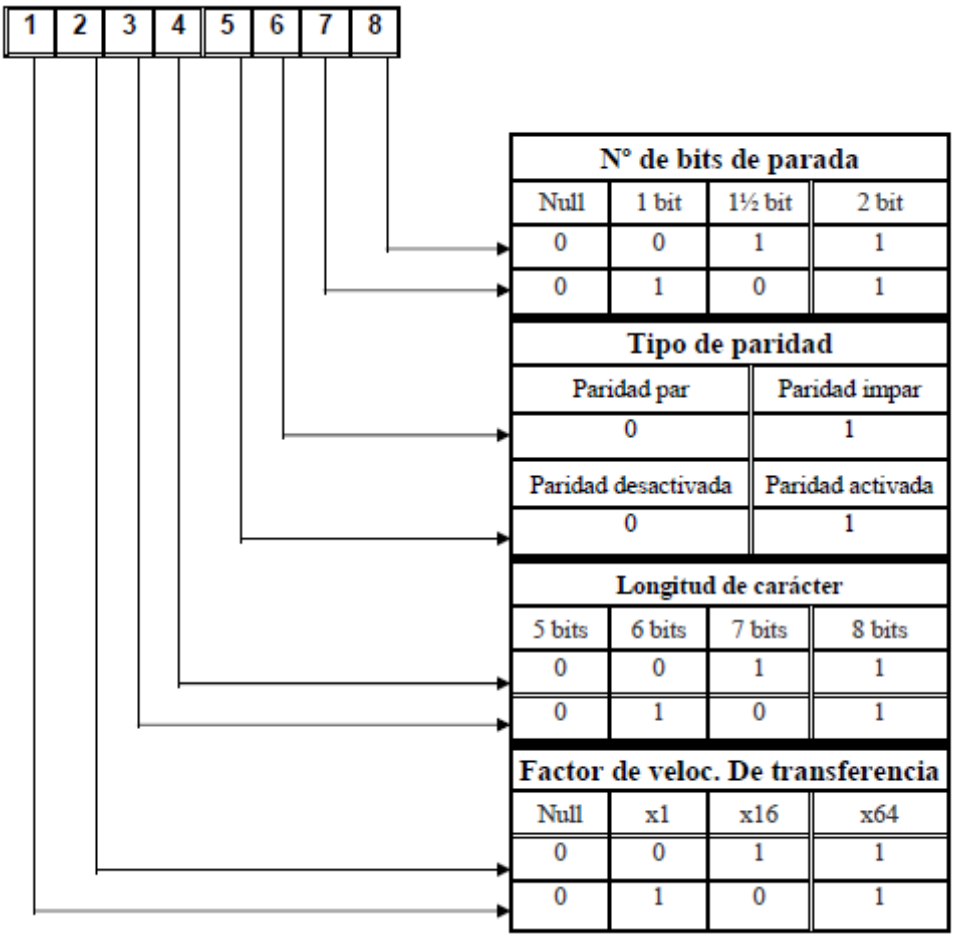


Figura 2.4 Bits del SW2: Transmisión asíncrona

Durante todo el proyecto hemos usado la configuración de la Figura 2.5

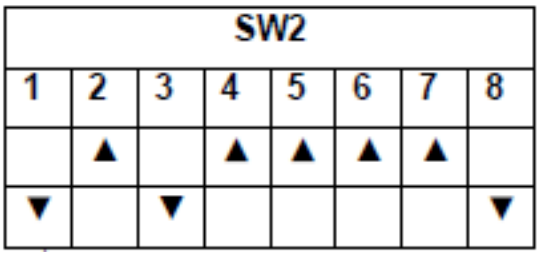


Figura 2.5 Configuración SW2 empleada

Lo que significa que trabajamos con 7 bits de datos, paridad par y 1 bit de parada.

SW3: Establecimiento de la velocidad media de transferencia

La velocidad de transferencia se puede definir mediante el interruptor SW3 de la unidad controladora. La Tabla 2.4 nos muestra la velocidad que se establece en función de cuál sea el bit que se encuentre en su posición de ON (superior). Ésta dependerá también del factor por el que se esté multiplicando (x1, x16 ó x64) cuyo valor viene dado por las posiciones de las lengüetas de los dos primeros bits del interruptor SW3, como se verá más adelante.

Tabla 2.4 Bits del SW3: Velocidad media de transferencia

SW3 bit nº	Factor de velocidad de transferencia		
	X1	X16	X64
1	1200	75	-
2	2400	150	-
3	4800	300	75
4	9600	600	150
5	-	1200	300
6	-	2400	600
7	-	4800	1200
8	-	9600	2400

Durante la realización del presente proyecto, se ha trabajado con la configuración de la Figura 2.6

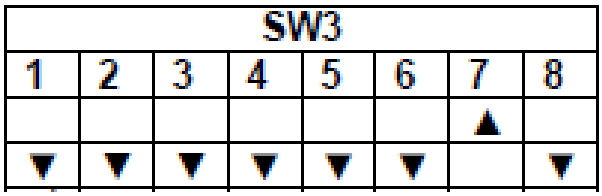


Figura 2.6 Configuración SW3 empleada

Comparando con la Tabla 2.4, deducimos que estamos trabajando a una **velocidad de 4800 baudios.**

2.8 Líneas de tiempo y funcionamiento de las señales

Como hemos visto, el flujo de información entre ordenador y robot se controla mediante la activación de señales, que indican a cada parte cuando la otra está preparada o no para recibir datos, o para enviarlos. Se detallan a continuación los pasos que dan tanto la controladora del robot como el ordenador personal para establecer esa comunicación distinguiendo dos casos: cuando la información va del ordenador al robot, que serán normalmente instrucciones de movimiento o control; y cuando el flujo se establece del robot al ordenador, que habitualmente serán datos de posición, o de error de algún tipo. Se muestran además gráficas temporales para las señales de control que clarifican la explicación del proceso (Figuras 2.7 y 2.8).

Transferencia de datos del ordenador personal al robot

¿Qué hacen las señales del **robot**?

- En el 1^{er} carácter: Las señales ER (DTR) y RS (RTS) se activan (elevan) durante 7 ms y 177 ms respectivamente. Si durante ese tiempo se introduce algún dato, las señales bajan permitiendo que dichos datos sean leídos.

- A partir del 2º carácter: ER (DTR) y RS (RTS) se elevan para aceptar datos de entrada. Si no entra ningún dato, las señales ER y RS se bajan se bajan permitiendo leer los datos que estén dentro. Esta operación se repite hasta que se envía bien un comando hexadecimal "OD" (es decir, un CR: retorno de carro) o bien un comando "OA" (FL: un avance de línea). Mientras el robot ejecuta los comandos, ER (DTR) y RS (RTS) están abajo.

¿Qué hacen las señales del **ordenador**?

- Transfiere el 1^{er} carácter mientras DR (DSR) está arriba.

- Transfiere el 2º carácter y subsiguientes cuando DR (DSR), después de haber bajado, se pone arriba de nuevo. (Si DR (DSR) está arriba, transferir una serie de caracteres causa error en el robot).

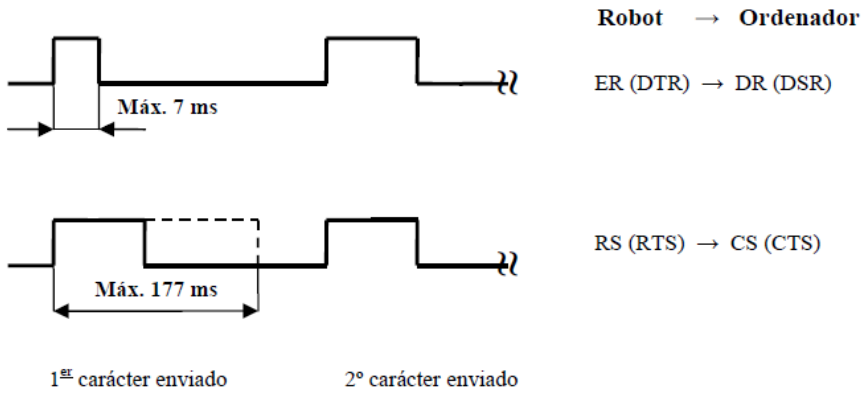


Figura 2.7 Línea temporal para la transmisión de datos del ordenador al robot

Transferencia de datos del robot al ordenador personal

¿Qué hacen las señales del **robot**?

Ahora la transferencia de datos se inicia cuando la señal ER (DTR) esté arriba. La señal ER (DTR) es bajada cuando el comando hexadecimal "OD" (retorno de carro) sea transferido.

¿Qué hacen las señales del **ordenador**?

Cuando el ordenador recibe datos, eleva la señal RS (RTS) en espera de dichos datos. Como ya vimos en el capítulo anterior, si el bit 1 del interruptor SW1 del panel lateral de la controladora está en su posición superior ON (como es nuestro caso), entonces el ordenador requiere un comando hexadecimal "OA" (LF: avance de línea) seguido del código "OD" (CR: retorno de carro) como señal de terminación para recibir datos.

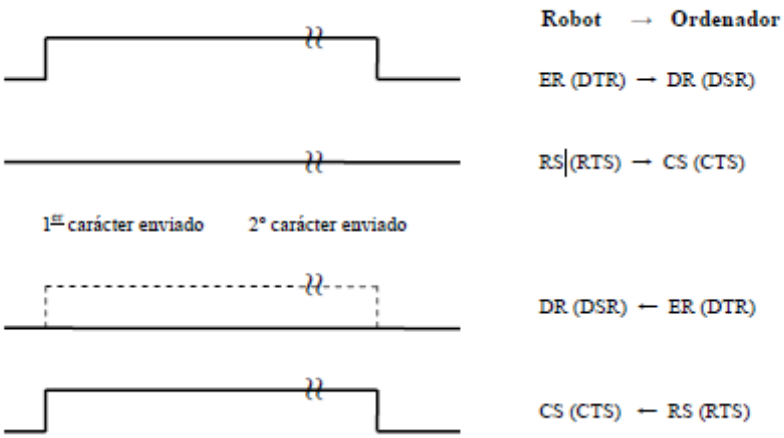


Figura 2.8 Línea temporal para la transmisión de datos del robot al ordenador

En el capítulo 3, dedicado al lenguaje de programación Python, explicaremos como hemos programado el establecimiento de la conexión bidireccional entre el ordenador y el robot para el envío de información entre el ordenador y el robot, y viceversa.

Como adelanto, diremos que se ha usado una librería de Python llamada Pyserial que permite comunicarse a través de intercambio de información por comunicación serie siguiendo el protocolo RS-232C.

Capítulo 3:

Lenguaje de programación Python

3.1 Introducción

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses "Monty Python". Es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python[1] y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional. El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C). Python también puede usarse como un lenguaje de extensiones para aplicaciones personalizables.

3.2 Principales características del lenguaje

Python es un *lenguaje de propósito general*, ya que se pueden crear todo tipo de programas. No es un lenguaje creado específicamente para la web, aunque entre sus posibilidades sí se encuentra el desarrollo de páginas.

Es *multiplataforma*, ya que hay versiones disponibles de Python en muchos sistemas informáticos distintos. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.

[1] http://www.python.org/

Es *interpretado*, lo que quiere decir que no se debe compilar el código antes de su ejecución. En realidad sí que se realiza una compilación, pero esta se realiza de manera transparente para el programador. En ciertos casos, cuando se ejecuta por primera vez un código, se producen unos bytecodes que se guardan en el sistema y que sirven para acelerar la compilación implícita que realiza el intérprete cada vez que se ejecuta el mismo código.

Python dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudarnos a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.

También es un lenguaje *orientado a objetos*. La programación orientada a objetos está soportada en Python y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables.

Dispone de *muchas funciones incorporadas* en el propio lenguaje, para el tratamiento de strings, números, archivos, etc. Además, existen muchas librerías que podemos importar en los programas para tratar temas específicos como la programación de ventanas o sistemas en red o cosas tan interesantes como crear archivos comprimidos en .zip. En concreto, en este proyecto utilizamos una librería para la comunicación serie y para la interfaz gráfica.

Por último, destacar que Python tiene una *sintaxis muy visual*, gracias a una notación identada (con márgenes) de obligado cumplimiento. En muchos lenguajes, para separar porciones de código, se utilizan elementos como las llaves o las palabras clave *begin* y *end*. Para separar las porciones de código en Python se debe tabular hacia dentro, colocando un margen al código que iría dentro de una función o un bucle. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.

Existen muchos otros lenguajes de programación, ¿por qué usar Python en este proyecto? Python presenta una serie de ventajas que lo hacen muy atractivo, tanto para su uso profesional como para el aprendizaje de la programación. Entre las más interesantes desde el punto de vista didáctico tenemos:

- Los programas Python son muy compactos: un programa Python suele ser bastante más corto que su equivalente en lenguajes como C. Python llega a ser considerado por muchos un lenguaje de programación de muy alto nivel.
- Es muy legible. La sintaxis de Python es muy elegante y permite la escritura de programas cuya lectura resulta más fácil que si utilizáramos otros lenguajes de programación.
- Ofrece un entorno interactivo que facilita la realización de pruebas y ayuda a despejar dudas acerca de ciertas características del lenguaje.

- El entorno de ejecución de Python detecta muchos de los errores de programación que escapan al control de los compiladores y proporciona información muy rica para detectarlos y corregirlos.
- Puede usarse como lenguaje imperativo procedimental o como lenguaje orientado a objetos y posee un rico juego de estructuras de datos que se pueden manipular de modo sencillo.

Algunos casos de éxito en el uso de Python son Google, Yahoo, la NASA, Industrias Light & Magic, y todas las distribuciones Linux, en las que Python cada vez representa un tanto por ciento mayor de los programas disponibles.

> *"Python ha sido parte importante de Google desde el principio, y lo sigue siendo a medida que el sistema crece y evoluciona. Hoy día, docenas de ingenieros de Google usan Python y seguimos buscando gente diestra en este lenguaje."* Peter Norvig, director de calidad de búsquedas de Google

> *"Python juega un papel clave en nuestra cadena de producción. Sin él, un proyecto de la envergadura de Star Wars: Episodio II hubiera sido muy difícil de sacar adelante. Visualización de multitudes, proceso de lotes, composición de escenas... Python es lo que lo une todo." T*ommy Brunette, director técnico senior de Industrial Light & Magic.

Como vemos, Python tiene un enorme potencial y es usado en muchas empresas líderes en sus sectores a nivel internacional. Existen diversas implementaciones del lenguaje:

- CPython es la implementación original, disponible para varias plataformas en el sitio oficial de Python.
- IronPython es la implementación para .NET
- Stackless Python es la variante de CPython que trata de no usar el stack de C .
- Jython es la implementación hecha en Java.
- Pippy es la implementación realizada para Palm.
- PyPy es una implementación de Python escrita en Python y optimizada mediante JIT.

El 13 de febrero de 2009 se lanzó una nueva versión de Python bajo el nombre clave *"Python 3000"* o, abreviado, *"Py3K"* Esta nueva versión incluye toda una serie de cambios que requieren reescribir el código de versiones anteriores. Una lista completa de los cambios puede encontrarse en la página oficial[2]

[2] https://docs.python.org/3.1/whatsnew/3.0.html

3.3 Principios básicos para empezar a programar con nuestro robot.

Con demasiada frecuencia, los programadores se involucran en un proyecto y codifican interactivamente con un API (Interfaz de programación de aplicaciones) a través del editor sin haber reflexionado suficientemente sobre el problema que intentan resolver. Para evitar confusiones y potenciales problemas, el programador debe recoger la información, cuando se trate de comunicación con dispositivos serie externos, que se indica seguidamente antes de iniciar el proyecto:

1. **Coger el manual del dispositivo y leer la sección sobre el interfaz RS-232 y su protocolo.** Muchos dispositivos tienen un protocolo que debe seguirse para poder establecer comunicación entendible con él. El dispositivo decodificará los datos que sigan ese protocolo, así que hay que poner atención en el envío y recepción de datos. Con una inicialización correcta no se asegura que la comunicación también lo sea, por ello hay que tomarse el tiempo necesario en hacer las pruebas que sea menester, incluso haciendo una aplicación de lo más simple, que escriba y lea datos a través del puerto serie usando Python. Para la comunicación con nuestro robot, se consultó el manual del MoveMaster en su apéndice de especificaciones RS-232-C.

2. **Consultar los ejemplos que proporcione el fabricante; incluso aunque se encuentren en otro lenguaje de programación,** siempre resultan útiles. Para el caso que nos ocupa se ha contado con la ayuda del proyecto anterior, desarrollado en lenguaje Java. Aunque no era del todo idéntico, al tener conocimientos de programación en C y en Python, fue más que suficiente.

3. **Buscar o codificar el ejemplo más simple posible para verificar la comunicación con el dispositivo.** En el caso de dispositivos serie, esto puede resultar altamente frustrante cuando se envía algo al dispositivo conectado al puerto serie y no sucede nada. Este es el resultado que se obtiene cuando las condiciones de la línea son incorrectas. La regla número uno de la programación con dispositivos es asegurarse que se puede establecer comunicación con el dispositivo. En el proyecto presente se ha estimado este punto como esencial para un correcto comienzo del código de programación. Por ello, la primera prueba fue la conexión y el envío de un comando (NT: mover el robot a su origen mecánico). Cuando logramos el envío correcto de parámetros al robot, el siguiente paso consistió en recibirlos.

4. Si el protocolo es muy complicado, hay que **considerar el uso de algún tipo de software que analice la línea RS-232,** el cual permite comprobar los datos que se intercambian. Por suerte no tuvimos que aplicar este punto

Para nuestro proyecto hemos utilizado la versión Python 3.4.0, ya que era la más reciente. Para crear y modificar los archivos python hemos empleado el software **Aptana Studio 3**, que es un entorno de desarrollo integrado de

software libre basado en eclipse que puede funcionar bajo Windows, Mac y Linux (en nuestro caso Windows) y provee soporte para lenguajes como: Php, Python, Ruby, CSS, Ajax, HTML y Adobe AIR. La interfaz del software Aptana Studio 3 empleado se muestra en la Figura 3.1.

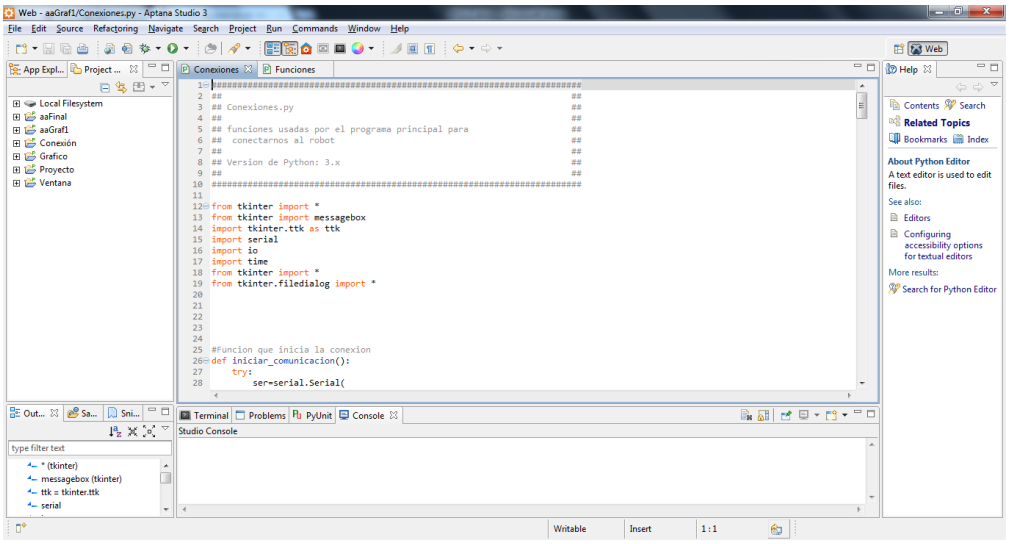


Figura 3.1 Interfaz de usuario de Aptana Studio 3

Como observamos, dispone de una interfaz sencilla, fácil de usar y de configurar, que nos ha ayudado a la hora de desarrollar el proyecto por su comodidad y por la forma de almacenar los archivos y acceder a ellos, mucho más sencilla que la consola del sistema desde la que se abre Python y que se muestra en la Figura 3.2.

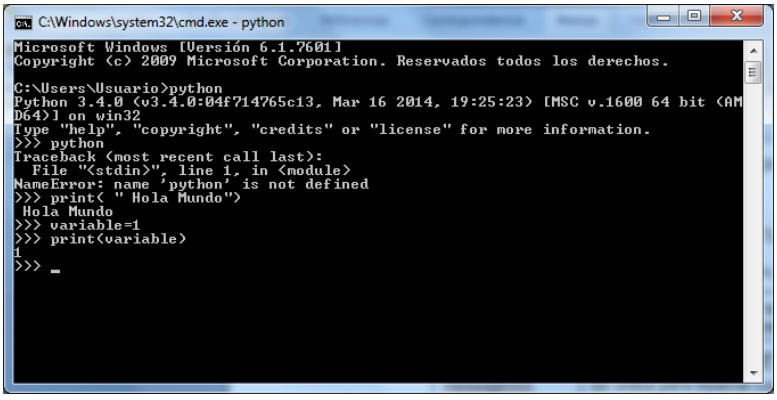


Figura 3.2 Consola de Python

Los módulos usados a la hora de programar se pueden observar en la Tabla 3.1

Tabla 3.1 Módulos usados en la programación

serial	Módulo para establecer la comunicación serie, hablaremos de él más adelante
tkinter	Módulo para la interfaz gráfica
messagebox	Se utiliza para mostrar cuadros de mensaje en sus aplicaciones, como son mensajes de error, de información o de confirmación
filedialog	Módulo proporciona las interfaces grafica de Python para abrir un archivo
io	Módulo para abrir, leer, escribir y cerrar archivos
time	Este módulo proporciona diversas funcionalidades relacionadas con el tiempo

3.4 Módulo PYSERIAL

Como ya avanzamos en el capítulo 2, para establecer la comunicación con el robot hemos usado el módulo Pyserial, exactamente la versión 2.7 compatible con la versión Python usada (3.4).

Es un módulo que se puede descargar de forma libre desde la página oficial[3] donde también se encuentran múltiples explicaciones y ejemplos de uso prácticos en los que nos hemos basado.

Establecimiento de la conexión: Para establecer la conexión serie, el comando usado es: `serial.Serial(parametros)`. Los parámetros a implementar se muestran en la tabla

Tabla 3.2 Parámetros para el establecimiento de la conexión

Port	Nombre del puerto serie (COM0, COM1…)
Baudrate	Velocidad de transmisión, en baudios
Bytesize	Número de bits de datos, los valores posibles son *FIVEBITS*,*SIXBITS*,*SEVENBITS*,*EIGHTBITS*
Parity	Habilitar o no habilitar la comprobación de paridad. Los posibles valores son: PARITY_NONE, PARITY_EVEN, PARITY_ODD,PARITY_MARK, PARITY SPACE
Stopbits	Número de bits de parada (Number of stop bits). *STOPBITS_ONE*,*STOPBITS_ONE_POINT_FIVE*, *STOPBITS_TWO*
Rtscts	Habilitar RTS/CTS para el control de información
Dsrdtr	Habilitar DSR/DTR para el control de información

[3] http://pyserial.sourceforge.net/

Escribir en el **puerto serie:** `write(datos)`, entre paréntesis se pondrán los datos o la información que queremos enviarle al robot.

Leer del puerto serie: `read(tamaño)`, entre paréntesis se pondrá el tamaño de la información a leer del puerto serie, como a priori no sabemos dicho tamaño, añadimos el siguiente comando: `inWaiting()`, que nos indica el número de datos existente en el buffer. De esta forma, combinando los comandos anteriores, se lee de forma correcta el tamaño del buffer.

Borrar datos del buffer: existe dos tipos de comandos, uno para borrar el buffer de entrada: `flushInput()` y otro para borrar el buffer de salida `flushOutput()`.

Finalmente, al obtener el valor de CTS (*Clear To Send*), con el comando getCTS nos va a permitir saber cuándo finaliza la ejecución de una orden previa para poder enviarle otra, ya que cuando está el robot ejecutando una acción el estado de CTS será False. Al finalizar dicha ejecución, será True.

Como ya vimos en el capítulo 1, la conexión con el robot deberá ser de 7 bits de datos, paridad par y 1 bit de parada. La velocidad será de 4800 baudios. El puerto donde está conectado el puerto serie es el puerto 0.

La función que hemos utilizado para iniciar la comunicación de forma correcta, establecimiento las características específicas comentadas, se muestran en el siguiente fragmento de código:

```
def iniciar_comunicacion():
    try:
        ser=serial.Serial(
                      port = 0, #port = "COM1"
                      baudrate = 4800,
                      bytesize = serial.SEVENBITS,
                      parity = serial.PARITY_EVEN,
                      stopbits = serial.STOPBITS_ONE,
                      rtscts=True,
                      dsrdtr=True,
                      )
        if ser.isOpen():
            ser.flushInput() ## vacia bufer entrada
            ser.flushOutput()##vacia bufer salida
        return ser
    except serial.SerialException:
        result = messagebox.showerror("Error al establecer conexión:",
                                      message="-Compruebe que el robot
está correctamente conectado al puerto\n"+"-Compruebe que el robot
está encendido\n"+"-Compruebe que el interruptor del teaching box
ON/OFF está en OFF\n"+"-Compruebe que  los interruptores ST1 Y ST2
están en su posición inferior ")
        return 0
        print ("error al abrir puerto serie: ")
```

En caso de que no logremos establecer conexión con el robot, se nos mostrará por pantalla un mensaje indicándonos que no se ha podido establecer conexión y las posibles causas.

Si se ha producido la conexión de forma correcta, el robot estará esperando a que le enviemos una orden, como puede ser la siguiente:

```
        ser.write(b'\rNT\r')
```

Esta orden envía al robot a su origen mecánico desde la posición en la que se encuentre. Tras dicha orden finalizaremos la comunicación con el siguiente fragmento de código:

```
def finalizar_comunicacion_simple(ser):
    while True :
        estado=ser.getCTS()
        if estado == True :
            break;
    if ser.isOpen():
        ser.close()
```

Tras enviarle la orden anterior, entramos en un bucle en la que analizamos el estado del CTS. Cuando el CTS se ponga en True, indicará que está listo para recibir otro comando, lo que viene a indicar que el robot ha llegado a la posición establecida. Tras esto, cerramos el puerto.

3.5 Interfaz gráfica: Módulo Tkinter

TkInter (de TK Interface) es un módulo que nos permite construir interfaces gráficas de usuario multiplataforma en Python utilizando el conocido toolkit Tk.

Python incluye este módulo por defecto, lo que hace que sea un toolkit muy popular. Además, es robusto, maduro y muy sencillo de aprender y de utilizar, contando con una amplia documentación, estos son los motivos por lo que nos hemos decantado por esta interfaz, frente a otras como pudieran ser wxPython, PyGTK y PyQt.

Nuestro programa principal es el siguiente:

```
if __name__ == '__main__':
    root = Tk()
    app = Mi_Aplicacion(root)
    app.mainloop()
```

Con este fragmento iniciamos el programa gráfico que veremos con más detalle en el siguiente capítulo, ya que este está dedicado a la programación.

La finalidad de emplear `if __name__ == '__main__'` es ejecutar el código solo si se invoca cuando se ejecuta el modulo directamente desde el IDE de

Aptana Studio 3, por lo que si lo importáramos no se ejecutaría tal como si sucede al no importarse.

La siguiente línea: `root = Tk()` crea la ventana principal la cual utilizaremos para enviarlo como parámetro a la clase `Mi_Aplicación` (esta clase la veremos más adelante).

Mediante `app = Mi_Aplicacion(root)`, llamamos a la clase que contiene todos los elementos para la creación de la ventana gráfica enviando como argumento la ventana principal

Finalmente, `mainloop()` es un bucle periódico de manejador de eventos que viene a significar lo siguiente:

```
while True:
    event=wait_for_event()
    event.process()
    if main_window_has_been_destroyed():
        break
```

Los eventos pueden ser clics de botones, movimientos de spinbox (veremos más adelante que es esto). Si pulsamos el botón de salir, sale del bucle finalizando la ejecución.

Widget de Tkinter.

Los "widgets", o también denominados "gadgets", proporcionados por la interfaz gráfica que hemos usado, han sido los siguientes:

- **Label**: se utiliza para mostrar texto o imagen en la pantalla.
- **Frame**: Es una región rectangular en la pantalla, un marco usado para proporcionar división en partes de la pantalla. Es similar a LabelFrame como veremos más adelante.
- **Button:** Nos permite establecer los botones en nuestro programa. Por cada botón se puede asociar una función de Python, y al pulsarse, se llama automáticamente a esa función.
- **LabelFrame:** Fusión entre Label y Frame, ya que es un recuadro con la posibilidad de insertar texto. Muy práctico para dividir la ventana.
- **Notebook:** Sirve para dividir la ventana en pestañas. Es muy útil ya que nos permite aumentar el espacio disponible.
- **Radiobutton:** Nos permite seleccionar una y solo una opción de las posibles tal y como podemos apreciar en la Figura 3.3.

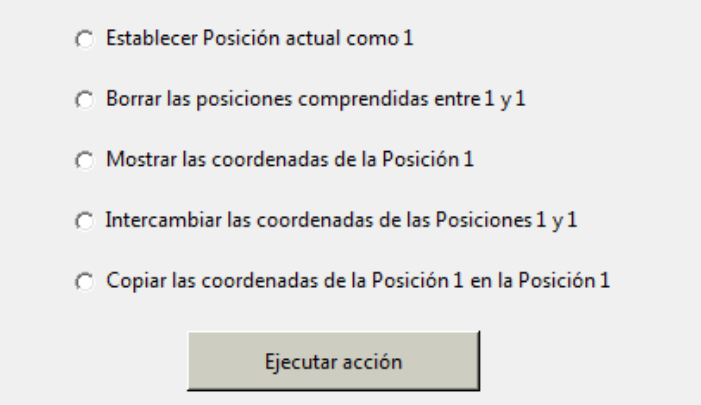


Figura 3.3 Radio-botones

- **Scale:** Barra móvil para variar el valor de una determinada variable (Figura 3.4).

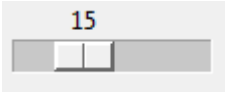


Figura 3.4 Escala

- **Spinbox:** Otra forma que tenemos para cambiar el valor de una variable, es con este widget, pero esta vez aumentamos o disminuimos dicho valor con los botones, tal y como vemos en la Figura 3.5.

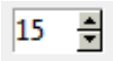


Figura 3.5 Spinbox

- **Scrollbar:** barra que nos permite desplazarnos por la ventana, en caso de que en ella haya contenido que ocupe mayor tamaño que el espacio disponible. Puede ser horizontal o vertical.

Figura 3.6 Scrollbar

A continuación vamos a mostrar el código de programación referente a la parte gráfica. Lo hemos dividido en dos clases:

La primera es la clase *Mi_Aplicacion*, y se llama en el código anterior ya comentado. Gracias a esta clase, creamos la pantalla de bienvenida del programa, de la que hablaremos en el capítulo 4, en el apartado 4.2.

```
##############################################################################
##                                                                          ##
##       VENTANA PRINCIPAL                                                  ##
##                                                                          ##
## Este programa sigue el patrón de mejores prácticas ya que                ##
## la aplicación se representa como una clase.                              ##
##                                                                          ##
## Version de Python: 3.x                                                   ##
##                                                                          ##
##############################################################################

class Mi_Aplicacion(Frame):
    '''clase que inicia la conexión con el robot: Esta clase nos permite
representar la ventana principal del programa. '''
    def __init__(self, master=None):
        #Asigno tamaño y título a la ventana principal#
        self.peq = master
        self.peq.geometry("50x50+500+350")
        self.peq.withdraw()
        ##Asigno tamaño y título a la ventana menú
        self.root = Toplevel()
        self.root.title("Robot Mitsubisi")
        self.root.geometry("650x700+200+0")
        self.root.config(bg="snow3")

        ## Invoca al constructor del master
        Frame.__init__(self, self.root)
        self.crea_widgets()

    def crea_widgets(self):
        """Crea los widgets en el frame correspondiente al objeto"""
        titulo = Label(self.root,bg="snow3",font= "Helvetica",
                       text="BIENVENIDO: \nPrograma de Control Remoto
                            del robot Mitsubishi MOveMaster EX RV-M1 ")
        titulo.pack(ipady=10)
        marco = Frame(self.root, highlightcolor="yellow", bd=5,
                      relief="groove")
        marco.pack()
        #insertamos la imagen
        self.fondo = PhotoImage(file="movemaster.gif")
        imagen = Label (marco,image=self.fondo).pack()
        titulo3 = Label(self.root,font= "Helvetica",
                        text= " Ver conexiones disponibles:",bg="snow3")
        titulo3.pack(ipady=5)
        boton1 = Button (self.root,bg="khaki",text="Ver",command =
                         Funciones.ventana)
        boton1.pack(ipadx=20,ipady=5)
        titulo2 = Label(self.root, bg="snow3", font= "Helvetica",
                        text='Elija un puerto de conexión:').pack()
        var = IntVar()
        r1 = Radiobutton(self.root, bg="snow3", text="COM0",
                         indicatoron=0,activebackground="snow3",
                         command=Conexiones.conex_com1,
                         variable=var, value=1 ).pack()
        r2 = Radiobutton(self.root,bg="snow3", text="COM1",
                         state="disabled",indicatoron=0,
                         activebackground="snow3",
```

```
                        variable=var, value=2).pack()

        r3 = Radiobutton(self.root,bg="snow3", text="COM2",
                        indicatoron=0, state="disabled",
                         activebackground="snow3",
                        variable=var, value=3).pack()

        r4 = Radiobutton(self.root,bg="snow3", text="COM3",
                        indicatoron=0,
                        state="disabled",
                        activebackground="snow3",
                        variable=var, value=4).pack()

        titulo4 = Label(self.root,font= "Helvetica",
                        text= " Salir del programa:",bg="snow3")
        titulo4.pack(ipady=5)
        boton2 = Button(self.root,text="salir",bg="tomato2",
                        command = lambda: Funciones.on_button(root))
        boton2.pack(ipadx=10, ipady=3)
        espacio = Label(self.root, bg="snow3", ).pack()
        autor = Label(self.root, bg="snow3", font= "Helvetica",
                     text='Autor: Rubén Poncelas Bodelón').pack()
```

La otra clase, es la *VentanaHija*, encargada de crear la pantalla principal donde están las ordenes de interacción con el robot. Hablaremos con más detalle en el capítulo cuarto, apartado *4.4*. Esta clase se inicia al establecer la conexión con el robot de forma correcta (en la pantalla de bienvenida).

```
###############################################################################
##                                                                           ##
##        VENTANA HIJA                                                       ##
##                                                                           ##
## Este programa sigue el patrón de mejores prácticas ya que                 ##
## la aplicación se representa como una clase.                               ##
##                                                                           ##
##    Version de Python: 3.x                                                 ##
##                                                                           ##
###############################################################################

class VentanaHija():
    def __init__(self):
#creamos una ventana hija y le aplicamos un titulo
#establecemos tamaño y que no se pueda modificar
        self.hija = Toplevel()
        self.hija.geometry('1024x725+0+0')
        self.hija.title("Software")
        self.hija.resizable(0,0)
        #Añadimos pestañas
        self.notebook = ttk.Notebook(self.hija)
        self.notebook.pack(fill="both", expand='yes')
        self.frame1=ttk.Frame(self.notebook,height=600,width=1024)
        self.frame2=ttk.Frame(self.notebook,height=600,width=1024)
        self.notebook.add(self.frame1,text='CONTROL DEL ROBOT')
        self.notebook.add(self.frame2,text='POSICIONESYPROGRAMAS')
        #Texto y Cuadros de Coordenadas de posición
```

```
        self.marco = LabelFrame(self.hija, text="Coordenadas actuales:",
                                font= "Helvetica",padx=0,pady=0)
        self.marco.place(height=95, width=1024, x=5, y=600)
        self.text1 = Label(self.hija, text= "Eje X :").place(x=100,
                                                             y=645)
        self.text2 = Label(self.hija, text= "Eje Y :").place(x=250,
                                                             y=645)
        self.text3 = Label(self.hija, text= "Eje Z :").place(x=410,
                                                             y=645)
        self.text4 = Label(self.hija, text= "Incl. Pinza:").place(x=570,
                                                                  y=645)
        self.text5 = Label(self.hija, text= "Giro Pinza :").place(x=740,
                                                                  y=645)
        self.marcot1 = Frame(self.hija, bd=1, relief="groove",
                             background="White")
        self.marcot2 = Frame(self.hija, bd=1, relief="groove",
                             background="White")
        self.marcot3 = Frame(self.hija, bd=1, relief="groove",
                             background="White")
        self.marcot4 = Frame(self.hija, bd=1, relief="groove",
                             background="White")
        self.marcot5 = Frame(self.hija, bd=1, relief="groove",
                             background="White")

        self.marcot1.place(height=25, width=70, x=150, y=645)
        self.marcot2.place(height=25, width=70, x=300, y=645)
        self.marcot3.place(height=25, width=70, x=460, y=645)
        self.marcot4.place( height=25, width=70,x=645, y=645)
        self.marcot5.place(height=25, width=70, x=815, y=645)

        #Botones Inicio,Reset, Home, Salir
        binicio =Button(self.hija,text="HOME",activebackground="Ivory4",
                        command = lambda: Conexiones.home(self.hija),
                        bg="Ivory3").place(height=90,
                                           width=90,x=920, y=100)
        bhome = Button(self.hija, text="Origen",
                       activebackground="Ivory4",
                       command = lambda:
                       Conexiones.origen_coord(self.hija),
                       bg="Ivory3").place(height=90, width=90,
                                          x=920,y=225)
        breset = Button(self.hija, text="RESET",
                        activebackground="Ivory4",
                        command = Conexiones.reset,
                        bg="Ivory3").place(height=90, width=90,
                                           x=920, y=350)
        bsalir = Button(self.hija, text="SALIR",
                        activebackground="tomato3",
                        command = lambda: Conexiones.salir(self.hija),
                        bg="tomato2").place(height=90, width=90,
                                            x=920, y=475)


    #funciones para crear las pestañas
        self.pestana2(self.frame2,self.hija)
        self.pestana1(self.frame1,self.hija)
```

```
    def pestana1(self,pestania,coordenadas):
        '''funcion que crea los elementos de la pestaña1'''
        #Creo los cuadros gráficos
        cuadro1 = LabelFrame(pestania, text = "Movimiento
                             Articulaciones",
                             font= "Helvetica").place(height=280,
                                                      width=275,
                                                      x=5 , y=10)
        cuadro2 = LabelFrame(pestania, text="Movimiento Cartesiano",
                             font= "Helvetica",).place(height=200,
                                                       width=275,
                                                       x=5, y=290)
        cuadro3=LabelFrame(pestania).place(height=75,width=275,
                                           x=5, y=500)
        cuadro4 = LabelFrame(pestania, text="Movimiento relativo",
                             font= "Helvetica").place(height=85,
                                                      width=630,
                                                      x=285, y=490)
        cuadro5 = LabelFrame(pestania, text="Movimientos de la pinza",
                             font= "Helvetica").place(height=200,
                                                      width=300,
                                                      x=615, y=290)
        cuadro6 = LabelFrame(pestania, text="Velocidad Robot",
                             font= "Helvetica").place(height=280,
                                                      width=300,
                                                      x=615, y=10)
#cuadro1  Movimiento Articulaciones: Inserto elementos en este cuadro
        bbmas = Button(pestania, text="B+", bg="Ivory3",
                       command = lambda:
                       Conexiones.b_mas(coordenadas,w1.get()),
                                        activebackground="Ivory4")
        bbmas.place(bordermode=OUTSIDE, height=40, width=40, y=40 ,x=20)
        bbmenos = Button(pestania, text="B-", activebackground="Ivory4",
                         bg="Ivory3",
                         command=lambda: Conexiones.b_menos(coordenadas,
                                                            w1.get()).
        bbmenos.place(bordermode=OUTSIDE,height=40,width=40, y=40, x=70)
        labelb= Label(pestania,
                      text="MovimientodelaCintura")
        labelb.place(bordermode=OUTSIDE, height=30,
                     width=160, y=40, x=110)
        bsmas = Button(pestania, bg="Ivory3", activebackground="Ivory4"
                       command = lambda:
                       Conexiones.s_mas(coordenadas,w1.get()),
                       text="S+",).place(bordermode=OUTSIDE, height=40,
                                         width=40, y=90,x=20)
        bsmenos = Button(pestania, text="S-", bg="Ivory3",
                         command=lambda:
Conexiones.s_menos(coordenadas,w1.get()),

activebackground="Ivory4").place(bordermode=OUTSIDE, height=40,
                                 width=40, y=90, x=70)
        labels = Label(pestania, text="Movimiento del Hombro")

        labels.place(bordermode=OUTSIDE, height=30, width=160,
                     y=90, x=110)
        bemas = Button(pestania,text="E+", bg="Ivory3",
```

```
                command = lambda:
                Conexiones.e_mas(coordenadas,w1.get()),
                activebackground="Ivory4").
 bemas.place(bordermode=OUTSIDE,height=40,
             width=40, y=140, x=20)
 bemenos=Button(pestania,text="E",bg="Ivory3",
                command=lambda:Conexión
                es.e_menos(coordenadas,w1.get()),
             activebackground="Ivory4").place(bordermode=OUTSIDE,
                                              height=40, y=140
                                              width=40, , x=70)
 labele = Label(pestania, ="Movimiento del Codo")
 labele.place(bordermode=OUTSIDE, height=30,
              width=160, y=140, x=110)
 bpmas = Button(pestania, text="P+", bg="Ivory3",
                command = lambda:
                Conexiones.p_mas(coordenadas,w1.get()),
                activebackground="Ivory4")
 bpmas.place(bordermode=OUTSIDE, height=40,
             width=40, y=190 , x=20)
 bpmenos = Button(pestania, text="P-", bg="Ivory3",
                  command=lambda: Conexiones.p_menos(coordenadas,
                                                     w1.get()),
                  activebackground="Ivory4")
bpmenos.place(bordermode=OUTSIDE, height=40,
              width=40, y=190, x=70)
 labelp = Label(pestania,
                text="Inclinación de la Pinza")
 labelp.place(bordermode=OUTSIDE, height=30,
              width=160, y=190, x=110)
 brmas = Button(pestania, text="R+", bg="Ivory3",
                command = lambda:
                Conexiones.r_mas(coordenadas,w1.get()),
                activebackground="Ivory4")
 brmas.place(bordermode=OUTSIDE, height=40,
             width=40,y=240, x=20)
 brmenos = Button(pestania,text="R-",bg="Ivory3",
                  command=lambda:
                  Conexiones.r_menos(coordenadas,w1.get()),
                  activebackground="Ivory4")
 brmenos.place(bordermode=OUTSIDE, height=40, width=40,
               y=240, x=70)

 labelr=Label(pestania,text="RotaciónPinza")
 labelr.place(height=30, bordermode=OUTSIDE, width=160,
              y=240, x=110)

 #cuadro2 Movimiento Cartesiano:Inserto elementos en este cuadro
 bxmas = Button(pestania,text="X+",bg="Ivory3",
                command = lambda:
                Conexiones.x_mas(coordenadas,w1.get()),
                activebackground="Ivory4")
bxmas.place(bordermode=OUTSIDE, height=40, width=40, y=325, x=20)
bxmenos = Button(pestania,text="X-",bg="Ivory3",
                 command=lambda:
                 Conexiones.x_menos(coordenadas,w1.get()),
                  activebackground="Ivory4")
```

```
        bxmenos.place(bordermode=OUTSIDE, height=40,
                      width=40, y=325, x=70)
        labelx= Label(pestania,text="Eje abscisas")
        labelx.place(bordermode=OUTSIDE, height=30,
                      width=160, y=325, x=110)
        bymas = Button(pestania,text="Y+",bg="Ivory3",
                       command = lambda:
                       Conexiones.y_mas(coordenadas,w1.get()),
                       activebackground="Ivory4")
        bymas.place(bordermode=OUTSIDE, height=40,
                     width=40, y=375 , x=20)
        bymenos = Button(pestania,text="Y-",bg="Ivory3",
                         command=lambda:
                         Conexiones.y_menos(coordenadas,w1.get()),
                         activebackground="Ivory4")
       bymenos.place(bordermode=OUTSIDE, height=40,
                       width=40, y=375, x=70)

labely=Label(pestania,text="Ejeordenadas").place(bordermode=OUTSIDE,
                                                 height=30, width=160,
                                                 y=375, x=110)
bzmas = Button(pestania,text="Z+",bg="Ivory3",
               command = lambda:Conexiones.z_mas(coordenadas,
                                                 w1.get()),
               activebackground="Ivory4").place(bordermode=OUTSIDE,
                                                height=40, width=40,
                                                y=425,x=20)
bzmenos = Button(pestania,text="Z-",bg="Ivory3",
                 command=lambda:
                 Conexiones.z_menos(coordenadas,w1.get()),
                 activebackground="Ivory4")
bzmenos.place(bordermode=OUTSIDE, height=40, width=40, y=425 , x=70)
labelz= Label(pestania,text="Eje de cotas")
labelz.place(bordermode=OUTSIDE, height=30,
             width=160, y=425, x=110)
#cuadro3 Avance: Inserto elementos en este cuadro
 vavance = IntVar()
 txt = Label(pestania,text="Avance:").place(x=10,y=525)
 savance = Spinbox(pestania,from_=0,to=50 , wrap = True,increment=5,
                   buttonbackground="Ivory3", textvariable=vavance)
 savance.place(bordermode=OUTSIDE,height=20,width=40,x=100,y=525)
 w1 = Scale(pestania, orient= HORIZONTAL,variable = vavance,
            from_ = 0, to = 50, activebackground="Ivory3")
        w1.set(15)
        w1.place(bordermode=OUTSIDE, x=150, y=505)

#cuadro4 Movimiento Relativo
 varrela = StringVar()
 varrela.set("0")
 varrelb = StringVar()
 varrelb.set("0")
 varrelc = StringVar()
 varrelc.set("0")
 varreld = StringVar()
 varreld.set("0")
 varrele = StringVar()
```

```
 varrele.set("0")
 bmover = Button(pestania, text="Mover",
                 activebackground="Ivory4",bg="Ivory3"
                 command=lambda: Conexiones.pos_rel(coordenadas,
                                                    ar.get(),
                                                    br.get(),
                                                    cr.get(),
                                                    dr.get(),
                                                     er.get()))

bmover.place(bordermode=OUTSIDE,height=60,width=60,x=330,y=510)
ar = Spinbox(pestania, from_=-60, to=60 , textvariable=varrela,
             wrap = True, increment=5,buttonbackground="Ivory3")
ar.place(bordermode=OUTSIDE, height=30,width=60,x=450, y=535)
textar = Label (pestania,text = "Cintura").place( x=455, y=510)

br = Spinbox(pestania, from_=-60, to=60 ,textvariable=varrelb,
             wrap = True, increment=5 , buttonbackground="Ivory3")
br.place(bordermode=OUTSIDE, height=30,width=60,x=530, y=535)
textbr = Label (pestania,text = "Hombro").place( x=545, y=510)

cr = Spinbox(pestania, from_=-60, to=60, wrap = True, increment=5 ,
             buttonbackground="Ivory3",textvariable=varrelc)
cr.place(bordermode=OUTSIDE, height=30,width=60,x=610, y=535)

textcr = Label (pestania,text = "Codo").place( x=620, y=510)
dr = Spinbox(pestania, from_=-60, to=60, wrap = True, increment=5 ,
             buttonbackground="Ivory3",textvariable=varreld)
dr.place(bordermode=OUTSIDE,height=30, width=60,x=690, y=535)
         textdr = Label (pestania,
                         text = "Inc.Pinza").place( x=690, y=510)
er = Spinbox(pestania, from_=-60, to=60, wrap = True, increment=5,
             buttonbackground="Ivory3",textvariable=varrele)
er.place(bordermode=OUTSIDE,height=30, width=60,x=770, y=535)
         texter = Label (pestania,text = "Giro Pinza").place( x=770,
y=510)

#cuadro5 Movimiento de la pinza
  tdesplaz = Label(pestania,
                   text="Velocidadddesplazamiento")
  tdesplaz.place(bordermode=OUTSIDE,y=50, x=625)
  sdesplaz = Spinbox(pestania, from_=0, to=9 , wrap = True,
                     buttonbackground="Ivory3")
  sdesplaz.place(bordermode=OUTSIDE,height=20,
                 width=40, x=800,y=50)
  ttiem = Label(pestania,text="TiempodeAceleración/Deceleración:")
  ttiem.place(bordermode=OUTSIDE, y=100, x=625)

  varbutton = IntVar()
  rbutton1 = Radiobutton(pestania, text="Rápido(H)",
                                variable=varbutton, value=1)
  rbutton1.place(y=125,x=740)
  rbutton2 = Radiobutton(pestania, text="Lento (L)",
                                variable=varbutton, value=2)
  rbutton2.place(y=150,x=740)
  bvel = Button(pestania,text="Establecer Velocidad",
                command=lambda:
```

```
                Conexiones.velocidad(sdesplaz.get(),varbutton.get()),
                activebackground="Ivory4",bg="Ivory3").
  bvel.place(bordermode=OUTSIDE,height=40,
                                    width=150,y=200, x=700)#cuadro6
Velocidad del Robot: Insertamos los elementos
  bopen = Button(pestania,text="Abrir Pinza",
                 activebackground="Ivory4",
                 command=lambda:
                 Conexiones.abrir_pinza(coordenadas,
                                        sinicial.get(),
                                        sretencion.get(),
                                        stiempo.get()),
                 bg="Ivory3").place(bordermode=OUTSIDE,
                                    height=40, width=100,
                                    y=325, x=650)

        bclose = Button(pestania,text="Cerrar pinza",
                        activebackground="Ivory4",
                        command=lambda:
                        Conexiones.cerrar_pinza(coordenadas,
                                                sinicial.get(),
                                                sretencion.get(),
                                                stiempo.get()),
                        bg="Ivory3").
        bclose.place(bordermode=OUTSIDE, height=40,
                     width=100, y=325, x=765)
        sinicial= Spinbox(pestania, from_=0, to=15 ,
                          wrap = True,buttonbackground="Ivory3")

        sinicial.place(bordermode=OUTSIDE,height=20,width=40,
                       x=825,y=375)
        tinicial=Label(pestania, text="FuerzadeAgarreInicial")
        tinicial.place(bordermode=OUTSIDE, y=375, x=650)
        sretencion= Spinbox(pestania, from_=0, to=15 ,
                            wrap = True,buttonbackground="Ivory3")
        sretencion.place(bordermode=OUTSIDE,height=20,
                         width=40,x=825, y=410)
        tretencion = Label(pestania,
                           text="Fuerza de Agarre de Retención")
        tretencion.place(bordermode=OUTSIDE, y=410, x=650)
        stiempo= Spinbox(pestania, from_=0, to=50 ,wrap = True,
                         buttonbackground="Ivory3")
        stiempo.place(bordermode=OUTSIDE,height=20,
                      width=40,x=825, y=445)
        ttiempo = Label(pestania,
                        text="Tiempo de retención ").
        Tiempo.place(bordermode=OUTSIDE, y=445, x=650)
        #Imagen
        photo = PhotoImage (file = "RobotP.GIF")
        image=Label(pestania,image=photo).place(x=280, y=20)
        mainloop()
    def pestana2(self,ventana,coordenadas):
        '''Se crean los elementos de la segunda pestaña'''
        #cuadros
        cuadro1 = LabelFrame(ventana,text = "Posiciones Empleadas",
                             font= "Helvetica").place(height=150,
                             width=450, x=5 , y=10)
```

```
  cuadro2 = LabelFrame(ventana,text="Control de posiciones",
                       font= "Helvetica").place(height=275,
                       width=450, x=5, y=160)
  cuadro3 = LabelFrame(ventana,text="Opciones correspondientes al
                       programa",font= "Helvetica")
  cuadro3.place(height=135, width=905, x=5, y=435)

  cuadro4 = LabelFrame(ventana,text="Movimientos del robot",
                       font= "Helvetica").place(height=210,
                       width=450, x=460 , y=10)
  cuadro5 = LabelFrame(ventana,text="Código archivo",
                       font= "Helvetica").place(height=215,
                       width=450, x=460, y=220 )


#cuadro 1 Posiciones Empleadas
  vars1 = IntVar()
  vars2 = IntVar()
  sposa = Spinbox(ventana, from_=1, to=29 , wrap = True,
                textvariable= vars1,buttonbackground="Ivory3",
                command=lambda:
                Funciones.Posicion(vars1.get(),vars2.get(),
                                   ventana,varpinza.get(),
                                   coordenadas))
  sposa.place(bordermode=OUTSIDE,height=30, width=80,x=200, y=50)
  tposa = Label(ventana,text="Posicion A")
  tposa.place(bordermode=OUTSIDE, y=50, x=25)
  scalea = Scale(ventana,variable=vars1,orient= HORIZONTAL,
                activebackground="Ivory3", from_ = 1 , to = 29,
                command=lambda x:
                Funciones.Posicion(vars1.get(),vars2.get(),
                                   ventana,varpinza.get(),
                                   coordenadas))
  scalea.place(bordermode=OUTSIDE, x=305, y=35)
  sposb= Spinbox(ventana, from_=1, to=29 , wrap = True ,
                buttonbackground="Ivory3", textvariable= vars2 ,
                command=lambda:
                Funciones.Posicion(vars1.get(),
                                   vars2.get(),
                                  ventana,varpinza.get(),
                                  coordenadas))
  sposb.place(bordermode=OUTSIDE,height=30,
              width=80,x=200,y=100)

  tposb = Label(ventana,text="Posicion B")
  tposb.place(bordermode=OUTSIDE, y=100, x=25)
  scaleb = Scale(ventana,variable=vars2,orient= HORIZONTAL,
                activebackground="Ivory3", from_ = 1 , to = 29,
                command=lambda x:
                Funciones.Posicion(vars1.get(),
                                   vars2.get(),
                                   ventana,varpinza.get(),
                                   coordenadas))
  scaleb.place(bordermode=OUTSIDE, x=305, y=85)

  #cuadro 2 Control de Posiciones
  varbutton = IntVar()
```

```
rbutton1 = Radiobutton(ventana, variable=varbutton,
                       value=1).place(y=200,x=50)
txtrb1 =  Label(ventana,
                text = "Establecer Posición actual como"
                +str(vars1.get()))
txtrb1.place (y=200,x=70)

rbutton2 = Radiobutton(ventana,  variable=varbutton,
                       value=2).place(y=235,x=50)
txtrb2 =  Label(ventana,
                text="Borrar las Posiciones comprendidas entre
                "+str(vars1.get()) +" y "+str(vars2.get()))
txtrb2.place(y=235,x=70)
rbutton3 = Radiobutton(ventana, variable=varbutton,
                       value=3).place(y=270,x=50)
txtrb3 =  Label(ventana,
                text="Mostrar las coordenadas de la Posición
                "+str(vars1.get()))
txtrb3.place (y=270,x=70)
rbutton4 = Radiobutton(ventana,
                       variable=varbutton,
                       value=4).place(y=305,x=50)
txtrb4 =  Label(ventana,
                text="Intercambiar las coordenadas de las
                Posiciones "+str(vars1.get())+" y
                "+str(vars2.get()))
txtrb4.place (y=305,x=70)
rbutton5 = Radiobutton(ventana, variable=varbutton,
                       value=5).place(y=340,x=50)
txtrb5 =  Label(ventana,
                text="Copiar las coordenadas de la Posición
                "+str(vars1.get())+" en la posición "
                 +str(vars2.get()))
txtrb5.place (y=340,x=70)
botonacep = Button (ventana, text= "Ejecutar
                    acción",bg="Ivory3",
                    command = lambda:
                    Conexiones.ejecutar_accion(coordenadas,
                                               varbutton.get(),
                                               scalea.get(),
                                               scaleb.get()),
                    activebackground="Ivory4").place(height=35,
                                        width=170, x=120, y=380)

#cuadro 3 Opciones correspondientes al programa
varenv = IntVar()
botonenv = Button (ventana, text="Enviar",
                   bg="Ivory3",state=DISABLED,
                   activebackground="Ivory4",
                   command=lambda : Conexiones.enviar(varenv))
botonenv.place(height=35, width=80,x=50, y=475)
marcoenv = Frame (ventana, bd=2 ,
                  relief = "groove").place(height=35, width=350,
                                           x=130, y=476)
rbenv = Radiobutton(ventana,
                    text= "Enviar puntos al robot",
                    state=DISABLED, value=1,
```

```
                        variable = varenv).place(x=140, y=480)
    rbenv2 = Radiobutton(ventana, text= "Enviar programa al robot" ,
                         state=DISABLED, value=2,
                         variable = varenv).place(x=300, y=480)
    varrec = IntVar()
    botonrec = Button (ventana, text= "Recibir",bg="Ivory3",
                       activebackground="Ivory4",
                       command=lambda:
                       Conexiones.recibir(varrec.get(),list1))
    botonrec.place(height=35, width=80, x=50, y=515)
    marcorec = Frame (ventana, bd=2 ,
                      relief = "groove").place(height=35,
                                               width=350, x=130, y=516)

     rbrec = Radiobutton(ventana, text= "Recibir puntos del robot",
                 variable = varrec,value=1).place(x=140, y=520)
    rbrec2 = Radiobutton(ventana, text= "Recibir programa del robot"
                         variable = varrec,value=2).place(x=300,
                                                          y=520)
    botonenv1 = Button (ventana, text= "Ejecutar \n código \n
                        Archivo",bg="Ivory3",
                        activebackground="Ivory4").place(height=80,
                                                         width=80,
                                                       x=660, y=475)
    botonenv2 = Button (ventana, text= "  Borrar \n memoria
                        \ncontrolador",bg="orange red",
                        activebackground="orange red3",
                        command=lambda :
                        Conexiones.borrar_mem(ventana,list1))

    botonenv2.place(height=80, width=80, x=560, y=475)
    botonenv3 = Button (ventana, text= "Borrar \n Código \n
                        Archivo",bg="Ivory3",command= lambda :
                        Funciones.borrar_listbox(list1),
                        activebackground="Ivory4").place( height=80,
                                               width=80, x=760, y=475)
#cuadro 4 -Movimiento del Robot
    varpinza = IntVar()
    botonmov = Button (ventana, text= " Mover hasta la Posición
                       "+str(vars1.get())+" ",
                      command=lambda:Conexiones.pos_mov(varpinza.get
                                                        (),vars1.get(),
                                                          coordenadas),
                      activebackground="Ivory4",bg="Ivory3").place(h
                      eight=25, width=200,x=600, y=50)

    rabuttono = Radiobutton(ventana, text= "Pinza abierta",
                            variable = varpinza, value=1,
                            indicatoron = 1,
                            command = lambda:
                            Funciones.Posicion(vars1.get(),
                                               vars2.get(),
                                               ventana,
                                               varpinza.get(),
                                               coordenadas))
    rabuttono.place(x=650, y= 100)
    rabuttonc = Radiobutton(ventana, text="Pinza Cerrada" ,
```

```
                                variable = varpinza, value=2,
                                indicatoron = 1, command = lambda:
                                Funciones.Posicion(vars1.get(),
                                                   vars2.get(),
                                                   ventana,
                                                   varpinza.get(),
                                                   coordenadas))
        rabuttonc.place(x=650, y= 130)
        botonmov = Button(ventana, text= " Incrementar Posición ",
                          command=lambda:
                          Conexiones.pos_sigui(vars1.get(),coordenadas),
                          bg="Ivory3",
                          activebackground="Ivory4").place(x=500,y=180)

  botonmov = Button (ventana, text= " Decrementar Posición ",
                           command=lambda:
                           Conexiones.pos_ant(vars1.get(),coordenadas),

bg="Ivory3",activebackground="Ivory4").place(x=700,y=180)

     #cuadro 5 Código Programa
        frame1=Frame(ventana)
        frame1.place(height=180, width=420,x = 465 , y = 240)
        frame2= Frame(ventana)
        frame2.place(height=180, width=25,x = 880 , y = 240)
        scroll1=Scrollbar(frame2)
        list1=Listbox(frame1, height=15)
        list1.pack()
        Funciones.colocar_scrollbar(list1,scroll1)
```

Capítulo 4:

Interfaz de Usuario para el control remoto del robot Mitsubishi.

4.1 Introducción

Con la interfaz gráfica tkinter de Python hemos realizado una de las partes más importantes del proyecto, la interfaz de usuario para la interacción entre el usuario y el robot a través del ordenador.

Ha sido una de las tareas que nos ha llevado más tiempo, junto con la comunicación serie con el robot, pero era de vital importancia su correcta realización ya que era indispensable que el usuario entendiera bien la interfaz para poderse manejar con facilidad, incluso sin leer el presente capítulo.

En todo momento intentamos que dicha interfaz fuera lo más sencilla, clara y que se nos mostrara el motivo, en caso de error, de la no ejecución de los comandos.

4.2 Pantalla de bienvenida

Al ejecutar el código de programación, desde Aptana Studio 3 nos va a salir la pantalla de la Figura 4.1 dándonos la bienvenida al programa de Control Remoto del Robot Mitsubishi Movemaster EX RV-M1 con una foto de dicho robot, para identificarlo en todo momento.

La pantalla de bienvenida nos indica que elijamos un puerto para realizar la conexión. Como en nuestro caso conocemos el puerto al que nos vamos a conectar, el puerto 0, los demás puertos se encuentran desactivados para evitar confusión al usuario.

Si por un casual estuvieran los posibles puertos de conexión activados, pulsando el botón “ver”, se nos abre una ventana emergente mostrándonos la lista de puertos serie disponibles, tal y como vemos en la Figura 4.2.

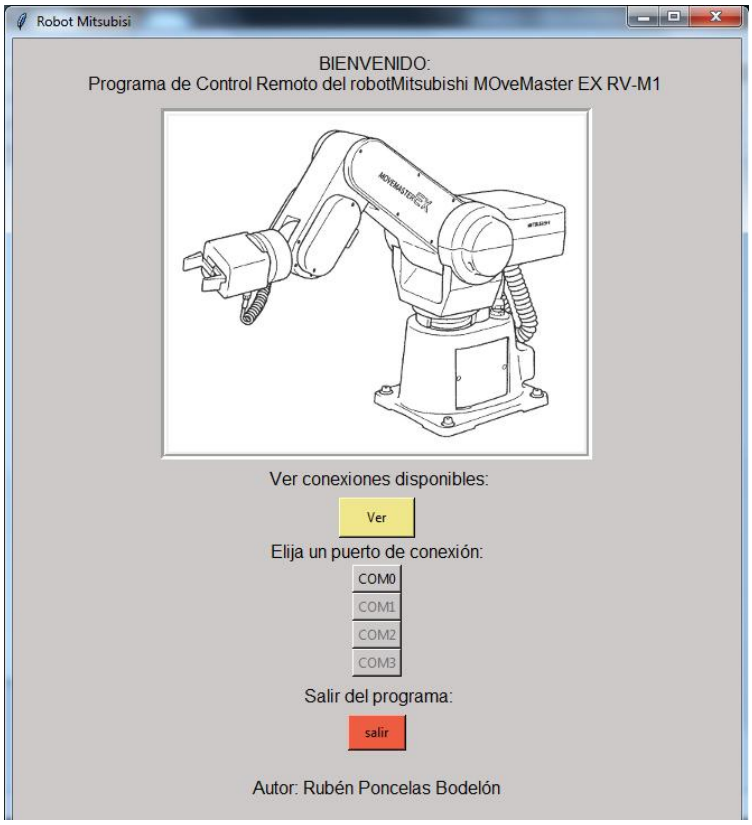


Figura 4.1 Pantalla de bienvenida

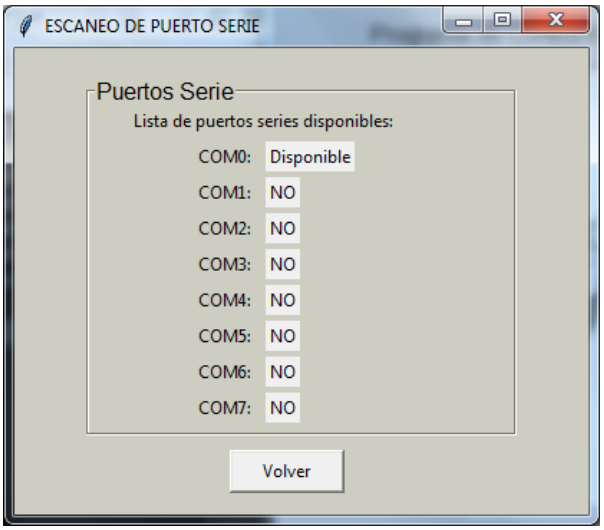


Figura 4.2 Lista de puertos disponibles

Como vemos, el COM 0 se encuentra disponible para la conexión, por lo que ya sabríamos a que puerto conectarnos. Nos ha ocurrido que tras un error, nos aparecía el COM 0 como no disponible (NO), si reiniciábamos Aptana Studio 3 volvía a estar "Disponible" de nuevo.

Si por un casual quisiéramos cerrar el programa, nos saldría un mensaje de confirmación por si le hubiéramos dado sin querer (véase Figura 4.3).

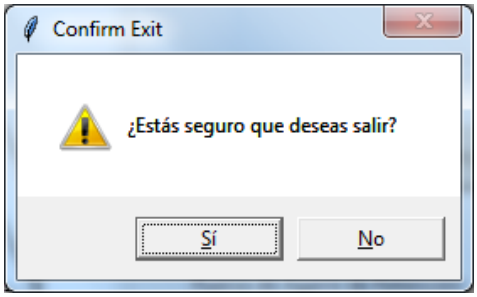


Figura 4.3 Mensaje de confirmación

El siguiente paso será ya conectarnos al robot Mitsubishi. Pueden ocurrir dos cosas:

1- **Que no se produzca la conexión de forma correcta**. Se nos notificará con el siguiente mensaje indicándonos posibles soluciones (Figura 4.4).

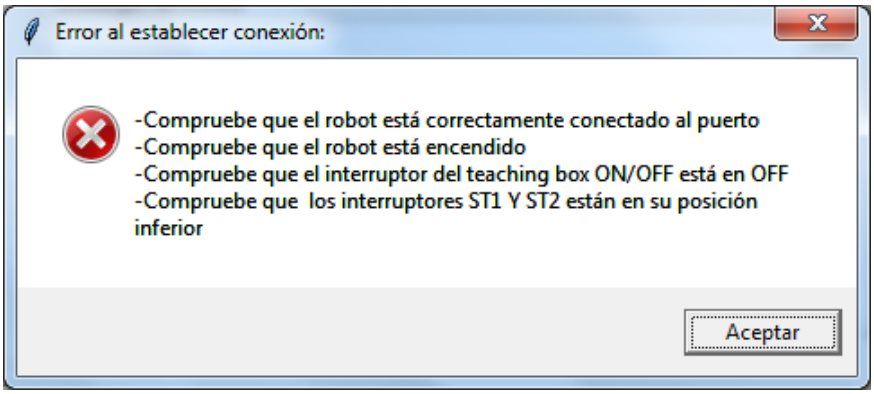


Figura 4.4 Mensaje de error

Las causas del posible error de conexión son:

a. El robot no está correctamente conectado al puerto: si esto ocurre habría que revisar la correcta conexión del cable serie y, posteriormente, comprobar las conexiones disponibles.
b. El robot no está encendido: tendríamos que ir al controlador del robot y activarlo.
c. El teaching box está en On: por lo que el teaching box es el encargado de transmitir órdenes al robot y no el ordenador, tendríamos que ponerlo en la posición OFF.
d. Los interruptores ST1 y ST2 no están en la posición inferior para que podamos controlar el robot desde el ordenador: el ST1 establece el modo de control a la computadora estando en la posición inferior mientras que el ST2 permite que no se transmitan

los datos de la EPROM a la memoria RAM de la controladora (ya que no disponemos de ella).

El mensaje de error de la Figura 4.4 también podrá aparecer en cualquier momento una vez iniciado el programa principal siempre que no logremos conectarnos al robot para enviarle un comando.

2- **Si se logra establecer la conexión correctamente.** Nos saldrá un mensaje de advertencia (véase Figura 4.5) para que nos aseguremos de que no existe peligro de colisión si el robot se desplaza a orígenes.

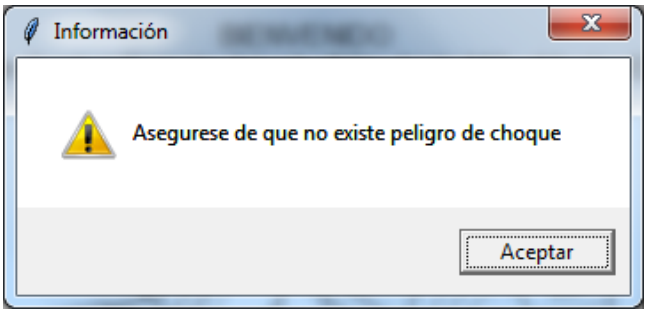


Figura 4.5 Mensaje de información

Al pulsar el botón aceptar, se iniciará el programa principal abriéndose una pantalla principal. Mientras tanto, el robot ira a la posición de origen mecánico.

4.3 Pantalla principal

Si se establece la conexión de forma correcta, se nos muestra la pantalla de control del robot de la Figura 4.6.

Como vemos en la parte superior, ésta se divide en dos pestañas: la pestaña de control del robot y la pestaña de posiciones y programas de las que hablaremos más adelante.

En la parte derecha se encuentran los botones de acceso. En primer lugar está el botón **HOME**, que lleva al robot a su origen mecánico siempre que se pulse. Debajo de éste se encuentra el botón **Origen**, que lleva al robot al origen de sistema cartesiano: en esta posición el brazo se encuentra estirado y en horizontal de forma que su coordenada Y coincide con su longitud total, la coordenada X sería cero y la Z valdría 300, que es la distancia en milímetros que el robot está elevado respecto al cero absoluto.

Durante estas operaciones, y en general durante cualquier operación de movimiento, la controladora no responderá a ninguna otra orden, pues el robot está ocupado en moverse y esta operación a veces puede durar algunos segundos.

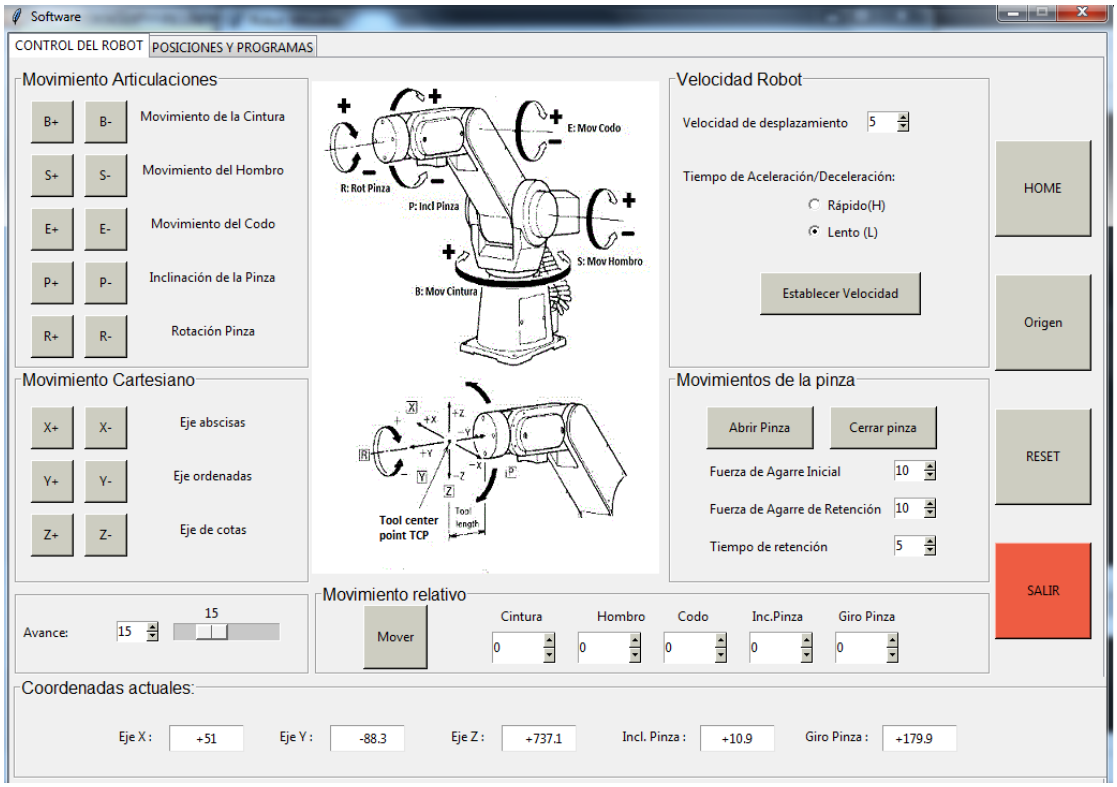


Figura 4.6 Pantalla principal del software del robot

En esta zona también encontramos el botón de **Reset** que tiene la misma función que el botón del mismo nombre situado en la parte frontal del controlador. Sirve para resetear un programa ejecutado, es decir, para volver al comienzo del mismo. Pero sobre todo, sirve para resetear un error de tipo II. En cuanto al error de tipo I, ya hemos dicho que sólo se elimina apagando la controladora y volviéndola a encender. La causa más típica que lo origina es el choque del brazo con alguno de los elementos que tiene alrededor (como la cinta transportadora, el torno y la fresa o la propia base del robot). En este caso, el programa se cuelga y se debe cerrarse la aplicación para volver a ejecutarla posteriormente.

Por último, encontramos el botón **Salir**, que abandona el programa principal volviendo a la pantalla de bienvenida.

En la parte inferior de la interfaz se muestra las coordenadas en las que se encuentra el robot en ese momento. Concretamente las coordenadas dadas en la Figura 4.7 se corresponden con la posición de origen mecánico.

Figura 4.7 Coordenadas del robot en su origen mecánico

Pestaña 1: Control del Robot

Como ya vimos en la Figura 4.6, la pantalla principal se dividía en dos pestañas. Ésta, en concreto, es la encargada de los movimientos del robot. Se divide en varias partes que pasamos a describir a continuación.

Movimiento articulaciones: Nos permite mover las diferentes articulaciones de forma individual tanto en el sentido positivo como negativo (véase Figura 4.8). Gracias a la imagen insertada en el medio de la pestaña, sabemos qué botones se corresponden con cada articulación así como su sentido.

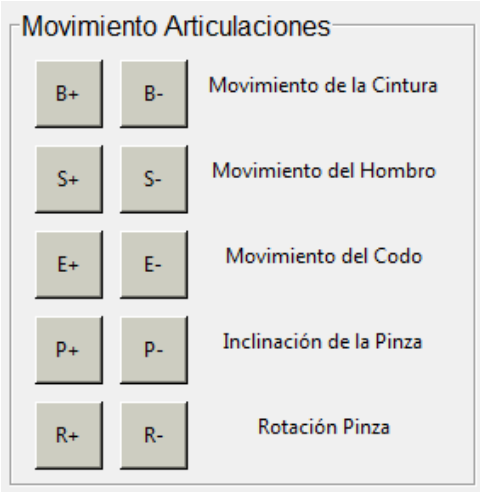


Figura 4.8 Movimiento Articulaciones

La magnitud de movimiento se controla con la barra de avance que se puede ver en la Figura 4.9. También se puede variar con el Spinbox.

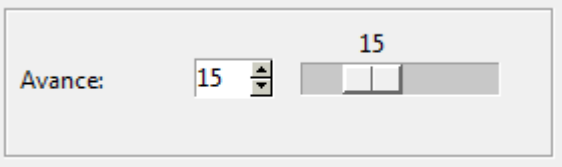


Figura 4.9 Barra de avance

Existen 5 posibles movimientos, uno por cada articulación del robot:

- Movimiento de la cintura, a través de los botones B+ y B-.
- Movimiento del hombro, controlada por los botones S+ y S-.
- Movimiento del codo, botones E+ y E-.
- Inclinación de la pinza, P+ y P-.
- Rotación pinza R+ y R-.

Movimiento Cartesiano: También existe la posibilidad de mover el robot en las direcciones cartesianas, siendo el punto de referencia la punta de la pinza del robot, como vimos en la Figura 1.15. Dicha imagen se ha adjuntado en el medio de la pestaña para ver la dirección y sentido a los que se corresponde cada botón. Los botones que hacen referencia a cada sentido se pueden observar en detalle en la Figura 4.10. Al igual que en los movimientos de las articulaciones, la variable avance controla la magnitud a mover.

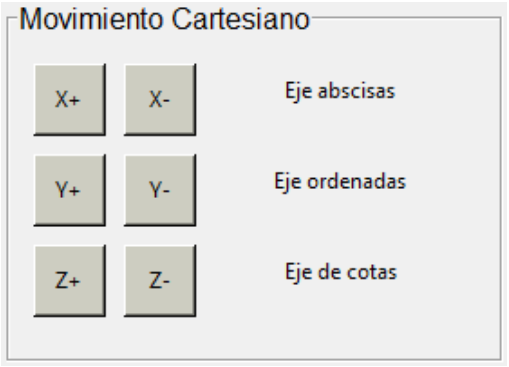


Figura 4.10 Movimiento Cartesiano

Pero moverse en el espacio XYZ no es tan intuitivo como seguir el movimiento de los grados de libertad del brazo. Esto conlleva que la utilización de este movimiento nos lleve a puntos fuera del volumen de trabajo de una forma más o menos fácil. Aunque el programa resetea el error, se recomienda que se use el movimiento de las articulaciones para llevar la pinza a un lugar determinado. Aun con todo, el usuario obviamente deberá usar la opción con la que más cómodo se encuentre.

Movimiento relativo: Existe la posibilidad de mover dos o más articulaciones a la vez, esto lo conseguimos con este comando, que disponiendo la magnitud a mover de cada articulación conseguiremos el movimiento de dichas articulaciones. La forma de mover cada articulación se puede observar en la Figura 4.11.

Figura 4.11 Movimiento relativo

Es un movimiento total, es decir, de todos los grados de libertad a la vez (en el caso que el valor de cada articulación sea distinto de cero). Por ello, se recomienda, como en el caso del movimiento en el sistema cartesiano, estudiar el movimiento que se vaya a hacer para evitar salirse continuamente del volumen de trabajo.

Velocidad Robot: Nuestro robots Mitsubishi tiene la posibilidad de moverse a varias velocidades, exactamente 10, pudiéndose establecer también dos tipos de aceleraciones, rápida o lenta. No se recomienda una rápida aceleración para movimientos cortos, ya que no da tiempo al robot a alcanzar la velocidad máxima. Para establecer la velocidad (Figura 4.12) seleccionamos el valor de dicha velocidad y el tiempo de aceleración/deceleración, y pulsamos el botón Establecer Velocidad.

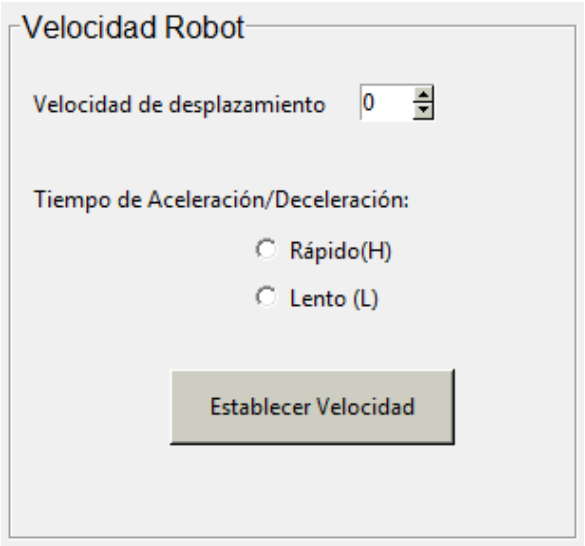


Figura 4.12 Velocidad del Robot

Movimiento de la pinza: En el extremo del robot disponemos de una pinza, que mediante esta serie de comandos podemos abrir, cerrar, configurar la fuerza de agarre inicial, de retención y el tiempo de establecimiento del agarre de retención tal y como vemos en la Figura 4.13.

La Fuerza de Agarre Inicial puede tomar un valor entre 0 y 15 y corresponde a la fuerza aplicada durante el tiempo establecido como *Tiempo de Retención de la Fuerza de Agarre Inicial*.

La Fuerza de Agarre de Retención, al igual que el anterior, toma valores comprendidos entre 0 y 15. Es la fuerza aplicada por la pinza desde que acaba el *Tiempo de Retención de la Fuerza de Agarre Inicial* hasta que acaba el movimiento de apertura o cierre. Si esta fuerza es muy pequeña, la pinza no se moverá pues no puede con las fuerzas de resistencia o lo hará de forma excesivamente lenta. Se deberá establecer una fuerza mayor o igual que 6 para que esto no suceda.

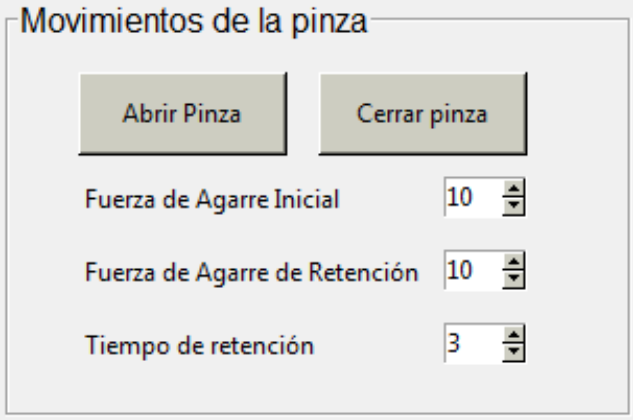


Figura 4.13 Movimientos de la pinza

El Tiempo de Retención de la Fuerza de Agarre Inicial, puede establecerse su valor en un rango de entre 0 y 99 aunque su valor real vendrá dado por la décima parte del valor seleccionado. Es decir, que su máximo real valdrá 9,9 segundos. Como su propio nombre indica, es el tiempo a partir del inicio del movimiento durante el cual se aplica la fuerza de agarre inicial.

Pestaña 2: Posiciones y programas

La segunda pestaña está centrada en el establecimiento de posiciones para interactuar a posteriori con ellas pero también en el envío y recibo de programas y puntos entre el robot y el usuario. Hemos añadido la posibilidad de ver esos puntos y programas en una ventana incluida en esta pestaña, sin necesidad de abrir los ficheros creados. La interfaz de la pestaña número dos se puede observar en la Figura 4.14

En primer lugar, en el cuadro de las **posiciones empleadas** de la Figura 4.15, existe la posibilidad de definir dos puntos o posiciones: A y B. Aunque para la mayoría de las acciones a realizar en esta pestaña sobra con el número de posición A, añadimos la posición B para realizar ciertas acciones, como "Borrar posiciones comprendidas entre A y B" o "Cambiar coordenadas de la posición B a la A",... Más adelante nos centraremos en esto.

Control de Posiciones. En este cuadro podremos seleccionar una de las cinco posibles opciones, tal y como vemos en la Figura 4.16, y a continuación:

Establecer Posición Actual como Posición A: Esta opción nos permite guardar en la controladora del robot la posición en la que se encuentra actualmente el robot.

Borrar las Posiciones Comprendidas entre A y B: Borra la posición definida en A, la definida en B y todas las intermedias.

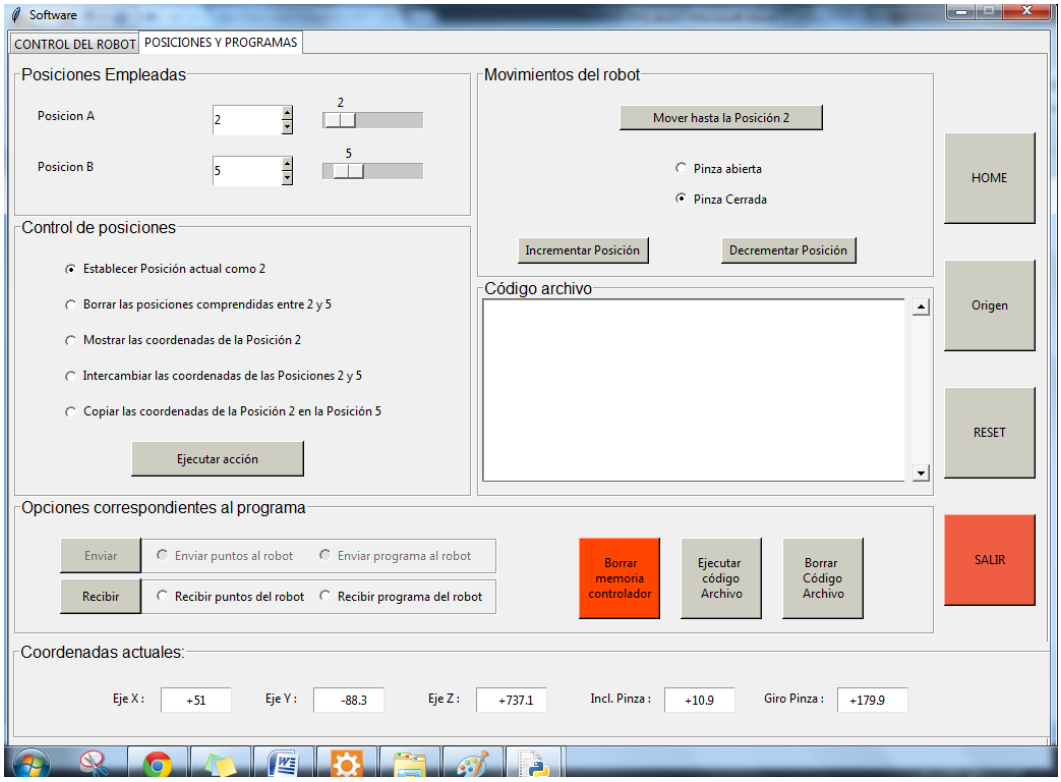


Figura 4.14 Pestaña de posiciones y programas

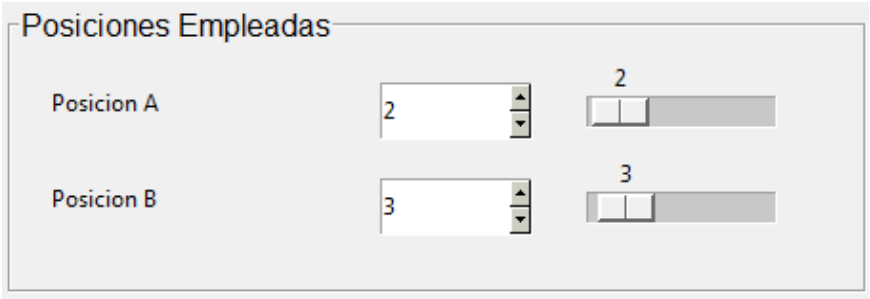


Figura 4.15 Posiciones empleadas

Mostrar las Coordenadas de la Posición A: Se lee la posición A y se muestra en los cuadros de la parte inferior. Es una opción muy útil para saber por ejemplo si una posición está definida o no (en cuyo caso se muestra un 0 en cada coordenada).

Intercambiar las Coordenadas de A y B: Cambia la posición A por la B y viceversa.

Copiar las coordenadas de la Posición A en la Posición B: La posición B toma las coordenadas de la posición A, y ésta no varía.

Para llevar a cabo cualquiera de las operaciones anteriores, tan sólo debemos seleccionar el botón de radio correspondiente y presionar el botón *Ejecutar Operación*.

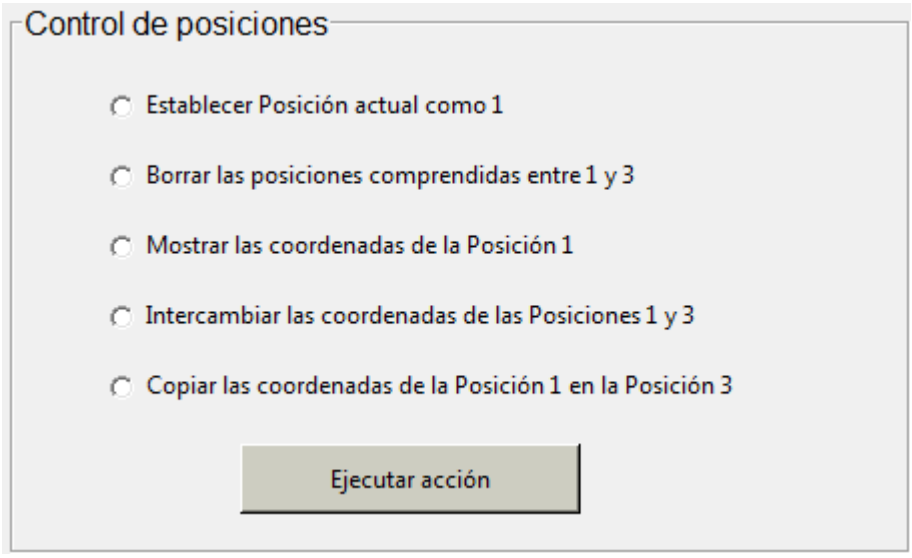


Figura 4.16 Control de posiciones

Movimiento del Robot: Nos permite desplazarnos hasta la posición definida como posición A. Además, nos da la opción de ir con la pinza abierta o cerrada (véase Figura 4.17).

Además, los botones de incrementar y decrementar la posición nos permiten desplazarnos a la posición inmediatamente superior o inmediatamente inferior a la que nos encontremos, pero para ello tendremos que haber guardado previamente alguna como ya hemos visto con anterioridad.

Si ordenamos el desplazamiento hacia una posición que no está definida, salta el error de tipo II y el pitido correspondiente.

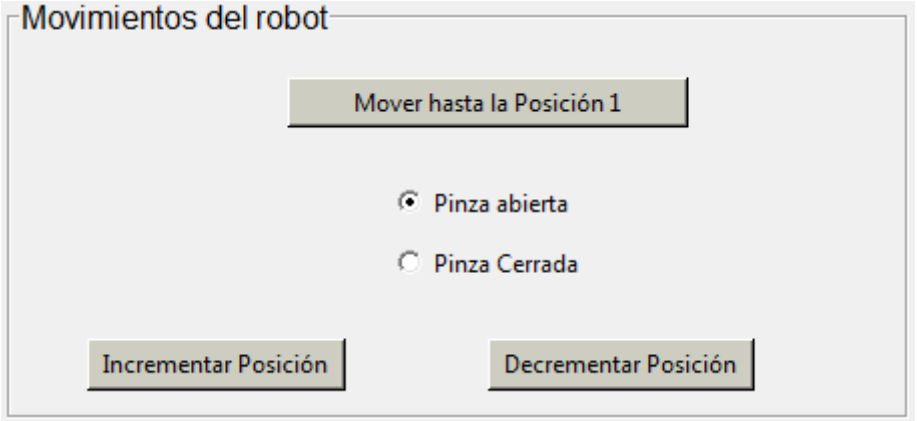


Figura 4.17 Movimientos del robot

Enviar y recibir programas del robot y puntos del robot. La controladora del robot Mitsubishi tiene una memoria interna capaz de almacenar tanto posiciones (como hemos visto, hasta un total de 629) como códigos de programas enteros (hasta 2048 líneas de programa). Esta pantalla nos va a permitir intercambiar estos datos de la controladora con ficheros de texto. El aspecto de esta pantalla se puede ver en la Figura 4.18.

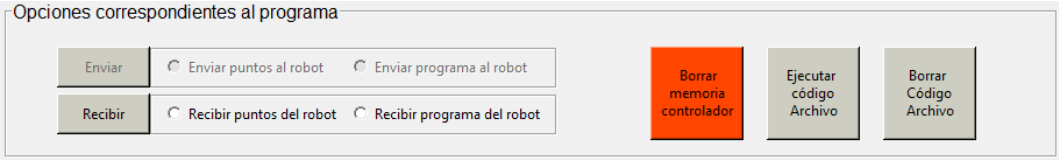


Figura 4.18 Opciones correspondientes al programa

Vamos a dividirlo en dos partes:

1- **Enviar**: Antes de poder enviar cualquier fichero a la memoria del controlador, deberemos limpiar dicha memoria. Para ello se presiona el botón de *Borrar Memoria Controlador*.
Una vez que se borre la memoria, se activará el botón de *Enviar*. El siguiente paso es seleccionar la información que queremos cargar en la controladora. Si seleccionamos ***Enviar Puntos al Robot***, y presionamos *Enviar*, aparecerá una pantalla de elección de ficheros (Figura 4.19) la cual tiene establecido un filtro gracias al cual sólo se muestran archivos de extensión **.crd**, que corresponde a la extensión de los archivos que contienen definiciones de puntos del robot (aunque también se tiene la opción de visualizar todos los archivos).

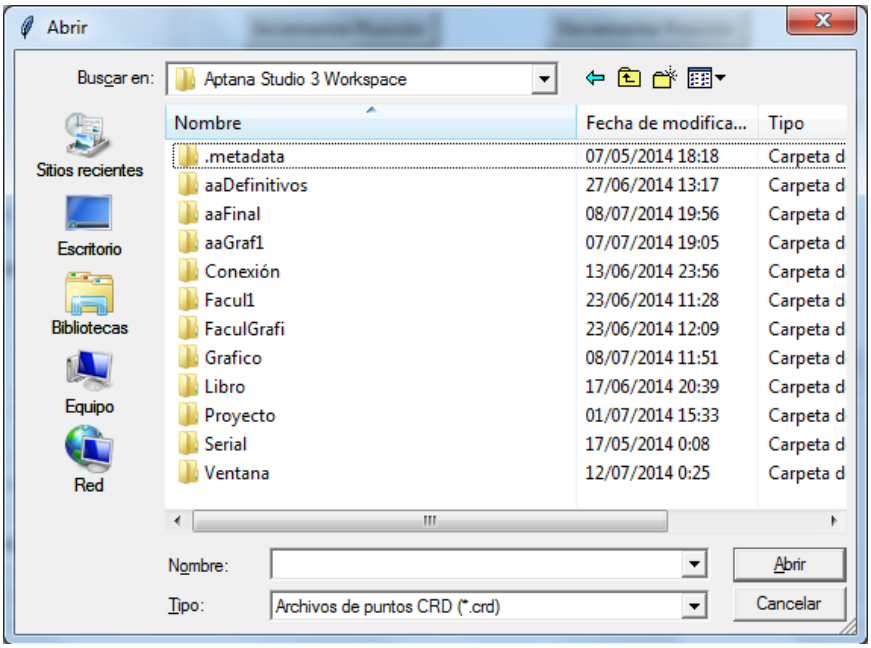


Figura 4.19 Abrir archivos de puntos CRD

Tras seleccionar y aceptar el fichero se nos mostrará por pantalla en una parte de la pestaña llamada código archivo de texto, véase Figura 4.20, que es desplazable lo que nos ayuda a visualizar lo que vamos a enviar al robot sin necesidad de abrir ningún editor de texto y nos permite realizar una comprobación. Esto también nos servirá para enviarle programas y recibir puntos y programas.

Figura 4.20 Código archivo

Si elegimos ***Enviar Programa al robot*** y apretamos *Enviar*, al igual que en el caso anterior aparecerá una pantalla de elección de archivos, pero en este caso con un filtro de extensión *.pgm, extensión correspondiente a los archivos con códigos de programas para el robot, tal y como vemos en la Figura 4.21.

No se permite la opción de ejecutar el programa, pues normalmente un programa que puede contener más de 2000 líneas de comando ocasiona algún error de tipo I ó II. Localizar dentro del código el error y gestionarlo excede los límites del presente proyecto, pero podría valer como un futuro proyecto el desarrollo, por ejemplo, de un editor de programas específico para el control del Mitsubishi.

2- **Recibir:** Si deseamos conocer las coordenadas de los puntos almacenados en el robot o recibir el programa que tiene actualmente almacenado, debemos recurrir a la opción *Recibir*. El proceso de recepción es muy similar al de envío, solo que ahora:

- No deberemos borrar la memoria del controlador, pues no recibiríamos nada.
- Internamente, al recibir datos de la controladora lo que estamos haciendo es ordenar al programa leer línea por línea el código del programa, el cual contiene 2048 líneas (o bien punto por punto, en el caso de *Recibir Puntos del Robot*, con 629 puntos), es decir, según la estructura de programación, estamos abriendo el puerto, abriendo el canal, enviando datos, cerrando en canal, cerrando el puerto, etc. hasta 2048 veces.

Esto se traduce en un tiempo de espera alto a la hora de recibir datos, que puede llegar hasta los 4 ó 5 minutos en el caso de *Recibir Programa del Robot*. Futuras mejoras podrían optimizar este tiempo de espera.

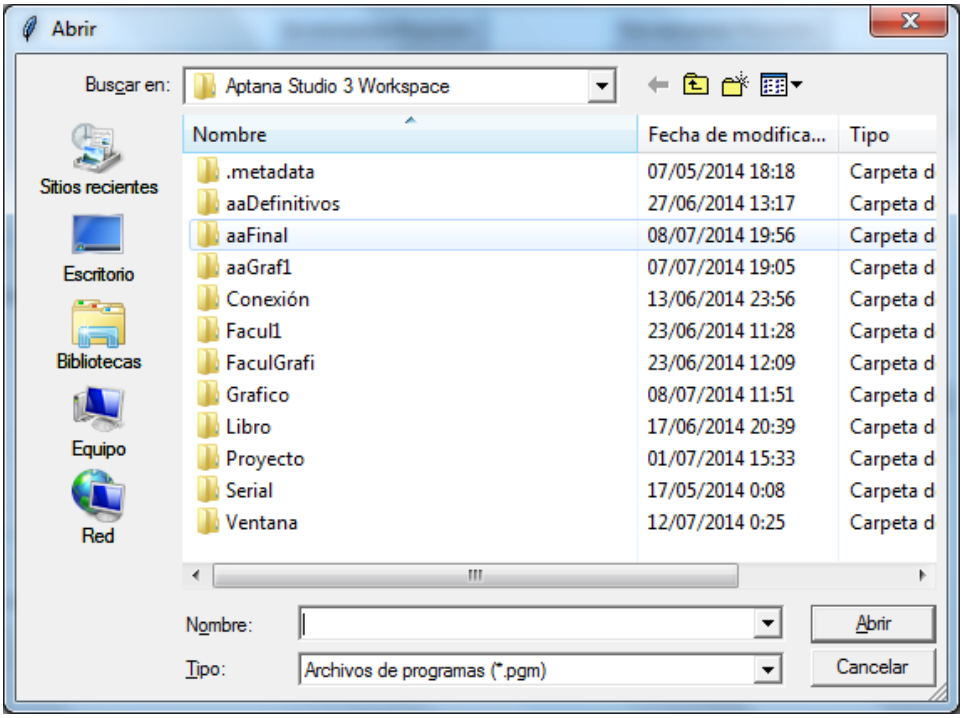


Figura 4.21 Abrir archivo de programas

Los datos recibidos se almacenan en un archivo de programas PGM si lo queremos recibir son archivos de programas, mientras que será en un archivo de puntos CRD si lo que recibimos son datos de puntos de programa. Al igual que ocurre al enviar, se nos muestra lo que recibimos en la pantalla de la pestaña (véase Figura 4.20) para que podamos ver la información recibida sin necesidad de abrir ningún archivo.

Capítulo 5:

Estudio económico del proyecto

5.1. Introducción

En este capítulo se va a explicar el estudio económico de este trabajo fin de grado, en el cual se desglosará el presupuesto necesario y además se estudiará su viabilidad económica.

A la hora de hacer el presupuesto económico de dicho trabajo tendremos que tener en cuenta las peculiares características que posee un proyecto de este tipo. A diferencia de otro tipo de proyectos, como puedan ser los electrónicos o mecánicos, aquí las mayores partidas no van a estar en los materiales consumidos o en el transporte, sino en la concepción, realización, aplicación y funcionamiento óptimo del programa a realizar.

Además, en el presupuesto inicial no se incluirá el coste de adquisición del robot, pues éste ya había sido adquirido con anterioridad por el Laboratorio de Organización e Ingeniería de Producción de la facultad, y se considera la realización de este proyecto como un trabajo subcontratado por el mismo y posterior a la compra.

5.2. Personal necesario

Las personas necesarias para la completa elaboración de este proyecto son cuatro:

- Director del proyecto
- Analista funcional
- Analista orgánico
- Programador

El organizador o **director del proyecto** es el responsable de la idea del proyecto y el encargado de llevar a cabo los contactos iniciales con los clientes. También realiza la planificación del proyecto y su presupuesto económico. Pero la misión más importante que lleva a cabo es la de dirigir y coordinar las diferentes personas que intervienen y sus funciones.

El **analista funcional** es el que detalla las propiedades del software a desarrollar. Es por tanto imprescindible que conozca el entorno de elaboración del proyecto, en este caso el campo de la robótica, así como un contacto perfecto con los clientes y sus necesidades. Es el encargado, junto con el analista orgánico, de la elección de los medios a utilizar, incluido el lenguaje de programación; para el presente proyecto, como ya sabemos el lenguaje es Python. En nuestro caso, un ingeniero industrial es el encargado al mismo tiempo de las tareas del director y del analista funcional.

El analista orgánico realiza el pseudocódigo, es decir, analiza desde el punto de vista de la programación las características del programa y su funcionamiento interno.

Por último, el programador escribe el código del programa en el lenguaje de programación elegido y basándose en lo planificado por el pseudocódigo del analista orgánico. Estas dos últimas tareas se le han sido encargadas, en nuestro caso, a un ingeniero informático especializado en lenguaje Python.

5.3 Etapas del proyecto

Como ya dijimos en la introducción, las características inherentes a un proceso de creación de software, hacen que este tipo de proyectos presenten unas peculiaridades y unas etapas de desarrollo propias. Una particularidad obvia es el bajo coste de los materiales físicos empleados. Es decir, un software no se fabrica (en sentido clásico), sino que **se desarrolla.** Las partidas más caras vienen dadas por horas de ingeniería empleadas y no, por ejemplo, por el coste de los medios empleados.

Otra diferencia estriba en la vida útil del producto final. Un software **no se degrada** (de nuevo, no en un sentido estricto) sino que, una vez detectado y corregido los errores iniciales, su vida es ilimitada. Esto en teoría. La realidad nos muestra que en los sistemas se van introduciendo correcciones y modificaciones que obligan al programa inicial a adaptarse y a corregir nuevos errores que puedan surgir, con el coste que ello conlleve. Además, la informática es un campo en continua evolución, por lo que lenguajes informáticos actualmente en uso pueden quedar relegados a segundos planos por otros lenguajes más versátiles o potentes. Por ello, es posible que nuestro programa deba en el futuro "reprogramarse" en otro lenguaje más novedoso (de cara, por ejemplo, a la gestión integral de nuestro sistema de control con otros dispositivos programados en otros lenguajes), quedando obsoleto el actual (de hecho, este proyecto "actualiza" en lenguaje Python un software anterior realizado en lenguaje Java).

Una tercera particularidad es el **mantenimiento** del producto. En un proyecto mecánico, cuando una pieza falla se sustituye por otra y se soluciona el problema. En nuestro caso, un error del programa significará probablemente

un error de diseño en código, lo cual es mucho más grave y costará más su detección y corrección.

Las fases de elaboración de un software pueden variar en función de las características particulares de cada proyecto, pero todos, incluido el nuestro, pasan por las etapas de la Figura 5.1 que posteriormente desglosaremos.

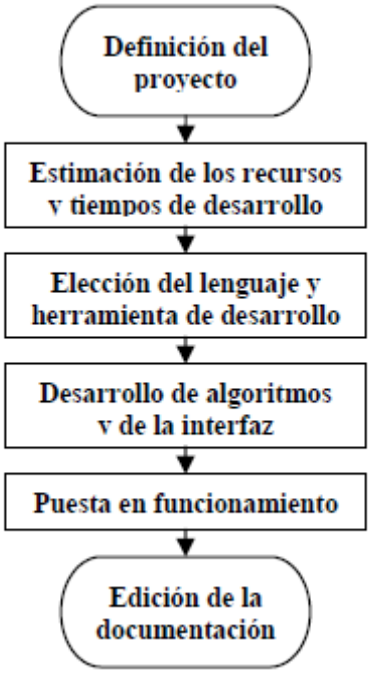


Figura 5.1 Fases de un proyecto de software

Fase 1: Definición del proyecto. En esta etapa se lleva a cabo un análisis general del problema. Se procede a la recopilación de la información necesaria referente al tema a tratar, así como a la consulta de bibliografía. También se buscan en el mercado temas relacionados. Basándonos en los datos obtenidos se formula el problema, se determina el alcance y las líneas generales del proyecto, se establece lo que hay que hacer (planificación de tareas) y quién tiene que hacerlo (asignación de recursos).

Fase 2: Estimación de los recursos y tiempos de desarrollo. Una vez abordado el problema general, se procede al análisis detallado del mismo. Se planifica su desarrollo en cuanto a tiempos, así como en cuanto a recursos necesarios. Es en ese momento cuando debe analizarse la viabilidad del proyecto. Detectar la inviabilidad del proyecto en etapas posteriores conduce a un aumento considerable de los costes.

Fase 3: Elección del lenguaje y de la herramienta de desarrollo visual. Dada la gran variedad tanto de lenguajes de programación como de herramientas de desarrollo visual existentes, es fundamental un estudio exhaustivo de todos

ellos para determinar cuál se adecua mejor a nuestros propósitos. La elección de un lenguaje frente a otros vendrá determinada por aspectos tales como la complejidad, la potencia, la flexibilidad frente a cambios o mejoras, etc. La elección de Python estándar como lenguaje a usar y sus ventajas (y desventajas) frente a otros ya ha sido expuesta en el tema correspondiente.

Fase 4: Desarrollo de algoritmos y de la interfaz de aplicación. Consiste en la elaboración de un organigrama tanto general como de cada una de las partes que pueda o quiera considerarse, especificando las entradas/salidas que pueda haber. También puede ayudarse de un lenguaje relacional o de modelado y de una herramienta visual para ayudarnos.

Fase 5: Puesta en funcionamiento. Se realizan pruebas de aplicación para detectar posibles errores y poder proceder a su corrección. En la práctica, durante la elaboración de nuestro software de control, la puesta en funcionamiento del programa – es decir, su ejecución – se realiza frecuentemente para ir comprobando que cada acción de control que se proporciona funciona correctamente. Es un trabajo "a pie de campo". No es un trabajo que se pueda hacer en un cuarto aislado con un computador. Es necesario estar en contacto continuo (o al menos, muy frecuente) con el elemento a controlar.

Fase 6: Edición de la documentación. Una vez terminado el programa, se redacta un estudio del mismo, así como un detallado manual de usuario.

5.4 Presupuesto del Trabajo Fin de Grado

En este apartado se va a realizar el presupuesto propiamente dicho del presente trabajo. En proyectos del tipo que nos concierne se usa una contabilidad por actividades, en la que se desglosan los costes de cada una de las actividades que se realizan hasta el producto final. Esto permite evaluar la influencia del coste de cada uno de los procesos que intervienen con relación al coste total del producto. La contabilidad por actividades desglosa los costes en 4 conceptos:

1. Cálculo de las horas efectivas anuales, para determinar las tasas por hora de los salarios.

2. Cálculo de las amortizaciones de los equipos.

3. Coste por hora de los materiales calificados como consumibles.

4. Coste por hora de los costes indirectos.

5. Horas de personal dedicadas a cada etapa.

Horas efectivas anuales y tasas horarias del personal

Debemos obtener las tasas por hora y por semana tanto de los salarios como de las amortizaciones de material, tanto en la fase de planificación como en la de desarrollo, para poder realizar evaluaciones presupuestarias.

Para hacernos una idea de las horas efectivas anuales vamos a ver la distribución de la jornada laboral para el año 2014, en el que nos encontramos, en el supuesto de que la jornada laboral se distribuya de lunes a viernes, véanse las Figura 5.2. y 5.3.

DÍAS HÁBILES 2014

(Empresas que distribuyan su jornada laboral de lunes a viernes)

ENERO	21	FEBRERO	20
MARZO	21	ABRIL	20
MAYO	21	JUNIO	21
JULIO	22	AGOSTO	20
SEPTIEMBRE	22	OCTUBRE	23
NOVIEMBRE	20	DICIEMBRE	21

Figura 5.2 Días hábiles

A DEDUCIR:

Vacaciones, días netos:	22
24 y 31 de Diciembre:	2
Fiestas Locales:	2

Total a deducir:	26

Figura 5.3 Días y horas laborables en el año 2014

De las Figura 5.2 y 5.3 se extraen las conclusiones recogidas en la Tabla 5.1.

Tabla 5.1 Días laborables, netos y horas al año disponibles

Días laborales totales:	252
Días netos laborables totales:	226
Horas/años	1.808

Para el cálculo de las tasas horarias de personal necesitamos saber las personas involucradas en el proyecto. En nuestro caso, se ha contado con:

- Un ingeniero industrial, que actúa como director del proyecto y analista funcional.
- Un ingeniero informático, encargado de realizar el código del programa.
- Un auxiliar administrativo, encargado de generar los informes correspondientes y los documentos de utilización del sistema.

Tal y como obtenemos del BOE Núm. 28, a fecha de sábado 1 de febrero de 2014 Sec. I. en el apartado TIPOS DE cotización. Artículo 4:

> *TIPOS DE COTIZACIÓN:*
> *- Para las contingencias comunes, el 28,30 por 100, del que el 23,60 por 100 será a cargo de la empresa y el 4,70 por 100 a cargo del trabajador.*

Los sueldos los hemos obtenido del convenio del metal de Valladolid del 2014. En la Tabla 5.2 podemos ver los cálculos realizados.

Tabla 5.2 Tasa horaria personal

Concepto	Ingeniero	Informático	Auxiliar Administrativo
Sueldo Bruto (€)	26.577,18	19.000,00	16.044,28
SS Empresa (23.60%) (€)	6.272,21	4.484,00	3.786,45
Total (€)	32.849,39	23.484,00	19.830,73
Coste Horario (€/h)	18,17	12,99	10,97

Cálculo de las amortizaciones del equipo

Para el equipo informático instalado vamos a considerar un período de amortización lineal a tres años. El equipo necesario se puede separar en dos grupos en función de su uso: el equipo destinado a realizar la elaboración del programa informático y al control del robot que denominaremos sistema de control o sistema de desarrollo; y el equipo destinado a la redacción e impresión de todos los documentos e informes relacionados con el proyecto, al cual denominaremos sistema de gestión de documentos o sistema de edición.

Cabe destacar que la mayor parte del software de uso en nuestro proyecto es de código libre y se puede descargar de forma gratuita. Además, la mayoría de los ordenadores vienen con la licencia de Windows incluida. Los cálculos se pueden ver en la Tabla 5.3.

Tabla 5.3 Amortizaciones de los equipos

Concepto	Coste (€)	Cantidad	Coste Total(€)
HP COMPAQ 100-200ES (E1-2500/4GB/500GB). Cpu + Monitor + Impresora	414,00	1	414
ASUS F552CL-SX238H (i3-3217U / 4GB / 1TB / W8). Portátil 15.6"	499,00	1	499,00
Aptana Studio 3	0,00	1	0,00
Licencia Office 365	297,00	1	297,00
Total			**796,00**
Amortización horaria a 3 años (€)			**0,147**

Costes del material consumible

El material consumible se refiere a todos aquellos activos que se consumen para la elaboración del proyecto en la utilización tanto del equipo de desarrollo como en el de edición, como el papel de impresora, los sistemas de almacenamiento informático externos, papelería, etc. Su coste se calcula como consumo medio por hora de trabajo, obteniéndose los resultados de la Tabla 5.4.

Tabla 5.4 Costes del material consumible

Concepto	Coste(€)
Papel para impresora	120,00
Toners	180,00
Cds y Pen-drives	140,00
Utiles de oficina	300,00
Total	740,00
Coste horario anual (€/h)	0,41

Costes Indirectos

Incluyen gastos comunes a todos los procesos, como son el consumo de electricidad, teléfono, calefacción, alquileres, internet, etc. Se incluyen en la Tabla 5.5.

Horas de personal dedicadas a cada etapa

Las horas efectivas reales que cada persona dedica a cada fase del proyecto es un parámetro difícil de medir y a veces sólo representativo. Mediante la realización de un estudio de tiempos y la comparación de otros proyectos con similares características a éste, se ha determinado la dedicación del personal a cada una de las etapas, las cuales se recogen en la Tabla 5.6. Conviene

recordar que estos datos no son cualitativamente exactos, pero sí son aproximativos y nos permite un posterior cálculo de asignación de costes a cada fase del proyecto, último paso de desarrollo de este presupuesto.

Tabla 5.5 Costes indirectos

Concepto	Coste(€)
Alquiler local trabajo	900,00
Electricidad	200,00
Otros	150,00
Total	1.250,00
Coste horario anual (€/h)	0,69

5.5 Costes asignados a cada fase del proyecto

Esta es una de las divisiones presupuestarias más útiles, pues permite la extrapolación de costes a otros proyectos, similares o no, pero que puedan contener etapas en su desarrollo parecidas a algunas de las que contiene el presente proyecto. Para asignar los costes de los recursos de cada fase, necesitaremos las horas dedicadas por cada persona en cada etapa, las tasas horarias de cada salario, las amortizaciones, y los costes de material consumible e indirecto. Todo ello ha sido previamente calculado.

Fase 1: Definición del proyecto

En esta etapa intervienen tanto el director del proyecto como el auxiliar administrativo. El director, a través de entrevistas personales, concreta con los responsables de la empresa cuáles son los objetivos a alcanzar. El auxiliar administrativo se encarga de las tareas de redacción de documentos y mecanografía requeridas en esta etapa.

En este momento es cuando se propone la realización de un programa informático de elaboración propia para el control del robot Mitsubishi. Como se puede uno imaginar, ésta es la fase en la que más interviene el director del proyecto, que en nuestro caso es un ingeniero industrial: recopila información en forma de bibliografía y artículos publicados que estén relacionados con el tema, define las líneas de actuación y orienta al equipo.

Este proceso llevó un total de 55 horas al director del proyecto. La elaboración de informes y documentos se llevó a cabo en 5 horas, por lo que en total la primera fase supuso 60 horas. Los costes de esta fase se reparten según se indica en la Tabla 5.6.

Tabla 5.6 Coste de la fase 1

Concepto	Horas	Coste Horario(€)	Coste Total(€)
Personal			
Ingeniero (Como director)	55	18,17	999,35
Informático			0
Aux. administrativo	5	10,97	54,85
Amortización			
Equipo de desarrollo			0
Equipo de edición	5	0,147	0,735
Material Consumible			
Varios	60	0,41	24,6
Costes indirectos	60	0,69	41,4
		COSTE TOTAL	**1.120,94**

Fase 2: Estimación de los recursos y tiempos de desarrollo

En esta etapa intervienen el ingeniero industrial, como director y como gestor de recursos, y el auxiliar administrativo. La estimación de los recursos requeridos es una labor complicada pero fundamental. Nos servirá para predecir el esfuerzo en tiempo, presupuesto y personal. Asimismo, la estimación de los tiempos de desarrollo necesarios para cada tarea va a ser igualmente imprescindible para completar el análisis. Mediante este estudio se puede valorar el riesgo de continuar con el proyecto y estimar las pérdidas en que pueden incurrirse en el caso más desfavorable. Ante dichas perspectivas ofrecidas, el director del proyecto da luz verde, o no, a desarrollo del mismo. En esta etapa el personal ha dedicado un total de 60 horas, y los costes se reparten según la Tabla 5.7.

Tabla 5.7 Costes de la fase 2

Concepto		Horas	Coste Horario(€)	Coste Total(€)
Personal				
Ingeniero	Como director	15	18,17	272,55
	Como analista	40	18,17	726,8
Informático				
Aux. administrativo		5	10,97	54,85
Amortización				
Equipo de desarrollo		40	0,47	18,8
Equipo de edición		5	0,147	0,735
Material Consumible				
Varios		60	0,41	24,6
Costes indirectos		60	0,69	41,4
			COSTE TOTAL	**1.139,74 €**

Fase 3: Elección del lenguaje y de la herramienta visual

Tal y como se ha comentado en varias ocasiones, la herramienta elegida aquí ha sido Aptana Studio 3 trabajando con Python versión 3.4. Podríamos haber utilizado cualquier otro entorno, como podría haber sido Vim, pero se eligió Aptana porque se adecuaba más a nuestras necesidades y era muy cómodo. Se han requerido 25 horas en esta fase, tal y como podemos observar en la Tabla 5.8.

Tabla 5.8 Coste de la fase 3

Concepto		Horas	Coste Horario(€)	Coste Total(€)
Personal				
Ingeniero	Como director	5	18,17	90,85
	Como analista	10	18,17	181,7
Informático		10	12,99	129,9
Aux. administrativo				
Amortización				
Equipo de desarrollo		20	0,47	9,4
Equipo de edición				
Material Consumible				
Varios		25	0,41	10,25
Costes indirectos		25	0,69	17,25
			COSTE TOTAL	439,35 €

Las tres primeras etapas vistas se deben dar siempre en el orden expuesto y no pueden solaparse; es decir, no se puede comenzar una fase sin la completa consecución de la anterior, ni tampoco la fase cuarta puede empezarse si no se han terminado las tres anteriores.

Fase 4: Desarrollo de algoritmos y de la interfaz

Esta etapa es eminentemente de aplicación. El responsable de llevarla a buen término es el informático, apoyado en las tomas de decisiones del analista, que en nuestro caso es el ingeniero industrial. La labor de este último en esta etapa se resume en los siguientes puntos:

- Determinar el organigrama del programa, tanto general como de cada parte que pueda considerarse.
- Estudiar y establecer las necesidades y requisitos de comunicación de las máquinas (robot y su deslizador).
- Establecer los parámetros de entrada y salida de las mismas.
- Diseñar las distintas pantallas y opciones que requieran los clientes u otras que él mismo considere convenientes.

Esta información la recibe el informático (de forma continua, no toda a la vez, e incluso de forma interactiva) que la escribe en lenguaje Python entendible por la herramienta. La Tabla 5.9 muestra los costes de esta fase.

Tabla 5.9 Coste de la fase 4

Concepto	Horas	Coste Horario(€)	Coste Total(€)
Personal			
Ingeniero (Como como analista)	75	18,17	1.362,75
Informático	120	12,99	1.558,8
Aux. administrativo			
Amortización			
Equipo de desarrollo	195	0,47	91,65
Equipo de edición			
Material Consumible			
Varios	195	0,41	79,95
Costes indirectos	195	0,69	134,55
		COSTE TOTAL	**3.227,7**

Fase 5: Puesta en funcionamiento

Se recuerda que aunque se ha hecho esta división en diferentes tareas con el fin de una mayor visualización de los pasos a dar y un posterior desglose en partidas de coste, esta fase no es posterior en sentido estricto a la anterior. Los costes asociados a esta fase, con un gasto horario de 50 horas, se muestran en la Tabla 5.10. Como ya se ha mencionado, en la práctica, la elaboración del código y la puesta en marcha de los distintos elementos de control generados son procesos que se alternan consecutivamente con el fin de ir depurando los errores a medida que éstos puedan ir surgiendo y no encontrarse con todos a la vez al acabar de programar.

Tabla 5.10 Coste de la fase 5

Concepto	Horas	Coste Horario(€)	Coste Total(€)
Personal			
Ingeniero (Como analista)	5	18,17	90,85
Informático	45	12,99	584,55
Aux. administrativo			
Amortización			
Equipo de desarrollo	50	0,47	23,5
Equipo de edición			
Material Consumible			
Varios	50	0,41	20,5
Costes indirectos	50	0,69	34,5
		COSTE TOTAL	**753,9 €**

Se pondrá especial atención en aquellos errores relacionados con la transferencia de datos entre el ordenador y el robot (por el puerto serie) y a

aquellos relativos al tratamiento de datos. Además, se recomienda siempre consultar al personal del taller, que son los que realmente "conviven" con el robot en su rutina laboral y han podido observar los errores más frecuentes o las necesidades diarias de las máquinas.

Fase 6: Edición de la documentación

La etapa de edición, en la que como se ha explicado se elaboran los documentos referentes al proyecto realizado (como un manual de usuario o un documento que muestre las mejoras realizadas en el sistema) es un trabajo principalmente del auxiliar administrativo, pero tendrá que estar continuamente asesorado por el informático y también por el analista. Se recomienda haber acabado la anterior fase del proyecto para comenzar la edición del mismo.

Sin embargo, es posible comenzar la documentación en algún punto anterior siempre y cuando cualquier modificación del proyecto quede reflejada en la documentación, lo cual en ocasiones puede conllevar un aumento del tiempo total empleado. Se han invertido 134 horas y los costes han sido los mostrados en la Tabla 5.11.

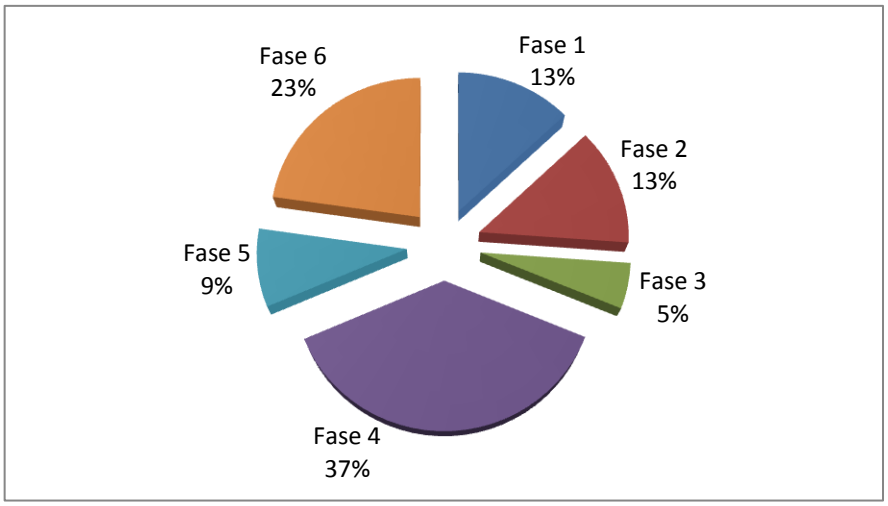
Tabla 5.11 Coste de la fase 6

Concepto	**Horas**	**Coste Horario(€)**	**Coste Total(€)**
Personal			
Ingeniero (Como analista)	21	18,17	381,57
Informático	35	12,99	454,65
Aux. administrativo	78	10,97	855,66
Amortización			
Equipo de desarrollo	56	0,47	26,32
Equipo de edición	78		
Material Consumible			
Varios	134	0,41	54,94
Costes indirectos	134	0,69	92,46
		COSTE TOTAL	**1.865,6**

El cálculo del coste total es el objetivo final del estudio económico. Se obtiene, claro está, como suma de los costes parciales de cada fase, calculados en el apartado anterior y que se recogen en la siguiente tabla (Tabla 5.12) y en un gráfico sectorial (Gráfica 5.1) que visualiza el peso de cada etapa en el coste final del proyecto

Tabla 5.12 Suma de los costes y horas de cada etapa

Fase	Horas	Coste (€)
Definición del proyecto	60	1.120,94
Estimación de los recursos y tiempos de desarrollo	60	1.139,74
Elección del lenguaje	25	439,35
Desarrollo de algoritmos e interfaz	195	3.227,7
Puesta en funcionamiento	50	753,9
Edición De la documentación	134	1.965,6
Coste Total	**524**	**8.647,23**



Gráfica 5.1 Distribución por sectores del coste de cada etapa

5.6 Estimación del precio de venta unitario

Para estimar el posible precio de venta de nuestro software se han de tener en cuenta las siguientes consideraciones:

- El software del Robot Mitsubishi se distribuye en un CD-ROM junto con su manual de usuario.
- Se calculan unas ventas de 25 copias con un coste (que comprende tanto la impresión del manual como la grabación del CD-ROM) de 49,99€ por unidad. La demanda se ha estimado en base a las empresas de la región susceptibles de adquirir un brazo robotizado tipo Mitsubishi MoveMasterEX RV-M1 o que les pueda ser útil en su cadena de producción.
- El margen de beneficio se fijará en un 13,5%.

Según esto, el coste total (incluyendo los costes de edición del paquete) se deduce de la Tabla 5.13.

Tabla 5.13 Cálculo coste total

Concepto	**Coste(€)**	**Cantidad**	**Total(€)**
Proyecto	8.647,23	1	8.647,23
Edición	49,99	25	1.249,75
		COSTE TOTAL	9.896,98**€**

Según lo reflejado en la tabla 5.13, el coste total es de 9.896,98 €.

En base a coste total calculado y a las consideraciones realizadas, el precio de venta unitario al público se estima en:

$$P.V.P = (9.896{,}98 \cdot 1.135)/25 = \boxed{449{,}32\ €}$$

Conclusiones y líneas futuras

Conclusiones

Este apartado del trabajo sirve como epílogo a toda la documentación y debe servir para evaluar si una vez acabado la elaboración del Trabajo Fin de Grado hemos alcanzado los objetivos que se proponían en la introducción. Vamos a enumerar las principales conclusiones que hemos podido extraer.

En primer lugar, se ha conseguido crear un programa en un lenguaje flexible y multiplataforma, como es el lenguaje Python, siendo este software capaz de controlar el robot Mitsubishi Movemaster modelo EX RV M1, por lo que se ha logrado el objetivo del presente Trabajo Fin de Grado.

Que sea un código independiente de la plataforma en que se ejecute hace posible que, como ya se ha comentado en esta documentación, nuestro programa pueda funcionar bajo Windows XP, Windows 7 o Windows 8. Pero también funciona con Linux, Solaris, Mac OS o cualquier otro sistema operativo que se quiera implantar (por razones de licencias, optimización del sistema o de la red local, etc.). Además, se ha realizado una comunicación mucho más simple y directa que los proyectos anteriores, gracias al lenguaje Python y en especial a su librería Pyserial.

Nuestro software nos permite el envío de programas ya realizados para su ejecución, envío de puntos, así como también es capaz de recibir puntos y programas. A parte, la interfaz de usuario realizada, es de lo más simple e intuitiva, lo que facilita la labor del operario, con múltiples opciones de acción para que este no tenga la necesidad de soltar el ratón lo que permite ganar en comodidad y rapidez.

Resulta normal, sobre todo hasta que uno se habitúa al manejo del robot, que nos salgamos del espacio de operaciones del mismo (volumen de trabajo). El programa diseñado informa y facilita la gestión de este molesto y habitual inconveniente ya que es capaz de resetear errores de software (Error tipo II) haciendo el software más completo.

Además, este programa tal y como ya se ha comentado, permite ver los puertos disponibles del ordenador con la posibilidad de selección del número de puerto de comunicación con el robot.

En lo que se refiere al código de programa, se ha intentado respetar al máximo la filosofía Python, haciéndolo de la forma más legible posible intentando comentar todos aquellos aspectos que sean importantes para comprender el funcionamiento tanto del programa, como de los elementos que lo componen. Además, se ha procedido a realizar varias revisiones con el fin de minimizar los recursos que utiliza el programa.

A parte del software, hemos añadido un capítulo para contextualizarnos con los robots industriales existentes, centrándonos en concreto en nuestro modelo. Existe otro capítulo específico dedicada a la comunicación serie, ya que es la forma de comunicación usada en este trabajo, lo que a posteriori nos ha permitido entender mejor la forma de controlar y programar nuestro robot.

Finalmente, hemos realizado un estudio económico, estimando las horas realizadas en cada fase y las personas necesarias para su elaboración y así ver la viabilidad de nuestro trabajo, objetivo indispensable en esta clase de proyectos.

Posibles mejoras

Una vez visto los objetivos alcanzados, comentaremos algunas de las posibles mejoras que pueden realizarse en el futuro con respecto a lo ya realizado. Se nos ocurren las siguientes.

- Realizar una aplicación para que nos muestre, de forma gráfica, la posición actual y movimiento del robot, lo que nos permitiría saber el estado del robot sin verlo y los límites del volumen de trabajo y la distancia a ellos.
- A veces ocurre que el cable que alimenta la pinza se queda trabado en el eje de la muñeca. En estos casos se produce un error de tipo I mucho más difícil de gestionar que los errores tipo II y sólo se soluciona apagando la controladora del robot y reiniciando el programa. Se propone para solucionar este inconveniente la creación de un programa que, establezca unos límites de operación, es decir, la creación de un área de trabajo en la que no se produzcan estos errores.
- Mejorar el programa encargado de enviar y recibir puntos y programas del robot, optimizarlo para hacerlo más eficiente.
- Añadir más funcionalidades al software de control del robot, incluyendo otra pestaña que incluya alguno de los comandos descritos en el primer capítulo.
- Por otro lado, y dado que la controladora del robot se sirve de una especie de lenguaje de programación «propio», sería posible la creación de una opción que permitiese ejecutar un programa de comandos en su totalidad, y que dicho programa pudiera editarse, cambiarse, grabarse y volverse a ejecutar de principio a fin. En caso de fallo o de incompatibilidades en la secuencia de órdenes, una pantalla de error nos informaría de la posible causa y el número de línea en el que dicho fallo se encuentra para poder corregirlo rápidamente.

Líneas futuras

A las posibles mejoras ya comentadas, podemos añadir unas líneas futuras de investigación para completar este trabajo con otros aprovechando los elementos del laboratorio.

En primer lugar, la realización del software de control de cada elemento de la célula de fabricación flexible del laboratorio, como son la fresadora, el almacén, etc., para integrarlos y elaborar ciertas rutinas de fabricación, todo ello sincronizado entre sí.

Para ello, podríamos aprovechar la comunicación serie entre los dispositivos ya que, como hemos comentado, una gran ventaja de Python es su modularidad. La parte gráfica del software podríamos basarla en la ya creada, adaptándola a las particularidades de casa elemento.

Finalmente, quedaría comunicar y coordinar los distintos componentes del laboratorio entre sí, integrarlos para poder elaborar ciertas rutinas de fabricación.

En definitiva, podemos ver que las ampliaciones y mejoras que se pueden realizar son muy numerosas y con muchas posibilidades. El programa desarrollado en este proyecto, a pesar de ser bastante flexible y permitir la introducción de alguna de ellas sin tener que modificar parte del código, necesita nuevos cambios y ampliaciones, a lo que hay que añadir la constante evolución del lenguaje de programación Python, por lo que es muy probable que en el futuro sea necesaria la creación de nuevas versiones o la actualización de la existente.

Bibliografía

Libros de consulta

Arteche, M. M. (2009). *Robótica.* Valencia: Universidad Politécnica.

Grayson, J. E. (2000). *Python and Tkinter Programming.* Manning Publications Company.

Industrial Micro-Robot Syste Model RV-M1, manual. Mitsubishi Electric Corporation.

Knowlton, J. (2008). *Python.* Madrid: Anaya.

Martelli, A. (2007). *Python : guía de referencia .* Madrid: Anaya.

Reyes Cortés, F. (2011). *Robótica : control de robots manipuladores.* Barcelona: Marcombo.

Stallings, W. (2004). *Comunicación y Redes de Computadores* (Séptima edición ed.). Madrid: PEARSON Prentice Hall.

Proyectos anteriores:

Ghezzi, I. V. (2012). *Desarrollo del agente controlador del robot Mitsubishi Movemaster RV-M1.* Valladolid.

Rodrigo, A. L. (2005). *Creación de una aplicación en java para el desarrollo de un software de control del robot industrial Mitsubishi.* Valladolid.

Documentos web:

- Duque, R. G. (2010). *Python para todos.* Disponible en: https://launchpadlibrarian.net/18980633/Python%20para%20todos.pdf
- Marzal, A.- Gracia, I. (2009). *Introducción a la programación con Python.* Disponible en: http://www.uji.es/bin/publ/edicions/ippython.pdf
- Shipma, J. W. (2013). *Tkinter 8.5 reference: a GUI for Python.* Disponible en: http://infohost.nmt.edu/tcc/help/pubs/tkinter/tkinter.pdf

Páginas web

- Comunicación serie. Disponible en: http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1. Última visita 12-5-2014

- El estándar RS-232C. Disponible en: http://www.euskalnet.net/shizuka/rs232.htm. Última visita 2-6-2014.

- Página oficial de Python. Disponible en: www.python.com. Última visita 26-4-2014.

- Pyserial, modulo para la comunicación serie en Python: Disponible en: http://pyserial.sourceforge.net/. Última visita 10-5-2014

- Robótica. Disponible en: www.monografías.com . Última visita 10-06-2014.

- Tkinter, modulo para la interfaz gráfica. Disponible en: http://es.wikibooks.org/wiki/Interfaz_gr%C3%A1fica_con_Tkinter/Gesti%C3%B3n_del_dise%C3%B1o/M%C3%A9todos. Última visita 2-6-2014