Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID
ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

CONTROL REMOTO DE UNA CÉLULA ROBOTIZADA MEDIANTE TECNOLOGÍA WI-FI

Autor:

Galindo de Santos, Álvaro

Tutor:

Herreros López, Alberto

Departamento de Ingeniería de Sistemas y Automática

Valladolid, Enero 2016.

Control remoto de una célula robotizada mediante tecnología Wi-Fi

Este proyecto ha sido redactado como Trabajo Fin de Grado por el alumno de la Escuela de Ingenierías Industriales, Sede Paseo del Cauce, de Valladolid, titulación Grado en Ingeniería en Electrónica Industrial y Automática, Álvaro Galindo de Santos.

La tutoría para la realización de este proyecto ha sido impartida por:

TUTOR/ES DEL PROYECTO

Fdo: Herreros López, Alberto

AUTOR DEL PROYECTO

Fdo: Galindo de Santos, Álvaro

AGRADECIMIENTOS

En primer lugar, quiero agradecer su ayuda y dedicación a mi tutor del Trabajo Fin de Grado, Alberto Herreros López.

En segundo lugar al Departamento de Ingeniería de Sistemas y Automática de la Universidad de Valladolid por darme la oportunidad de realizar este proyecto poniendo a mi disposición todos los medios necesarios.

Como no, quiero agradecer a mi familia su gran apoyo y esfuerzo a lo largo de mi vida como estudiante, sin los cuales no habría llegado a donde he llegado. No hay palabras suficientes para mostrar mi agradecimiento hacia ellos.

Y por último a mi novia, amigos y compañeros de la facultad, presentes en los buenos y malos momentos.

PALABRAS CLAVE:

Robótica*, Sockets, Android Studio, RobotStudio, ABB IRB 120.*

RESUMEN:

El departamento tiene una célula robotizada basada en un robot ABB IRB 120 y varios componentes relacionados con la educación. Este proyecto tiene dos objetivos, el modelado de la célula robotizada utilizando RobotStudio, y la comunicación a través de socket entre el robot y una *tablet* con sistema operativo Android.

El modelado de la célula robotizada será útil para simular el movimiento de la célula robotizada real. La comunicación entre el robot y la *tablet* será útil para mover el robot, utilizando para ello una aplicación Android.

El proyecto presenta varios problemas para poner a prueba las herramientas desarrolladas. El primer problema que se plantea es el juego de las “Tres en Raya” entre el robot real y un jugador usando el dispositivo Android. El segundo problema planteado es el desarrollo de un teclado virtual en la *tablet* para enviar señales digitales al robot. El tercer problema planteado es el desarrollo de un teclado virtual de movimiento usando la *tablet* para mover el robot IRB 120.

Los resultados obtenidos en todos estos problemas han sido muy positivos y muestran el rendimiento de las comunicaciones entre robots y *tablet*.

KEYWORDS:

Robotic, *Sockets, Android Studio, RobotStudio, ABB IRB 120.*

ABSTRACT:

The department has a robotic cell based on an ABB robot IRB120 and several education components. This project has two objectives, the modelling of the robotic cell using Robot-Studio, and the communication via socket between the robot and an Android tablet.

The modelling of the robotic cell will be useful to simulate the move of the real robotic cell. The communication between the robot and a tablet will be useful to move the robot using an Android application.

The project presents several problems to test the developed tools. The first posed problem is the “Noughts and crosses” game between the real robot and a player using the tablet. The second posed problem is the development of a virtual keypad in the tablet to send digital signals to the robot. The third posed problem is the development of a movement virtual keypad using the tablet to move the robot IRB 120.

The results obtained in all these problems have been very positive and they show the performance of the communications between robots and tablets.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE DIAGRAMAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 INTRODUCCIÓN

En primer lugar y dado que el presente proyecto tiene como objeto de estudio y desarrollo la estación de trabajo compuesto por un robot ABB IRB 120 y sus componentes externos, empezaremos definiendo el término "Robótica".

La robótica es una ciencia o rama de la tecnología, que estudia el diseño operación, disposición estructural, manufactura y aplicación de los robots. La definición de robot más comúnmente aceptada y que es proporcionada por la Asociación de Industrias Robóticas (RIA), dice que: "Un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas".

En la Robótica se combinan varias disciplinas al mismo tiempo, como son la mecánica, la electrónica, la inteligencia artificial, la informática y la ingeniería de control. Al mismo tiempo, resulta fundamental el aporte que recibe y extrae de campos tales como el álgebra, los autómatas programables y las máquinas de estados.

La implantación de un robot industrial en un determinado proceso exige un detallado estudio previo del proceso en cuestión, examinando las ventajas e inconvenientes que conlleva la introducción del robot. Será preciso siempre estar dispuesto a admitir cambios en el desarrollo del proceso primitivo (modificaciones en el diseño de piezas, sustitución de unos sistemas por otros, etc.) que faciliten y hagan viable la aplicación del robot. A la hora de elegir el robot industrial adecuado, habrá que considerar aspectos importantes como el espacio de trabajo disponible, velocidad de carga, capacidad de control, coste, etc.

Las aplicaciones más importantes y más manejadas en la actualidad por los sistemas robotizados son las siguientes:

- Trabajos en fundición
- Soldadura
- Aplicación de materiales
- Aplicación de sellantes y adhesivos
- Alimentación de máquinas
- Procesado
- Corte
- Montaje
- Paletización
- Control de calidad

Con base en los resultados preliminares de las estadísticas mundiales sobre robots industriales, la International Federation of Robotics (IFR) estima que alrededor de 229.000 fueron vendidas en 2014, un 27% más que en 2013 (ver Figura 1.1). Los principales impulsos vinieron de Asia, especialmente de China y Corea del Sur. En total, alrededor de 140.000 unidades fueron vendidas en toda esta región, con mucho, el volumen más alto jamás registrado. Las ventas de robots en América y Europa también alcanzaron nuevos niveles máximos.

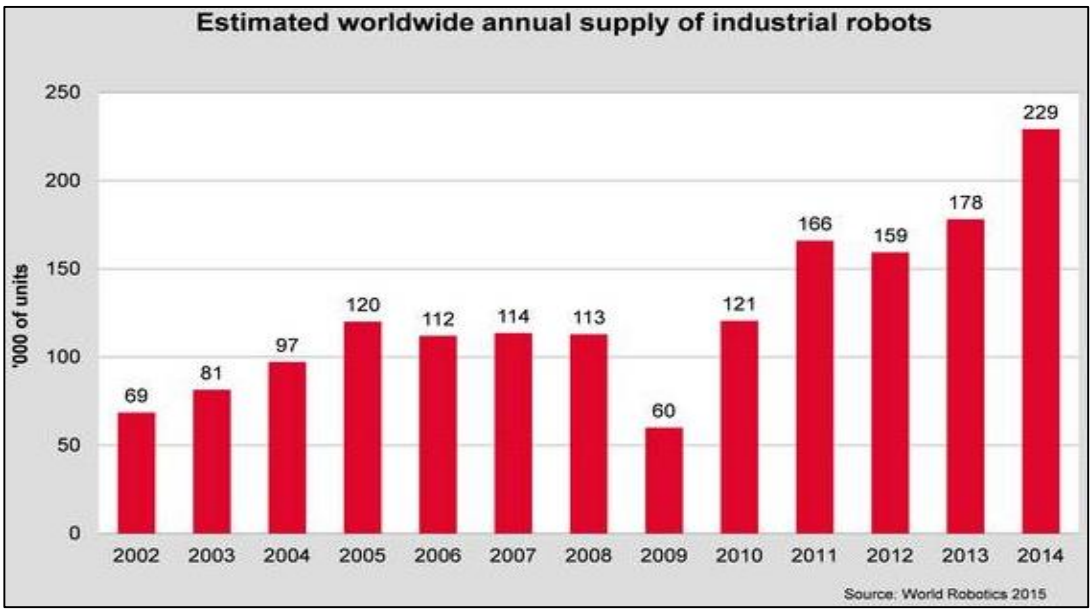


Figura 1.1: Suministro anual mundial de robots industriales

En España, el parque de robots industriales ronda según el estudio desarrollado por la IFR las 29.000 unidades y nos posiciona como octava potencia mundial (ver Figura 1.2). Una gran cantidad de los robots operativos en España (casi 19.000 unidades) trabajan en el sector automovilístico.

Estimated operational stock of multipurpose industrial robots at year-end in selected countries. Number of units

Country	2013	2014	2015*	2018*
America	**226,071**	**248,430**	**272,000**	**343,000**
Brazil	8,564	9,557	10,300	18,300
North America (Canada, Mexico, USA)	215,817	236,891	259,200	323,000
Other America	1,690	1,982	2,500	1,700
Asia/Australia	**689,349**	**785,028**	**914,000**	**1,417,000**
China	132,784	189,358	262,900	614,200
India	9,677	11,760	14,300	27,100
Japan	304,001	295,829	297,200	291,800
Republic of Korea	156,110	176,833	201,200	279,000
Taiwan	37,252	43,484	50,500	67,000
Thailand	20,337	23,893	27,900	41,600
other Asia/Australia	29,188	43,871	60,000	96,300
Europe	**392,227**	**411,062**	**433,000**	**519,000**
Czech Rep.	8,097	9,543	11,000	18,200
France	32,301	32,233	32,300	33,700
Germany	167,579	175,768	183,700	216,800
Italy	59,078	59,823	61,200	67,000
Spain	28,091	27,983	28,700	29,500
United Kingdom	15,591	16,935	18,200	23,800
other Europe	81,490	88,777	97,900	130,000
Africa	**3,501**	**3,874**	**4,500**	**6,500**
not specified by countries**	21,070	32,384	40,500	41,500
Total	**1,332,218**	**1,480,778**	**1,664,000**	**2,327,000**

Sources: IFR, national robot associations.

*forecast

** reported and estimated sales which could not be specified by countries

Figura 1.2: Stock estimado de robots industriales según países

España ocupa además el quinto lugar en densidad de robots por cada 10.000 empleados y en los próximos tres años las inversiones rondarán los 2.500 millones de euros en nuestro país. La clave parece residir en que a pesar de ser el cuarto país europeo en consumo, un mercado que factura solo en robots más de 4.000 millones de dólares al año, 15.000 millones si incluimos el software y la ingeniería, España apenas los fabrica. Los robots españoles solo facturan unos 400 millones al año.

1.2 JUSTIFICACION DEL PROYECTO

La principal razón por la que se desarrolla este proyecto es la de modelar la estación de trabajo robotizada disponible en el laboratorio de prácticas del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Valladolid. Para modelar la estación de trabajo nos ayudaremos de RobotStudio, que es el software de simulación y programación fuera de línea proporcionado por el fabricante ABB. La ventaja de utilizar una programación fuera de línea es que permite efectuar cambios en la programación del robot en un ordenador sin interrumpir el ciclo de funcionamiento normal de la célula robotizada.

Una vez tengamos la estación de trabajo modelada, se desarrollará una aplicación Android que nos permitirá comunicarnos con el controlador de la del robot ABB IRB 120 mediante tecnología Wi-Fi, utilizando para ello la comunicación por Sockets. La aplicación Android se encargará de simular el panel exterior de entradas y salidas digitales disponible en el laboratorio de prácticas, teniendo de esta manera dos alternativas para visualizar las salidas digitales o para cambiar el estado de las entradas digitales.

Al avanzar en el presente proyecto veremos cómo los resultados obtenidos mediante la simulación proporcionada por RobotStudio y los resultados obtenidos de la célula robotizada real del laboratorio coinciden.

1.3 OBJETIVOS

El presente proyecto tiene como finalidad modelar los diferentes elementos que componen la estación de trabajo del laboratorio de prácticas del Departamento de Ingeniería de Sistemas y Automática, utilizando para ello el software RobotStudio.

Una vez modelada la estación de trabajo se desarrollará una aplicación móvil basada en el sistema operativo Android, que nos permitirá comunicarnos e intercambiar información con la célula robotizada mediante el uso de mecanismos de comunicación denominados “Sockets”. Este mecanismo permite la comunicación entre diferentes máquinas a través del protocolo de comunicación de red TCP/IP.

Los sockets permiten implementar una arquitectura cliente-servidor. La comunicación debe ser iniciada por uno de los programas que se denomina programa "cliente" (en nuestro caso la aplicación Android). El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa "servidor" (en nuestro caso el programa RAPID de RobotStudio).

Con el objetivo de comprobar el correcto funcionamiento y sincronismo entre la estación simulada y la estación de trabajo real, se realizaran una serie de aplicaciones didácticas:

- En primer lugar se desarrollará una aplicación que permita jugar con la célula robotizada al famoso juego de las “Tres en Raya”, comprobando de tal forma que la estación de trabajo y la real trabajan en sincronismo.
- En segundo lugar se desarrollará una aplicación de uso genérico que permita simular el comportamiento de la botonera de entradas/salidas digitales de que dispone el departamento en el laboratorio de prácticas.
- Por último, se programará el uso de la botonera de entradas/salidas digitales para poder manejar el movimiento del robot IRB 120 a través de una aplicación móvil Android.

1.4 ESTRUCTURA DE LA MEMORIA

1.4.1 Introducción y objetivos

En este capítulo se presentan las motivaciones que me han llevado a realizado este proyecto, así como la justificación y los objetivos que se pretenden alcanzar con la elaboración del mismo. Por otra parte también se realiza una visión general de la estructura que tendrá la memoria, describiendo brevemente la temática de cada capítulo.

1.4.2 Elementos de la célula robotizada

En este capítulo se realiza una descripción del robot industrial ABB IRB 120 y de todos los elementos que componen la célula robotizada (panel exterior de entradas y salidas digitales, rotuladores, soporte de los rotuladores, cubos, Tablet Kindle Fire HD 7", etc.).

1.4.3 Estado del arte

En este apartado se realiza una descripción del software y tecnologías utilizados durante el desarrollo del proyecto. Se presentarán en este capítulo el software de simulación "RobotStudio" proporcionado por el fabricante ABB y el entorno de desarrollo integrado para la plataforma Android conocido como "Android Studio". También se explicará brevemente los principales aspectos de la comunicación inalámbrica y la comunicación por Sockets.

1.4.4 Modelado de la célula robotizada

En esta sección se explicará el proceso de modelado, realizado con el software RobotStudio, de los elementos de la estación de trabajo y de su interconexionado para conseguir que la estación simulada y la célula robotizada real tengan el mismo comportamiento.

1.4.5 Programación de la Aplicación Android

En este capítulo se describirá de una manera general la estructura principal y las diferentes herramientas de programación utilizadas para implementar la aplicación Android que se encargará de manejar las órdenes que le mandemos al controlador del robot a través de la tecnología de comunicación inalámbrica Wi-Fi.

1.4.6 Aplicaciones didácticas

En este apartado se desarrollarán una serie de aplicaciones didácticas que nos permitirán comprobar que la estación simulada y la célula robotizada real trabajan de forma síncrona y realizan los mismos procesos. El objetivo de estas aplicaciones es comprobar el correcto funcionamiento de la comunicación entre el controlador del robot y la aplicación Android creada para simular el uso de la botonera de entradas/salidas digitales disponible en el laboratorio de prácticas del departamento de Ingeniería de Sistemas y Automática.

1.4.7 Conclusiones y líneas futuras de investigación

En este capítulo se extraerán las conclusiones obtenidas tras la realización del presente proyecto, así como las posibles líneas futuras de investigación que se podrían llevar a cabo tomando como base el contenido del proyecto desarrollado a partir de la estación de trabajo robotizada disponible.

1.4.8 Bibliografía

En este apartado se enumerarán las fuentes consultadas como apoyo para la elaboración del proyecto.

1.4.9 Anexos

En este capítulo se mostrarán algunos de las hojas de características de los elementos utilizados en el proyecto (Robot industrial ABB IRB 120, Controlador IRC5 Compact) y los códigos de programación utilizados para la realización del proyecto.

MEMORIA

2 ELEMENTOS DE LA CÉLULA ROBOTIZADA

La célula robotizada en la que se basa el proyecto se encuentra situada en el laboratorio de prácticas del Departamento de Ingeniería de Sistemas y Automática situado en la planta sótano de la sede del Paseo del Cauce de la Escuela de Ingenierías Industriales. Esta célula robotizada se utiliza con fines didácticos para realizar algunas de las prácticas de la asignatura de Sistemas Robotizados correspondiente al cuarto curso del Grado de Ingeniería en Electrónica Industrial y Automática, además de utilizarse con fines de investigación. Por esta razón el tamaño del robot es relativamente pequeño si lo comparamos con los robots industriales comúnmente utilizados en la industria, los cuales deben soportas cargas mucho mayores de las que se le exige que soporte un robot con fines didácticos como con el que trabajaremos en el presente proyecto (ver Figura 2.1).

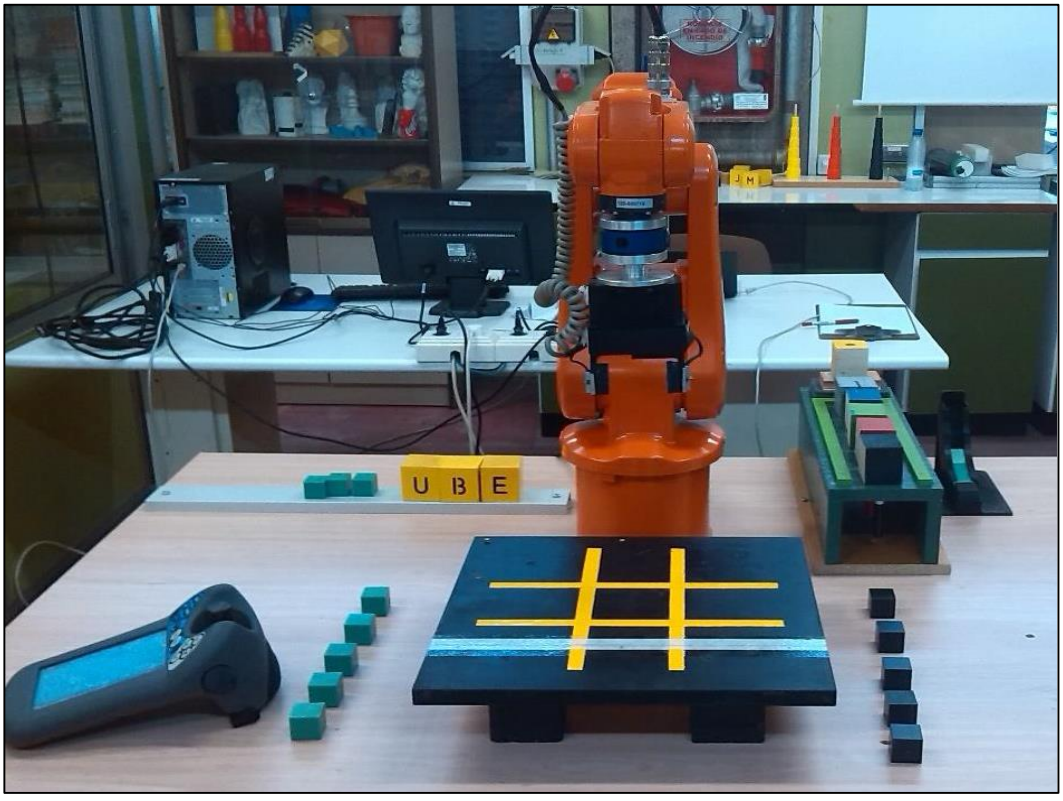


Figura 2.1: Estación de trabajo disponible en el laboratorio

A continuación se describirán los elementos de los que está compuesta la célula robotizada.

2.1 ROBOT ABB IRB 120

El laboratorio de prácticas del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Valladolid cuenta con un robot ABB IRB 120. Se trata del robot manipulador multiusos más pequeño desarrollado por ABB en su cuarta generación de tecnología robótica, que consta de 6 grados de libertad correspondientes a cada uno de los seis ejes del manipulador.

El robot industrial ABB IRB 120 tan solo pesa 25 kg y puede soportar una carga de 3 kg (4 kg en posición vertical de la muñeca) con un alcance de 580 mm. Es una elección fiable y rentable para generar una gran capacidad de producción a cambio de una baja inversión, proporcionando una extrema flexibilidad, precisión y rapidez, razón por la cual es utilizado en aplicaciones de pick & place, sobre todo en las industrias de alimentos, bebidas y de embalaje. Estos robots industriales pueden ser utilizados en aplicaciones de montaje para los fabricantes de componentes eléctricos, electrónicos y de automoción. También hay disponible una versión en acabado blanco Clean Room ISO 5 (clase 100) con certificado de la IPA.

Figura 2.2: Robot IRB 120, Controlador IRC5 Compact y FlexPendant

El robot IRB 120 viene acompañado del controlador IRC5 Compact, el cual ofrece todas las ventajas del controlador de robot líder del mercado utilizado en la mayoría de las industrias. El controlador ABB IRC5 Compact incluye el control de trayectorias TrueMove, la unidad de programación táctil FlexPendant, el software RobotStudio que permite una programación fuera de línea, la flexibilidad del lenguaje RAPID y potentes capacidades de comunicación entre otras características (ver Figura 2.2).

La unidad de programación FlexPendant se basa en la tecnología más avanzada de Microsoft, el sistema operativo Windows® CE, con su estilo Windows de símbolos en color e interfaz que facilitan el uso para el operario.

Con el fin de formar a los futuros ingenieros y especialistas técnicos con las habilidades necesarias, ABB ofrece a los institutos y universidades un nuevo paquete robótico diseñado para demostrar y enseñar los conceptos de programación de la vida real industrial en las aulas. De esta manera, utilizando la misma tecnología que en la industria, el paquete educativo de ABB ofrece a los estudiantes la oportunidad de aprender las habilidades necesarias para la gestión de la robótica moderna en los sistemas de fabricación.

El paquete de robótica de ABB con fines educativos está diseñado para ofrecerá a los estudiantes una visión de:

- Como operan los robots industriales.
- El diseño y los principios de control de los robots industriales.
- El diseño y los principios de control de las células de trabajo robóticas.
- La programación de los robots ABB utilizando la programación fuera de línea con el software RobotStudio y el lenguaje de programación RAPID.
- Los peligros de la célula robotizada en cuanto a salud, seguridad y mantenimiento.

2.2 PINZA DE AGARRE

La herramienta de trabajo que utilizaremos para nuestra estación de trabajo robotizada es una pinza de agarre (ver Figura 2.3), la cual se extrajo de un antiguo robot manipulador y se ha acoplado para que realice las funciones de manipulación de objetos en nuestra célula robotizada.

Figura 2.3: Herramienta de trabajo (Pinza de agarre)

La pinza de agarre se encuentra ligada al robot mediante un sensor de fuerza-par JR3, teniendo de esta manera la posibilidad de obtener los datos relativos a las fuerzas de reacción que se producen en la superficie de separación entre el sensor de fuerza JR3 y la muñeca del robot (Eje 6).

La estructura de la herramienta de trabajo se basa en un soporte, al cual se le acoplan dos eslabones que se desplazan horizontalmente realizando de esta forma la apertura o cierre de la pinza. El peso de la pinza es de 2 kg y permite manipular objetos de pequeñas dimensiones. La apertura total de la pinza, medida desde un eslabón lateral a su opuesto, es de 64 mm. La carga que puede soportar la pinza para asegurar una correcta fijación del objeto manipulado a los eslabones laterales, sin riesgo de que éste se "suelte", es de 1 kg.

En un eslabón lateral de la pinza se encuentra un sensor de presión, el cual se activa cuando el objeto que manipulamos oprime la superficie donde se encuentra el sensor. También se ha dotado a la pinza de un sensor laser de posición todo/nada que se activa ante la presencia de un objeto. En caso de no detectar ningún objeto, la señal del sensor se encuentra en nivel lógico bajo. La presencia de un objeto se determina a partir de un haz laser horizontal situado en la parte superior de los eslabones laterales. En la figura 2.4 se puede apreciar el principio de funcionamiento del sensor de posición laser que utiliza la herramienta de trabajo.

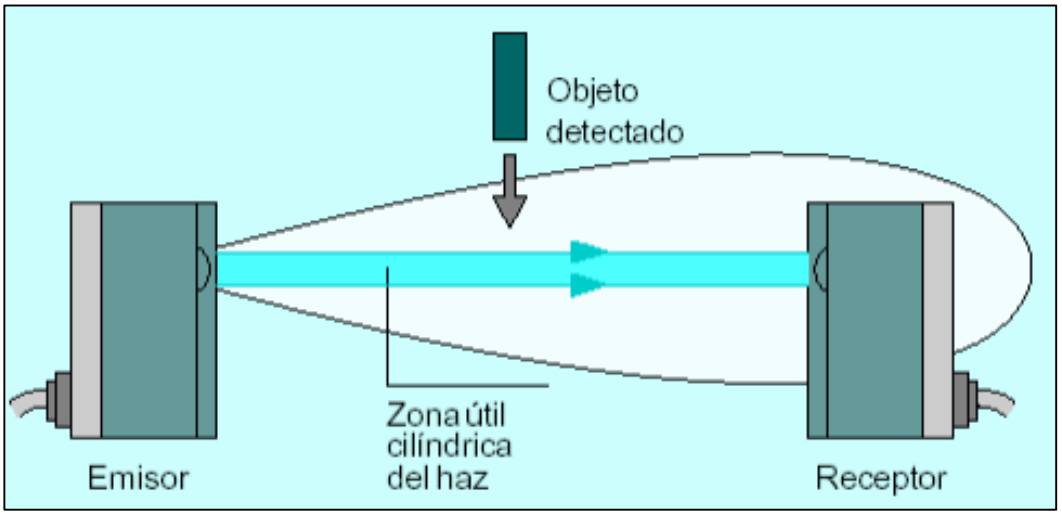


Figura 2.4: Principio funcionamiento sensor laser de posición

De esta manera, la pinza de agarre nos proporciona a la estación robotizada 4 señales digitales, que son:

- Señal de entrada digital que controla la apertura de la pinza
- Señal de entrada digital que controla el cierre de la pinza.
- Señal de salida digital que indica la existencia o no de un objeto ejerciendo presión sobre los eslabones.
- Señal de salida digital que indica la presencia o no de un objeto cortando el haz laser del sensor de posición.

2.3 PANEL EXTERNO DE ENTRADAS Y SALIDAS DIGITALES

El controlador IRC 5 Compact desarrollado por ABB está equipado con conectores externos para todas las señales y un sistema de E/S integrado ampliable de 16 entradas y 16 salidas. Este sistema de E/S es el módulo DSQC 652 de 24 V consta de 5 conectores tal y como se aprecia en la Figura 2.5, utilizando la tecnología DeviceNet.

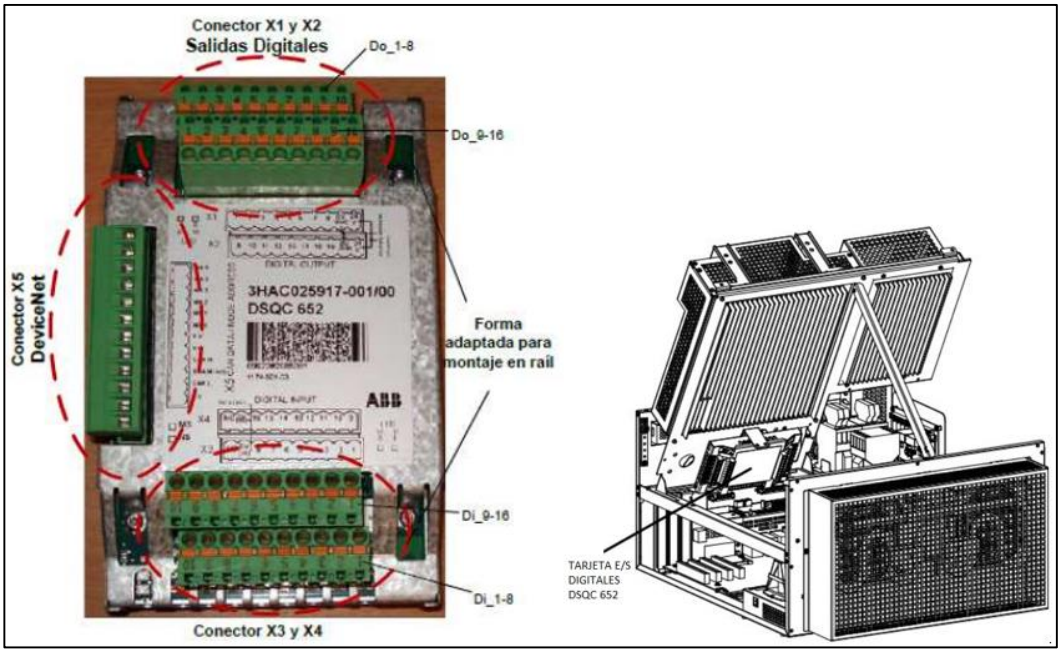


Figura 2.5: Módulo E/S Digitales 24 V (DSQC 652)

La tecnología DeviceNet es una red digital, multi-punto para conexión entre sensores, actuadores y sistemas de automatización industrial en general. Esta tecnología fue desarrollada para tener máxima flexibilidad entre los equipos de campo e interoperabilidad entre diferentes fabricantes. La red DeviceNet está clasificada en el nivel de red llamada DeviceBus, cuyas características principales son: alta velocidad, comunicación a nivel de byte (incluye comunicación con equipos discretos y analógicos) y alto poder de diagnóstico de los dispositivos de la red.

Los conectores que se aprecian en la figura 2.5 son:

- Conectores X1 y X2: son los conectores utilizados para las salidas digitales que se utilizan en la célula. Se usan todas las señales, de la señal do1 hasta do8 en el caso de X1 y de la do9 hasta do16 en el caso de X2. En total se proporcionan 16 señales digitales de salida.
- Conectores X3 y X4: son los conectores utilizados para las entradas digitales que se utilizan en la célula. Se usan todas las señales, de la señal di1 hasta di8 en el caso de X3 y de la di9 hasta di16 en el caso de X4. En total se proporcionan 16 señales digitales de salida.
- Conector X5: es el encargado de la conexión lógica del módulo E/S al bus de campo DeviceNet y a su vez, de la conexión del módulo E/S con la CPU del controlador IRC5. El controlador incluye una tarjeta PCI certificada por ABB que hace posible montar las unidades de E/S dentro del armario o fuera de él, usando un cable o bus para conectar la unidad de E/S al armario, ya sea mediante la opción DeviceNet o Profibus DP. En este caso se usa la opción DeviceNet, dentro de la cual ABB suministra un módulo de E/S digitales de 24 Vcc y que proporciona 16 entradas y 16 salidas. El límite de corriente que proporciona este módulo es de 8 A.

Las características de las señales utilizadas por la tarjeta de entradas y salidas digitales DSQC 652 se pueden apreciar en la Figura 2.6 y en la Figura 2.7, respectivamente.

Entradas digitales (opción 716-1, 717-2, 718-2)	Valores
24 V CC con aislamiento óptico	
Tensión nominal	24 V CC
Niveles de tensión lógica	"1" de 15 a 35 V "0" de -35 a 5 V
Intensidad de entrada con la tensión de entrada nominal	6 mA
Diferencia de potencial	Máx. 500 V
Retardos de tiempo	Filtro físico = 5 ms (± 0,5 ms) Retardo de software ≤ 0,5 ms[i]
Variaciones de tiempo	-1 ms +2 ms

[i] El tiempo de retardo de software depende del tipo de conexión. El tiempo indicado aquí corresponde a los parámetros predeterminados, cambio de estado con tiempo de inhibición de producción de 10 ms.

Figura 2.6: Características señales entradas digitales DSQC 652

Salidas digitales (opción 716-1, 717-2)	Valores
24 V CC con aislamiento óptico	Protegidas contra cortocircuitos, protección de polaridad de alimentación
Tensión de alimentación	De 19 a 35 V
Tensión nominal	24 V CC
Niveles de tensión lógica	"1" de 18 a 34 V "0" < 7 V
Salida de intensidad	Máx 0,5 A/canal
Diferencia de potencial	Máx. 500 V
Retardos de tiempo	Hardware ≤ 0,5 ms Software ≤ 1 ms
Variaciones de tiempo	-1 ms + 2 ms

Figura 2.7: Características señales salidas digitales DSQC 652

El conector XS7 que se encuentra en la parte frontal del controlador IRC5 está conectado internamente a la tarjeta de E/S digitales DSQC 652, tal y como se aprecia en el esquema eléctrico de la Figura 2.8.

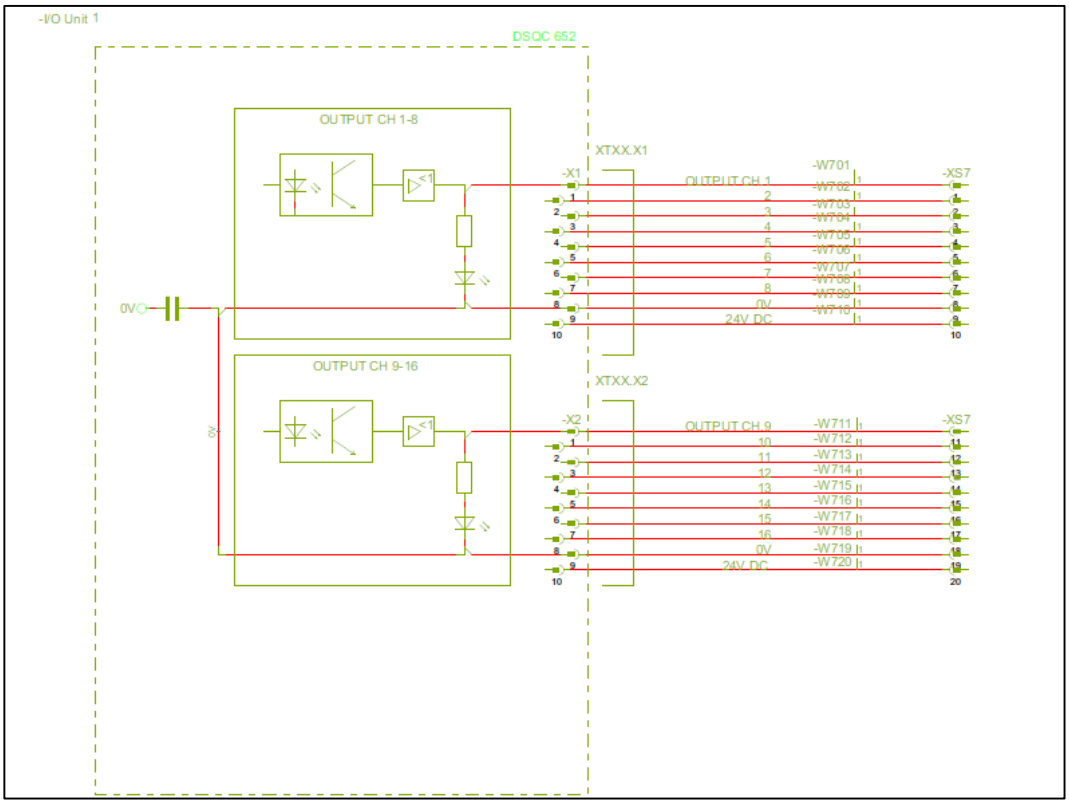


Figura 2.8: Esquema eléctrico conector XS7

Por lo tanto, será al conector XS7 donde deberemos conectar nuestro panel externo de entradas y salidas digitales (ver Figura 2.10) para interactuar con la célula robotizada y obtener los datos que maneja la tarjeta DSQC 652.

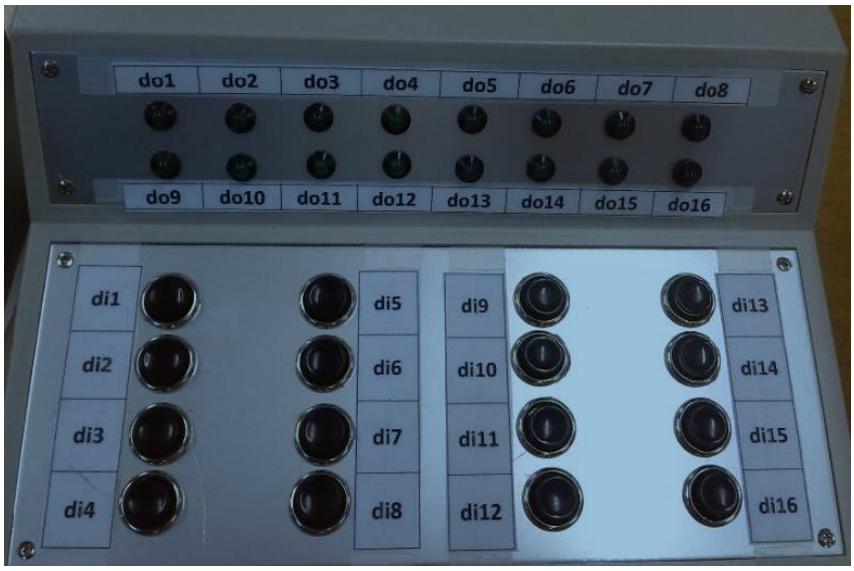


Figura 2.9: Panel externo E/S digitales disponible en el laboratorio

El panel externo de que dispone el laboratorio dispone de 16 botones correspondientes con cada una de las entradas digitales “di” que se activan/desactivan en la tarjeta de E/S. Dicho panel también cuenta con 16 leds correspondientes a cada una de las salidas digitales “do”, de modo que si la salida digital toma un valor alto “1” el led se iluminará y en caso contrario el led permanecerá apagado.

2.4 TABLET KINDLE FIRE HD 7”

Para la realización de del presente proyecto se ha utilizado una Tablet Kindle Fire HD 7” (ver Figura 2.11), que tiene como principal finalidad el poder mandar las instrucciones y señales mediante la tecnología Wi-Fi así como recibir posibles eventos que se produzcan en la célula robotizada. Es imprescindible que el dispositivo utilizado para comunicarnos con la célula robotizada permita la conexión Wi-Fi, ya que mediante “Sockets” enviamos la información necesaria desde una aplicación Android desarrollada con dicho propósito.

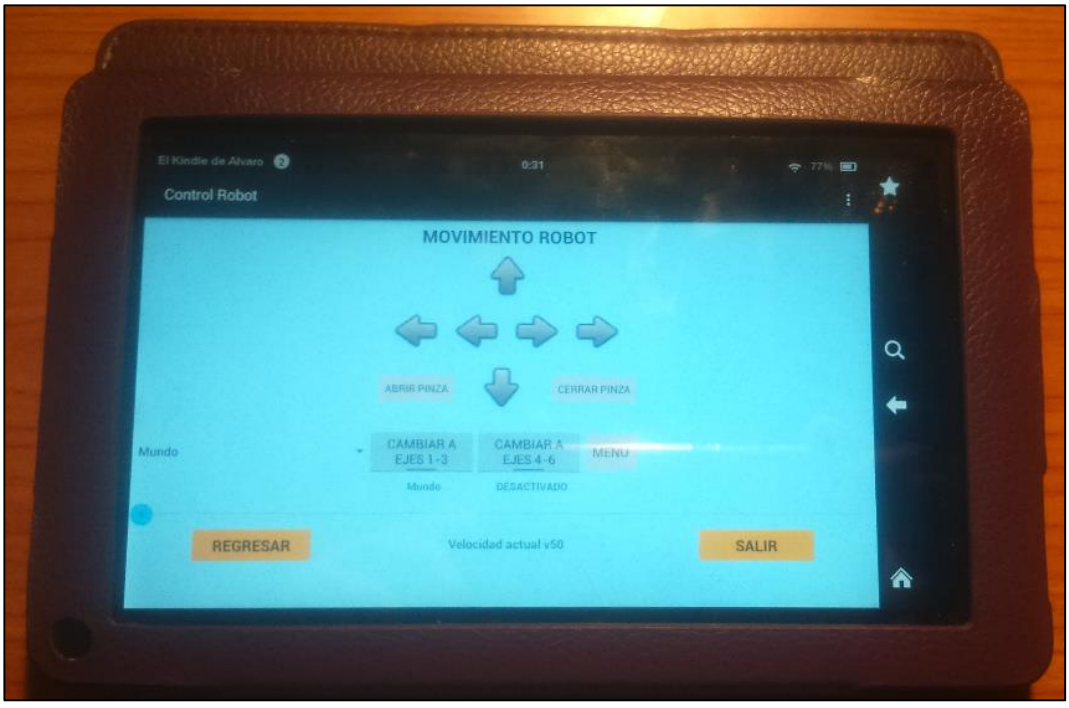


Figura 2.10: Tablet Kindle Fire HD 7"

Las características de la Tablet Kindle Fire HD utilizada son:

- Pantalla IPS de 7" (17,7 cm)
- Resolución: 1024 x 600 (171 ppp)
- Procesador: Quad-Core 1,3 GHz
- Almacenamiento 8 GB (hasta 128 GB extra con microSD)
- Conectividad Wi-Fi
- Dimensiones: 191 x 115 x 10,6 mm
- Peso: 313 gramos

2.5 SOPORTES DE TRABAJO

En la célula robotizada disponible en el laboratorio de prácticas disponemos de dos soportes de trabajo, que los hemos denominado como "Mesa Negra" y "Mesa Inclinada".

2.5.1 Mesa Negra

Este soporte de trabajo se está compuesto por un tablero de madera cuyas dimensiones son 403x398x18 mm, y por un "taco" de madera de 80mm de alto. El tablero de color negro se apoya sobre el "taco" de madera, siendo la altura total de la mesa de trabajo de 98 mm (ver Figura 2.11).

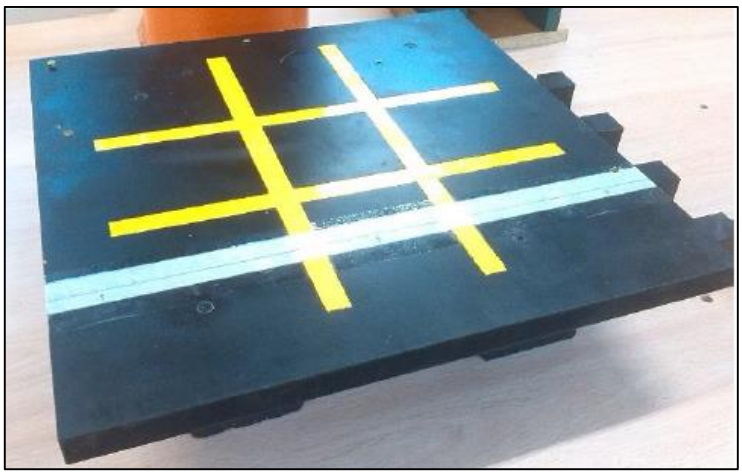


Figura 2.11: Mesa Negra de trabajo

Al tablero superior de la mesa de trabajo se le han añadido 4 rayas amarillas (2 horizontales y 2 verticales) que dividen la superficie en 9 porciones cuadradas, que nos servirán en un futuro para simular las casillas de un tablero del conocido juego de "Tres en Raya". La función principal de la mesa negra es la servir de soporte de las piezas que manipule el robot.

2.5.2 Mesa Inclinada

Este soporte de trabajo está compuesto por un tablero de color verde cuyas dimensiones son 500x386x25 mm, y por una tabla, que se encuentra colocado en posición vertical, de 100 mm de alto. El tablero de color verde se apoya en un extremo de éste sobre la tabla vertical, quedando el soporte de trabajo con una inclinación de 11,6º respecto de la superficie horizontal de referencia (ver Figura 2.12).

Figura 2.12: Mesa inclinada de trabajo

Debido a su inclinación y a un panel interior de color blanco que se ha colocado en el interior del tablero verde, esta mesa de trabajo tiene unas características idóneas para ser una superficie donde el robot pueda, a través de unos rotuladores existentes en el laboratorio, dibujar textos y figuras.

2.6 PIEZAS

En la célula robotizada disponible en el laboratorio de prácticas hay disponibles un conjunto de piezas que el robot puede manipular. Este conjunto de piezas está compuesto por 5 rotuladores, un soporte para los rotuladores y 10 cubos de pequeñas dimensiones.

2.6.1 Rotuladores

En la estación de trabajo dispones de 4 rotuladores de diferentes colores (negro, rojo, verde, azul) y de un rotulador con la punta vacía que se utiliza para comprobar posiciones. La estructura de los rotuladores se basa en un taco de madera 60x48x58 mm, al cual se le realiza un taladrado interior de 16mm, que es lo suficientemente grande para que en su interior se inserte un rotulador de 15,2 mm de diámetro. La fijación del rotulador al taco de madera se realiza mediante un tornillo pasante lateral que presiona al rotulador una vez introducido éste en el agujero del taco (ver Figura 2.13).

Figura 2.13: Rotuladores de color rojo y negro

El robot IRB 120 manipula los rotuladores "agarrándolos" con la pinza por los lados del cubo sin color y es capaz de escribir texto o dibujar figuras, gracias a su movimiento de arrastre sobre una superficie, una vez que entran en contacto la punta del rotulador y la superficie donde se quiera escribir o dibujar.

2.6.2 Soporte de rotuladores

El soporte de rotuladores se ha diseñado con la finalidad de tener un lugar donde alojar los rotuladores y que el robot IRB 120 conozca en todo momento las coordenadas a las que debe dirigirse para "agarrar" y "soltar" los rotuladores. Al soporte de rotuladores se la han realizado 5 taladrados superiores de 20 mm de diámetro, que son donde se apoyaran los rotuladores, estando definidas de antemano las posiciones de los diferentes rotuladores de colores (ver Figura 2.14).

Figura 2.14: Soporte de rotuladores

Como se puede apreciar en la figura 2.14, se ha creado también un borrador que pueda ser manipulado por el robot. Para ello, se ha unido un "borrador clásico", utilizado normalmente en las aulas de enseñanza, a un taco de madera de semejantes características de los descritos anteriormente en el caso de los rotuladores.

2.6.3 Cubos

En la estación de trabajo disponemos de 10 cubos de plástico de pequeñas dimensiones (28x28x28 mm), con el objetivo de que el robot pueda manipularlos fácilmente con la pinza de agarre (ver Figura 2.15). Los cubos tienen como finalidad simular las fichas utilizadas en una aplicación didáctica desarrollada que se basa en el juego "Tres en Raya".

Figura 2.15: Cubo de color verde

3 ESTADO DEL ARTE

En esta sección se van a enumerar y a describir las tecnologías utilizadas para la realización del presente Trabajo Fin de Grado. Las tecnologías que veremos serán las siguientes:

- RobotStudio
- Android Studio
- Comunicación inalámbrica
- Comunicación por Sockets

3.1 ROBOTSTUDIO

3.1.1 Introducción

El fabricante ABB ha diseñado el software RobotStudio con el fin de ser una herramienta útil a la hora de simular y programar fuera de línea una estación robotizada. La programación fuera de línea es la mejor manera de maximizar el rendimiento de los sistemas robotizados. El software de simulación y programación fuera de línea de ABB, RobotStudio, permite efectuar la programación del robot en un ordenador, sin necesidad de interrumpir para ello el ciclo normal de operación del robot.

El objetivo de la simulación mediante el software de programación fuera de línea RobotStudio es el de analizar, verificar y, en su caso, corregir el comportamiento de los sistemas con el fin de elaborar estaciones de trabajo más precisas, útiles y eficaces. El beneficio de utilizar un software de simulación, radica en el hecho de que no se expone la planta real de trabajo a posibles fallos, evitando de esta forma daños mecánicos, daños personales u otro tipo de daño que pudieran ocurrir en una estación de trabajo real. RobotStudio se ha construido en el VirtualController de ABB, una copia exacta del software real que hace funcionar su robot en producción. Esto permite simulaciones muy realistas, con archivos de configuración y programas de robot reales e idénticos a los utilizados en su instalación real.

El software RobotStudio permite trabajar con un controlador fuera de línea, que constituye un controlador IRC5 virtual que se ejecuta localmente en el PC. Este controlador fuera de línea también se conoce como el controlador virtual (VC). El software también permite trabajar con un controlador IRC5 físico real, que simplemente se conoce como el controlador real.

Cuando RobotStudio se utiliza con controladores reales, se conoce el modo de funcionamiento como online. Al trabajar sin conexión a un controlador real o mientras está conectado a un controlador virtual, se dice que RobotStudio se encuentra en el modo fuera de línea.

Para que la célula robotizada realice determinadas tareas es necesario dotar al software RobotStudio de un programa con las instrucciones precisas que queremos que realice nuestro robot. Este programa debe ser implementado con un lenguaje de programación de alto nivel denominado RAPID, que es el utilizado por los robots industriales ABB.

La estructura del software RobotStudio, se compone principalmente de dos archivos ejecutables, encargándose cada uno de ellos de unas tareas determinadas, tal y como podemos apreciar en la Figura 3.1.

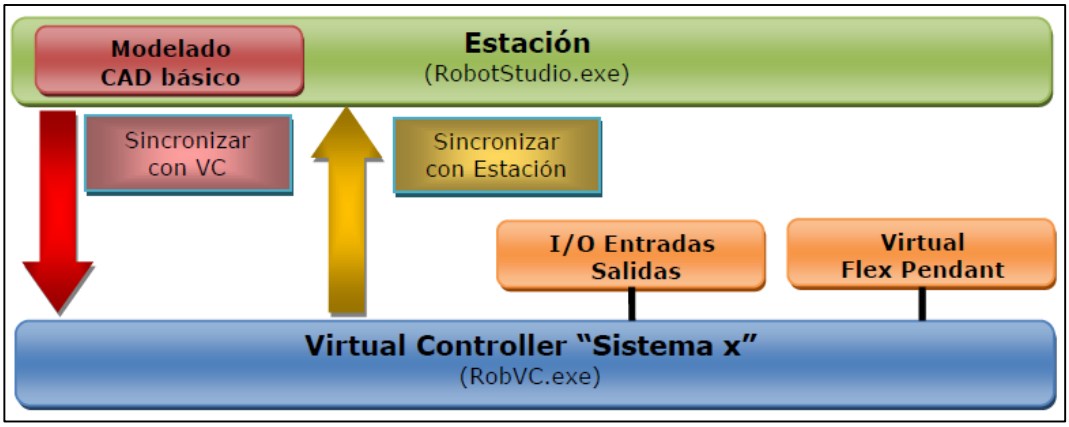


Figura 3.1: Estructura software RobotStudio

Estos dos archivos ejecutables en los que se basa RobotStudio son:

- Estación: La parte que se encarga de la creación del sistema, diseño, programación de trayectorias, movimiento manual de ejes, modelado etc. Es decir las tareas comprendidas en los menús "Inicio" y "Modelado".
- Controlador virtual: La parte que se encarga del programa RAPID, edición y compilación del programa, como tarea fundamental. Es decir las tareas comprendidas en los menús "Controlador" y "Rapid".

3.1.2 Interfaz gráfica

La interfaz gráfica de usuario que presenta el software RobotStudio es bastante intuitiva y de fácil manejo. En la figura 3.2 se pueden apreciar la cinta, las pestañas y los grupos de la interfaz gráfica de usuario, donde se han enumerado las pestañas que se describirán a continuación.

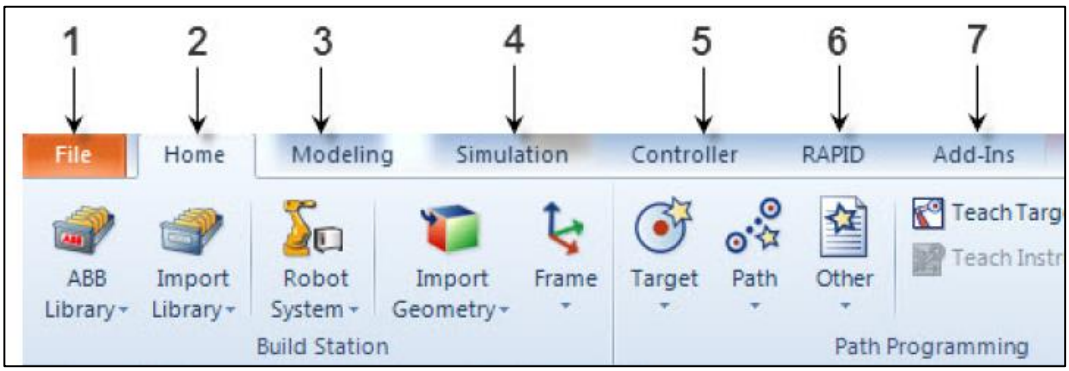


Figura 3.2: Cinta, pestañas y grupos de la interfaz gráfica

Las diferentes pestañas que disponibles en la interfaz gráfica son:

1. Archivo

Contiene las opciones necesarias para crear una nueva estación, crear un sistema de robot, conectarse a un controlador, guardar la estación como un visor y otras opciones.

2. Inicio

Contiene los controladores necesarios para construir estaciones, crear sistemas, programar trayectorias y colocar elementos.

3. Modelado

Contiene los controles necesarios para crear y agrupar componentes, crear cuerpos, mediciones y operaciones de CAD.

4. Simulación

Contiene los controles necesarios para crear, configurar, controlar, monitorizar y grabar simulaciones.

5. Controlador

Contiene los controles necesarios para la sincronización, configuración y tareas asignadas al controlador virtual (VC). También contiene controladores para gestionar los controladores reales.

6. RAPID

Contiene el editor de RAPID integrado utilizado para editar todas las tareas del robot distintas de las de movimiento del robot.

7. Complementos

Contiene los controles de los PowerPacs y las herramientas de migración y predicción de calor de la caja reductora.

3.1.3 Lenguaje RAPID

El software RobotStudio utiliza RAPID, que es un lenguaje de programación textual de alto nivel utilizado por los robots industriales ABB. Una aplicación de RAPID contiene una secuencia de instrucciones que sirven para controlar el robot, de forma que realice las operaciones deseadas por el usuario final de la célula robotizada. El lenguaje RAPID se basa en vocablos del idioma inglés y permite ejecutar instrucciones, tales como activar o desactivar salidas, leer entradas, tratamiento de eventos o comunicarse con el operador del sistema entre otras funciones. Una aplicación RAPID está dividida en un “programa” y “módulos del sistema”, estando a su vez el “programa” dividido en diferentes módulos, tal y como se puede apreciar en la Figura 3.3.

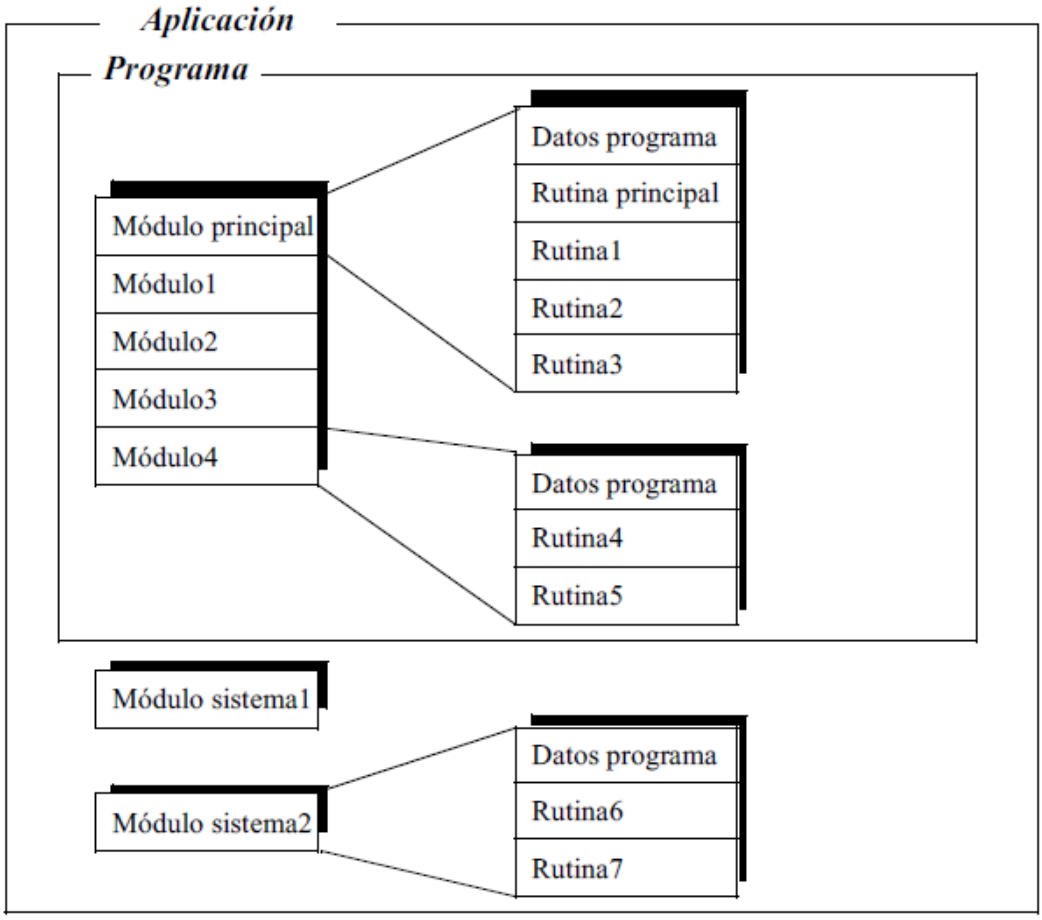


Figura 3.3: Estructura de una aplicación RAPID

3.1.3.1 Identificadores

Los identificadores sirven para nombrar los módulos, las rutinas, los datos y las etiquetas. El primer carácter de un identificador deberá ser siempre una letra, siendo el resto de caracteres letras, cifras o "barra baja".

```
Ejemplo:    MODULE      nombre_modulo
            PROC        nombre_rutina()
            VAR pos     nombre_variable;
            nombre_etiqueta;
```

El lenguaje RAPID tiene palabras reservadas que no se podrán utilizar como identificadores, ya que tienen un significado especial dentro del lenguaje de programación. Estas palabras reservadas se pueden observar en la Figura 3.4.

ALIAS	AND	BACKWARD	CASE
CONNECT	CONST	DEFAULT	DIV
DO	ELSE	ELSEIF	ENDFOR
ENDFUNC	ENDIF	ENDMODULE	ENDPROC
ENDRECORD	ENDTEST	ENDTRAP	ENDWHILE
ERROR	EXIT	FALSE	FOR
FROM	FUNC	GOTO	IF
INOUT	LOCAL	MOD	MODULE
NOSTEPIN	NOT	NOVIEW	OR
PERS	PROC	RAISE	READONLY
RECORD	RETRY	RETURN	STEP
SYSMODULE	TEST	THEN	TO
TRAP	TRUE	TRYNEXT	VAR
VIEWONLY	WHILE	WITH	XOR

Figura 3.4: Palabras reservadas lenguaje RAPID

3.1.3.2 Módulos

Existen dos clases de módulos en una aplicación RAPID: módulos de programa y módulos del sistema.

3.1.3.2.1 Módulos del programa

Un módulo de programa puede estar formado de diferentes datos y rutinas. Cada módulo o el programa entero, podrá ser copiado en una unidad de almacenamiento externo, en la memoria RAM, etc., y viceversa. Uno de los módulos contiene el procedimiento de entrada, un procedimiento global llamado “main” (principal). Al ejecutar el programa, lo que se está haciendo realmente es ejecutar el procedimiento principal “main”. El programa puede incluir varios módulos, pero sólo uno de ellos debe contener el procedimiento principal.

3.1.3.2.2 Módulos del sistema

Los módulos del sistema sirven para la definición de los datos y rutinas normales, específicos del sistema, como por ejemplo, las herramientas. No serán incluidos cuando se salva un programa, lo cual implica que cualquier actualización realizada en un módulo del sistema afectará a todos los programas que se encuentran en él.

3.1.3.3 Rutinas

Existen tres tipos de rutinas (subprogramas): los procedimientos, las funciones y las rutinas de tratamiento de interrupciones.

- Los procedimientos no devuelven ningún valor y se utilizan en el contexto de las instrucciones.
- Las funciones devuelven un valor de un tipo específico y se utilizan en el contexto de las expresiones.
- Las rutinas de tratamiento de interrupciones proporciona una manera de procesar las interrupciones. Una rutina de tratamiento de interrupciones puede estar asociada a una interrupción específica y en el caso de que dicha interrupción ocurra durante la simulación, se volverá a ejecutar automáticamente.

3.1.3.4 Tipos de datos

En el lenguaje RAPID existen tres tipos de datos: variables, persistentes y constantes.

- Un dato variable podrá tener asignado un valor nuevo durante la ejecución del programa.
- Un dato persistente puede definirse como un “variable persistente”, que tiene la particularidad de que su valor d inicialización se actualiza a medida que se ejecuta el programa. Cuando se guarda un programa, el valor de inicialización de una variable persistente, refleja el último valor que ha tomado el dato persistente.
- Un dato constante representa un valor estático al que no se le podrá asignar ningún valor nuevo.

3.1.3.5 Instrucciones

El programa RAPID se ejecuta generalmente de forma secuencial, es decir, instrucción por instrucción. En algunas ocasiones, para hacer frente a diferentes situaciones que surgen durante la ejecución del programa, son necesarias instrucciones que interrumpen esta ejecución secuencial y que llaman a otras instrucciones. El flujo de programa puede ser controlado a partir de 5 situaciones, que son:

- Llamando a otra rutina (PROC) y, una vez ejecutada esta rutina, el sistema continúa la ejecución con la siguiente instrucción de la llamada a la rutina.
- Ejecutando diferentes instrucciones según si una condición dada se cumple o no.
- Repitiendo una secuencia de instrucciones un cierto número de veces o hasta que se haya cumplido una condición dada.
- Saltando a una etiqueta dentro de la misma rutina.
- Parando la ejecución del programa.

Las instrucciones de las que dispone el lenguaje RAPID para controlar la ejecución de un programa dentro de una rutina se muestran en la Figura 3.5.

Instrucción	Sirve para:
Compact IF	Ejecutar una instrucción únicamente si se ha cumplido una condición
IF	Ejecutar una secuencia de instrucciones diferentes dependiendo de si se ha cumplido o no una condición
FOR	Repetir una sección del programa un cierto número de veces
WHILE	Repetir una secuencia de instrucciones diferentes mientras se cumpla una condición dada.
TEST	Ejecutar diferentes instrucciones dependiendo del valor de una expresión
GOTO	Saltar a una etiqueta
label	Especificar una etiqueta (nombre de línea)

Figura 3.5: Instrucciones ejecución lenguaje RAPID

3.1.3.6 Interrupciones

Las interrupciones son utilizadas por el programa para permitirle tratar directamente cualquier evento ocurrido en el sistema, independientemente de la instrucción que se está ejecutando cuando ocurre la interrupción. El programa es interrumpido cuando se produce un evento definido por una interrupción. En este momento se ejecuta un tipo especial de rutina de tratamiento de interrupción, y una vez se ha ejecutado la esta rutina, la ejecución del programa prosigue a partir de donde se había interrumpido.

Cada interrupción tiene asignada una identificación de interrupción propia, que se obtiene creando una variable (tipos de datos “intnum”) y relacionándola con una rutina de tratamiento de interrupción. La identificación de la interrupción, que es una variable, se utiliza entonces para dar la orden a una interrupción. La orden de una interrupción puede ser cualquiera de los siguientes acontecimientos:

- Una entrada o salida está activada en 0 o en 1.
- Ha pasado un cierto periodo de tiempo después de que se haya dado la orden a una interrupción.
- Se ha alcanzado una posición específica.

La conexión de las interrupciones a las rutinas de tratamiento de interrupción se realiza a través de la instrucción “CONNECT” que conecta una variable (identificación de la interrupción) a una rutina de tratamiento de la interrupción.

La petición de interrupciones se realiza a través de las siguientes instrucciones:

- ISignalDI: solicita una interrupción desde una señal de entrada digital
- ISignalDO: solicita una interrupción desde una señal de salida digital.
- ITimer: solicita una interrupción temporizada.
- IError: solicita una interrupción para errores.
- IPers: solicita una interrupción desde una variable persistente.
- TriggInt: solicita una interrupción de una posición fija de movimiento.

3.1.3.7 Multitarea RAPID

La función multitarea RAPID sirve para ejecutar programas de una forma pseudo paralela respecto a la ejecución normal. La ejecución arranca al activar el sistema y seguirá funcionando indefinidamente. Un programa paralelo puede situase en primer plano o en segundo plano respecto a otro programa.

Para poder utilizar esta función, el robot deberá ser configurado con una tarea adicional para cada programa de segundo plano, pudiéndose ejecutar hasta 10 tareas diferentes en pseudo paralelo.

Cada tarea está formada por un conjunto de módulos locales, siendo los datos variables y constantes locales para cada tarea, pero los persistentes no. Un dato persistente con el mismo nombre y del mismo tipo es alcanzable en todas las tareas. Todas las declaraciones deberán de especificar un valor de inicio a la variable persistente, pero sólo será usado por el primer módulo cargado con la declaración.

Cada tarea adicional es arrancada en la secuencia de arranque del sistema. En el caso de que la tarea sea de tipo STATIC será rearrancada en la posición actual; sin embargo, si la tarea es de tipo SEMISTATIC será rearrancado desde el principio, cada vez que se active el sistema.

También se podrá activar la tarea en el tipo NORMAL, y entonces tendrá el mismo comportamiento que la tarea principal. Desde la unidad de programación no se podrá arrancar ninguna tarea excepto la tarea 0 (tarea principal), por lo que la única manera de arrancar otra tarea tipo NORMAL es a través del software de comunicación.

3.1.3.7.1 Comunicación entre tareas

Todos los tipos de datos pueden ser enviados entre 2 o más tareas con variables persistentes. Una variable persistente es global en todas las tareas. Para ello, la variable persistente deberá ser del mismo tipo y tamaño (dimensión de matriz) en todas las tareas que la declaran. De lo contrario, se producirá un error de tiempo de funcionamiento.

3.2 ANDROID STUDIO

3.2.1 Introducción

Android Studio es un entorno de desarrollo integrado (IDE), basado en IntelliJ IDEA de la compañía JetBrains. Este entorno de desarrollo integrado proporciona varias mejoras con respecto al plugin ADT (Android Developer Tools) para Eclipse, utilizado hasta el momento para desarrollar aplicaciones Android. El programa Android Studio utiliza una licencia de software libre Apache 2.0, está programado con un lenguaje de programación Java y está disponible para las plataformas Microsoft Windows, Mac OS X y GNU/Linux.

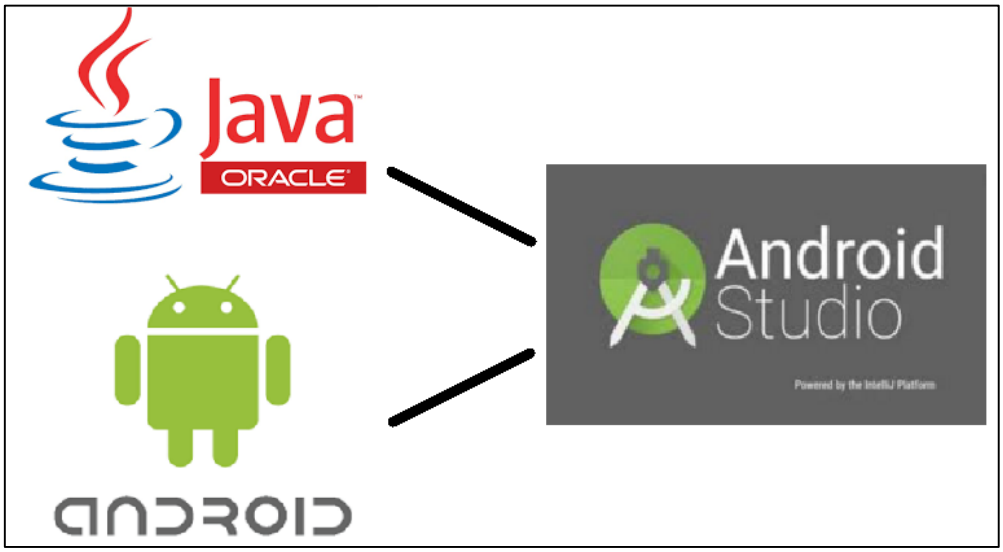


Figura 3.6: Unión Java + Android = Android Studio

El software Android Studio fue presentado por Google el 16 de mayo del 2013 en el congreso de desarrolladores Google I/O, con el objetivo de crear un entorno dedicado en exclusiva a la programación de aplicaciones para dispositivos Android, proporcionando a Google un mayor control sobre el proceso de producción. Se trata pues de una alternativa real a Eclipse, el IDE recomendado por Google hasta la fecha del lanzamiento de Android Studio, pero que presentaba problemas debido a su lentitud en el desarrollo de versiones que solucionaran las carencias actuales (es indispensable recordar que Eclipse es una plataforma de desarrollo, diseñada para ser extendida a través de plugins).

Android Studio se mantuvo hasta el 8 de diciembre de 2014 en su versión beta, momento en el cual se lanzó la versión estable de Android Studio 1.0. Desde la compañía Google recomendó desde ese momento al software Android Studio como el IDE para desarrollar aplicaciones para su sistema operativo, dejando el plugin ADT para Eclipse de estar en desarrollo activo.

Las características y herramientas que ofrece Android Studio son las siguientes:

- Herramientas Lint (detecta código no compatible entre arquitecturas diferentes o código confuso que no es capaz de controlar el compilador) para detectar problemas de rendimiento, usabilidad y compatibilidad de versiones.
- Utiliza ProGuard para optimizar y reducir el código del proyecto al exportar a APK (muy útil para dispositivos de gama baja con limitaciones de memoria interna).
- Integración de la herramienta Gradle encargada de gestionar y automatizar la construcción de proyectos, como pueden ser las tareas de testing, compilación o empaquetado.
- Nuevo diseño del editor con soporte para la edición de temas.
- Nueva interfaz específica para el desarrollo en Android.
- Permite la importación de proyectos realizados en el entorno Eclipse, que a diferencia de Android Studio (Gradle) utiliza ANT.
- Posibilita el control de versiones accediendo a un repositorio desde el que poder descargar Mercurial, Git, Github o Subversion.
- Alertas en tiempo real de errores sintácticos, compatibilidad o rendimiento antes de compilar la aplicación.
- Vista previa en diferentes dispositivos y resoluciones.
- Integración con Google Cloud Platform, para el acceso a los diferentes servicios que proporciona Google en la nube.
- Editor de diseño que muestra una vista previa de los cambios realizados directamente en el archivo xml.

Para una mayor comprensión de las diferencias y novedades que presenta Android Studio con respecto al anterior IDE Eclipse, y más concretamente con el ADT para Android, a continuación se enumeran las ventajas de utilizar Android Studio.

1. Android Studio ha pasado a ser el entorno recomendado para el desarrollo de aplicaciones en Android, al tratarse de un IDE oficial de Google en colaboración con JetBrains (compañía de desarrollo software especializada en diseño de IDEs).

2. Android Studio permite la creación de nuevos módulos dentro de un mismo proyecto, sin necesidad de estar cambiando de espacio de trabajo para el manejo de proyectos, algo habitual en Eclipse.

3. Con la simple descarga de Android Studio se disponen de todas las herramientas necesarias para el desarrollo de aplicaciones para la plataforma Android.

4. Su nueva forma de construir los paquetes .apk, mediante el uso de Gradle, proporciona una serie de ventajas más acorde a un proyecto Java:

    - Facilita la distribución de código, y por lo tanto el trabajo en equipo.
    - Reutilización de código y recursos.
    - Permite compilar desde línea de comandos, para aquellas situaciones en las que no esté disponible un entorno de desarrollo.
    - Mayor facilidad para la creación de diferentes versiones de la misma aplicación, que proporciona numerosas ventajas como puede ser la creación de una versión de pago y otra gratuita, o por ejemplo diferentes dispositivos o almacén de datos.

3.2.2 Android

3.2.2.1 Introducción e historia

El sistema operativo Android está basado en el núcleo Linux y fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tablets o tabléfonos; y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, compró.

Finalmente, Android fue presentado en 2007 junto la fundación del Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles. Google liberó la mayoría del código de Android bajo la licencia Apache, una licencia libre y de código abierto. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008.

El 25 de junio de 2014 en la Conferencia de Desarrolladores Google I/O, Google mostró una evolución de la marca Android, con el fin de unificar tanto el hardware como el software y ampliar mercados. En la actualidad, según un estudio de la compañía Gartner, los dispositivos Android son los más dominantes, ocupando un 82.2% de la cuota de mercado, seguido por Apple con un 14.6% y con Microsoft con un 2.5%.

Worldwide Smartphone Sales to End Users by Operating System in 2Q15 (Thousands of Units)

Operating System	2Q15 Units	2Q15 Market Share (%)	2Q14 Units	2Q14 Market Share (%)
Android	271,010	82.2	243,484	83.8
iOS	48,086	14.6	35,345	12.2
Windows	8,198	2.5	8,095	2.8
BlackBerry	1,153	0.3	2,044	0.7
Others	1,229.0	0.4	1,416.8	0.5
Total	**329,676.4**	**100.0**	**290,384.4**	**100.0**

Source: Gartner (August 2015)

Figura 3.7: Ventas mundiales de Smartphones por S.O.

Android tiene una gran comunidad de desarrolladores creando aplicaciones para extender la funcionalidad de los dispositivos. En la actualidad, se ha llegado ya a 1.000.000 de aplicaciones disponibles para la tienda de aplicaciones oficial de Android "Google Play", sin tener en cuenta aplicaciones de otras tiendas no oficiales para Android como la tienda de aplicaciones "Samsung Apps" de Samsung, "Slideme" de java y "Amazon Appstore". Los programas de las aplicaciones Android están escritos en el lenguaje de programación Java.

3.2.2.2 Arquitectura del sistema operativo Android

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un *framework Java* de aplicaciones orientadas a objetos sobre el núcleo de bibliotecas de Java en una máquina virtual *Dalvik* con compilación en tiempo de ejecución. Las bibliotecas escritas en lenguaje C incluyen un administrador de interfaz gráfica (Surface manager), un *framework OpenCore,* una base de datos relacional *SQLite,* una interfaz de programación de API gráfica *OpenGL ES 2.0 3D,* un motor de renderizado *WebKit, un motor gráfico SGL, SSL* y una biblioteca estándar de *C Bionic*. El sistema operativo está compuesto por 12 millones de líneas de código, incluyendo 3 millones de líneas de XML, 2,8 millones de líneas de lenguaje C, 2,1 millones de líneas de Java y 1,75 millones de líneas de C++.

En la figura 3.7 se puede observar un gráfico de la arquitectura empleada en Android. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores. Una de las características más importantes es que todas las capas están basadas en “software libre”.

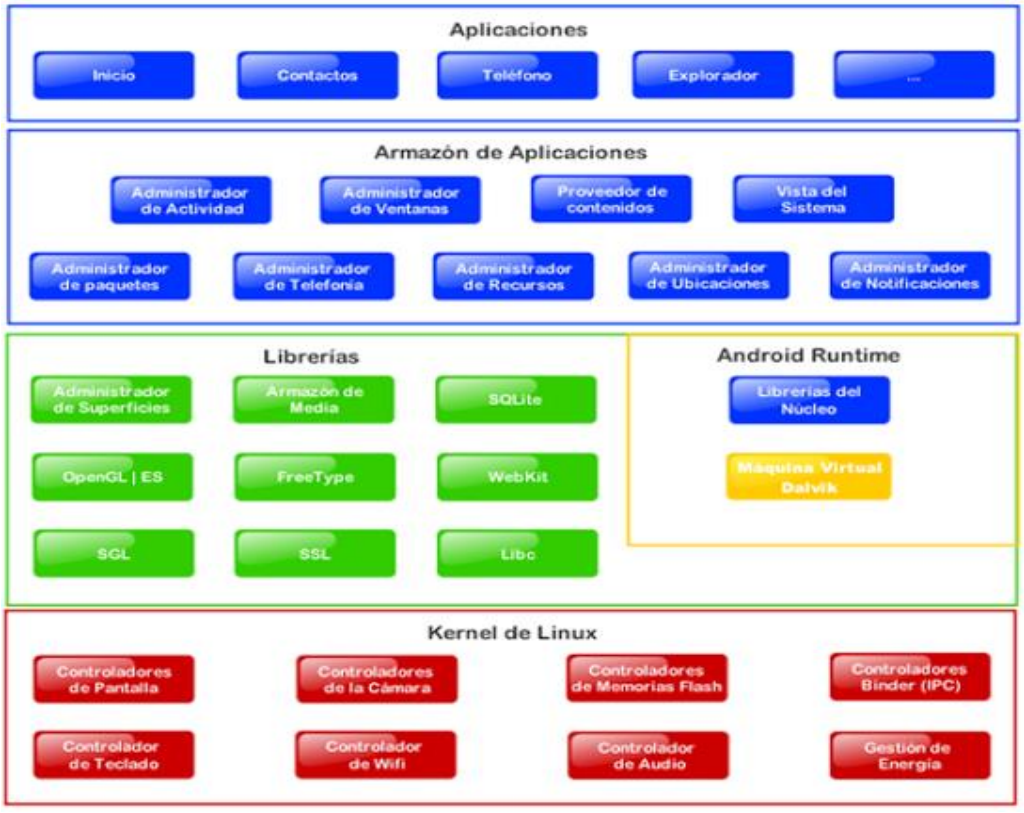


Figura 3.8: Arquitectura del sistema Android

a) El núcleo Linux

El núcleo de Android está formado por el sistema operativo Linux versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de drivers para dispositivos.

Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del hardware.

b) Runtime de Android

Está basado en el concepto de máquina virtual utilizado en Java. Dado las limitaciones de los dispositivos donde ha de correr Android (poca memoria y procesador limitado) no es posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual Dalvik, que respondiera mejor a estas limitaciones.

Algunas características de la máquina virtual Dalvik que facilitan esta optimización de recursos son: que ejecuta ficheros Dalvik ejecutables (.dex), un formato optimizado para ahorrar memoria. Además, está basada en registros y cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik, delegando al kernel de Linux algunas funciones como threading y el manejo de la memoria a bajo nivel.

A partir de Android 5.0 se reemplaza Dalvik por ART, que es una nueva máquina virtual que consigue reducir el tiempo de ejecución del código Java hasta en un 33%. También se incluye en el Runtime de Android el "core libraries" con la mayoría de las librerías disponibles en el lenguaje Java.

c) Librerías nativas

Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compiladas en código nativo del procesador y muchas de las librerías utilizan proyectos de código abierto. Algunas de estas librerías son:

- System C library: una derivación de la librería BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.
- Media Framework: librería basada en PacketVideo's OpenCORE; soporta códecs de reproducción y grabación de multitud de formatos de audio vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- Surface Manager: maneja el acceso al subsistema de representación gráfica en 2D y 3D.

- WebKit: soporta un moderno navegador Web utilizado en el navegador Android y en la vista Webview. Se trata de la misma librería que utiliza Google Chrome y Safari de Apple.
- Librerías 3D: implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador harware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- FreeType: fuentes en bitmap y renderizado vectorial.
- SQLite: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- SSL: proporciona servicios de encriptación Secure Socket Layer (capa de conexión segura).

d) Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, etc.).

Esta capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Este mismo mecanismo permite a los usuarios reemplazar componentes.

Los servicios más importantes que incluye son:

- Views: extenso conjunto de vistas, (parte visual de los componentes).
- Resource Manager: proporciona acceso a recursos que no son en código.
- Activity Manager: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
- Notification Manager: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- Content Providers: mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

e) Aplicaciones

Este nivel está formado por el conjunto de aplicaciones instaladas en una máquina Android. Todas las aplicaciones han de correr en la máquina virtual Dalvik para garantizar la seguridad del sistema. Normalmente las aplicaciones Android están escritas en Java. Para desarrollar aplicaciones en Java podemos utilizar el Android SDK.

3.2.3 Java

La tecnología Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. El lenguaje de programación Java es de propósito general, concurrente, orientado a objetos y fue diseñado específicamente para tener las mínimas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo ("write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Para conseguir la portabilidad de los programas Java se utiliza el entorno de ejecución Java Runtime Environment (JRE) disponible para los principales sistemas operativos. Esto asegura que el mismo programa Java se pueda ejecutar en Windows, MAC OS, Linux, Solaris (ver Figura 3.9).

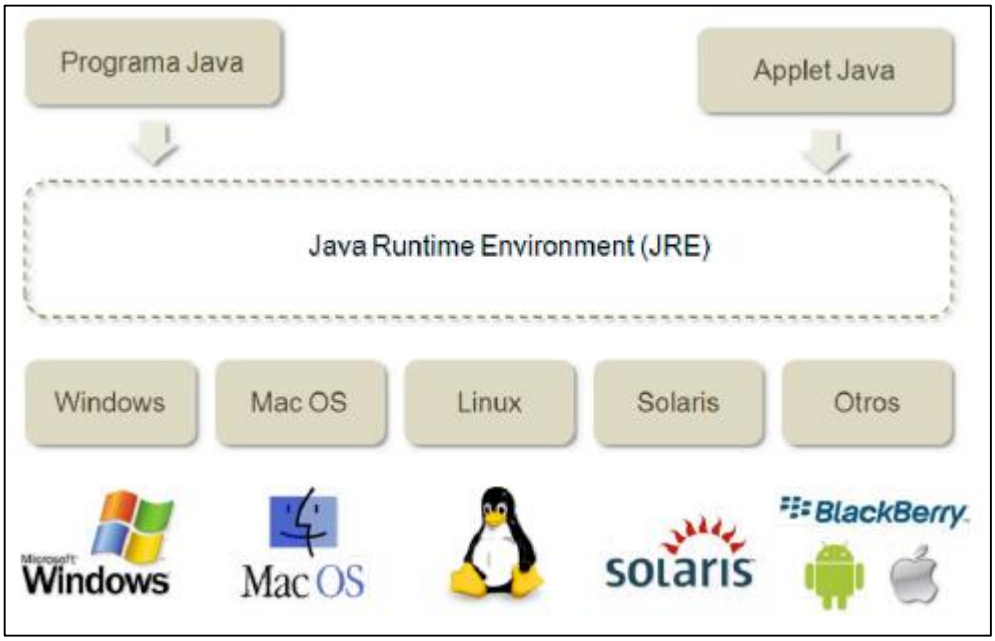


Figura 3.9: Sistemas operativos compatibles con JRE

Esta tecnología es la base para prácticamente todos los tipos de aplicaciones de red, además del estándar global para desarrollar y distribuir aplicaciones móviles y embebidas, juegos, contenido basado en web y software de empresa.

Con más de 9 millones de desarrolladores en todo el mundo, Java permite desarrollar, implementar y utilizar de forma eficaz interesantes aplicaciones y servicios.

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

El lenguaje Java se creó con cinco objetivos principales:

- Debería usar el paradigma de la programación orientada a objetos.
- Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
- Debería incluir por defecto soporte para trabajo en red.
- Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
- Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

La tecnología Java, en la actualidad se utiliza en la mayoría de los equipos. A continuación se enumeran algunos datos indicativos de lo presente que está Java en la actualidad en los equipos.

- El 97% de los escritorios empresariales ejecutan Java
- El 89% de los escritorios (o computadoras) en Estados Unidos ejecutan Java
- 9 millones de desarrolladores de Java en todo el mundo
- La primera opción para los desarrolladores
- La primera plataforma de desarrollo
- 3000 millones de teléfonos móviles ejecutan Java
- El 100% de los reproductores de Blu-ray incluyen Java
- 5000 millones de Java Cards en uso
- 125 millones de dispositivos de televisión ejecutan Java
- Los 5 principales fabricantes de equipos originales utilizan Java.

3.2.4 Interfaz gráfica Android Studio

El aspecto general de la ventana principal del programa Android Studio se puede observar en la Figura 3.10.

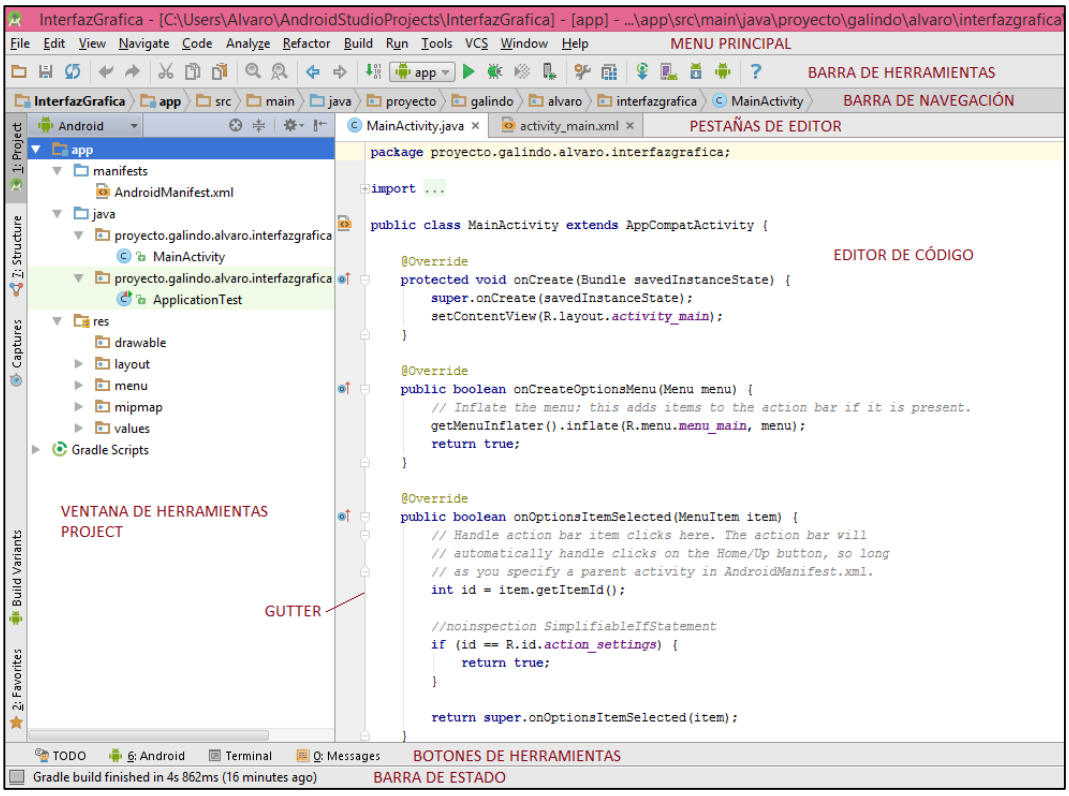


Figura 3.10: Interfaz gráfica Android Studio

Los diferentes elementos de la interfaz gráfica se describirán a continuación y son:

- Menú Principal
- Barra de herramientas
- Barra de navegación
- Pestañas del editor
- Ventana de herramientas Project
- Editor de código y gutter

3.2.4.1 Barra Menú Principal

La barra de menú principal como se observa en la figura 3.10, se encuentra en la parte más alta de la accesibilidad de Android Studio (ver Figura 3.11).

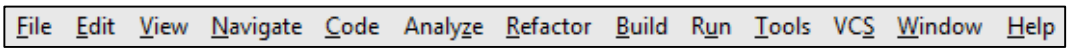


Figura 3.11: Barra Menú Principal Android Studio

Con esta barra podemos realizar todas aquellas acciones que puedan existir dentro del entorno de desarrollo, tales como abrir, crear proyectos, refactorizar código, correr y depurar aplicaciones, controlar la versión de los archivos, integrar herramientas, etc. Esta barra no puede ocultarse a diferencia de las demás barras o ventanas que tenemos en la interfaz de usuario.

3.2.4.2 Barra de herramientas

La barra de herramientas contiene botones con las acciones usadas más frecuentes para tener un acceso más rápido (ver Figura 3.12). Se puede ocultar o hacer visible la barra de herramientas desde el menú principal en la pestaña View > Toolbar.

Figura 3.12: Barra de herramientas Android Studio

Desde esta barra se puede acceder a acciones de administración de archivos como abrir, copiar, pegar, cortar, deshacer y rehacer; o a acciones para correr rápidamente las aplicaciones, iniciar el SDK, emuladores, buscar elementos, etc.

3.2.4.3 Barra de navegación

La barra de navegación muestra la ruta en la que se encuentra alojado el fichero que ha sido abierto en el editor de código. La profundidad de los directorios se expresa de izquierda a derecha (ver Figura 3.13).

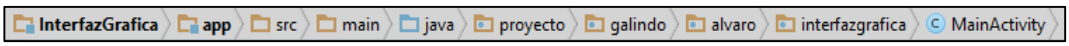


Figura 3.13: Barra navegación Android Studio

Alternativamente si seleccionas un segmento de la ruta en la barra de navegación se abrirá un menú contextual para mostrar de forma vertical las ramificaciones existentes. La barra de navegación se puede mostrar u ocultar desde el menú principal en la pestaña View > Navigation Bar.

3.2.4.4 Pestañas del editor

Al abrir un proyecto nuevo en Android Studio se nos crean por defecto dos pestañas, una para “MainActivity.java” y otra para “actividad_main” (ver Figura 3.14).

Figura 3.14: Pestañas del editor Android Studio

Si quieres navegar entre una pestaña y otra puedes usar la combinación “Alt + (Izquierda)” o “Alt + (Derecha), siempre y cuando las tabs del editor el foco seleccionado. Existen gran cantidad de acciones que podemos realizar con las pestañas, estando disponibles desde el menú principal en la ruta Window>Editor Tabs.

3.2.4.5 Ventana de herramientas Project

La ventana de herramientas “Project” es la ventana más intuitiva y la más utilizada en cualquier IDE. Su objetivo es mostrar el alcance global de todos los archivos del proyecto a través de una vista de árbol o jerarquía.

En esta ventana se puede observar el código fuente, recursos, librerías, archivos de construcción y directorios de forma segmentada para tener un manejo rápido y flexible de tus acciones (ver Figura 3.15).

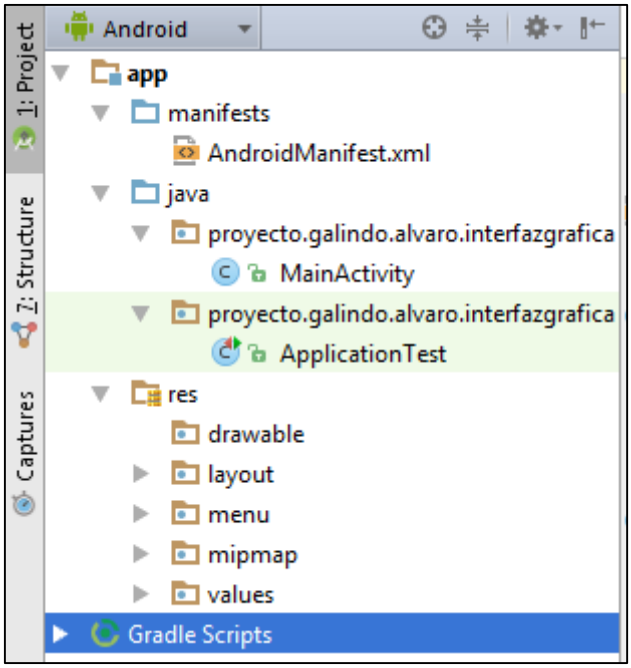


Figura 3.15: Ventana herramientas Project Android Studio

Analizando el módulo por defecto "app" que aparece en la ventana de herramientas, vemos como presenta tres subdirectorios. El primero es "manifests", el cual contiene el archivo "AndroidManifest.xml" del proyecto, que es un archivo XML que contiene todas las características de una aplicación Android, tales como los building blocks existentes, la versión de SDK usada, los permisos necesarios para ejecutar algunos servicios, etc. Luego está el subdirectorio "java" para los archivos de programación que se utilizan en la aplicación con lenguaje Java. El tercer subdirectorio es "res", que son los recursos (archivos o datos externos) que soportan el comportamiento de nuestra aplicación Android.

En la parte inferior tendremos otra subdivisión llamada "Gradle Scripts", donde están todos los archivos asociados al sistema de construcción.

3.2.4.6 Editor de código y gutter

El editor de código es la ventana con tabs que se ve en la parte central de interfaz gráfica de Android Studio (ver Figura 3.16). Prácticamente todo el proyecto depende de la programación que se encuentra en este espacio.

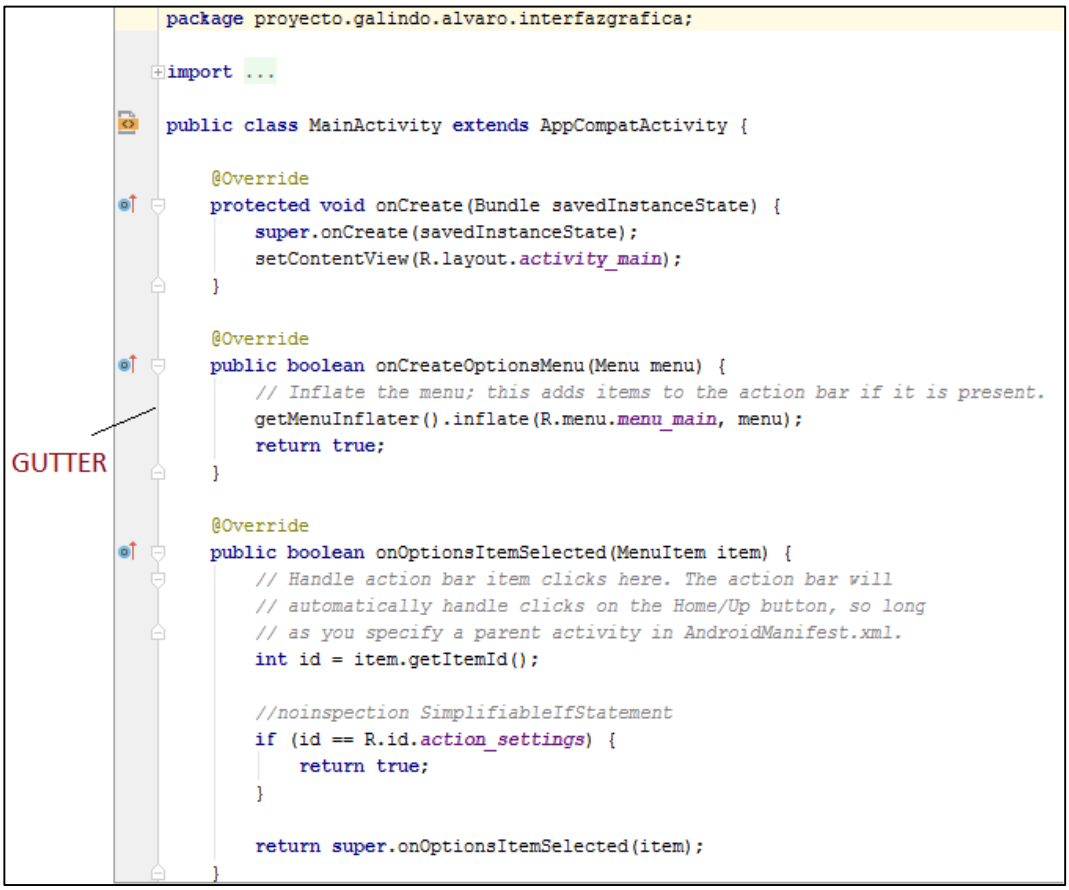


Figura 3.16: Editor de código Android Studio

La sección lateral izquierda "Gutter" del editor de código, se utiliza para mostrar información sobre los recursos y controles asociados al código. Una de sus funciones es mostrarnos una vista rápida de los iconos y colores que tenemos asociados al código. Además de ayudar a previsualizar un icono utilizado por la aplicación, también se puede acceder directamente al recurso con tan solo clickear la imagen. Otra opción que podemos utilizar es usar la barra para expandir o contraer bloques de códigos contenidos en sentencias de apertura y cierre.

3.2.5 Introducción lenguaje programación Java

En esta sección trataré de explicar de una manera sencilla y clara los principios de la programación Java.

3.2.5.1 Identificadores y palabras reservadas

En Java los identificadores comienzan por una letra del alfabeto inglés, un subrayado "_" o el símbolo de dólar "$". Los siguientes caracteres del identificador pueden ser letras o dígitos. El lenguaje de programación Java cuenta con un conjunto de palabras reservadas (ver Figura 3.17), por lo que ningún identificador puede llevar el nombre de una palabra reservada.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Figura 3.17: Palabras reservadas lenguaje programación Java

3.2.5.2 Tipos de datos primitivos

Los tipos de datos primitivos utilizados por el lenguaje de programación Java se pueden observar en la Figura 3.18.

Tipo	Definición
boolean	`true o false`
char	Carácter Unicode de 16 bits
byte	Entero en complemento a dos con signo de 8 bits
short	Entero en complemento a dos con signo de 16 bits
int	Entero en complemento a dos con signo de 32 bits
long	Entero en complemento a dos con signo de 64 bits
float	Real en punto flotante según la norma IEEE 754 de 32 bits
double	Real en punto flotante según la norma IEEE 754 de 64 bits

Figura 3.18: Tipos de datos primitivos Java

Los datos primitivos en Java, al contrario que en C o C++, el tamaño no depende del sistema operativo o de la arquitectura. En todas las arquitecturas y bajo todos los sistemas operativos el tamaño en memoria es el mismo.

3.2.5.3 Estructuras de control en Java

Las estructuras de control en Java presentan escasas diferencias con respecto a C/C++, no obstante existen diferencias. Se llama programación estructurada al uso correcto de las estructuras de control, que se resume en que toda estructura de control debe tener un único punto de entrada y un único punto de salida. En el lenguaje de programación Java tenemos tres tipos de estructuras de control:

- Estructuras condicionales
- Estructuras de repetición
- Uso “break” y “continue"

3.2.5.3.1 Estructuras condicionales

Las estructuras condicionales pueden ser de bifurcación o de selección.

- **Bifurcación “if-else” e “if-else if”:**

En esta estructura de control es necesario que la condición sea una variable o expresión booleana. Si sólo existe una instrucción en el bloque, las llaves no son necesarias. No es necesario que existe un bloque else. La sintaxis es la siguiente:

```
if(condicion) {
        instruccion1();
        instruccion2();
        // etc
} else {
        instruccion1();
        instruccion2();
        // etc
}
```

- **Selección múltiple "switch":**

La expresión de esta estructura de control ha de ser una variable de tipo entero o una expresión de tipo entero. Cuando se encuentra coincidencia con un case se ejecutan las instrucciones a él asociadas hasta encontrar el primer break. Si no se encuentra ninguna coincidencia se ejecutan las instrucciones en default. La sintaxis del "switch" es la siguiente:

```
switch (expresion) {
   case valor1:
            instrucciones();
            break;
   case valor2:
            instrucciones();
            break;
   default:
   instrucciones();
   }
```

3.2.5.3.2 Estructuras de repetición

Las estructuras de repetición en el lenguaje de programación Java, al igual que en C y C++, pueden ser de dos tipos: repetición sobre un rango determinado o repetición condicional.

- **Repetición sobre un rango determinado "for":**

En la parte de iniciación se puede declarar una variable de control del bucle cuyo ámbito será el bucle. Tanto en la parte de iniciación como de incremento se puede incluir varias expresiones separadas por comas, pero nunca en la parte de condición. La condición ha de ser una variable booleana o una expresión que se evalúe a un valor booleano. No es necesario que la condición se base exclusivamente en la variable de control del bucle. La sintaxis del bucle "for" es la siguiente:

```
for(iniciación; condición; incremento) {
   // Bloque de instrucciones
}
```

- **Repetición condicional "while" y "do-while"**

La estructura de control "while" evalúa la condición antes de ejecutar el bloque de la estructura; en la estructura de control "do...while" se evalúa la condición después de la ejecución del bloque. Igual que en el caso del for la condición ha de ser una variable booleana o una expresión que se evalúe a un valor booleano. La sintaxis de ambos bucles es la siguiente:

```
while(condición) {
   // Bloque de instrucciones
}
do {
   // Bloque de instrucciones
} while(condición);
```

3.2.5.3.3 Uso "break" y "continue"

La palabra reservada "break" además de para indicar el fin del bloque de instrucciones en una instrucción de selección múltiple "switch", sirve para forzar la salida del bloque de una estructura de repetición. La palabra reservada "continue", dentro del bloque de una estructura de repetición condicional, sirve para forzar la evaluación de la condición.

3.2.5.4 Clases en Java

Java es un lenguaje de programación orientado a objetos (POO). Todo en Java, con la excepción de los tipos primitivos, es un objeto. La programación orientada a objetos (POO) abstrae las entidades del mundo real como objetos y las relaciones entre ellos como paso de mensajes. Los objetos son instancias o ejemplares de una clase o plantilla y poseen como características atributos (valores) y métodos (acciones).

Para declarar una clase debes usar la palabra reservada “class” y luego definir un bloque de código que comprenda sus atributos y métodos. En la Figura 3.19 podemos apreciar la estructura que sigue una clase con un ejemplo que describe a un cliente.

```
public class Cliente {
    /*
    Atributos
     */
    private int idCliente;
    private String nombre;
    private String telefono;
    private float saldo;

    // Método que modifica al campo saldo
    public void acumularSaldo(float excedente) {
        saldo += excedente;
    }
}
```

Figura 3.19: Ejemplo clase en Java

Se puede restringir el alcance de la clase con modificadores de acceso, que determinan el alcance de atributos y métodos hacia otras clases. Existen 4 posibilidades de implementación.

- **public:** Permite que una pieza de código sea visible en todos los niveles.
- **protected**: Con este modificador solo las clases que se encuentren en el mismo paquete pueden ver y acceder a este código.
- **sin modificador**: Permite que solo las clases del paquete accedan al código. A diferencia de “protected”, este modificador no permite que las subclases tengan privilegios del código de su superclase.
- **private**: Solo la clase donde está contenido el código, puede tener acceso.

3.2.5.5 Crear objetos en Java

Una vez tienes tu clase definida, ya es posible instanciar objetos que comiencen a contribuir a la aplicación. La creación de un nuevo objeto se lleva a cabo del operador "new". Por ejemplo, crear un nuevo objeto de la clase Cliente utilizada como ejemplo de clase en el apartado anterior sería:

Cliente clienteActual = new Cliente();

Si deseas personalizar la inicialización de los objetos de una clase, se puede declarar una variación del constructor por defecto. Un constructor es un mecanismo para crear tus objetos y tienen casi la misma forma de un método. La diferencia está en que no tiene tipo de retorno especificado y su nombre es el mismo de la clase.

En la Figura 3.20 se puede apreciar un constructor que permite asignar valores a todos los campos del cliente.

```
public class Cliente {
    /*
    Atributos
     */
    private int idCliente;
    private String nombre;
    private String telefono;
    private float saldo;

    public Cliente(int idCliente, String nombre, String telefono, float saldo) {
        this.idCliente = idCliente;
        this.nombre = nombre;
        this.telefono = telefono;
        this.saldo = saldo;
    }

    // Método que modifica al campo saldo
    public void acumularSaldo(float excedente) {
        saldo += excedente;
    }
}
```

Figura 3.20: Ejemplo constructor Java

3.2.6 Manejo de hilos en Android Studio

Un hilo es una unidad de ejecución asociada a una aplicación y es la estructura de la programación concurrente, la cual tiene como objetivo dar la percepción al usuario que el sistema que ejecuta realiza múltiples tareas a la vez.

Aunque los hilos se benefician de las tecnologías multinúcleos y multiprocesamiento, no significa que una arquitectura simplista no se beneficie de la creación de hilos.

Cuando se construye una aplicación Android, todos los componentes y tareas son introducidos en el hilo principal o hilo de UI (UI Thread). Si las operaciones introducidas en el hilo principal son rápidas no se encontrará ningún problema; pero normalmente en los proyectos reales necesitamos realizar procesos más costosos, y por lo tanto el hilo principal, encargado de mostrar la interfaz de usuario, quedaría bloqueado, con la consiguiente lentitud de cara al usuario. Es aquí donde interviene el sistema operativo Android, monitorizando todos los procesos que están en ejecución, y forzando a salir de la aplicación para aquellos que superen los 5 segundos.

Para evitar este tipo de situaciones, Android proporciona una serie de clases, que permiten trabajar en segundo plano, para aquellas operaciones que necesiten un mayor tiempo para ser procesadas, que son:

- AsyncTask
- Thread
- Handler

3.2.6.1 AsyncTask

La clase “AsyncTask” permite comunicarse con el subproceso del hilo de interfaz de usuario hilo principal. Para ello realiza operaciones en segundo plano, mostrando los resultados en subprocesos de la interfaz de usuario (ver Figura 3.21).

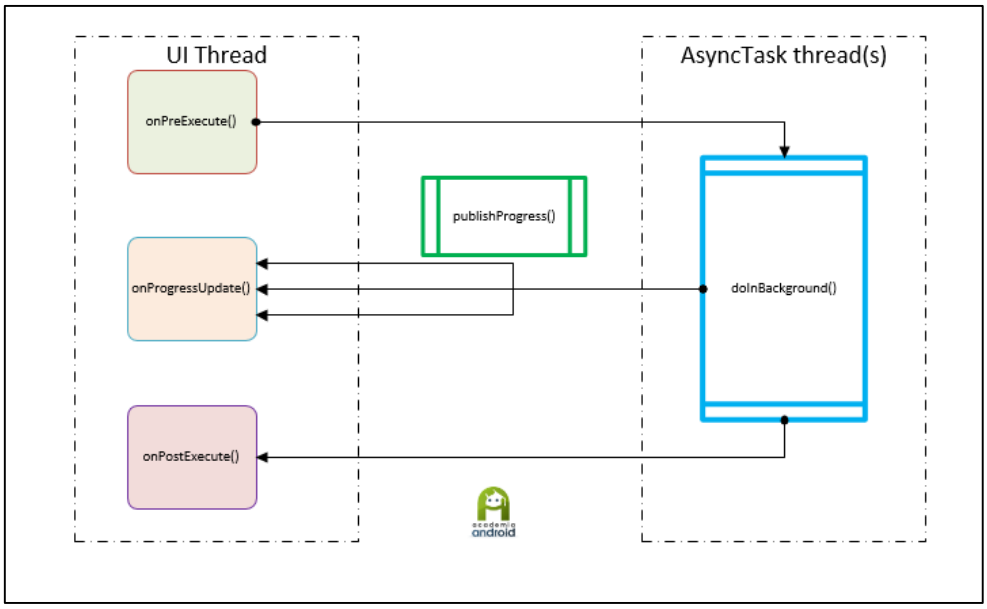


Figura 3.21: Ciclo de ejecución clase AsyncTask

Las características de la clase "AsyncTask" son:

- Proporciona un mayor control y orden en la ejecución de nuestros procesos en segundo plano.
- Marco de trabajo para operaciones no muy costosas.
- A partir de la versión Android 3.0, las tareas se ejecutan en un sólo hilo, para evitar errores derivados de ejecuciones en paralelo.
- Posee métodos que permiten coordinar la ejecución de las tareas asíncronas que se quieran ubicar en segundo:

    - onPreExecute(): En este método van todas aquellas instrucciones que se ejecutarán antes de iniciar la tarea en segundo plano. Normalmente es la inicialización de variables, objetos y la preparación de componentes de la interfaz.
    - doInBackground(Parámetros...): Recibe los parámetros de entrada para ejecutar las instrucciones específicas que irán en segundo plano, después de terminar el método onPreExecute(). Dentro de él podemos invocar un método auxiliar llamado publishProgress(), el cual transmitirá unidades de progreso al hilo principal. Estas unidades miden cuanto tiempo falta para terminar la tarea, de acuerdo a la velocidad y prioridad que se está ejecutando.

- onProgressUpdate(Progreso...): Este método se ejecuta en el hilo de UI después de llamar al método publishProgress(). Su ejecución se prolongará lo necesario hasta que la tarea en segundo plano haya sido terminada. Recibe las unidades de progreso, así que podemos usar algún View para mostrarlas al usuario para que este sea consciente de la cantidad de tiempo que debe esperar.
- onPostExecute(Resultados...): En este método se publican todos los resultados retornados por doInBackground() hacia el hilo principal.
- onCancelled(): Ejecuta las instrucciones que el usuario desee realizar al cancelar la tarea asíncrona.

3.2.6.2 Thread

La clase "Thread" proporciona su propia unidad concurrente de ejecución, y se puede definir como la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo. En la Figura 3.22 podemos observar el ciclo de vida de un "Thread" Una de sus principales características es permitir a una aplicación realizar varias tareas de manera simultánea. Define sus propios argumentos, variables y pila de llamada a métodos.

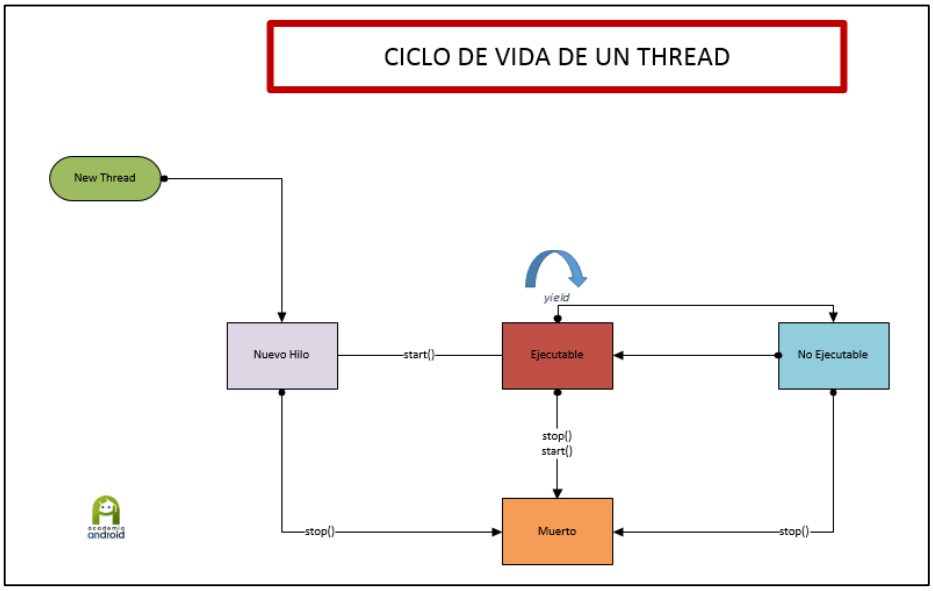


Figura 3.22: Ciclo de vida de un Thread

Las características de la clase “Thread” son:

- Ejecución de tareas en paralelo o de manera concurrente.
- Los hilos de ejecución comparten el espacio de memoria.
- El conjunto de hilos en ejecución que comparten los mismos recursos es conocido como un proceso.
- Cualquier modificación de un dato en memoria, por parte de un hilo, permite el acceso al dato modificado para el resto de hilos de manera inmediata.
- Cada hilo presenta de manera propia el contador de programa, la pila de ejecución y el estado de la CPU.
- Un proceso seguirá en ejecución si al menos uno de sus hilos de ejecución sigue activo. Si un proceso finaliza, todos sus hilos de ejecución también lo harán.

3.2.6.3 Handler

La clase Handler es aquella que permite manejar y procesar mensajes, proporcionando un mecanismo para su envío (como si se tratara de un puente) entre threads o hilos, y así poder enviar mensajes desde nuestro hilo secundario al UIThread o hilo principal. En la Figura 3.23 se puede observar el ciclo de ejecución de la clase Handler.

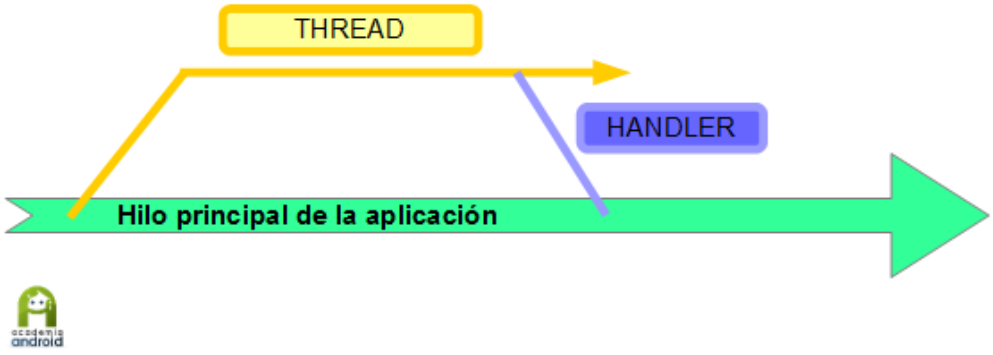


Figura 3.23: Ciclo de ejecución clase Handler

Las características de la clase "Handler" son:

- Permiten poner en cola una acción que se realiza en un subproceso distinto al suyo.
- Cada instancia de la clase Handler está asociada a un sólo hilo y a la cola de mensajes de este.
- Comunicación del hilo secundario con el hilo principal o UIThread a través de un controlador.
- Proporciona varios métodos para poner en cola un mensaje en la parte delantera o enviar al final de la cola después de un determinado tiempo:

    - sendMessageAtFrontOfQueue (Message msg)
    - sendMessageAtTime (Message msg, long uptimeMillis).

3.3 COMUNICACIÓN INALAMBRICA

3.3.1 Introducción

La historia del origen y desarrollo de la tecnología inalámbrica se remonta al año 1880 cuando Alexander Graham Bell y Charles Summer Tainter inventaron el fotófono, el primer aparato de transmisión de sonido mediante luz sin necesidad de utilizar cables, tan solo 8 años después el físico alemán Rudolf Hertz utilizó ondas de radio para realizar la primera comunicación inalámbrica.

La comunicación inalámbrica o sin cables es aquella en la que la comunicación (emisor/receptor) no se encuentra unida por un medio de propagación físico, sino que se utiliza la modulación de ondas electromagnéticas a través del espacio. Las transmisiones inalámbricas constituyen una eficaz herramienta que permite la transferencia de voz, datos y vídeo sin la necesidad de cableado

La comunicación que se realiza a través de ondas de radiofrecuencia, facilita la operación en lugares donde un dispositivo no se encuentra en una ubicación fija (almacenes, oficinas de varios pisos, etc.). Actualmente las redes cableadas presentan ventaja en cuanto a transmisión de datos sobre las inalámbricas. Mientras que las redes cableadas proporcionan velocidades de hasta 1 Gbit/s (Red Gigabit), las inalámbricas alcanzan sólo hasta 600 Mbit/s. (Wi-Fi N).

3.3.2 Tecnología Wi-Fi

La tecnología Wi-Fi es utilizada para conectar e intercambiar información entre dispositivos electrónicos sin necesidad de conectarlos mediante el uso de cables físicos. Wi-Fi pertenece al conjunto de tecnologías conocidas como Wireless (sin cables) con mayor aceptación y uso en la mayoría de dispositivos electrónicos (smartphones, tablets, ordenadores de sobremesa, portátiles, etc.) gracias al cual podemos disponer de una red de comunicación entre varios dispositivos y con acceso a Internet.

Las bases del Wi-Fi actual datan del año 1985 cuando la comisión de comunicaciones de los Estados Unidos estableció las características que debían tener una red inalámbrica, asignando las frecuencias en las que trabaja esta tecnología conocidas como bandas ISM (Industrial, Scientific, Medical) destinadas al uso en redes inalámbricas en el campo industrial, científico y médico.

En 1991 las empresas norteamericanas AT&T y NCR desarrollaron las bases del estándar 802.11 que establece la normativa en la comunicación inalámbrica, en esta época las velocidades de transmisión eran realmente bajas del orden de 5 Mb/s hasta que en 1993 el Ingeniero Jhon O´Sullivan desarrolló una tecnología para el sector astrofísico que fue implementada en las redes inalámbricas permitiendo alcanzar velocidades de transmisión eficientes.

En 1997 se lanza el estándar 802.11 por parte del IEEE (Instituto de ingenieros eléctricos y electrónicos). Posteriormente en el año 1999 la empresa finlandesa Nokia junto con la empresa americana Symbol Technologies y la empresa Intersil, crean la asociación sin ánimo de lucro WECA (Wireless Ethernet Compatibility Alliance), con la finalidad de fomentar el desarrollo de dispositivos electrónicos que sean compatibles con el estándar IEEE 802.11. En el año 2003 se rebautizó con el nombre Wi-Fi Alliance.

De esta forma, en abril de 2000 WECA certifica la interoperabilidad de equipos según la norma IEEE 802.11b, bajo la marca Wi-Fi. Esto quiere decir que el usuario tiene la garantía de que todos los equipos que tengan el sello Wi-Fi pueden trabajar juntos sin problemas, independientemente del fabricante de cada uno de ellos.

La familia de estándares 802.11 ha ido naturalmente evolucionando desde su creación, mejorando el rango y velocidad de la transferencia de información, su seguridad, entre otras cosas. En la figura 3.24 se pueden apreciar la evolución de los estándares IEEE 802.11.

Estándar	Nombres comerciales	Año	Velocidad máxima teórica
IEEE 802.11	-	1997	2 Mbps
IEEE 802.11a	802.11A	1999	54 Mbps
IEEE 802.11b	802.11B, Wi-Fi B	1999	11 Mbps
IEEE 802.11g	802.11G, Wi-Fi G	2003	54 Mbps
IEEE 802.11n	802.11N, Wi-Fi N	2009	600 Mbps
IEE 802.11ac	802.11AC, Wi-Fi AC	2014	1.3 Gbps (*)

Figura 3.24: Evolución estándares IEEE 802.11

3.4 COMUNICACIÓN POR SOCKETS

Los sockets (también llamados conectores) son un mecanismo de comunicación entre procesos que permiten la comunicación bidireccional tanto entre procesos que se ejecutan en una misma máquina como entre procesos lanzados en diferentes máquinas. Cuando los procesos están en máquinas distintas la comunicación se lleva a cabo a través de redes de ordenadores. Para lograr esta comunicación se utilizan los protocolos de comunicación TCP y UDP.

El protocolo TCP (Transmission Control Protocol) establece un conducto de comunicación punto a punto entre dos computadoras, es decir, cuando se requiere la transmisión de un flujo de datos entre dos equipos, el protocolo TCP establece un conducto exclusivo entre dichos equipos por el cual los datos serán transmitidos y este perdurará hasta que la transmisión haya finalizado, gracias a esto TCP garantiza que los datos enviados de un extremo de la conexión lleguen al otro extremo y en el mismo orden en que fueron enviados. Las características que posee TCP hacen que el protocolo sea conocido como un protocolo orientado a conexión.

También existe un protocolo no orientado a la conexión llamado UDP. El protocolo UDP no es orientado a la conexión debido a que la forma de trasmitir los datos no garantiza en primera instancia su llegada al destino, e incluso si este llegara al destino final, tampoco garantiza la integridad de los datos. El protocolo UDP hace la transmisión de los datos sin establecer un conducto de comunicación exclusiva como lo hace TCP, además utiliza datagramas, los cuales contienen una porción de la información y que son enviados a la red en espera de ser capturados por el equipo destino. Cuando el destino captura los datagramas debe reconstruir la información, para esto debe ordenar la información que recibe ya que la información transmitida no viene con un orden específico, además se debe tener conciencia de que no toda la información va a llegar.

La comunicación entre procesos a través de sockets se basa en la filosofía CLIENTE-SERVIDOR. El programa servidor comienza a "escuchar" en un puerto determinado, y posteriormente el programa "cliente" debe conocer la dirección IP del servidor y el puerto que está escuchando, al saber esto simplemente solicita establecer una conexión con el servidor. El servidor acepta esa conexión y se puede decir que estos programas están "conectados", de este modo pueden intercambiar información. En la figura 3.25 se puede observar el diagrama de flujo básico para una comunicación orientada a la conexión por sockets entre un servidor y un cliente.

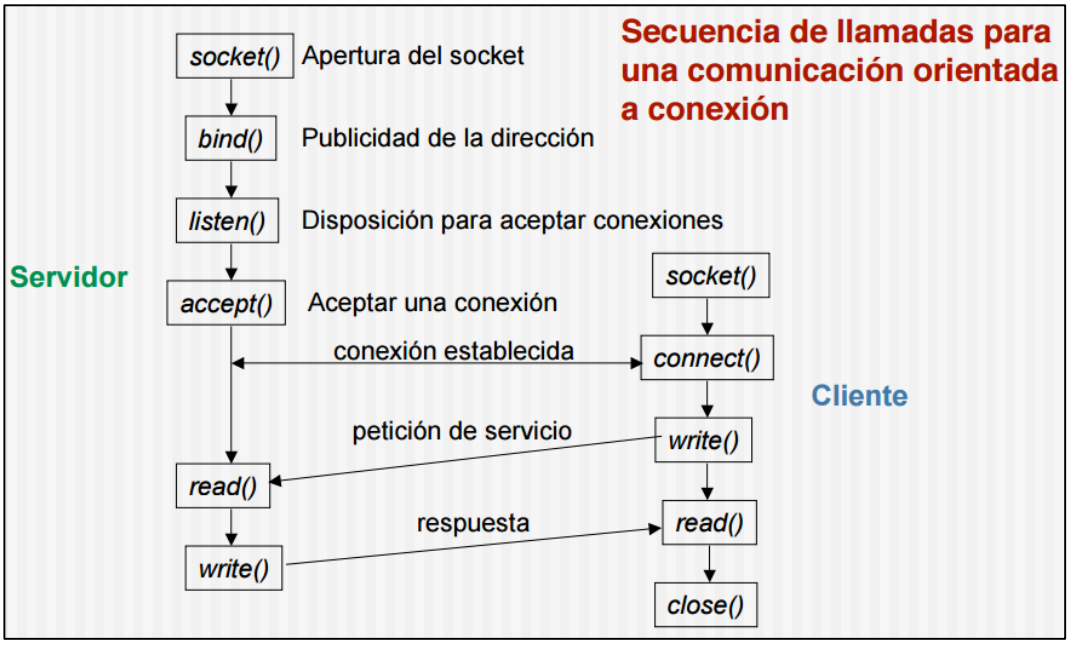


Figura 3.25: Comunicación cliente-servidor por Sockets

En el caso particular del presente proyecto, el servidor es el programa creado en el robot que se encuentra en una red y tiene asignada una IP local y un punto de acceso Wi-Fi, mientras que el cliente es el programa instalado en Android.

Una vez creado el socket en el servidor, y el cliente ha enlazado su socket con el del servidor, es posible el envío de datos entre ambos programas. El programa de RAPID interpreta los datos enviados por Android para realizar las operaciones solicitadas. A su vez el robot enviará datos al terminal Android cuando el programa desarrollado en RAPID necesite mandar instrucciones de control al dispositivo Android. El controlador del robot no siempre envía datos, solo cuando se le solicita. Con las pruebas realizadas, cuando esto ocurría, se ralentizaban mucho los movimientos del robot. Por lo tanto se ha optado solo a realizar envío de paquetes de datos del servidor al cliente, cuando este lo solicite.

4 MODELADO DE LA CÉLULA ROBOTIZADA

4.1 INTRODUCCIÓN

Con el objetivo de realizar simulaciones fiables con el software RobotStudio de la célula robotizada disponible en el laboratorio de prácticas del Departamento de Ingeniería de Sistemas y Automática, se realiza el modelado y la interconexión de los elementos de la estación de trabajo (ver Figura 4.1).

Durante la programación o simulación en RobotStudio se necesitarán los modelos de sus piezas de trabajo y equipos. Los modelos de equipos estándar se instalan como bibliotecas o geometrías junto con el software RobotStudio. Si el fabricante nos proporciona los modelos CAD de sus piezas de trabajo, se puede importar como geometría desde RobotStudio. En lo que nos ocupa a nuestro proyecto, como no disponemos de modelos CAD proporcionados por el fabricante, deberemos crear por nuestra cuenta los modelos CAD, para ello utilizaremos la pestaña de modelado de RobotStudio, que contiene los controles necesarios para crear componentes, crear cuerpos, mediciones y operaciones de CAD.

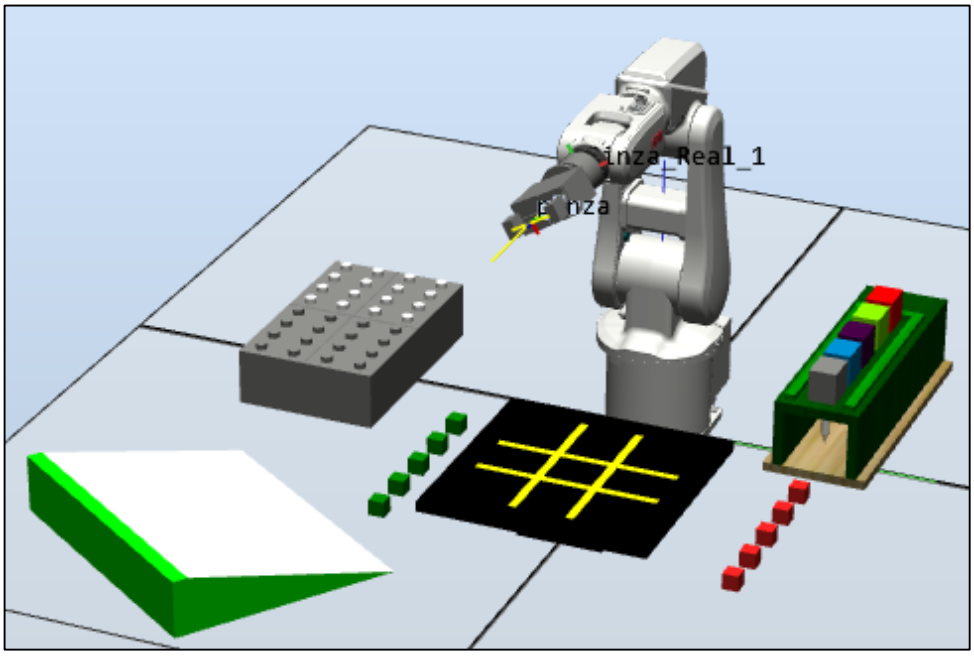


Figura 4.1: Célula robotizada del laboratorio en RobotStudio

Los elementos de la estación de trabajo utilizada para el proyecto son:

- Robot ABB IRB 120
- Pinza de agarre
- Panel externo de entradas y salidas digitales
- Mesa inclinada de trabajo
- Mesa negra de trabajo
- Soporte rotuladores
- Rotuladores de diferentes colores
- Piezas

De los elementos nombrados anteriormente el Robot ABB IRB 120 se encuentra modelado en la biblioteca proporcionada en el software RobotStudio, por lo que nos limitaremos a modelar únicamente el resto de elementos de la célula robotizada.

4.2 MODELADO DE LA PINZA DE AGARRE

La herramienta de trabajo que utilizaremos para nuestra estación de trabajo robotizada es una pinza de agarre, que realiza las funciones de manipulación de objetos. El proceso de modelar la pinza de agarre se realiza en los siguientes pasos:

- Creación de la geometría de la pinza
- Creación de la herramienta a partir de la geometría de la pinza.
- Añadir componentes inteligentes a la pinza.

4.2.1 Creación de la geometría de la pinza

La pinza modelada está compuesta por 11 sólidos simples (3 cilindros y 8 tetraedros) conectados entre sí para obtener la geometría deseada. En la Figura 4.2 se puede apreciar la geometría final de la pinza y los 11 sólidos simples que la componen.

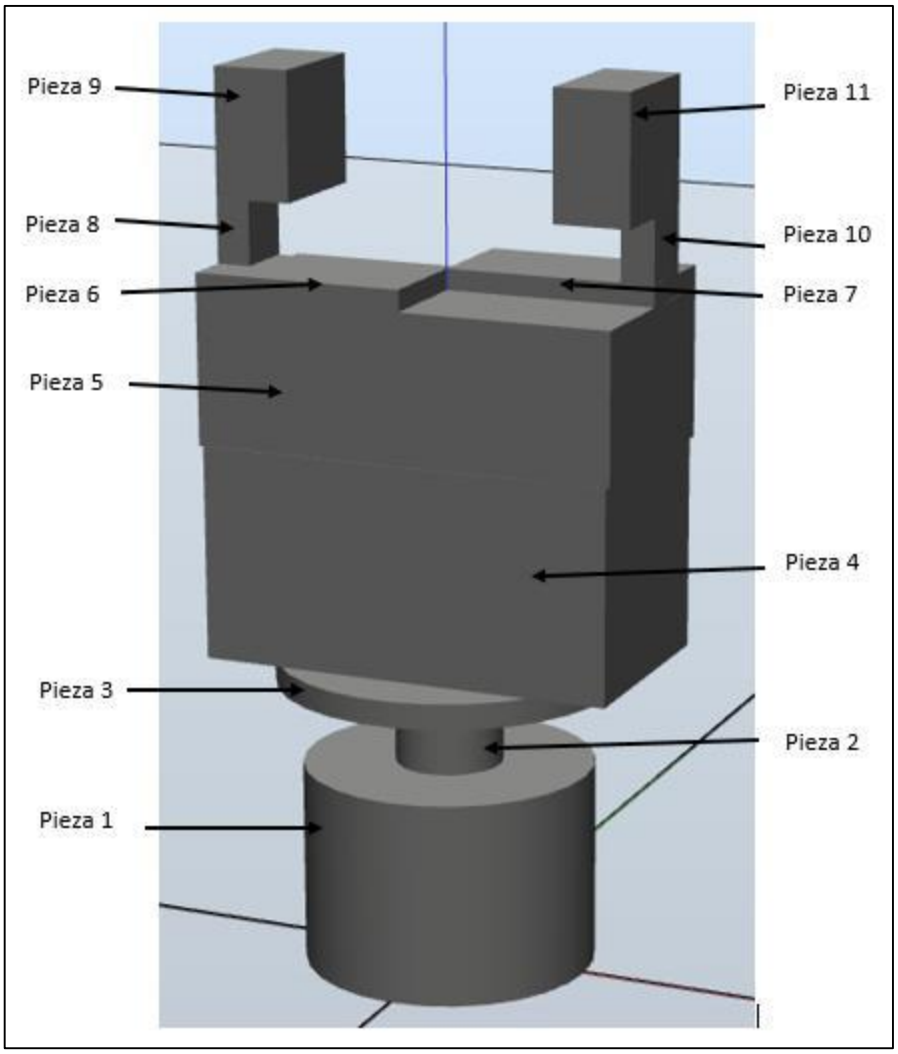


Figura 4.2: Geometría de la pinza

Para crear un sólido desde la pestaña de Modelado debemos hacer clic en "Sólido" y seleccionar de la lista desplegable el tipo de sólido que queremos introducir en la geometría. En nuestro caso, para la geometría de la pinza sólo utilizaremos sólidos de tipo cilindro y tetraedro. Una vez seleccionamos el sólido que utilizaremos se nos abre una ventana de diálogo donde deberemos introducir los valores necesarios para crear el sólido: referencia, punto esquina, orientación, longitud, anchura, altura, etc. En la Tabla 4.1 podemos obtener la información de las piezas simples que componen la geometría de la pinza.

PIEZA	TIPO SÓLIDO	DIMENSIONES (mm)
Pieza 1	Cilindro	68 x 48 (Diámetro x Altura)
Pieza 2	Cilindro	25 x 20 (Diámetro x Altura)
Pieza 3	Cilindro	80 x 6 (Diámetro x Altura)
Pieza 4	Tetraedro	96 x 52 x 50 (Largo x Ancho x Altura)
Pieza 5	Tetraedro	100 x 52 x 35 (Largo x Ancho x Altura)
Pieza 6	Tetraedro	50 x 26 x 5 (Largo x Ancho x Atura)
Pieza 7	Tetraedro	50 x 26 x 5 (Largo x Ancho x Atura)
Pieza 8	Tetraedro	8 x 15 x15 (Largo x Ancho x Altura)
Pieza 9	Tetraedro	18 x 30 x 30 (Largo x Ancho x Altura)
Pieza 10	Tetraedro	8 x 15 x15 (Largo x Ancho x Altura)
Pieza 11	Tetraedro	18 x 30 x 30 (Largo x Ancho x Altura)

Tabla 4.1: Información geometrías simples de la pinza

4.2.2 Creación de la herramienta a partir de la geometría

En esta sección se explicará cómo crear un mecanismo a partir de la geometría de la pinza creada anteriormente. Desde la pestaña de Modelado de RobotStudio accedemos a menú de creación de mecanismos haciendo clic en "Crear Mecanismo" (ver Figura 4.3).

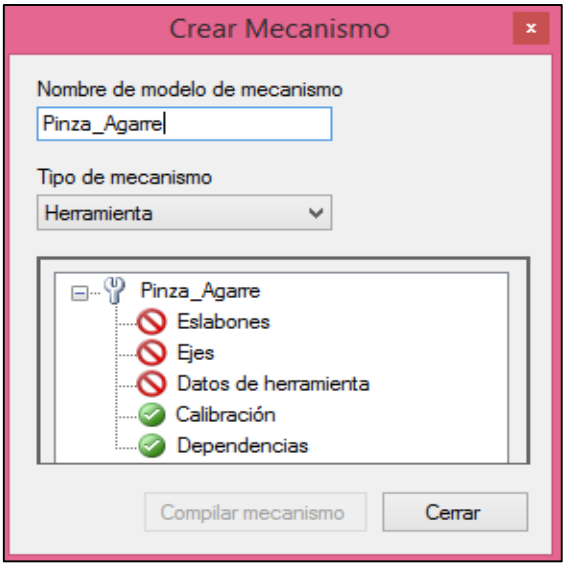


Figura 4.3: Ventana de dialogo "Crear Mecanismo"

El mecanismo que queremos realizar se tiene que comportar como una herramienta de trabajo, por lo que a la hora de elegir el tipo de mecanismo marcamos la opción de herramienta.

El siguiente paso a realizar es configurar los eslabones del mecanismo, para ello accedemos a través de la estructura de árbol a la ventana de dialogo "Crear Eslabón". Para crear la herramienta de la pinza de agarre consideramos 3 eslabones, a los cuales debemos añadir las geometrías simples creadas en la pestaña de modelado. Los 3 eslabones creados y las piezas que los componen se pueden apreciar en la Figura 4.4.

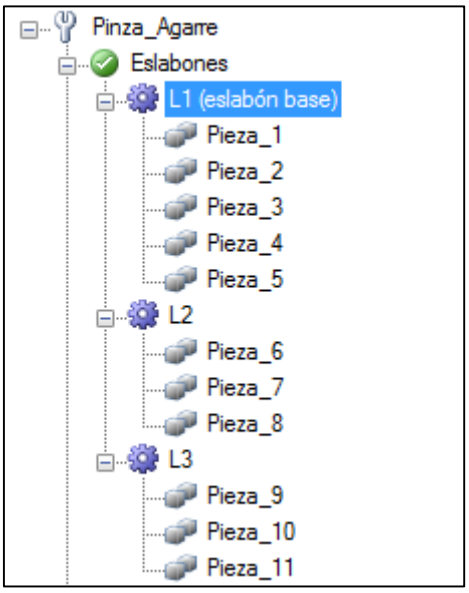


Figura 4.4: Eslabones de la pinza

Se ha considerado el eslabón L1 como eslabón base, formado por el conjunto de las piezas 1, 2, 3, 4 y 5; siendo los eslabones L2 y L3 los eslabones laterales que en su movimiento relativo respecto al eslabón base, simularan la apertura y el cierre de la pinza. El eslabón L2 está compuesto por las piezas 6, 7 y 8; mientras que el eslabón L3 está compuesto por las piezas 9,10 y 11.

A continuación, definimos los ejes de la herramienta, que en nuestro caso serán 2. Uno para mover el eslabón L2 respecto al eslabón base, y el otro para mover el eslabón L3 respecto al eslabón base.

Para definir los ejes accedemos a través de la estructura de árbol a la ventana de dialogo “Crear Eje”, donde podremos configurar el eje antes de crearlo. En la Figura 4.5 se puede observar la configuración elegida para los ejes J1 y J2 de los que consta nuestra pinza de agarre.

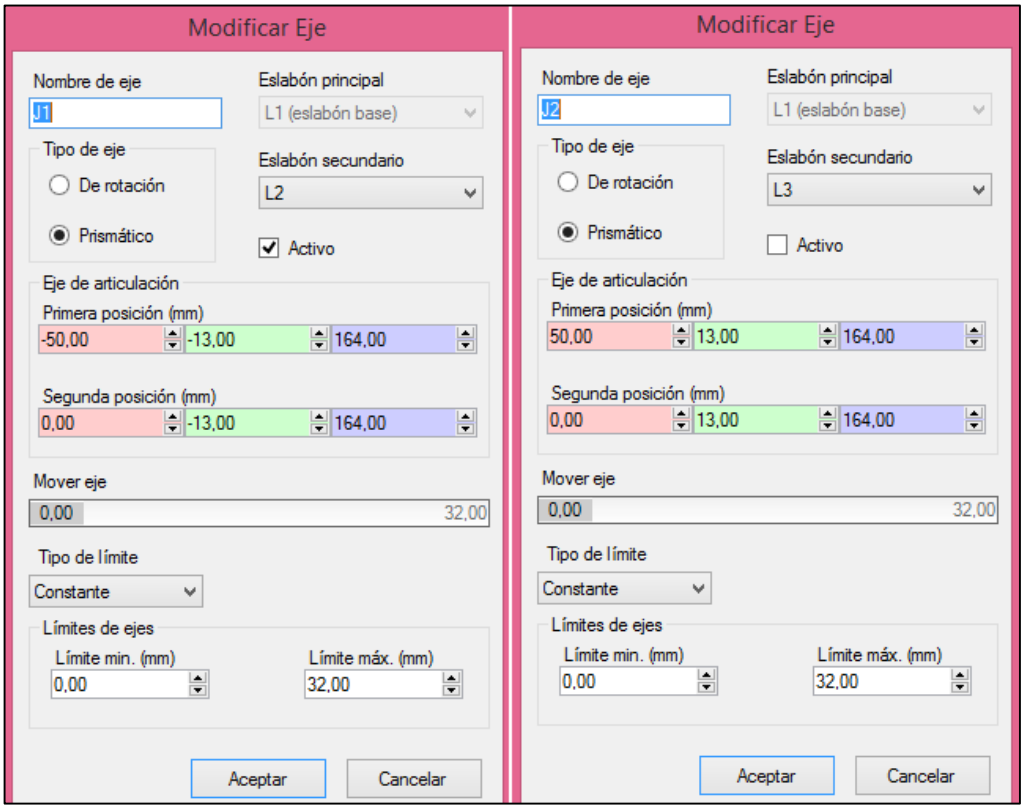


Figura 4.5: Ejes de la pinza

El eje J1 es de tipo prismático y permite el movimiento lineal entre el eslabón principal (L1) y el eslabón secundario (L2). El eje J2 también es de tipo prismático y permite el movimiento lineal entre el eslabón principal (L1) y el eslabón secundario (L3). La configuracion de estos ejes hace que ambos ejes se muevan en sentido contrario. Cuando los eslabones se acercan entre sí se produce el cierre de la pinza y cuando los eslabones se alejan entre sí se produce la apertura de la pinza.

El siguiente paso es modificar los datos de la herramienta, para ello accedemos a través de la estructura de arbol a la ventana de dialogo “Modificar Datos de la Herramienta” y completamos los datos que nos piden (ver Figura 4.6).

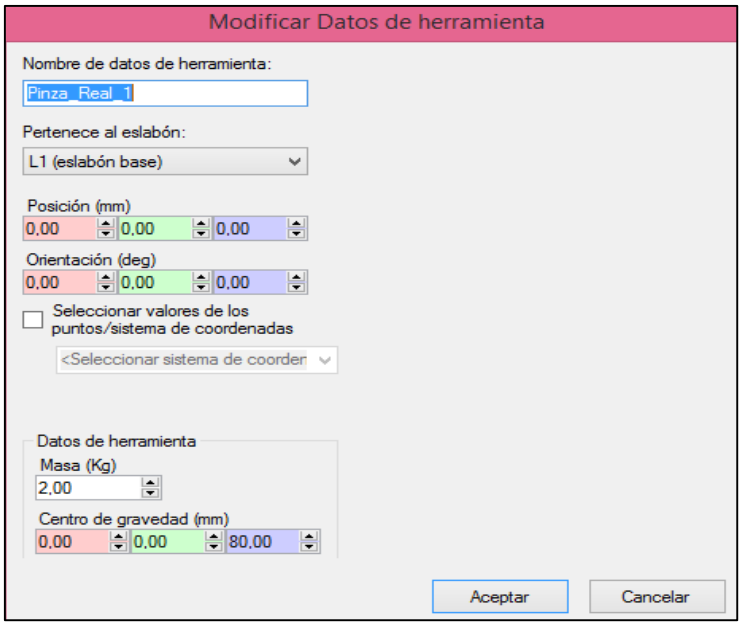


Figura 4.6: Datos de la herramienta

Los datos de la herramienta que debemos introducir son básicamente la masa de la pinza, el centro de gravedad y el momento de inercia. La masa que introducimos en los datos de la herramienta es la masa de la pinza real utilizada en la estación de trabajo, que es de 2 kg.

El siguiente paso es establecer la dependencia entre los ejes J1 y J2. Para que el movimiento de apertura y cierre de la pinza esté sincronizado y los eslabones se muevan con la misma velocidad, se establece una dependencia del eje J2 respecto al eje J1 con un factor de dependencia de valor 1.

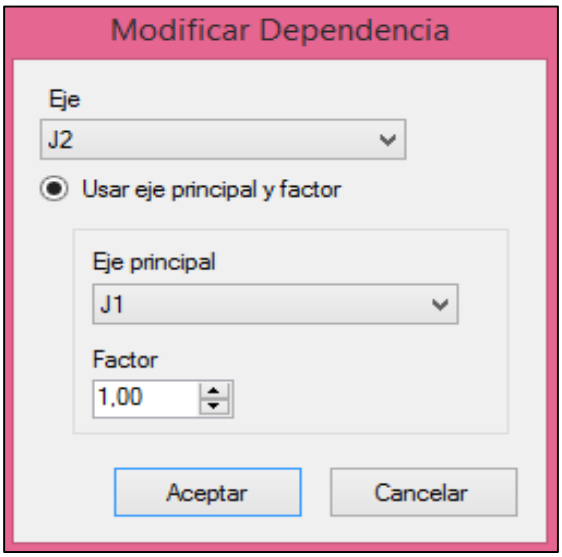


Figura 4.7: Dependencia de los ejes de la pinza

RobotStudio te indica si has cumplido todos los pasos necesarios para crear un mecanismo mediante un aspa de color verde al lado de cada uno de las configuraciones que se muestran en la estructura de árbol. Una vez que hemos configurado todas las características anteriores ya estamos en disposición de crear nuestra herramienta (ver Figura 4.8). El siguiente paso es hacer clic en "Compilar mecanismo" y ya tendremos a nuestra disposición la herramienta para utilizarla.

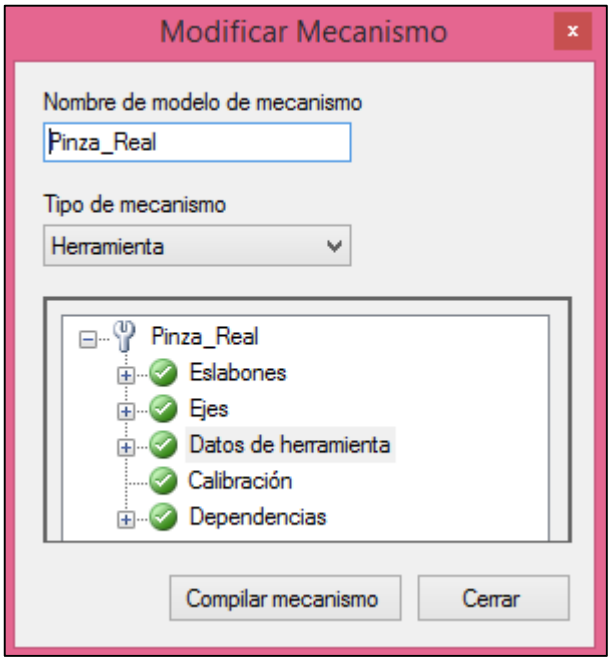


Figura 4.8: Compilar mecanismo pinza

4.2.3 Añadir componentes inteligentes a la pinza

La herramienta de trabajo creada anteriormente no sería de gran utilidad a la hora de realizar una simulación de RobotStudio, ya que no tiene definida el comportamiento que debe tener la pinza ni las señales de los sensores de los que dispone. Esto se consigue mediante el uso de los componentes inteligentes, que es un objeto de RobotStudio (con o sin representación gráfica en 3D) que presenta el comportamiento que puede implementarse mediante la clase code-behind y/o agregación de otros componentes inteligentes. El Editor de componentes inteligentes permite crear, editar y agregar componentes inteligentes mediante una interfaz gráfica de usuario y es una alternativa al uso de un compilador de bibliotecas basado en XML.

4.2.3.1 Crear componente inteligente

En primer lugar creamos nuestro componente inteligente desde la pestaña "Modelado". Para ello hacemos clic en "Crear Componente Inteligente" y nos aparecerá en la ventana de modelado nuestro componente inteligente vacío, al que nombraremos como "Componente inteligente Pinza". El siguiente paso es asociarle la geometría que queremos que tome nuestro componente inteligente, que en este caso es la herramienta "Pinza_real" creada anteriormente.

A continuación creamos otros dos componentes inteligentes nuevos, que llamaremos "Componente inteligente Pinza_1" y "Componente inteligente Pinza_2", que arrastraremos hacia el componente principal para hacerles a estos componentes subordinados de nuestro componente principal. El resultado de la organización se puede observar en la Figura 4.9, que se corresponde con la ventana de modelado.

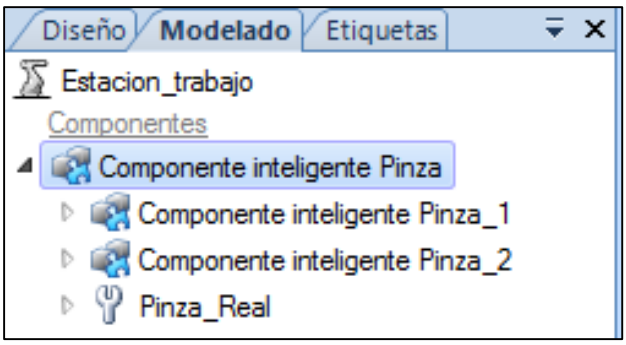


Figura 4.9: Estructura componente inteligente pinza

Los componentes inteligentes subordinados al principal se han creado con el propósito de que la pinza se comporte de diferente manera según el eslabón que antes entre en contacto con la pieza que se quiere manipular. Cuando el primer eslabón en entrar en contacto con una pieza es el eslabón L3, el componente inteligente se comporta según la programación definida en "Componente inteligente Pinza_1". Por el contrario, si el primer eslabón en entrar en contacto con una pieza es el eslabón L2, el componente inteligente se comporta según la programación definida en "Componente inteligente Pinza_2". El diagrama de flujo general para comprender como se comporta el componente inteligente de la pinza según un objeto colisione con los eslabones laterales se puede observar en el Diagrama 4.1.

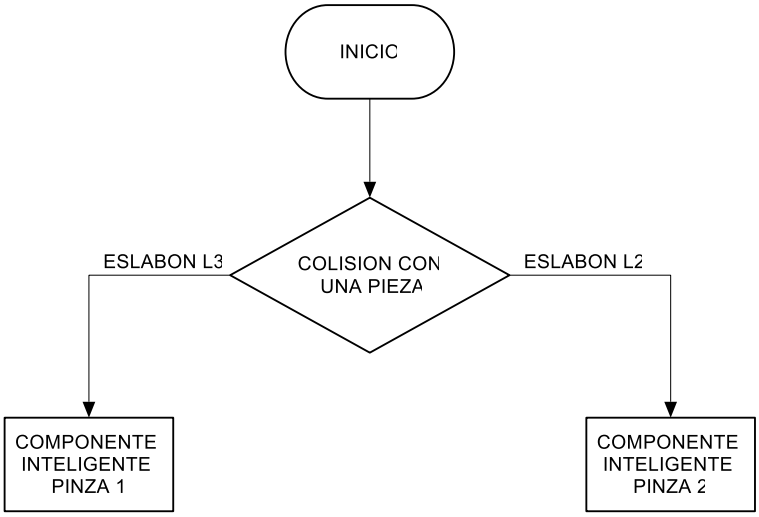


Diagrama 4.1: Comportamiento de la pinza ante colisiones

4.2.3.2 Añadir señales E/S de la pinza

El siguiente paso es dotar al componente inteligente de las señales de E/S, que se corresponderán con las señales reales de que dispone la pinza real disponible en el laboratorio de prácticas. En la Tabla 4.2 se enumeran las señales de E/S del componente inteligente de la pinza.

SEÑAL	TIPO DE SEÑAL	DESCRIPCION
PiezaDetectada	Salida digital	Cuando el sensor lineal detecta un objeto activa la señal digital
PiezaAgarrada	Salida digital	Cuando una pieza es "agarrada" activa la señal digital
AbrirPinza	Entrada digital	Señal que controla la apertura de la pinza
CerrarPinza	Entrada digital	Señal que controla el cierre de la pinza
ActivarSensores	Entrada digital	Señal interna del componente inteligente que se utiliza para iniciar todos los sensores.

Tabla 4.2: Señales de E/S del componente inteligente pinza

4.2.3.3 Componente inteligente Pinza_1

En esta sección vamos a describir como construir el componente inteligente para la pinza cuando el eslabón L3 de la herramienta entra en contacto con algún objeto que se quiera manipular. Para configurar el componente inteligente nos valemos de los componentes inteligentes básicos disponibles en RobotStudio.

A continuación se nombrarán y describirán brevemente la función que realizan algunos de estos componentes inteligentes básicos que se utilizarán para modelar la pinza.

- **LineSensor**: se define una línea de radio variable definida por un punto de inicio "Start" y un punto final "End", que actúa como un sensor. Cuando la señal "Active" se encuentra en nivel alto, el sensor detecta los objetos que están en intersección con la línea del sensor. Cuando se produce una intersección, se activa la señal de salida "Sensor Out" y el objeto que se encuentra en intersección con el sensor se almacena en la propiedad "Sensed Part". En la Figura 4.10 se puede observar las propiedades del sensor de la pinza modelada.

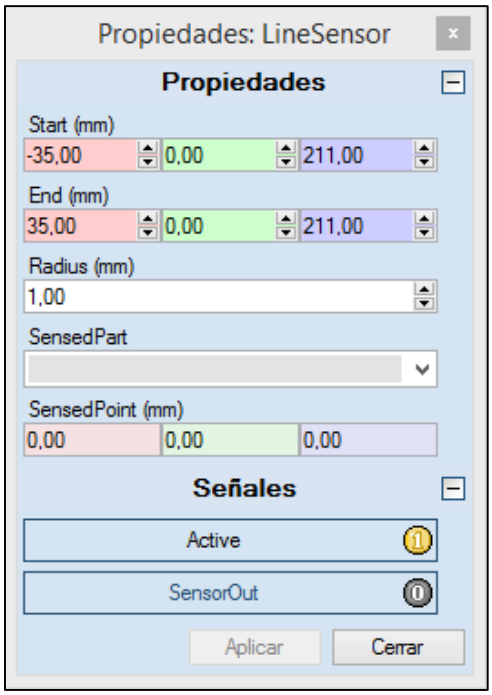


Figura 4.10: Propiedades LineSensor

- **CollisionSensor:** detecta colisiones y eventos de casi colisión entre el primer objeto y el segundo. Si no se especifica uno de los objetos, el otro se comprueba frente a toda la estación. Si la señal "Active" está en el nivel alto y se produce una colisión o un evento de casi colisión y el componente está activo, la señal "SensorOut" se activa y las piezas implicadas en la colisión o en el evento de casi colisión se indican como primera pieza en colisión y segunda pieza en colisión del Editor de propiedades. En la Figura 4.11 se puede observar las propiedades del sensor de la pinza modelada.

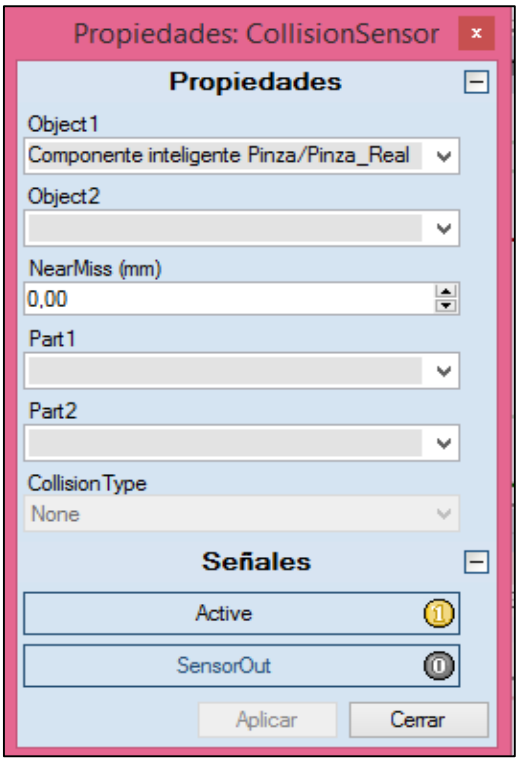


Figura 4.11: Propiedades CollisionSensor

- **LogicGate**: la señal "Output" es activada por la operación lógica especificada Operator en las dos señales "Input A" e "Input B", con el retardo especificado en Delay. Las operaciones lógicas que se pueden realizar son: AND, OR, NOT, XOR Y NOP.

- **LogicSRLatch**: mantiene estable el pulso de entrada.

  Si Set=1, entonces Output=1 y InvOutput=0

  Si Reset=1, entonces Output=0 y InvOutput=1

- **Expression:** se puede crear una expresión utilizando literales numéricos (incluidos el número PI), paréntesis, operadores matemáticos y funciones matemáticas. El resultado de la expresión se muestra en la señal de salida "Resultado".

- **VectorConverter:** convierte un vector de coordenadas en sus respectivas coordenadas X, Y, Z por separado. Esto hace posible que podamos utilizar estos valores en cálculos posteriores.

- **Comparer:** este componente compara el primer valor "Value A" con el segundo valor "Value B" utilizando el Operator. La señal de salida "Output" cambia a valor alto si se cumple la condición.

- **PositionSensor:** este componente monitoriza la posición de un objeto, que sólo se actualizará durante la simulación. En la Figura 4.12 se muestra las propiedades del componente PositionSensor utilizado en el modelado de la pinza para monitorizar la posición del objeto que "agarra" la pinza.

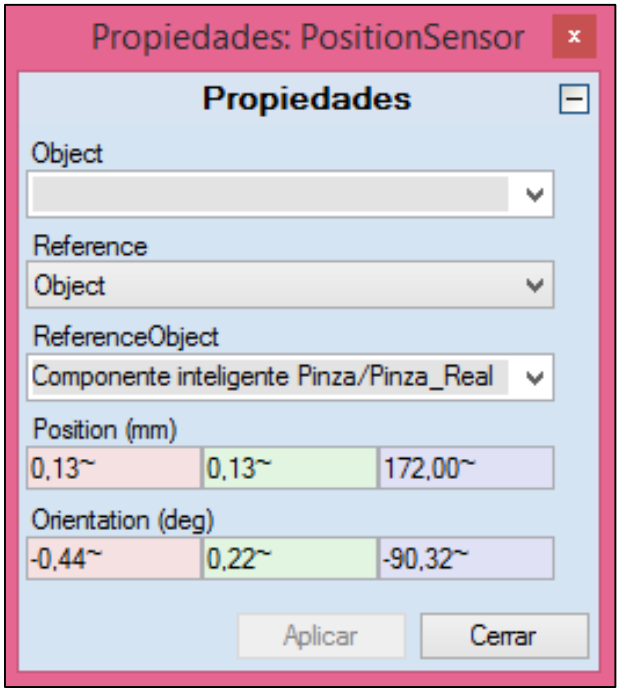


Figura 4.12: Propiedades PositionSensor

- **Attacher:** este componente conectará el objeto "Child" a "Parent" cuando se activa la señal de entrada "Execute". Si "Parent" es un mecanismo, también es necesario especificar la brida "Flange" a la que conectarse. Cuando se activa la entrada "Execute", el objeto subordinado se conecta al objeto superior, además si "Mount" está activado, el objeto subordinado también se montará sobre el objeto superior, con los parámetros "Offset" y "Orientation" especificados. La señal de salida "Executed" se activará al finalizar de unir el objeto "Child" a "Parent". En la Figura 4.13 se puede observar las propiedades del componente Attacher utilizado en el modelado de la pinza, cuya finalidad es unir una pieza al eslabón que entra en contacto con la misma.

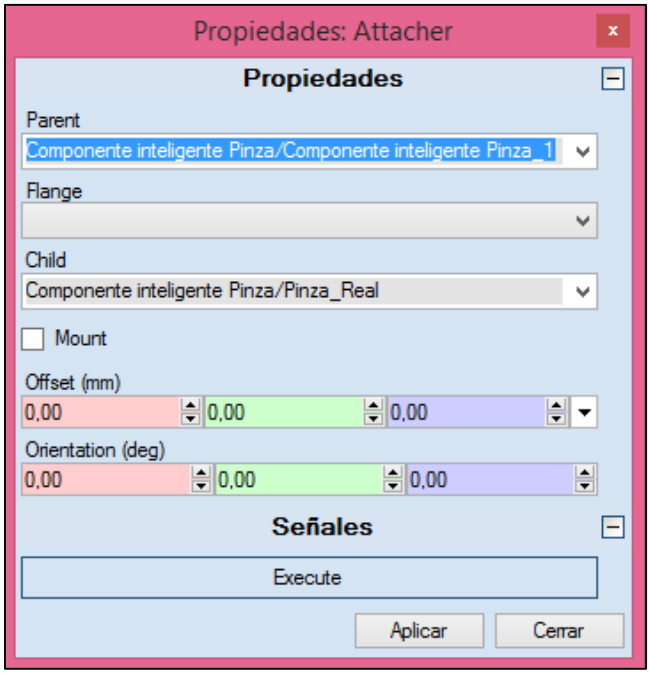


Figura 4.13: Propiedades Attacher

- **JointMover:** este componente utiliza como propiedades un mecanismo, un conjunto de valores de eje y una duración. Cuando se activa la señal de entrada "Execute", los valores de eje del mecanismo se mueven hasta la posición indicada. Una vez alcanzada la posición, se activa la señal de salida "Executed". La señal "GetCurrent" obtiene los valores de eje actuales del mecanismo.

En la Figura 4.14 se pueden apreciar las propiedades del componente JointMoverClose y JointMoverOpen utilizados en el modelado de la pinza. El componente JointMoverClose se utiliza para cerrar la pinza, hasta el momento en el que la pieza "agarrada" se encuentra justo en la posición central de la pinza, que es cuando se cancela el movimiento de cierre de la pinza y se detiene el JointMoverClose. El componente JointMoverOpen se utiliza para abrir la pinza una vez que el objeto se ha separado de los eslabones laterales.

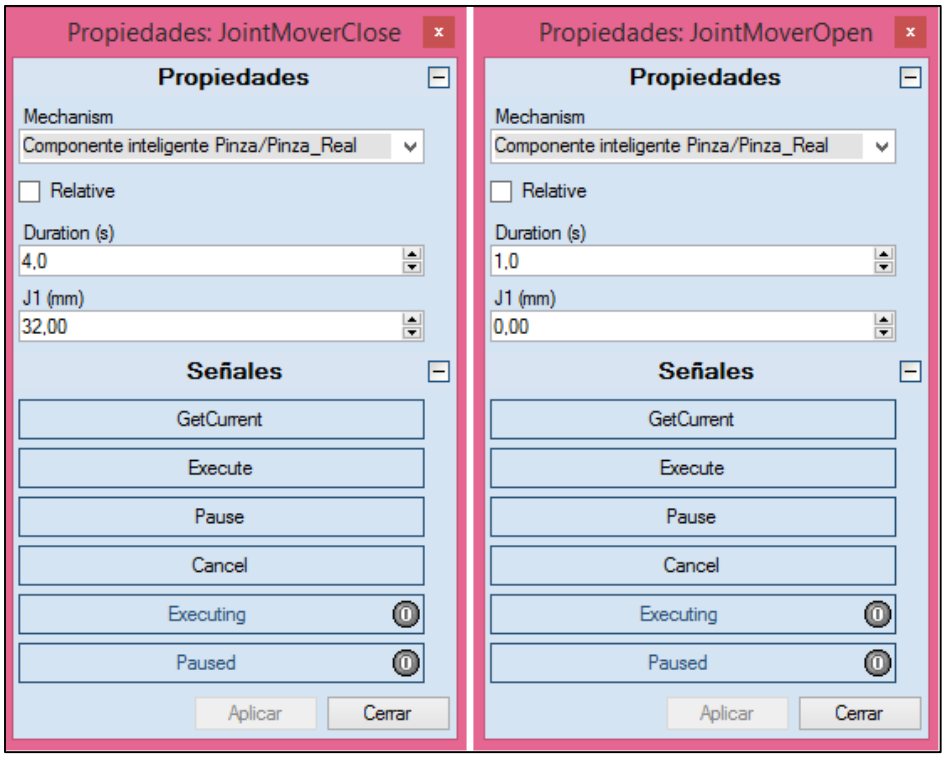


Figura 4.14: Propiedades JointMoverClose y JointMoverOpen

- **Detacher:** este componente desconectará el objeto "Child" del objeto cuando se activa la señal de entrada "Execute". Si "Keep position" está activado, la posición se mantendrá. De lo contrario, el objeto subordinado se posiciona con respecto a su objeto superior. Al finalizar, la señal de salida "Executed" se activa. En la Figura 4.14 se muestra las propiedades del componente Detacher, siendo el valor del objeto "Child" el que le envía el componente Attacher.

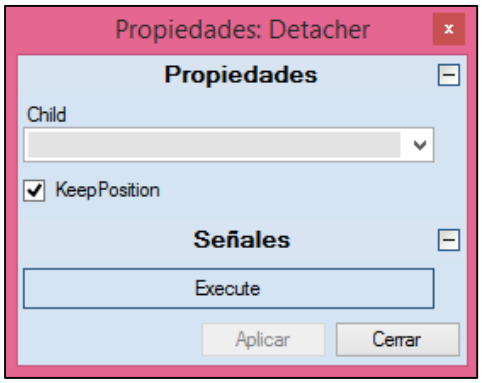


Figura 4.15: Propiedades detacher

Todos los componentes básicos que se acaban de introducir se deben de conectar entre sí y configurar para que lleven a cabo las acciones que deseamos. Las conexiones entre las señales de los componentes básicos se gestionan desde el menú del componente inteligente en la pestaña "Señales y conexiones"; mientras que los enlazamientos de propiedades entre los diferentes componentes inteligentes básicos se realiza en la pestaña "Propiedades y enlazamientos".

Los enlazamientos de señales del modelado de la pinza se muestran en la Tabla 4.3.

OBJETO ORIGEN	SEÑAL ORIGEN	OBJETO DESTINO	SEÑAL DESTINO
C.I. Pinza_1	ActivaSensores	LineSensor	Active
LineSensor	SensorOut	C.I. Pinza_1	PiezaDetectada
C.I. Pinza_1	AbrirPinza	Detacher	Execute
C.I. Pinza_1	AbrirPinza	JointMoverOpen	Execute
Comparer	Output	JointMoverClose	Cancel
Comparer	Output	LogicGate [NOT]	Input A
LogicGate [NOT]	Output	CollisionSensor	Active
C.I. Pinza_1	CerrarPinza	LogicGate [AND]	Input A
LogicGate [NOT]	Output	LogicGate [AND]	Input B
LogicGate [AND]	Output	JointMoverClose	Execute
CollisionSensor	SensorOut	Attacher	Execute
Attacher	Executed	LogicSRLatch	Set
LogicSRLatch	Output	C.I. Pinza_1	PiezaCogida
JointMoverOpen	Executing	Detacher	Execute
JointMoverOpen	Executing	LogicSRLatch	Reset

Tabla 4.3: Señales componente inteligente pinza 1

Los enlazamientos de propiedades de los objetos utilizados en el modelado de la pinza se muestran en la Tabla 4.4:

OBJETO ORIGEN	PROPIEDAD ORIGEN	OBJETO DESTINO	PROPIEDAD DESTINO
CollisionSensor	Part 2	Attacher	Child
Attacher	Child	Detacher	Child
PositionSensor	Position	VectorConverter	Vector
CollisionSensor	Part 1	Attacher	Parent
LineSensor	SensedPart	CollisionSensor	Objetc2
LineSensor	SensedPart	PositionSensor	Object
Expression	Result	VectorConverter	Transform.X
Expression	Result	Comparer	Value A

Tabla 4.4: Propiedades componente inteligente pinza 1

4.2.3.4 Componente inteligente Pinza_2

El componente inteligente Pinza_2 simula el comportamiento de la herramienta cuando el eslabón L2 es el primero en establecer contacto con el objeto que queremos manipular. Su estructura y conexiones son similares a lo descrito en la sección anterior. La única propiedad que cambia y que hace que se active este componente inteligente es el signo del comparador de la posición del objeto que se une al eslabón lateral. En la Figura 4.15 se puede observar como la única diferencia radica en el signo que se evalúa en el comparador, del componente inteligente Pinza_1 (en la izquierda de la Figura 4.15) y el comparador del componente inteligente Pinza_2 (en la derecha de la Figura 4.15).

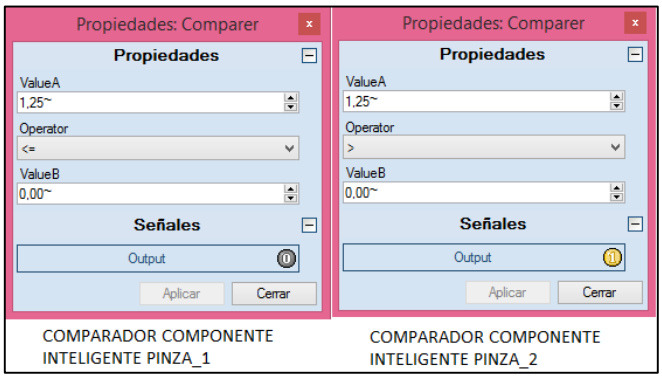


Figura 4.16: Propiedades de los comparadores

4.2.3.5 Diagramas de flujo de funcionamiento de la pinza

El que se ejecute el componente inteligente subordinado Pinza_1 o el componente inteligente subordinado Pinza_2 depende de la posición inicial de la pieza que corta el sensor de posición. Si la pieza se encuentra en un principio más cerca del eslabón L3, implica que entrará en contacto antes con dicho eslabón, ejecutándose si se cumplen estas premisas el componente subordinado Pinza_1. Si por el contrario, la colisión se produce antes con el eslabón L2, se ejecuta el componente subordinado Pinza_2. A continuación se muestra el diagrama de flujo simplificado del funcionamiento del componente inteligente de la pinza.

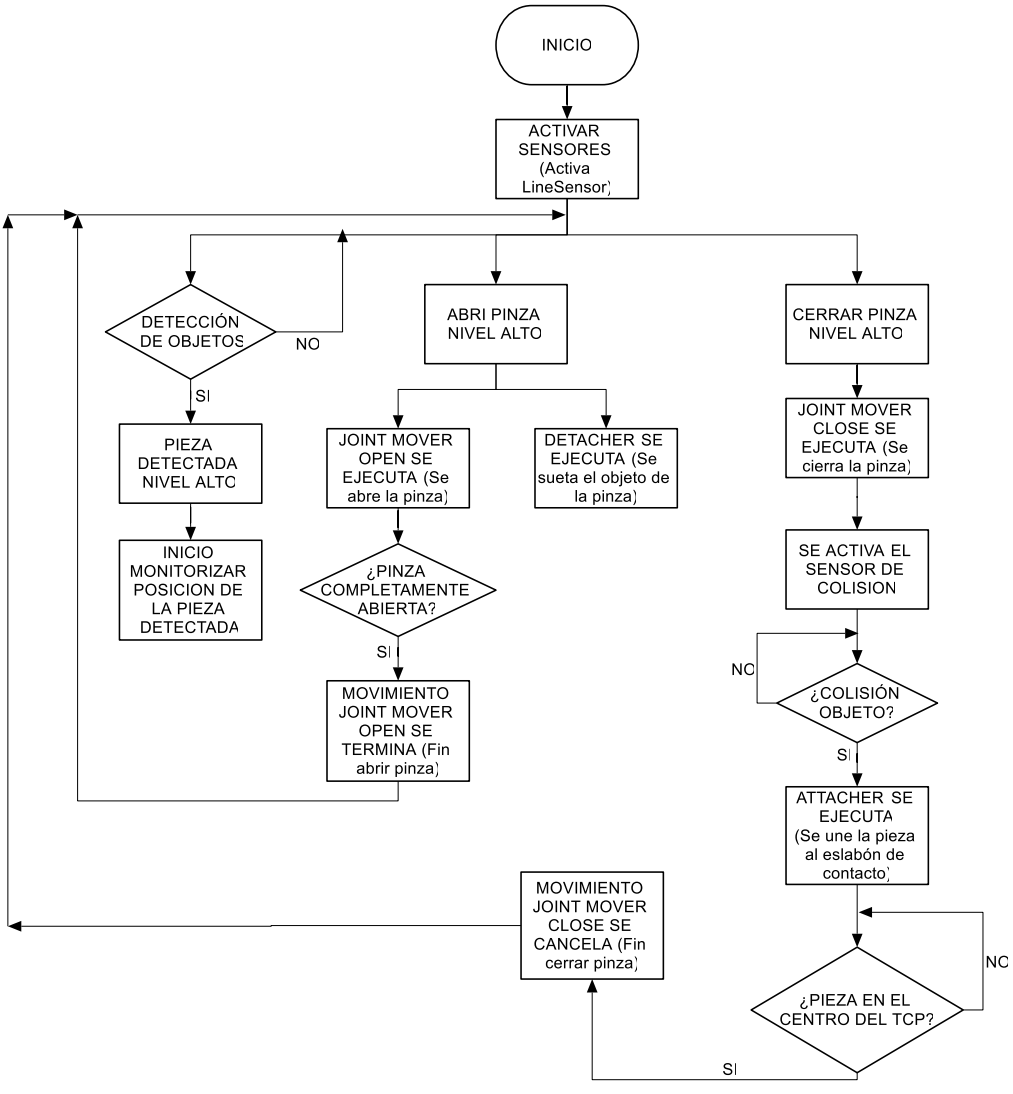


Diagrama 4.2: Diagrama simplificado de la pinza

4.2.3.6 Configurar la lógica de la estación

Una vez tenemos modelada la pinza de la estación de trabajo, ya sólo nos queda conectar las señales de E/S del componente inteligente de la pinza con las señales de E/S del controlador del robot IRB 120, con el fin de que puedan compartir las mismas señales que se utilizan en la estación de trabajo real. En la tabla 4.5 se pueden observar las conexiones que se realizan en la lógica de la estación.

OBJETO ORIGEN	SEÑAL ORIGEN	OBJETO DESTINO	SEÑAL DESTINO
irb120_profibus	do5	C.I. Pinza	AbrirPinza
irb120_profibus	do6	C.I. Pinza	CerrarPinza
C.I. Pinza	PiezaDetectada	irb120_profibus	di7
C.I. Pinza	PiezaAgarrada	irb120_profibus	di8

Tabla 4.5: Lógica de la estación (Pinza)

En la Figura 4.17 se puede observar un esquema de las conexiones que genera RobotStudio de la lógica de la estación. En la figura se pueden ver gráficamente las conexiones entre el componente inteligente de la pinza y el controlador del robot IRB 120, que anteriormente se han detallado en la Tabla 4.5.

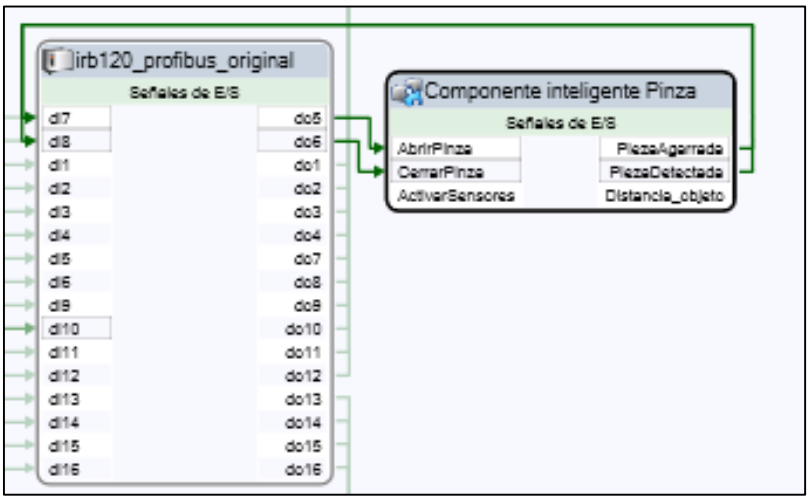


Figura 4.17: Esquema de conexiones estación-pinza

Al conectar las señales del componente inteligente con la lógica de la estación, se pueden usar las funcionalidades de la pinza al realizar las simulaciones en RobotStudio. Para ello basta con darle las instrucciones de abrir y cerrar la pinza a través de las salidas digitales (do5 para abrir la pinza y do6 para cerrar la pinza) del controlador IRB 120. Las instrucciones que habría que implementar en RAPID para abrir y cerrar la pinza se pueden observar en la Figura 4.18.

```
! Abre la pinza

PROC AbrirPinza()

    SetDO do5,1;
    WaitTime 1;
    SetDO do5,0;
    WaitTime 1;

ENDPROC
```

```
! Cierra la pinza

PROC CerrarPinza()

    SetDO do6,1;
    WaitTime 1;
    SetDO do6,0;
    WaitTime 1;

ENDPROC
```

Figura 4.18: Ejemplo RAPID abrir y cerrar pinza

4.3 MODELADO DEL PANEL EXTERNO DE E/S

El panel externo disponible en el laboratorio de prácticas dispone de 16 botones correspondientes con cada una de las entradas digitales “di” que se activan/desactivan en la tarjeta de E/S del controlador del robot. Dicho panel también cuenta con 16 leds correspondientes a cada una de las salidas digitales “do” del controlador, de modo que si la salida digital toma un valor alto “1” el led se iluminará y en caso contrario el led permanecerá apagado.

El proceso de modelar la pinza de agarre se realiza en los siguientes pasos:

- Creación de la geometría del panel externo de E/S
- Añadir componentes inteligentes a la pinza.

4.3.1 Creación de la geometría del panel externo de E/S

El panel externo modelado está compuesto por 34 sólidos simples (32 cilindros y 2 tetraedros) conectados entre sí para obtener la geometría deseada. De los 32 cilindros que hay en el panel externo, 16 de ellos representan los leds del panel externo (salidas digitales del controlador) y los otros 16 representan los botones del panel externo (entradas digitales del controlador). Los dos tetraedros son el soporte donde se apoyan los botones y los led, siendo la pieza 1 el soporte de los leds y la pieza 2 el soporte de los botones. En la Figura 4.2 se puede apreciar la geometría final de la pinza y los 34 sólidos simples que la componen.

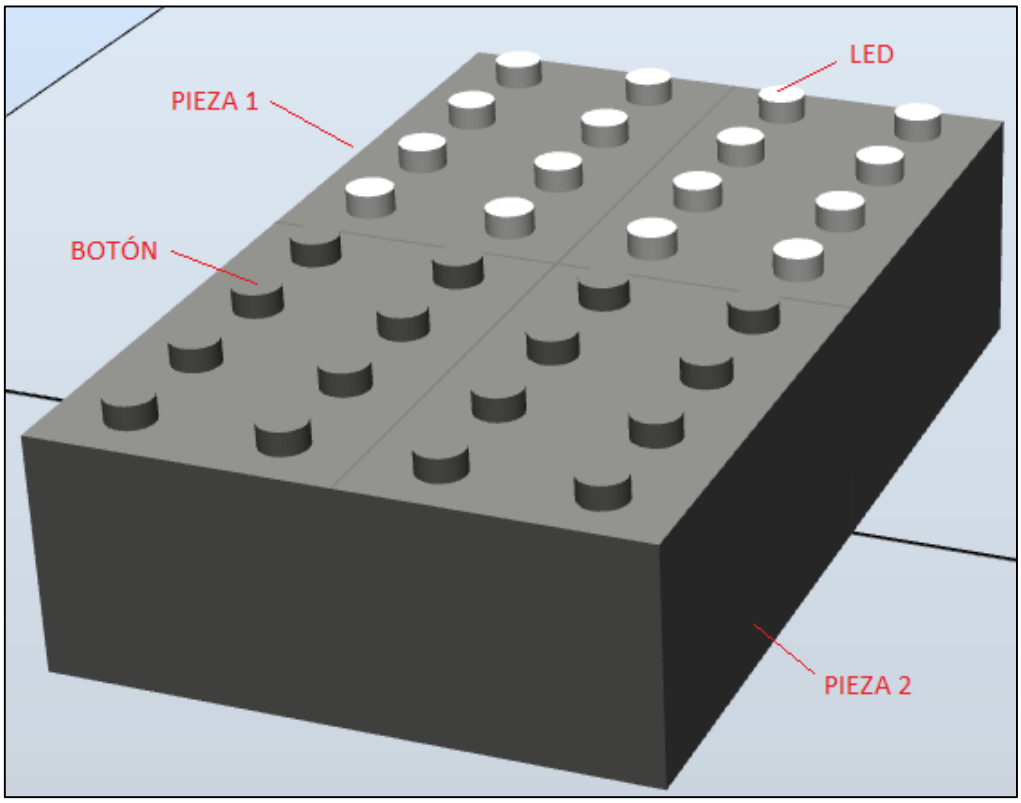


Figura 4.19: Geometría panel externo E/S

Para crear un sólido desde la pestaña de Modelado debemos hacer clic en "Sólido" y seleccionar de la lista desplegable el tipo de sólido que queremos introducir en la geometría. En nuestro caso, para la geometría del panel externo sólo utilizaremos sólidos de tipo cilindro y tetraedro.

Una vez seleccionamos el sólido que utilizaremos se nos abre una ventana de diálogo donde deberemos introducir los valores necesarios para crear el sólido: referencia, punto esquina, orientación, longitud, anchura, altura, etc. En la Tabla 4.6 podemos obtener la información de las piezas simples que componen la geometría del panel externo.

PIEZA	TIPO SÓLIDO	DIMENSIONES (mm)
Pieza 1	Tetraedro	250 x 200 x 100 (Largo x Ancho x Altura)
Pieza 2	Tetraedro	250 x 200 x 100 (Largo x Ancho x Altura)
LED	Cilindro	20 x 10 (Diámetro x Altura)
BOTÓN	Cilindro	20 x 10 (Diámetro x Altura)

Tabla 4.6: Información geometrías simples del panel externo E/S

4.3.2 Añadir componentes inteligentes al panel externo

La geometría del panel externo creada anteriormente no sería de gran utilidad a la hora de realizar una simulación de RobotStudio, ya que no tiene definida el comportamiento que debe tener el panel ni las señales que comparte con la célula robotizada. Esto se consigue mediante el uso de componentes inteligentes y enlazando las señales del componente inteligente de la botonera con las señales del controlador del robot IRB 120.

4.3.2.1 Crear componente inteligente

En primer lugar creamos nuestro componente inteligente desde la pestaña “Modelado”. Para ello hacemos clic en “Crear Componente Inteligente” y nos aparecerá en la ventana de modelado nuestro componente inteligente vacío, al que nombraremos como “Componente inteligente botonera”. El siguiente paso es asociarle la geometría que queremos que tome nuestro componente inteligente, que en este caso son la piezas 1, la pieza 2, los 16 botones y los 16 leds.

A continuación creamos 16 componentes inteligentes nuevos, que se encargaran de simular el comportamiento real de los botones. Estos 16 componentes creados los arrastraremos hacia el componente principal para hacerles componentes subordinados de nuestro componente principal.

4.3.2.2 Añadir señales E/S panel externo

El siguiente paso es dotar al componente inteligente de las señales de E/S, que se corresponderán con las señales reales de que dispone el panel externo real disponible en el laboratorio de prácticas. En la Tabla 4.7 se enumeran las señales de E/S del componente inteligente del panel externo.

SEÑAL	TIPO DE SEÑAL	DESCRIPCION
[di1 … di16]	Entrada digital	Señal que se conecta con las salidas digitales del controlador y que cuando están en nivel alto enciende los leds
[do1 … do16]	Salida digital	Señal que se conecta con las entradas digitales del controlador y que cuando está en nivel alto activa las entradas digitales del controlador del robot.
[Boton_do1 … Boton_16]	Entrada digital	Señal interna que utiliza el componente inteligente para simular la situación real de pulsar el botón del panel. Al activar señal el componente inteligente muestra visualmente el botón pulsado.

Tabla 4.7: Señales de E/S del componente inteligente botonera

4.3.2.3 Componente inteligente "Boton_do"

En esta sección vamos a describir como construir el componente inteligente subordinado "boton_do", que simula el comportamiento real de presionar un botón del panel externo. Nos limitaremos a analizar y describir solamente el componente "botón_do1", ya que la estructura es igual para el resto de botones utilizaos en el modelado del panel externo. Para configurar el componente inteligente nos valemos de los componentes inteligentes básicos disponibles en RobotStudio.

A continuación se nombrarán y describirán brevemente la función que realizan algunos de estos componentes inteligentes básicos que se utilizarán para modelar el componente subordinado "boton_do".

- **Positioner:** este componente toma un objeto, una posición y una orientación como propiedades. Cuando se activa la señal de entrada "Execute", el objeto es reposicionado en la posición determinada con respecto a Reference. Al finalizar, se activa la señal de salida "Executed". ". En la Figura 4.20 se puede observar las propiedades del componente Positioner utilizado en el modelado del panel, cuya finalidad es desplazar el cilindro que representa el botón hacia dentro del soporte, dando la sensación visual de estar "presionando" el botón. A la izquierda en la figura podemos ver las propiedades del Positioner cuando "pulsamos" el botón; mientras que en la derecha de a figura vemos las propiedades del Positioner cuando "soltamos" el botón.

Figura 4.20: Propiedades positioner

- **Highlighter:** este componente cambia temporalmente el color del objeto seleccionado "Object" a los valores RGB especificados en la propiedad "Color". El color se mezcla con el color original de los objetos tal y como se definen por "Opacity". Cuando la señal de entrada "Active" se desactiva, el objeto vuelve a sus colores originales. En la figura 4.21 se pueden observar las propiedades del componente Highlighter del botón_do1, cuya función es cambiar el color del botón cuando se "presiona" el mismo. De esta manera se puede apreciar visualmente que el componente del botón está activo. En este caso se ha escogido la combinación de colores RGB [255,0,0], que se corresponde con el color primario rojo.

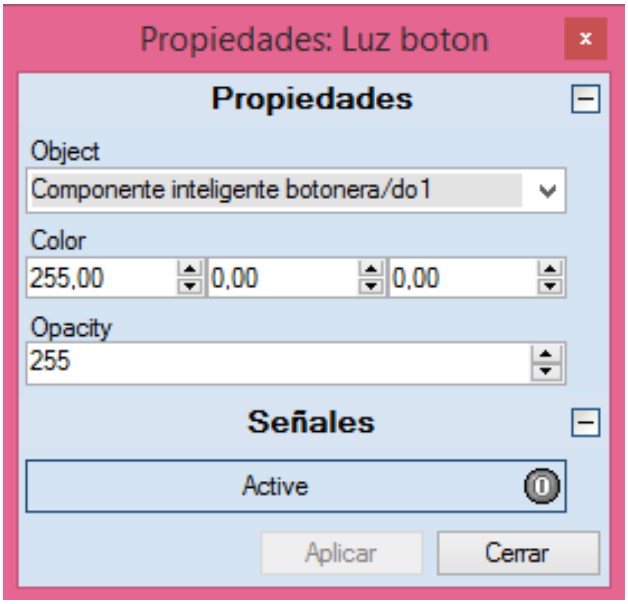


Figura 4.21: Propiedades highlighter boton_do

- **LogicGate:** la señal "Output" es activada por la operación lógica especificada Operator en las dos señales "Input A" e "Input B", con el retardo especificado en Delay. Las operaciones lógicas que se pueden realizar son: AND, OR, NOT, XOR Y NOP.

Todos los componentes básicos que se acaban de introducir se deben de conectar entre sí y configurar para que lleven a cabo las acciones que deseamos.

Las conexiones entre las señales de los componentes básicos se gestionan desde el menú del componente inteligente en la pestaña “Señales y conexiones”

Los enlazamientos de señales del modelado del componente inteligente “Boton do1” se muestran en la Tabla 4.8.

OBJETO ORIGEN	SEÑAL ORIGEN	OBJETO DESTINO	SEÑAL DESTINO
Boton do1	boton_do1	LogicGate [NOT]	Input A
LogicGate [NOT]	Output	Positioner OFF	Execute
Boton do1	boton_do1	Luz boton	Execute
Boton do1	boton_do1	Positioner OFF	Execute

Tabla 4.8: Señales componente inteligente "Boton do1"

4.3.2.4 Componente inteligente “botonera”

En esta sección vamos a describir como construir el componente inteligente principal “botonera”, que simula el comportamiento del panel externo. Este componente inteligente está formado por los 16 componentes inteligentes subordinados definidos en la sección anterior y por 16 componentes básicos Highlighter denominados “Luz di”. Nos limitaremos a analizar y describir solamente el componente “Luz di1”, ya que la estructura es igual para el resto de leds utilizados en el modelado del panel externo.

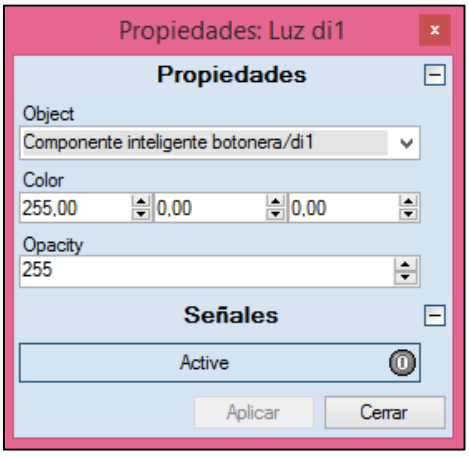


Figura 4.22: Propiedades highlighter leds

Para modelar la iluminación de los leds solo se utilizará el componente inteligente básico Highlighter, que se encarga de cambiar el color del cilindro que representa los respectivos leds del panel. En la Figura 4.22 se puede observar las propiedades de este componente, que se activa cuando se pone a nivel alto una salida digital del controlador. Al igual que en el caso de los botones, se ha elegido una combinación de colores RGB [255, 0, 0], que se corresponde con el color primario rojo.

Las conexiones entre las señales de los componentes básicos se gestionan desde el menú del componente inteligente en la pestaña "Señales y conexiones". Para el modelado del panel externo, los enlazamientos de señales del modelado del componente inteligente se muestran en la Tabla 4.9. En esta tabla solo se han mostrado las conexiones para el "Botón 1" y el "Led 1", siendo tres el número de conexiones de la tabla. Las conexiones del resto de botones y leds y es análoga a la mostrada en la tabla 4.9, por lo que el número total de conexiones que hay en el componente inteligentes es de 48.

OBJETO ORIGEN	SEÑAL ORIGEN	OBJETO DESTINO	SEÑAL DESTINO
C.I. Botonera	boton_do1	C.I. Botonera	do1
C.I. Botonera	di1	Luz di1	Active
C.I. Botonera	boton_do1	Boton do1	boton_do1

Tabla 4.9: Señales componente inteligente "botonera"

4.3.2.5 Diagrama de flujo de funcionamiento del panel externo

El funcionamiento que se pretende al modelar el panel externo es crear una comunicación con la tarjeta de E/S digitales del controlador del robot IRB 120. Al presionar un botón del panel se deber de activa una entrada digital en el controlador del robot; mientras que si se activa una salida digital del controlador, se tiene que activar el led correspondiente en el panel externo. A continuación se muestra el diagrama de flujo simplificado del funcionamiento del componente inteligente del panel externo de E/S.

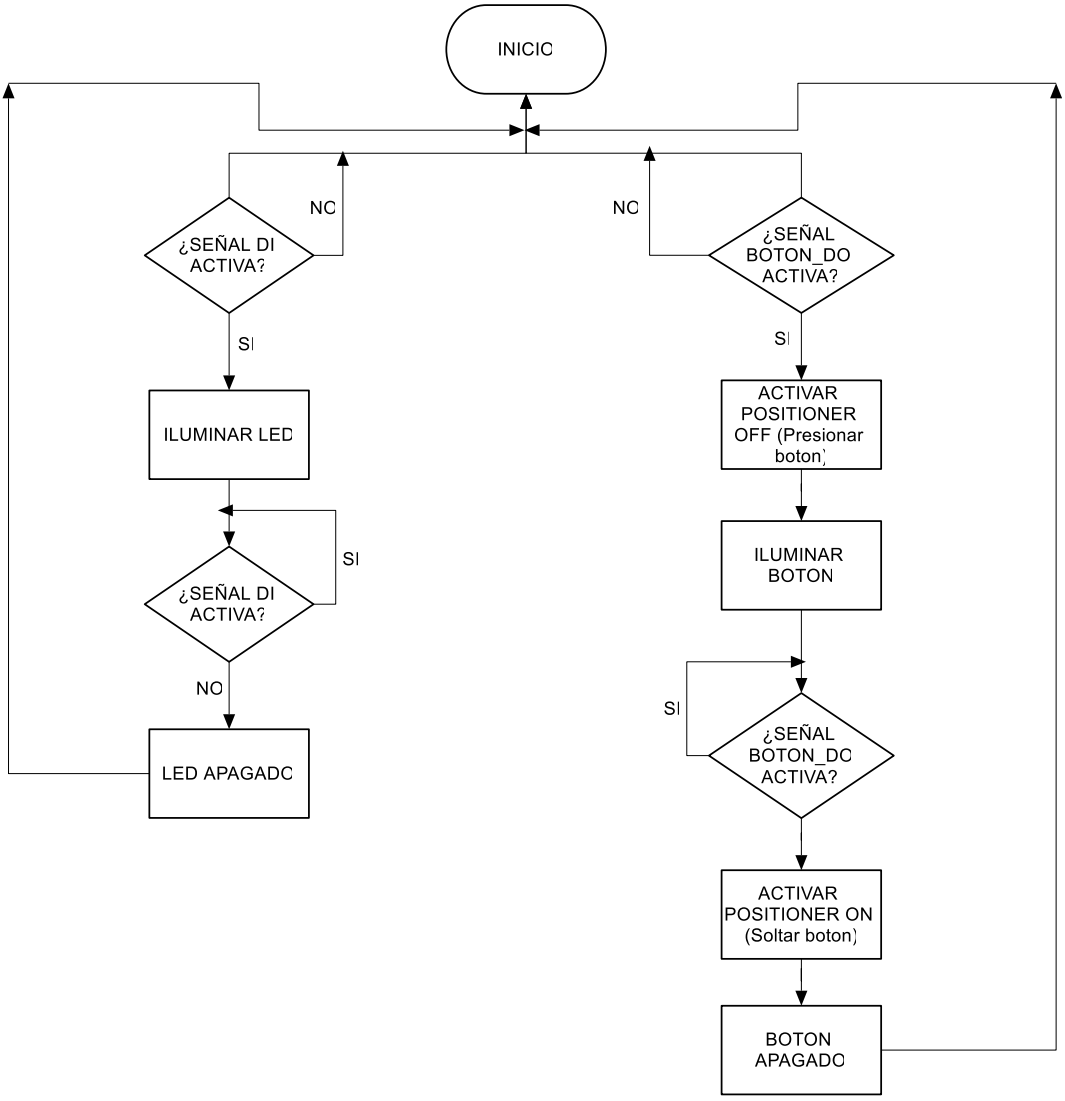


Diagrama 4.3: Diagrama simplificado panel externo E/S

4.3.2.6 Configurar la lógica de la estación

Una vez tenemos modelado el panel exterior de E/S de la estación de trabajo, ya sólo nos queda conectar las señales de E/S del componente inteligente con las señales de E/S del controlador del robot IRB 120, con el fin de que puedan compartir las mismas señales que se utilizan en la estación de trabajo real. En la tabla 4.10 se pueden observar las conexiones que se realizan en la lógica de la estación.

OBJETO ORIGEN	SEÑAL ORIGEN	OBJETO DESTINO	SEÑAL DESTINO
C.I. Botonera	do1	irb120_profibus	di1
C.I. Botonera	do2	irb120_profibus	di2
C.I. Botonera	do3	irb120_profibus	di3
C.I. Botonera	do4	irb120_profibus	di4
C.I. Botonera	do5	irb120_profibus	di5
C.I. Botonera	do6	irb120_profibus	di6
C.I. Botonera	do7	irb120_profibus	di7
C.I. Botonera	do8	irb120_profibus	di8
C.I. Botonera	do9	irb120_profibus	di9
C.I. Botonera	do10	irb120_profibus	di10
C.I. Botonera	do11	irb120_profibus	di11
C.I. Botonera	do12	irb120_profibus	di12
C.I. Botonera	do13	irb120_profibus	di13
C.I. Botonera	do14	irb120_profibus	di14
C.I. Botonera	do15	irb120_profibus	di15
C.I. Botonera	do16	irb120_profibus	di16
irb120_profibus	do1	C.I. Botonera	di1
irb120_profibus	do2	C.I. Botonera	di2
irb120_profibus	do3	C.I. Botonera	di3
irb120_profibus	do4	C.I. Botonera	di4
irb120_profibus	do5	C.I. Botonera	di5
irb120_profibus	do6	C.I. Botonera	di6
irb120_profibus	do7	C.I. Botonera	di7
irb120_profibus	do8	C.I. Botonera	di8
irb120_profibus	do9	C.I. Botonera	di9
irb120_profibus	do10	C.I. Botonera	di10
irb120_profibus	do11	C.I. Botonera	di11
irb120_profibus	do12	C.I. Botonera	di12
irb120_profibus	do13	C.I. Botonera	di13
irb120_profibus	do14	C.I. Botonera	di14
irb120_profibus	do15	C.I. Botonera	di15
irb120_profibus	do16	C.I. Botonera	di16

Tabla 4.10: Lógica de la estación (Panel exterior)

Como podemos observar en la tabla, las señales de salida del componente inteligente se conectan a las señales de entrada del controlador, y las señales de salida del controlador se conectan a las señales de entrada del componente inteligente.

En la Figura 4.23 se puede observar un esquema de las conexiones que genera RobotStudio de la lógica de la estación. En la figura se pueden ver gráficamente las conexiones entre el componente inteligente de la pinza y el controlador del robot IRB 120, que anteriormente se han detallado en la Tabla 4.10.

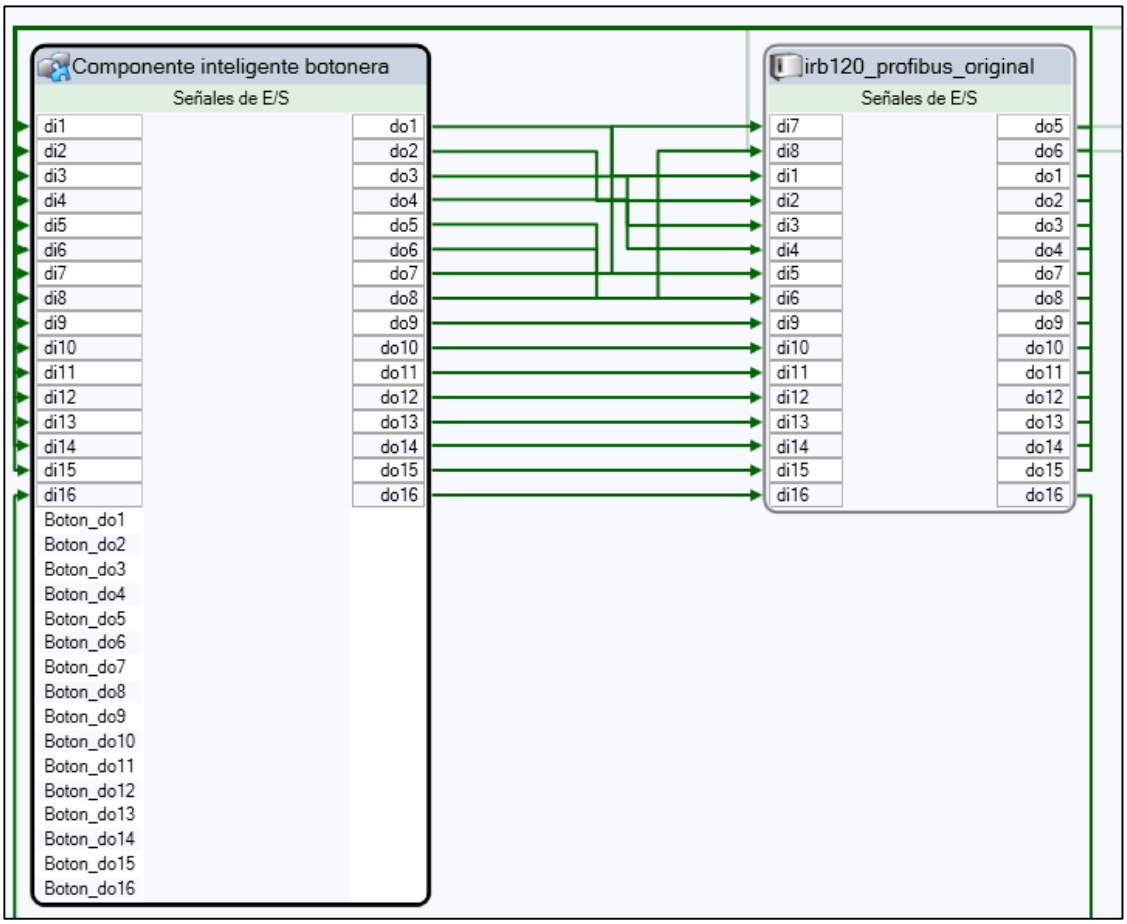


Figura 4.23: Esquema de conexiones estación-panel exterior E/S

4.4 MODELADO DE LOS OBJETOS DE TRABAJO

Un objeto de trabajo es un sistema de coordenadas utilizado para describir la posición de una pieza de trabajo. El objeto de trabajo se compone de dos bases de coordenadas: la base de coordenadas del usuario y la del objeto. Todas las posiciones que programe serán relativas a la base de coordenadas del objeto, que a su vez es relativa a la base de coordenadas del usuario, a su vez relativo al sistema de coordenadas mundo.

Para crear un nuevo objeto de trabajo debemos acceder a la pestaña “Inicio”, en el grupo “Programación de Trayectorias” y seleccionar “Crear objeto de trabajo”. Al realizar esto, aparecerá una ventana de diálogo “Crear objeto de Trabajo”, donde debemos modificar los datos y características para ajustar el objeto de trabajo a nuestras necesidades (ver Figura 4.24).

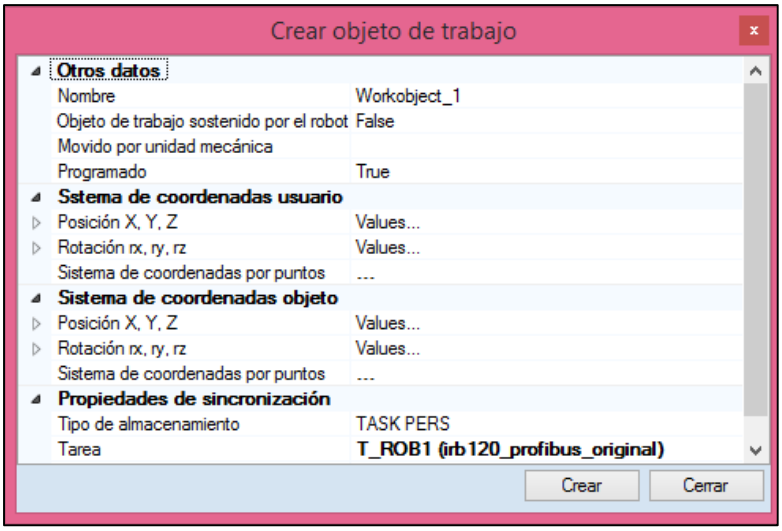


Figura 4.24: Ventana de diálogo Crear objeto de trabajo

La célula robotizada del laboratorio de prácticas consta de dos objetos de trabajo:

- Mesa negra
- Mesa inclinada

4.4.1 Mesa Negra

El objeto de trabajo "Mesa Negra" está compuesto por un tablero de madera cuyas dimensiones son 403x398x18 mm, y por un "taco" de madera de 80mm de alto. El tablero de color negro se apoya sobre el "taco" de madera, siendo la altura total de la mesa de trabajo de 98 mm. Para modelar este objeto en RobotStudio lo primero que hacemos es crear la geometría de la mesa negra, que se basará en dos tetraedros. La pieza 1 hará la función del tablero de madera y la pieza 2 hará la función del "taco" de madera sobre el que se apoya el tablero.

En la figura 4.25 se puede apreciar la geometría del objeto de trabajo "Mesa Negra".

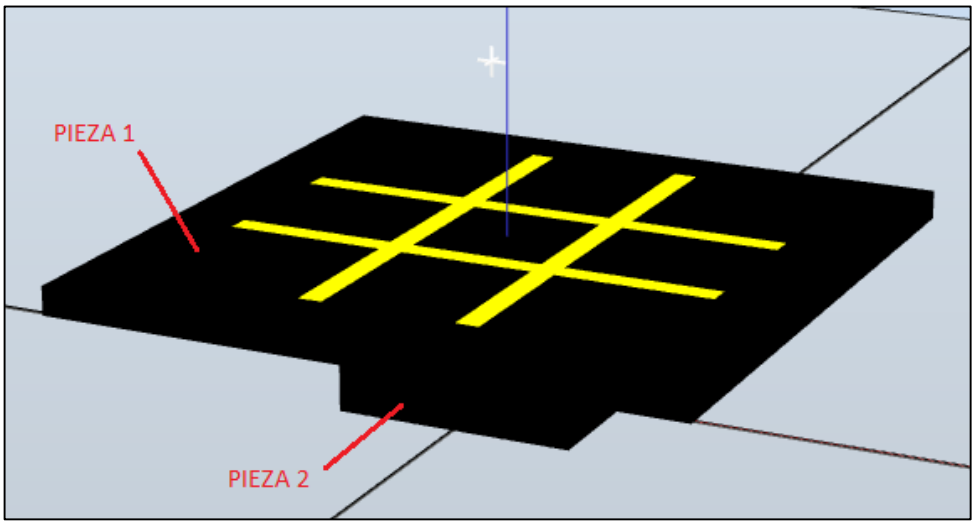


Figura 4.25: Geometría Mesa Negra

En la Tabla 4.11 podemos obtener la información de las piezas simples que componen la geometría de la mesa negra.

PIEZA	TIPO SÓLIDO	DIMENSIONES (mm)
Pieza 1	Tetraedro	403x398x18 (Largo x Ancho x Altura)
Pieza 2	Tetraedro	150 x 150 x 80 (Largo x Ancho x Altura)

Tabla 4.11: Información geometrías simples de la mesa negra

Una vez tenemos la geometría del objeto de trabajo ya podemos crear el objeto de trabajo desde la ventana de diálogo "Crear Objeto de trabajo", introduciendo el nombre del objeto de trabajo y el sistema de coordenadas del objeto. En el sistema de coordenadas introducimos el valor de la posición [X, Y, Z] a partir de tres puntos tomados sobre la mesa negra en la ventana gráfica. De estos tres puntos 2 corresponden al eje X y el otro al eje Y.

La ventana de diálogo para configurar el objeto de trabajo "Mesa Negra" se puede observar en la Figura 4.26.

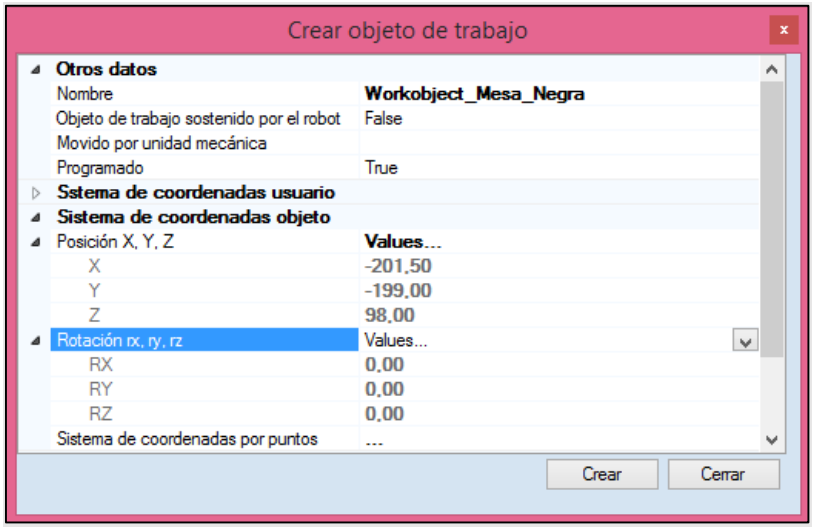


Figura 4.26: Crear objeto trabajo Mesa Negra

4.4.2 Mesa inclinada

El objeto de trabajo "Mesa inclinada" está compuesto por un tablero de color verde cuyas dimensiones son 500x386x25 mm, y por una tabla, que se encuentra colocado en posición vertical, de 100 mm de alto. El tablero de color verde se apoya en un extremo de éste sobre la tabla vertical, quedando el soporte de trabajo con una inclinación de 11,6º respecto de la superficie horizontal de referencia.

Para modelar este objeto en RobotStudio lo primero que hacemos es crear la geometría de la mesa inclinada, que se basará en un tetraedro y en un polígono de superficie, al que posteriormente extruiremos su superficie, convirtiéndolo así en un sólido triangular.

La pieza 1 hará la función del tablero colocado con una inclinación respecto a la horizontal; mientras que la pieza 2 hará la función del tablero de madera sobre el que se apoya el tablero.

En la figura 4.27 se puede apreciar la geometría del objeto de trabajo "Mesa Inclinada".

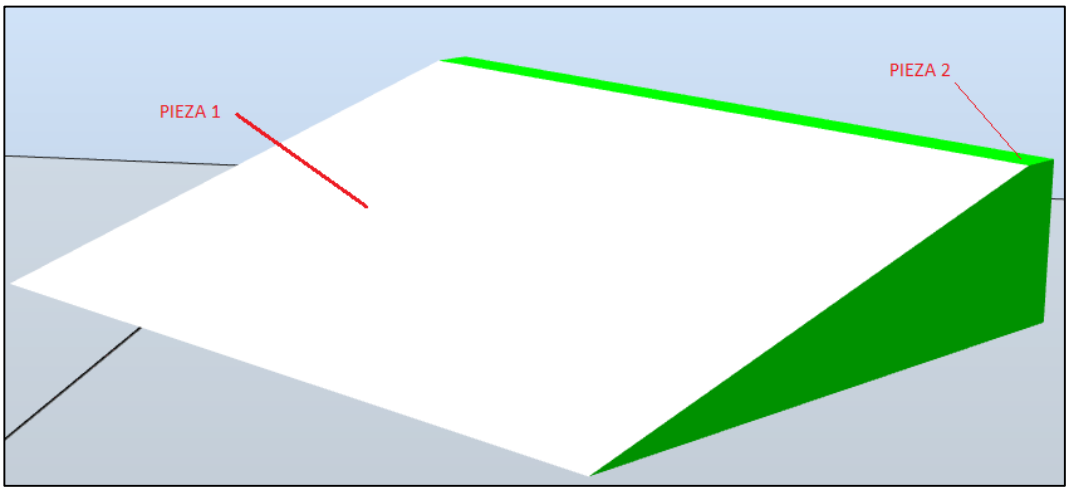


Figura 4.27: Geometría Mesa Inclinada

En la Tabla 4.12 podemos obtener la información de las piezas simples que componen la geometría de la mesa inclinada.

PIEZA	TIPO SÓLIDO	DIMENSIONES (mm)
Pieza 1	Solido triangular	100 x 373 (Base x Altura)
Pieza 2	Tetraedro	500 x 27 x 100 (Largo x Ancho x Altura)

Tabla 4.12: Información geometrías simples de la mesa inclinada

Una vez tenemos la geometría del objeto de trabajo ya podemos crear el objeto de trabajo desde la ventana de diálogo "Crear Objeto de trabajo", introduciendo el nombre del objeto de trabajo y el sistema de coordenadas del objeto. En el sistema de coordenadas introducimos el valor de la posición [X, Y, Z] a partir de tres puntos tomados sobre la mesa negra en la ventana gráfica. De estos tres puntos 2 corresponden al eje X y el otro al eje Y.

La ventana de diálogo para configurar el objeto de trabajo "Mesa Negra" se puede observar en la Figura 4.28.

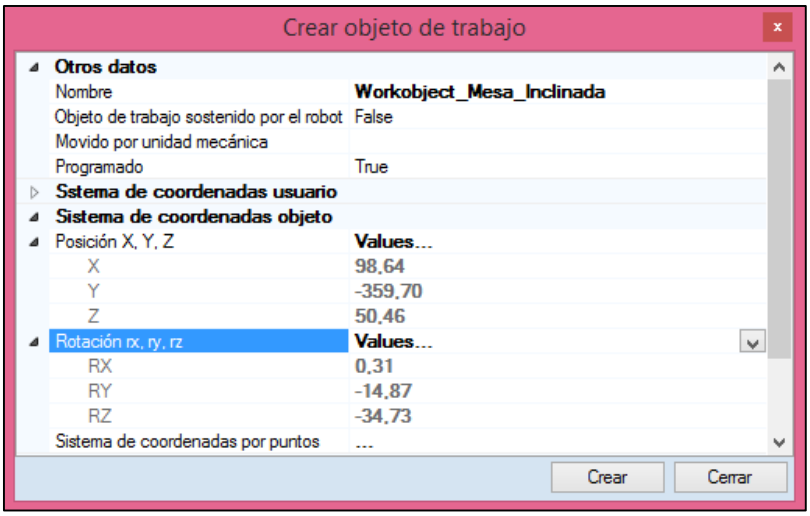


Figura 4.28: Crear objeto Mesa Inclinada

4.5 MODELADO DE LAS PIEZAS

En la célula robotizada disponible en el laboratorio de prácticas hay disponibles un conjunto de piezas que el robot puede manipular. Este conjunto de piezas está compuesto por 5 rotuladores, un soporte para los rotuladores y 10 cubos de pequeñas dimensiones.

4.5.1 Modelado de los rotuladores

En la estación de trabajo dispones de 4 rotuladores de diferentes colores (negro, rojo, verde, azul) y de un rotulador con la punta vacía que se utiliza para comprobar posiciones. La estructura de los rotuladores se basa en un taco de madera 60x48x58 mm, al cual se le realiza un taladrado interior de 16mm, que es lo suficientemente grande para que en su interior se inserte un rotulador de 15,2 mm de diámetro. La fijación del rotulador al taco de madera se realiza mediante un tornillo pasante lateral que presiona al rotulador una vez introducido éste en el agujero del taco.

Para modelar los rotuladores en RobotStudio lo único que debemos hacer es crear su geometría, que se basará en un tetraedro, dos cilindros y un cono. La pieza 1 hará la función del "taco" de madera, que se introduce para manipular más fácilmente el rotulador con la pinza de agarre.

La pieza 2 hará la función de la estructura del rotulador; mientras que la pieza 3 hace la función de la punta de rotulador. La pieza 4 simula el tornillo pasante lateral que presiona al rotulador una vez introducido éste en el agujero del "taco". En la figura 4.29 se puede observar la geometría de un rotulador de color azul.

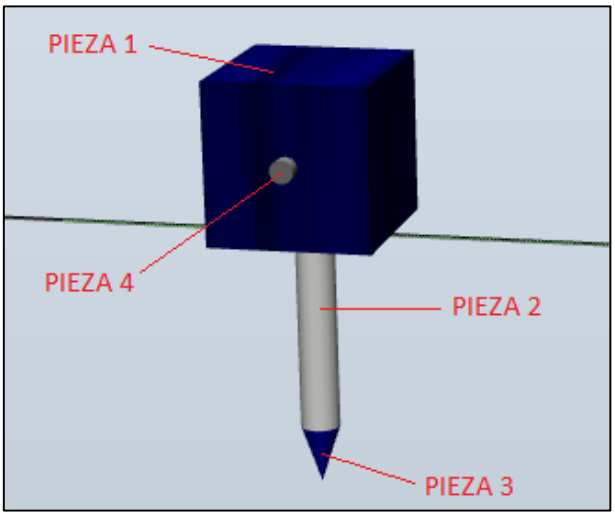


Figura 4.29: Geometría rotulador color azul

En la Tabla 4.13 podemos obtener la información de las piezas simples que componen la geometría de cada uno de los rotuladores de diferentes colores que se encuentran en la estación de trabajo.

PIEZA	TIPO SÓLIDO	DIMENSIONES (mm)
Pieza 1	Tetraedro	60x48x58 (Largo x Ancho x Altura)
Pieza 2	Cilindro	15 x 70 (Diámetro x Altura)
Pieza 3	Cono	15 x 14 (Base x Altura)
Pieza 4	Cilindro	8 x 5 (Diámetro x Altura)

Tabla 4.13: Información geometrías simples de un rotulador

4.5.2 Modelado del soporte de rotuladores

El soporte de rotuladores se ha diseñado con la finalidad de tener un lugar donde alojar los rotuladores y que el robot IRB 120 conozca en todo momento las coordenadas a las que debe dirigirse para "agarrar" y "soltar" los rotuladores. Al soporte de rotuladores se la han realizado 5 taladrados superiores de 20 mm de diámetro, que son donde se apoyaran los rotuladores, estando definidas de antemano las posiciones de los diferentes rotuladores de colores. En la figura 4.30 se puede observar la geometría del soporte de rotuladores.

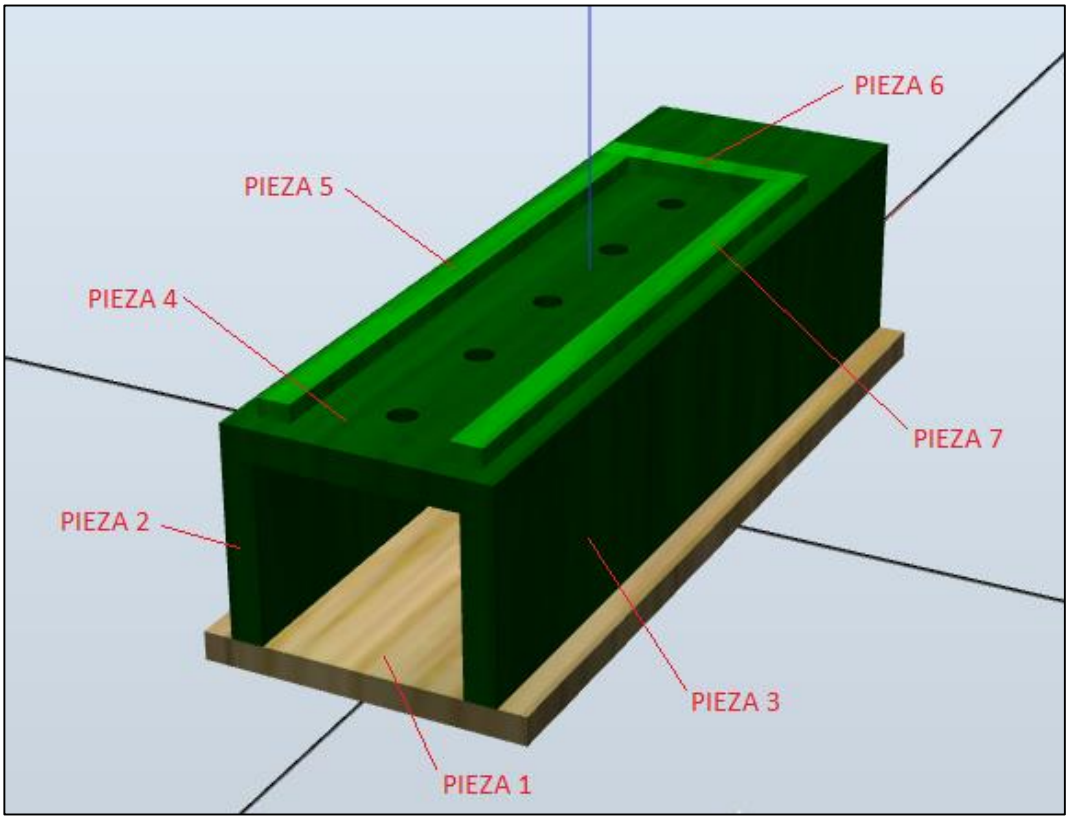


Figura 4.30: Geometría soporte de rotuladores

En la Tabla 4.14 podemos obtener la información de las piezas simples que componen la geometría del soporte de rotuladores.

PIEZA	TIPO SÓLIDO	DIMENSIONES (mm)
Pieza 1	Tetraedro	492 x 180 x16 (Largo x Ancho x Altura)
Pieza 2	Tetraedro	492 x 18 x 100 (Largo x Ancho x Altura)
Pieza 3	Tetraedro	492 x 18 x 100 (Largo x Ancho x Altura)
Pieza 4	Tetraedro	492 x 148 x 18 (Largo x Ancho x Altura)
Pieza 5	Tetraedro	368 x 18 x 10 (Largo x Ancho x Altura)
Pieza 6	Tetraedro	18 x 122 x 10 (Largo x Ancho x Altura)
Pieza 7	Tetraedro	368 x 18 x 10 (Largo x Ancho x Altura)

Tabla 4.14: Información geometrías simples soporte rotuladores

4.5.3 Modelado de los cubos

En la estación de trabajo disponemos de 10 cubos de plástico de pequeñas dimensiones, con el objetivo de que el robot pueda manipularlos fácilmente con la pinza de agarre. Los cubos tienen como finalidad simular las fichas utilizadas en una aplicación didáctica desarrollada que se basa en el juego "Tres en Raya". Los cubos modelados en RobotStudio se pueden observar en la Figura 4.31 y su geometría se basa en un tetraedro cuyas dimensiones son 28 x 28 x 28 mm.

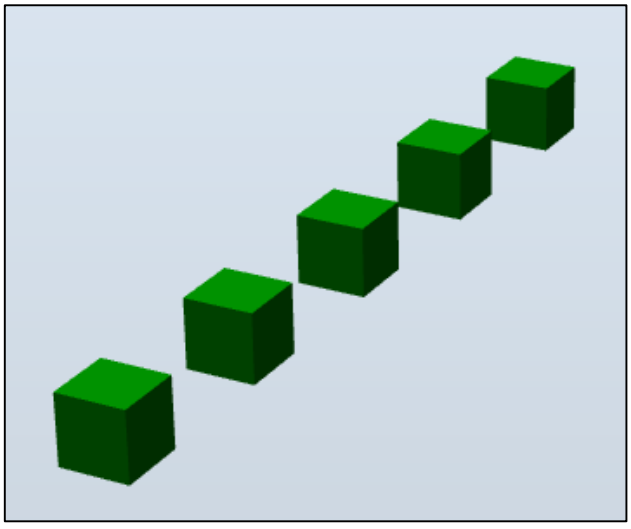


Figura 4.31: Geometría de los cubos

5 APLICACIONES DIDÁCTICAS EN ROBOTSTUDIO

5.1 INTRODUCCIÓN

En este apartado se desarrollarán una serie de aplicaciones didácticas que nos permitirán comprobar que la estación simulada y la célula robotizada real trabajan de forma síncrona y realizan los mismos procesos. El objetivo de estas aplicaciones es comprobar el correcto funcionamiento de la comunicación entre el controlador del robot y la aplicación Android creada y para simular el uso de la botonera de entradas/salidas digitales disponible en el laboratorio de prácticas del departamento.

Las aplicaciones didácticas que se desarrollaran son el famoso juego de "Tres en Raya" y una aplicación para mover el robot. Mediante el uso de variables persistentes se consigue la comunicación entre tareas y por medio de las aplicaciones Android, que veremos más adelante, podremos interactuar con el programa creado en RobotStudio.

5.2 JUEGO DE LAS "TRES EN RAYA"

Mediante la programación en RAPID del conocido juego de las "Tres en Raya" podremos comprobar que la comunicación mediante sockets se realiza correctamente y que la estación simulada se comporta de igual manera que la estación real de trabajo.

5.2.1 Organización de la estación de trabajo

La organización de la estación de trabajo se realiza de tal forma que la mesa negra se sitúa en la parte central de la célula; mientras que los cubos que simulan las piezas de cada jugador, se sitúan al lado derecho y al lado izquierdo de la mesa negra. De esta manera, habrá 5 piezas de color verde correspondientes a las piezas del primer turno de juego y otras 5 piezas de color rojo correspondientes a las piezas del segundo turno de juego. En la Figura 5.1 se puede observar la organización de la estación de trabajo en RobotStudio.

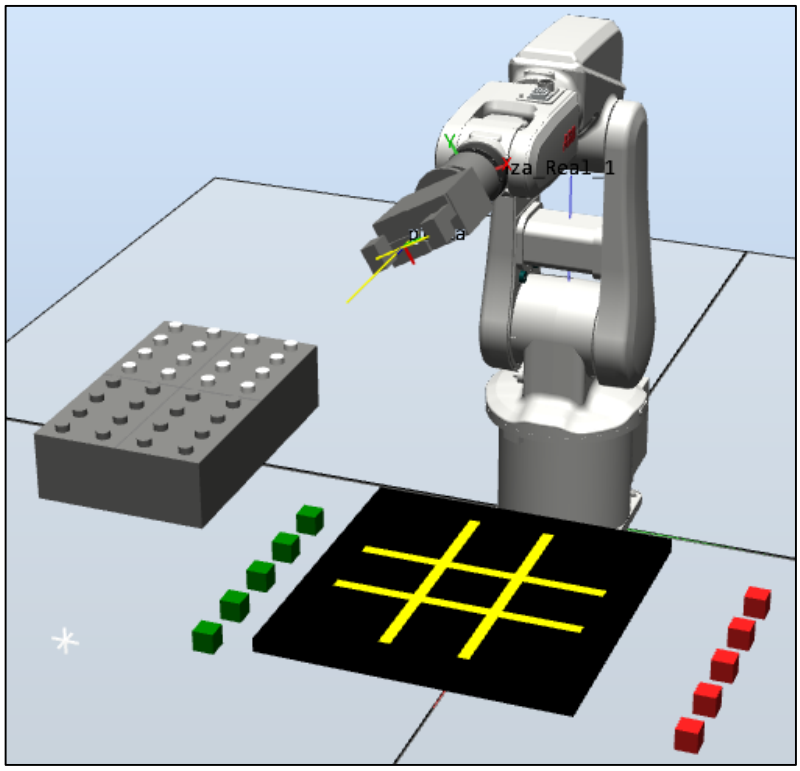


Figura 5.1: Estación de trabajo juego "Tres en Raya"

5.2.2 Simulación del juego

Al iniciar la simulación en RobotStudio del juego de las “Tres en Raya”, nos aparece en la ventana del operador de RobotStudio un mensaje que nos pide elegir la calibración de la estación (ver Figura 5.2).

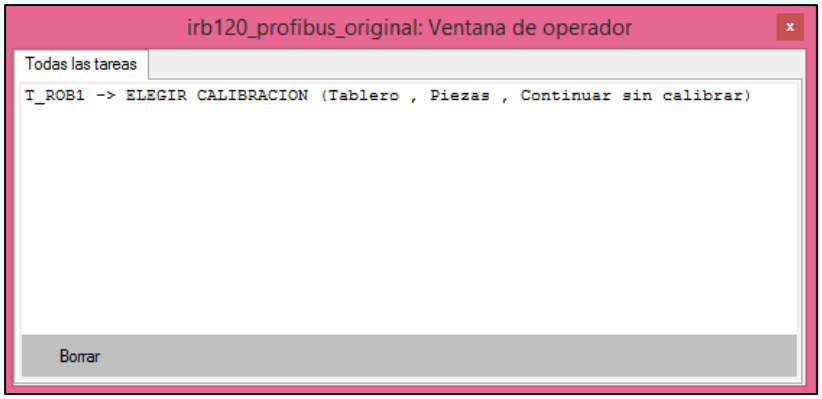


Figura 5.2: Ventana de operador (Elegir calibración)

Las opciones que tenemos en la ventana de calibración son:

- **Calibración del tablero:** se ejecuta una calibración del tablero para asegurarnos de que no se encuentra ningún obstáculo en él antes de comenzar la partida.
- **Calibración de las piezas:** se ejecuta una calibración de las posiciones de las piezas en la estación. Para ello el robot se posiciona en las posiciones iniciales donde deberían estar las piezas y comprueba que estas se encuentren en la posición exacta.
- **Continuar sin calibrar:** no se ejecuta ninguna calibración y avanza al siguiente paso.

Para elegir cualquiera de las opciones anteriores debemos mandarle desde la aplicación Android una serie de caracteres específicos. En la figura 5.3 se puede observar el código de RAPID que verifica la cadena de caracteres enviada por la aplicación y en función de la cadena recibida da un valor a la variable "opcion_calibracion", que se puede compartir entre los diferentes módulos de la tarea "T_ROB1" y que controla el flujo de la secuencia de calibración.

```
IF receivedString="T_CalibrarTablero\0A" THEN

     TPWrite "CALIBRAR TABLERO SELECCIONADO";
     opcion_calibracion:=1;

ELSEIF receivedString="T_CalibrarPiezas\0A" THEN

    TPWrite "CALIBRAR PIEZAS SELECCIONADO";
    opcion_calibracion:=2;

ELSEIF receivedString="T_Continuar\0A" THEN

    opcion_calibracion:=3;
```

Figura 5.3: Código RAPID para calibración de la estación

Tras la calibración de las piezas y del tablero la siguiente opción que se muestra en la ventana del operador es la elección de los jugadores. Se puede elegir entre el usuario, un robot inteligente o un robot aleatorio (ver Figura 5.4).

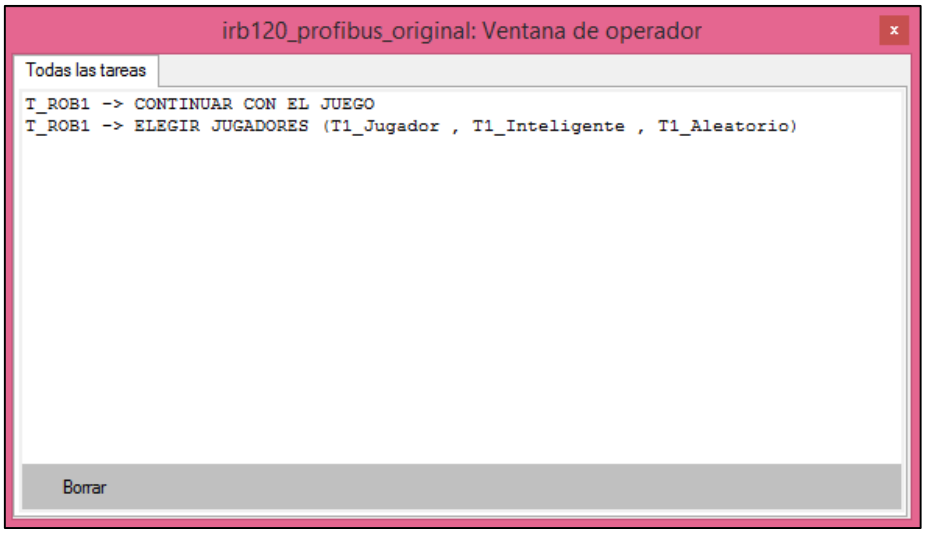


Figura 5.4: Ventana de operador (Elegir jugador)

Las opciones que tenemos en la ventana de selección de jugador son:

- **Jugador:** el usuario de la aplicación Android será el encargado de decidir las posiciones de cada jugada en el tablero del juego.
- **Robot inteligente:** el programa en RAPID posee un algoritmo que se encarga de decidir la mejor posición posible en el tablero de juego. El mejor resultado posible al jugar contra el robot inteligente es un empate.
- **Robot aleatorio:** el programa en RAPID posee un algoritmo que se encarga de decidir la posición en el tablero de juego de forma aleatoria, eso sí siempre intentando hacer la mejor jugada posible para ganar.

Para elegir cualquiera de las opciones anteriores debemos mandarle desde la aplicación Android una serie de caracteres específicos. En la figura 5.5 se puede observar el código de RAPID que verifica la cadena de caracteres enviada por la aplicación y en función de la cadena recibida da un valor a la variable “opción_menu1” para el caso del jugador 1 y “opción_menu2”, que se pueden compartir entre los diferentes módulos de la tarea “T_ROB1”.

```
ELSEIF receivedString="T1_Jugador\0A" THEN

    TPWrite "JUGADOR 1 ELEGIDO";
    opcion_menu1:=1;

ELSEIF receivedString="T1_Inteligente\0A" THEN

    TPWrite "ROBOT 1 INTELIGENTE ELEGIDO";
    opcion_menu1:=2;

ELSEIF receivedString="T1_Inteligente\0A" THEN

    TPWrite "ROBOT 1 ALEATORIO ELEGIDO";
    opcion_menu1:=3;

 ELSEIF receivedString="T2_Jugador\0A" THEN

    TPWrite "JUGADOR 2 ELEGIDO";
    opcion_menu2:=1;

ELSEIF receivedString="T2_Inteligente\0A" THEN

    TPWrite "ROBOT 2 INTELIGENTE ELEGIDO";
    opcion_menu2:=2;

ELSEIF receivedString="T2_Aleatorio\0A" THEN

    TPWrite "ROBOT 2 ALEATORIO ELEGIDO";
    opcion_menu2:=3;
```

Figura 5.5: Código RAPID para la selección de los jugadores

Una vez se han seleccionado los jugadores 1 y 2, la partida comienza. Si el jugador es el usuario de la aplicación Android, éste deberá introducir por pantalla la posición donde quiere colocar su dado y el robot se encargará de manipular el dado para colocarlo en la posición deseada. En caso de que el jugador sea el algoritmo del robot inteligente o el robot aleatorio, éste escogerá la opción que el algoritmo considera más adecuada y el robot se encargará de manipular el dado hasta su posición en el tablero de juego. En la Figura 5.6 se puede observar la ventana de simulación de RobotStudio y la ventana del operador, desde donde se puede seguir la ejecución del juego.

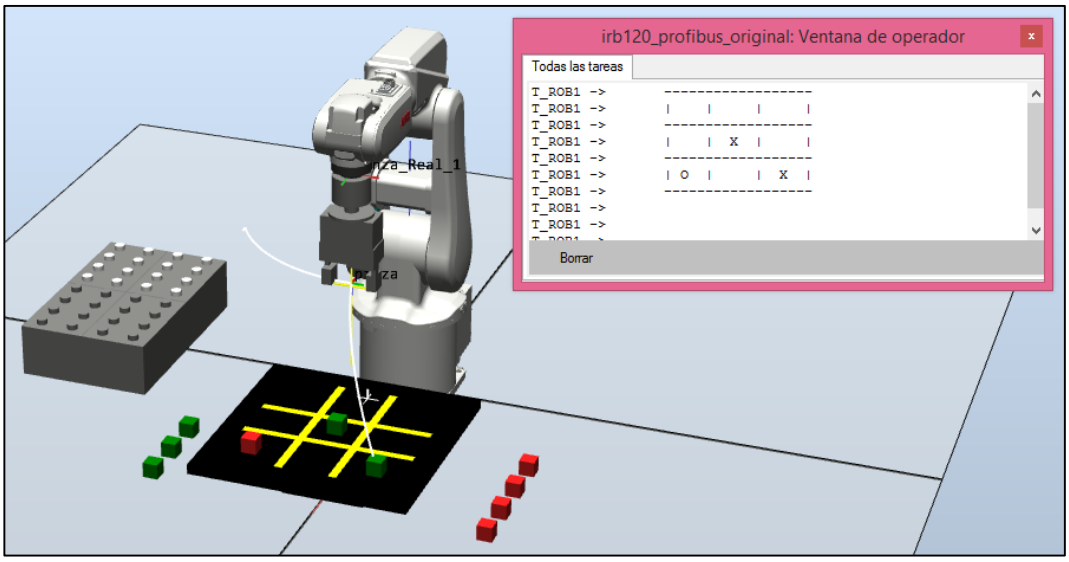


Figura 5.6: Simulación juego "Tres en Raya"

La selección de la jugada elegida por el usuario de la aplicación Android se envía mediante Sockets por medio de una cadena de caracteres que el código RAPID verifica y en función de la cadena recibida da un valor a la variable "jugada", que se puede compartir entre los diferentes módulos de la tarea "T_ROB1". En la figura 5.7 se puede observar el código de RAPID que verifica la cadena de caracteres enviada por la aplicación para asignar el valor de la variable "jugada".

```
ELSEIF receivedString="Jugada_1\0A" THEN

    TPWrite "NÚMERO 1 ELEGIDO";
    jugada:=1;

ELSEIF receivedString="Jugada_2\0A" THEN

    TPWrite "NÚMERO 2 ELEGIDO";
    jugada:=2;

ELSEIF receivedString="Jugada_3\0A" THEN

    TPWrite "NÚMERO 3 ELEGIDO";
    jugada:=3;
```

Figura 5.7: Código RAPID para la selección de la jugada del usuario

Cuando la partida finaliza, se muestra en la ventana del operador el resultado de la partida y se nos pregunta si queremos que el robot recoja las piezas (ver Figura 5.8).

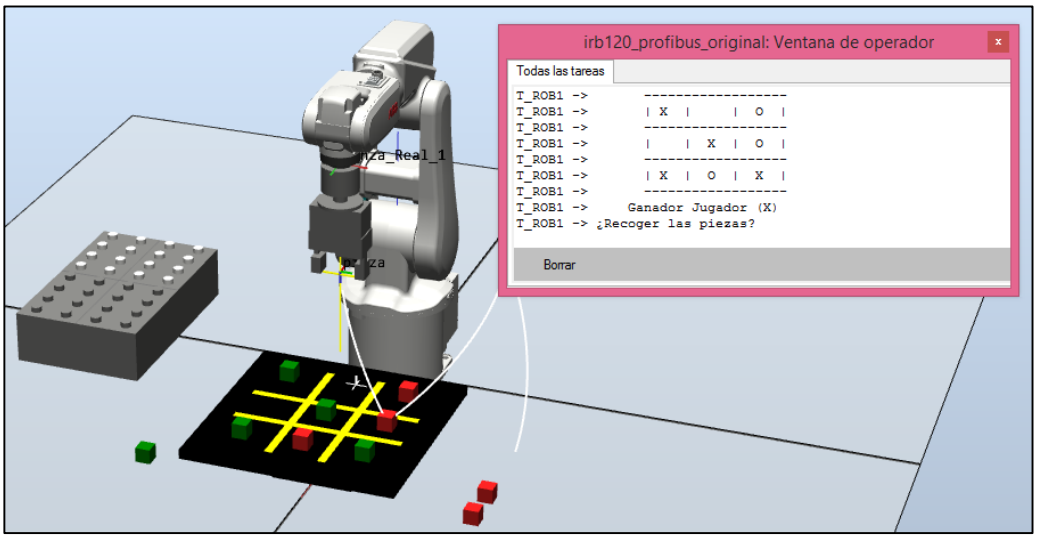


Figura 5.8: Fin juego "Tres en Raya" (Recoger Piezas)

En la figura 5.9 se puede observar el código de RAPID que verifica la cadena de caracteres enviada por la aplicación y en función de la cadena recibida da un valor a la variable “opcion_recoger” que se puede compartir entre los diferentes módulos de la tarea “T_ROB1”. Si elegimos la opción de recoger las piezas, el robot coloca todas las piezas en su posición inicial y la estación de trabajo se encontrará otra vez en condiciones de empezar una partida nueva.

```
ELSEIF receivedString="Recoge_si\0A" THEN

    TPWrite "Recoger piezas del tablero";
    opcion_recoger:=1;

ELSEIF receivedString="Recoge_no\0A" THEN

    TPWrite "No se recogeran las piezas.";
    opcion_recoger:=2;
```

Figura 5.9: Código RAPID recoger las piezas

5.2.3 Diagrama de flujo del juego

A continuación de muestra el diagrama de flujo simplificado de la ejecución del juego de las “Tres en Raya”.

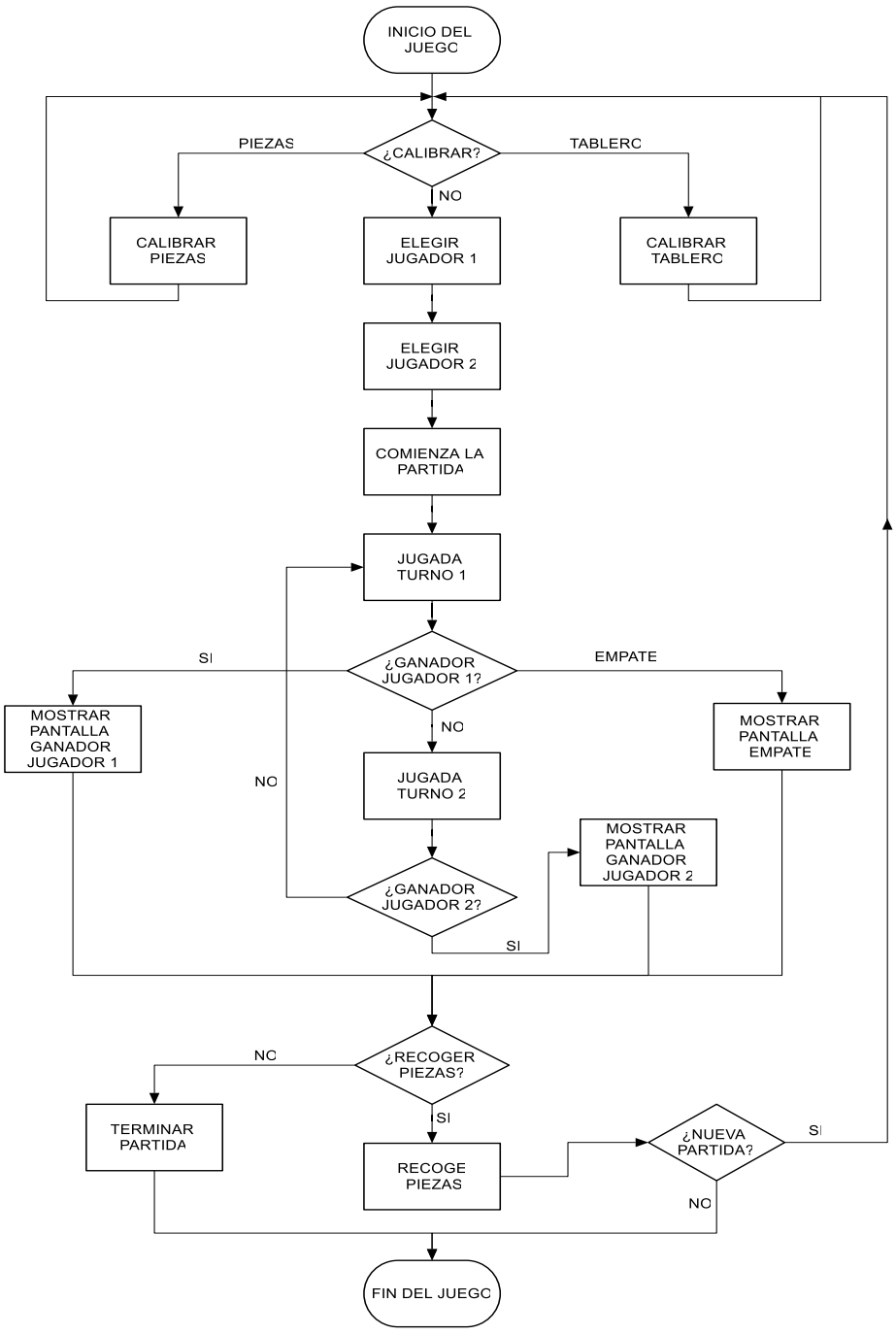


Diagrama 5.1: Diagrama simplificado del juego "Tres en Raya"

5.3 APLICACIÓN PARA MOVER EL ROBOT

En este apartado se explicará la aplicación desarrollada en RAPID con el objetivo de mover manualmente el robot ABB IRB 120. Cuando el robot se encuentre en posición manual, éste se puede controlar con el joystick del FlexPendant. El objetivo de esta aplicación es mover el robot manualmente y utilizar las propiedades de la pinza de agarre, pero en vez de utilizar el joystick del FlexPendant, usar una aplicación Android que simule el panel externo de E/S disponible en el laboratorio de prácticas. Antes de realizar el movimiento indicado, el programa RAPID será capaz de detectar si el movimiento es viable y realizándolo en caso de que lo sea.

5.3.1 Organización de la estación de trabajo

En esta ocasión, al ser una aplicación muy general y que puede mover el robot con libertad la organización de la estación no es de mucha importancia por lo que escogemos una organización de la estación de trabajo donde estén disponibles todos los elementos modelados. En la Figura 5.10 se puede observar la organización de la estación de trabajo.

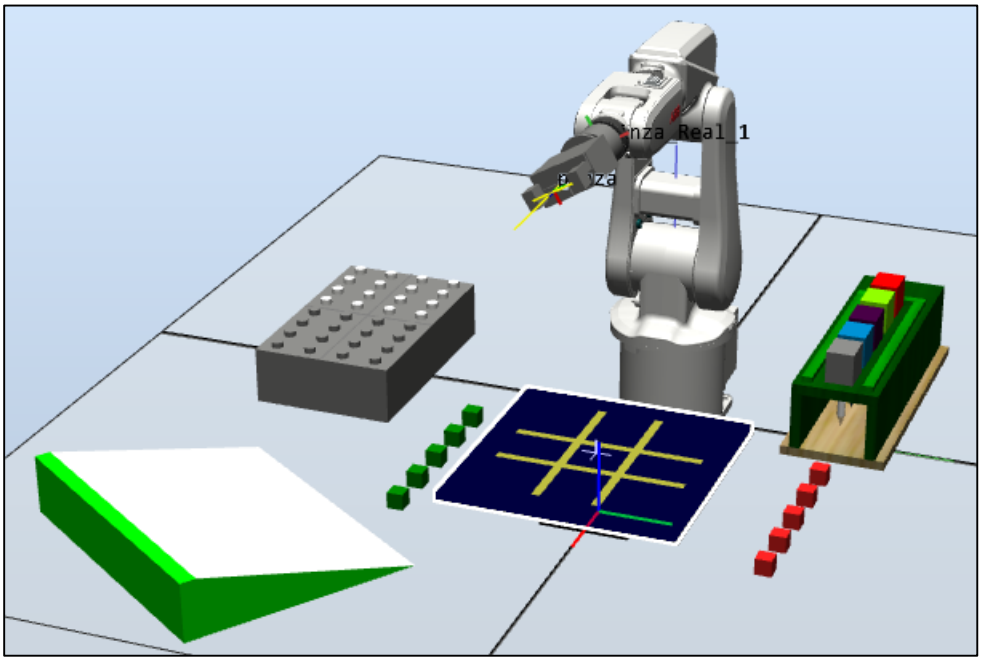


Figura 5.10: Organización estación aplicación mover el robot

La estación de trabajo como podemos observar está compuesta por:

- Mesa negra
- Mesa inclinada
- 10 cubos
- Soporte de rotuladores
- 5 rotuladores
- Panel externo de E/S

5.3.2 Simulación de la aplicación

Los tipos de movimientos que se permiten hacer con el joystick del FlexPendant son los siguientes:

- **Ejes**: El movimiento se robot se realiza a través de los ejes 1-3 o 4-6 del robot con el joystick.
- **Lineal**: El robot se mueve linealmente respecto de un sistema de referencia elegido.
- **Reorientación**: El robot se mueve en torno al TCP con distintas orientaciones.

Uno de los problemas que se presentan a la hora de realizar simulaciones en RobotStudio es la imposibilidad de usar el joystick del FlexPendant virtual para ejecutar movimientos del robot. Por eso, es interesante para determinadas aplicaciones didácticas poder mover el robot durante la simulación de manera manual, mediante el uso de una aplicación Android que simula el comportamiento del panel externo de E/S disponible en el laboratorio de prácticas.

En la aplicación realizada para mover el robot se podrán realizar el movimiento por ejes y el movimiento lineal respecto del sistema de referencia mundo. Además el robot se podrá mover con diferentes velocidades a través de selector de velocidades que almacena la velocidad que se le indica en una variable persistente.

Al iniciar la simulación lo primero que se hace es declarar las variables persistentes que se utilizarán para comunicarse entre la tarea T_ROB1 y la tarea T_COM. La tarea T_ROB1 tiene acceso al movimiento del robot; mientras que la tarea T_COM tiene como finalidad establecer la comunicación con la aplicación Android mediante socket.

La tarea T_COM será la encargada de cambiar el valor de las variables persistentes según la cadena de caracteres que se le envíen, y al ser variables persistentes los cambios de éstas pueden ser vistos desde las dos tareas, siendo el uso de estas variables persistentes la única forma de comunicarse entre las tareas T_ROB1 y T_COM.

Un cambio en alguna de las variables persistentes declaradas para ambas tareas inicia el tratamiento de la rutina TRAP correspondiente para cada variable, que serán declaradas en el procedimiento "Conexión". En la Figura 5.11 se puede observar la declaración de variables persistentes utilizadas por la tarea T_ROB1 y el procedimiento "main" de la aplicación, que accede la declaración de las rutinas de tratamiento de interrupciones mediante el procedimiento "Conexion". Mediante el procedimiento "Inicio" y "Movimiento" el robot se prepara y se coloca en la posición inicial declarada.

```
! Variables de conexión con la partición de comunicación
PERS bool pers_di{16}:=[FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE];
PERS num ref_seleccion:=1;
PERS num Vel_Act:=1;


PROC main_MANUAL_PERS()

    pers_di:=[FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE];
    Conexion;
    Inicio;
    Movimiento;

ENDPROC
```

Figura 5.11: Declaración de variables e inicio tarea T_ROB1

Las 16 variables persistentes "pers_di" de tipo booleano se corresponden y se comportan como si fueran los 16 botones del panel externo de E/S que se simularan mediante el uso de una aplicación Android. La variable persistente "ref_seleccion" almacenará el valor correspondiente al sistema de referencia de la herramienta que deseemos utilizar; mientras que la variable persistente "Vel_Act" almacenará la posición de un vector definido con las posibles velocidades a las que puede moverse el robot.

Si cualquiera de las 16 variables persistentes "pers_di" cambia de valor se inicia el tratamiento de una interrupción y dependiendo de cuál sea la variable afectada se puede realizar diferentes acciones. En la Figura 5.12 se muestra la relación de botones disponible en la aplicación y que acción ejecuta cada una.

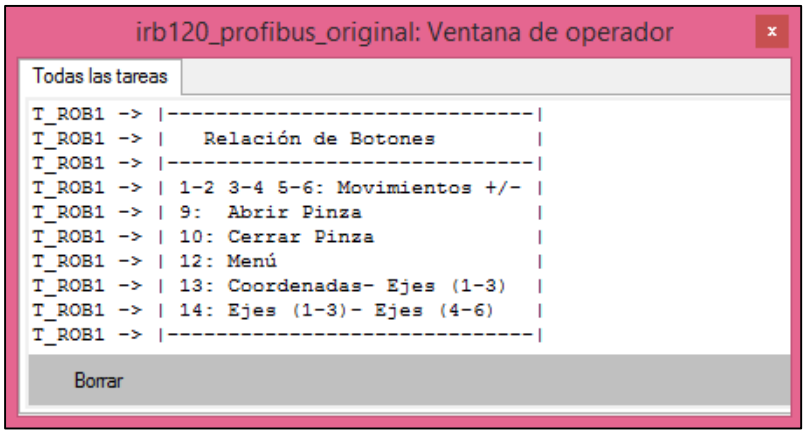


Figura 5.12: Relación de botones disponibles en RAPID

A continuación se muestra una tabla con las variables persistentes disponibles en el programa RAPID y una breve descripción de la función de la cada una.

VARIABLE PERSISTENTE	DESCRIPCIÓN
pers_di1	Si el movimiento seleccionado es mundo mueve el robot en sentido positivo en la dirección X. Si el movimiento seleccionado es ejes 1-3, mueve el eje 1 en sentido positivo; mientras que si el movimiento seleccionado es ejes 4-6, mueve el eje 4 en sentido positivo.
pers_di2	Si el movimiento seleccionado es mundo mueve el robot en sentido negativo en la dirección X. Si el movimiento seleccionado es ejes 1-3, mueve el eje 1 en sentido negativo; mientras que si el movimiento seleccionado es ejes 4-6, mueve el eje 4 en sentido negativo.
pers_di3	Si el movimiento seleccionado es mundo mueve el robot en sentido positivo en la dirección Y. Si el movimiento seleccionado es ejes 1-3, mueve el eje 2 en sentido positivo; mientras que si el movimiento seleccionado es ejes 4-6, mueve el eje 5 en sentido positivo.
pers_di4	Si el movimiento seleccionado es mundo mueve el robot en sentido negativo en la dirección Y. Si el movimiento seleccionado es ejes 1-3, mueve el eje 2 en sentido negativo; mientras que si el movimiento seleccionado es ejes 4-6, mueve el eje 5 en sentido negativo.

VARIABLE PERSISTENTE	DESCRIPCIÓN
pers_di5	Si el movimiento seleccionado es mundo mueve el robot en sentido positivo en la dirección Z. Si el movimiento seleccionado es ejes 1-3, mueve el eje 3 en sentido positivo; mientras que si el movimiento seleccionado es ejes 4-6, mueve el eje 6 en sentido positivo.
pers_di6	Si el movimiento seleccionado es mundo mueve el robot en sentido negativo en la dirección Z. Si el movimiento seleccionado es ejes 1-3, mueve el eje 3 en sentido negativo; mientras que si el movimiento seleccionado es ejes 4-6, mueve el eje 6 en sentido negativo.
pers_di9	Activa y desactiva el movimiento de apertura de la pinza
pers_di10	Activa y desactiva el movimiento de cierre de la pinza
pers_di12	Si estamos en simulación muestra en la ventana del operador el menú con la relación de botones; mientras que en la estación real lo muestra en la FlexPendant
pers_di13	Permite cambiar entre el movimiento con referencia (Mundo por defecto) y el movimiento por ejes
pers_di14	Permite cambiar entre el movimiento por ejes 1-3 y el movimiento por ejes 4-6
ref_seleccion	Permite cambiar el sistema de referencia de la herramienta. Dependiendo del valor se elige uno y otro sistema de referencia almacenados en RAPID
Vel_Act	Indica la posición de un vector que almacena las velocidades de movimiento disponibles en el robot.

Tabla 5.1: Descripción variables persistentes mover robot

Las variables persistentes 7 y 8 se han reservado ya que se corresponden con las señales utilizadas por el sensor laser de contacto de la pinza y con el sensor de presión que indica si una pieza ha sido cogida.

Para garantizar que el robot no sobrepasa los límites de sus ejes y se produzca algún fallo, antes de que el robot realice el movimiento, la función "Ejes_lim" verifica que el robot no va a sobrepasar los límites de sus ejes.

Si la función determina que se van a sobrepasar los límites, impide el movimiento y muestra un mensaje de error en la pantalla del FlexPendant.

5.3.3 Comunicación RAPID – Android

Para el desarrollo de la aplicación por un lado tenemos la comunicación entre la aplicación Android con la tarea T_COM y por otro lado la comunicación entre tareas que se lleva a cabo a partir de compartir las variables persistentes. La aplicación Android envía una cadena de caracteres a la tarea T_COM, que se encarga de desencriptar el mensaje y actualizar el valor de las variables persistentes. Al variar el valor de una variable persistente se ejecuta su correspondiente rutina de interrupción, haciendo posible mover el robot, cambiar la velocidad del movimiento, cambiar la referencia del movimiento y controlar la pinza de agarre. A continuación se muestra el diagrama de flujo simplificado de las tareas que se ejecutan en RAPID y la aplicación Android desarrollada.

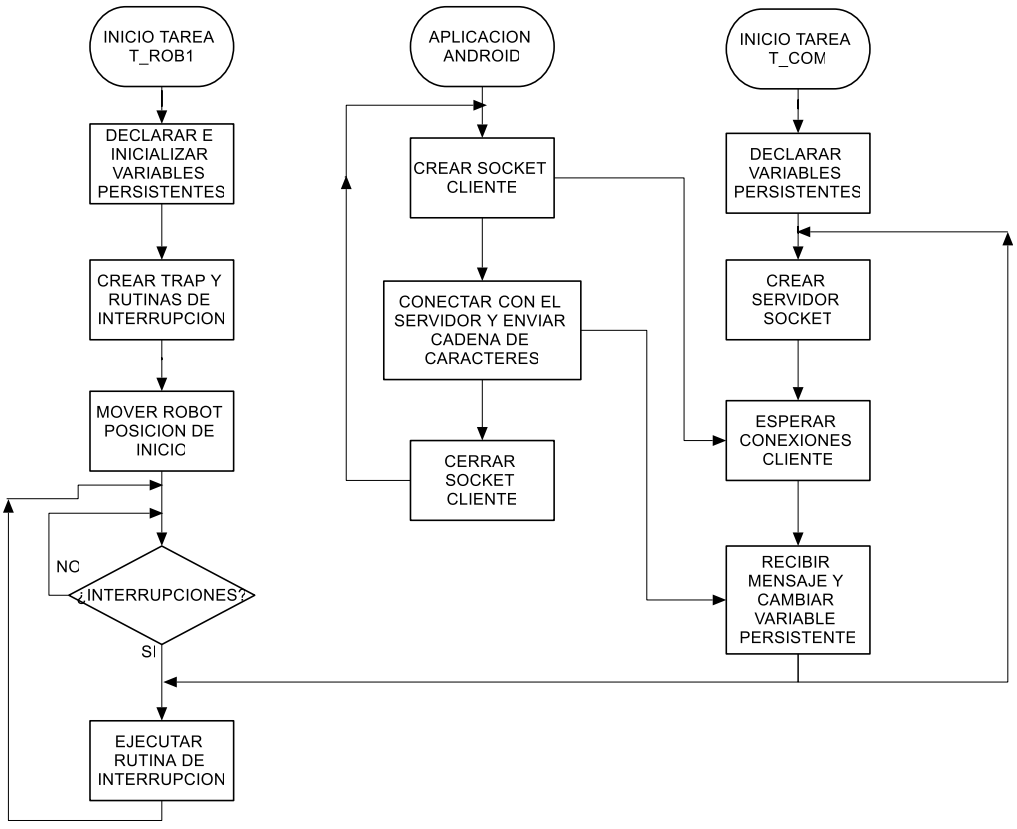


Diagrama 5.2: Comunicación entre tareas y aplicación Android

6 APLICACIONES ANDROID

6.1 INTRODUCCIÓN

En este capítulo se describirá de una manera general la estructura de las pantallas y los diferentes recursos utilizados para implementar las dos aplicaciones Android que se encargarán de mandar y recibir información al controlador del robot a través de la tecnología de comunicación inalámbrica Wi-Fi. Las dos aplicaciones Android desarrolladas son:

- Tres en Raya
- Control Robot

6.2 APLICACIÓN ANDROID TRES EN RAYA

La aplicación Android "Tres en Raya.apk" desarrollada tiene como finalidad interactuar con la aplicación didáctica del juego de las "Tres en Raya" de RobotStudio, y que el usuario sea capaz de controlar por completo la ejecución del programa mediante la comunicación entre ambas aplicaciones. Para la correcta comunicación entre la aplicación Android y el controlador del robot, el dispositivo Android donde se ejecuta la aplicación y el controlador del robot deben de estar conectados al mismo punto de acceso de red Wi-Fi.

La aplicación Android "Tres en Raya" está compuesta por una serie de pantallas "layout" con diferentes objetos visuales que mediante la oportuna programación son capaces de comunicarse con el controlador IRC5 compact mediante el uso de Sockets.

Las pantallas desarrolladas en la aplicación Android son las siguientes:

- Activity_cliente_android.xml
- Pantalla_calibrar.xml
- Pantalla_elegir_jugador1.xml
- Pantalla_elegir_jugador2.xml
- Pantalla_tres_raya_tablero.xml
- Pantalla_tres_raya_fin.xml

6.2.1 Activity_cliente_android.xml

La pantalla correspondiente con el archivo "Activity_cliente_android.xml" es la pantalla principal de la aplicación, y por tanto, es la primera pantalla que nos aparece al ejecutar la aplicación en nuestro dispositivo Android. En la Figura 6.1 se puede observar la estructura de esta pantalla.

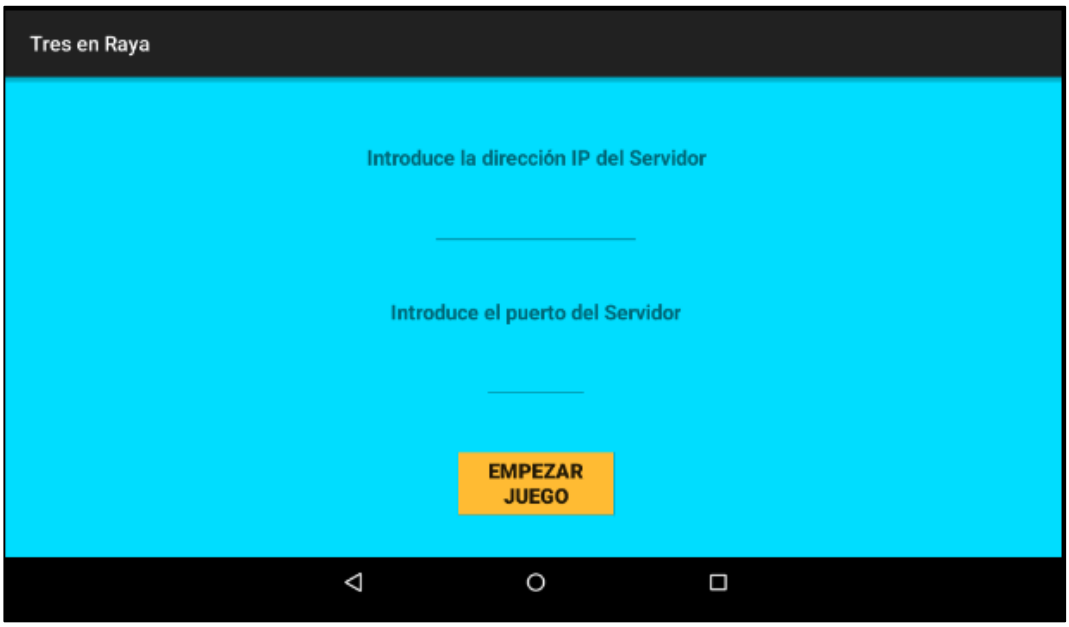


Figura 6.1: Pantalla Activity_cliente_android.xml

Esta pantalla está compuesta por dos cuadros de texto (TextView) que nos solicita que introduzcamos la dirección IP del servidor y el puerto al que nos conectaremos. La pantalla "Activity_cliente_android.xml" también dispone de dos cuadros de entrada de texto (EditText) donde podremos introducir la dirección IP del servidor y el puerto.

La dirección IP del servidor que hay que introducir es la dirección IP estática del controlador del robot ABB IRB 120 que se encuentra en el laboratorio de prácticas, cuando trabajemos con la estación real; mientras que si trabajamos con la estación simulada en RobotStudio, la dirección IP del servidor que habrá que introducir es la dirección IP del ordenador personal que estemos utilizando para ejecutar el software RobotStudio.

Por defecto, al iniciar la aplicación la dirección IP que aparece en los EditText es la del controlador del laboratorio de prácticas ("157.88.201.130") y el puerto utilizado el 80, que es un puerto de libre acceso TCP/IP proporcionado por la red "eduroam" de la Escuela de Ingenierías Industriales. Si queremos trabajar con la estación virtual podemos cambiar esta dirección IP y el puerto utilizado haciendo clic en el EditText correspondiente.

Una vez que ya hemos configurado la dirección IP y el puerto que queremos utilizar podemos acceder a la siguiente pantalla a través del botón "Empezar Juego" que se encuentra en la parte inferior de la pantalla. Al hacer clic en este botón los datos contenidos en los EditText de la dirección IP y el puerto se almacenan en las variables "IP" y "puerto" que podrán ser utilizadas posteriormente.

El diagrama de flujo de la pantalla "Activity_cliente_android.xml" es el siguiente:

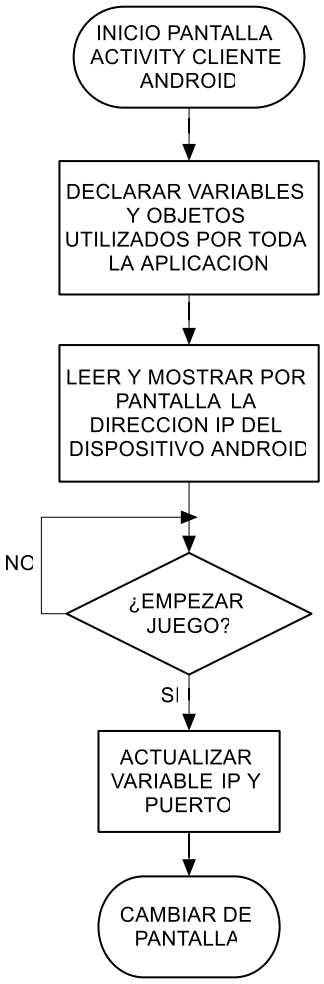


Diagrama 6.1: Pantalla Activity_cliente_android.xml

6.2.2 Pantalla_calibrar.xml

La siguiente pantalla que nos aparece en la aplicación Android es una pantalla donde podremos realizar la calibración de la estación de trabajo para el juego de las "Tres en Raya". Esta pantalla está compuesta por un cuadro de texto en la parte superior, que nos informa de la pantalla en la que estamos actualmente y de 5 botones. En la Figura 6.2 se puede observar el layout "Pantalla_calibrar.xml"

Figura 6.2: Pantalla_calibrar.xml

A continuación se explicará la función de cada uno de estos botones.

- **Calibrar Piezas:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que creará un socket cliente para conectarse al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "T_CalibrarPiezas" al servidor. El servidor al recibir la cadena la desencripta e inicia el calibrado de las piezas del juego.

Al terminar de mandar la cadena "T_CalibrarPiezas" se cierra el cliente socket, se espera un tiempo razonable para no saturar la comunicación y se vuelve a conectar un nuevo socket cliente al socket servidor.

Al terminar el robot IRB 120 el proceso de calibrar las piezas, el controlador envía al cliente socket la cadena de caracteres "Fin_calibrar". La aplicación Android en este momento muestra por pantalla un mensaje indicando que la calibración de las piezas ha terminado y continúa su ejecución normal en el hilo principal.

- **Calibrar Tablero:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que creará un socket cliente para conectarse al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "T_CalibrarTablero" al servidor. El servidor al recibir la cadena la desencripta e inicia el calibrado del tablero.

Al terminar de mandar la cadena "T_CalibrarTablero" se cierra el cliente socket, se espera un tiempo razonable para no saturar la comunicación y se vuelve a conectar un nuevo socket cliente al socket servidor.

Al terminar el robot IRB 120 el proceso de calibrar el tablero, el controlador envía al cliente socket la cadena de caracteres "Fin_calibrar". La aplicación Android en este momento muestra por pantalla un mensaje indicando que la calibración del tablero ha terminado y continúa su ejecución normal en el hilo principal.

- **Continuar:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "T_Continuar" al servidor. El servidor al recibir la cadena la desencripta y continua con el juego de las "Tres en Raya".

Al terminar de mandar la cadena "T_Continuar" se cierra el cliente socket y se continúa la ejecución de la aplicación, pasando esta forma a la siguiente pantalla del juego.

- **Regresar:**

Al hacer clic en este botón, la aplicación regresa a la pantalla "Activity_cliente_android.xml" e inicializa todas las variables y objetos utilizados por la aplicación

- **Salir**

Al hacer clic en este botón la aplicación se detiene y fuerza su salida al menú principal del dispositivo Android que estemos utilizando.

El diagrama de flujo del layout “Pantalla_calibrar.xml” es el siguiente:

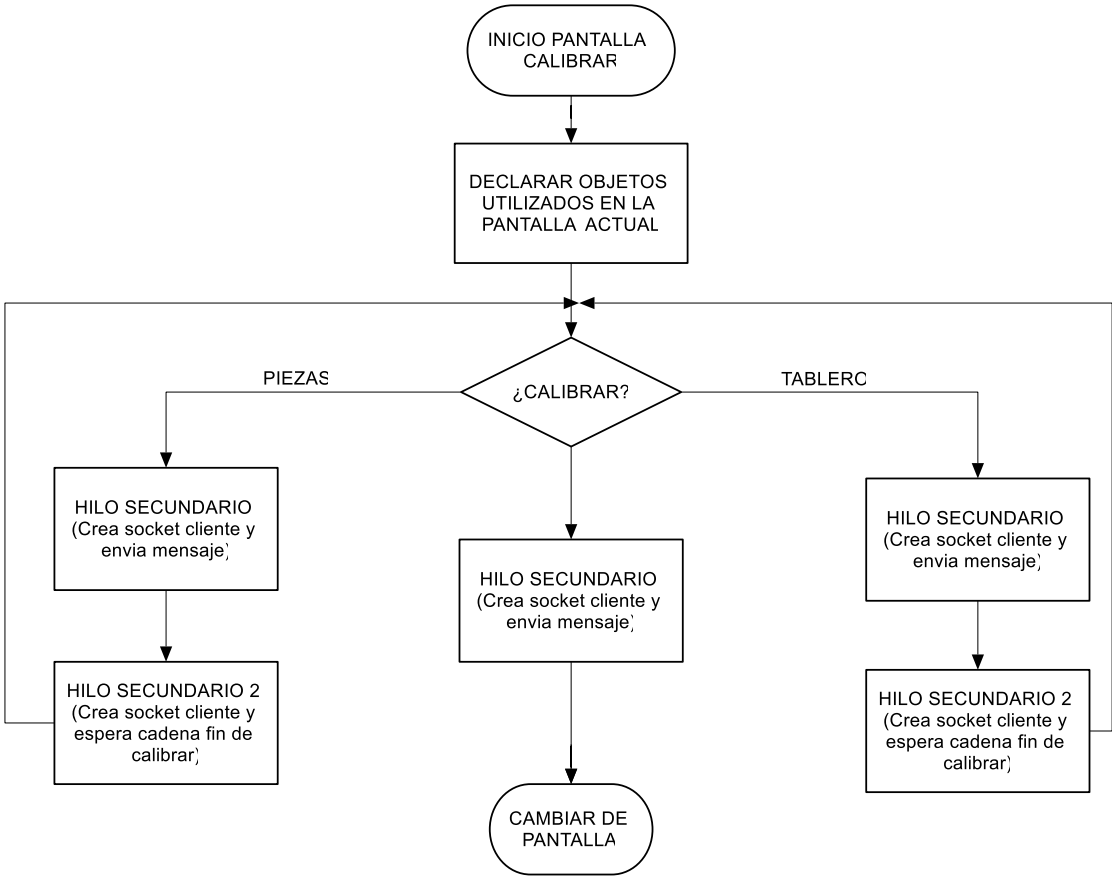


Diagrama 6.2: Pantalla_calibrar.xml

6.2.3 Pantalla_elegir_jugador1.xml

En la siguiente pantalla que nos aparece en la aplicación Android podremos elegir quien será el jugador del turno 1. Las opciones que nos ofrece la aplicación Android para el turno 1 son el usuario, un robot inteligente o un robot aleatorio.

Esta pantalla está compuesta por un cuadro de texto en la parte superior, que nos informa de la pantalla en la que estamos actualmente y de 5 botones. En la Figura 6.3 se puede observar la distribución del layout “Pantalla_elegir_jugador1.xml”.

Figura 6.3: Pantalla_elegir_jugador1.xml

A continuación se explicará la función de cada uno de estos botones.

- **Jugador 1:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "T1_Jugador" al servidor. El servidor al recibir la cadena la desencripta y asigna al turno 1 el usuario.

Al terminar de mandar la cadena "T1_Jugador" se cierra el cliente socket y se continúa la ejecución de la aplicación, pasando esta forma a la siguiente pantalla del juego.

- **Robot Inteligente 1:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "T1_Inteligente" al servidor. El servidor al recibir la cadena la desencripta y asigna al turno 1 el robot inteligente.

Al terminar de mandar la cadena "T1_Inteligente" se cierra el cliente socket y se continúa la ejecución de la aplicación, pasando esta forma a la siguiente pantalla del juego.

- **Robot Aleatorio 1:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "T1_Aleatorio" al servidor. El servidor al recibir la cadena la desencripta y asigna al turno 1 el robot inteligente.

Al terminar de mandar la cadena "T1_Aleatorio" se cierra el cliente socket y se continúa la ejecución de la aplicación, pasando esta forma a la siguiente pantalla del juego.

- **Regresar:**

Al hacer clic en este botón, la aplicación regresa a la pantalla "Activity_cliente_android.xml" e inicializa todas las variables y objetos utilizados por la aplicación

- **Salir**

Al hacer clic en este botón la aplicación se detiene y fuerza su salida al menú principal del dispositivo Android que estemos utilizando.

El diagrama de flujo del layout "Pantalla_elegir_jugador1.xml" es el siguiente:

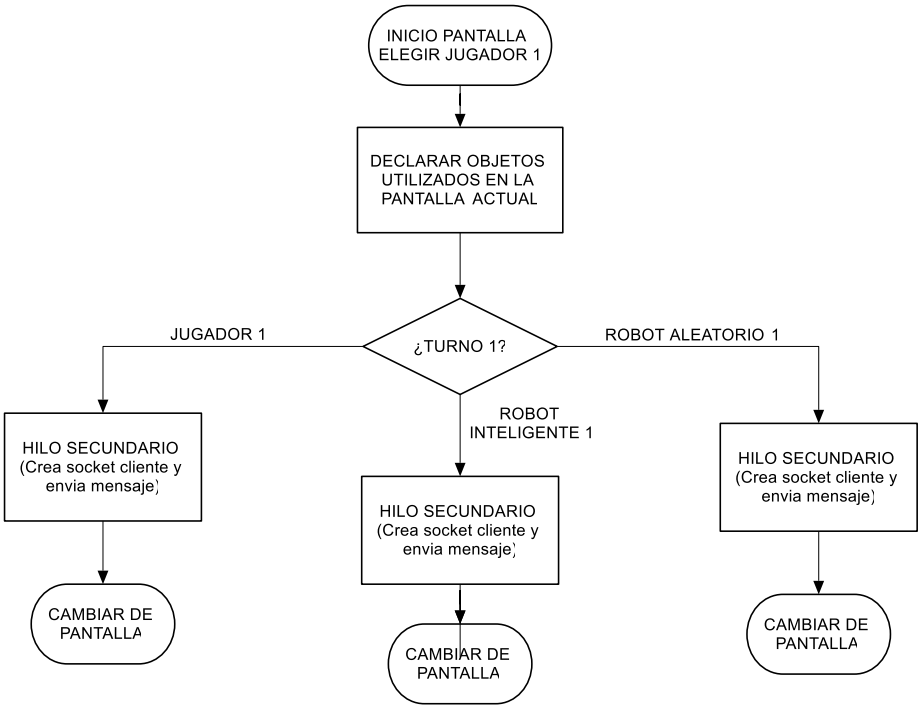


Diagrama 6.3: Pantalla_elegir_jugador1.xml

6.2.4 Pantalla_elegir_jugador2.xml

En la siguiente pantalla que nos aparece en la aplicación Android podremos elegir quien será el jugador del turno 2. Las opciones que nos ofrece la aplicación Android para el turno 2 son el usuario, un robot inteligente o un robot aleatorio.

Esta pantalla está compuesta por un cuadro de texto en la parte superior, que nos informa de la pantalla en la que estamos actualmente y de 5 botones. En la Figura 6.4 se puede observar la distribución del layout "Pantalla_elegir_jugador2.xml".

Figura 6.4: Pantalla_elegir_jugador2.xml

A continuación se explicará la función de cada uno de estos botones.

- **Jugador 2:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "T2_Jugador" al servidor. El servidor al recibir la cadena la desencripta y asigna al turno 2 el usuario.

Al terminar de mandar la cadena “T2_Jugador” se cierra el cliente socket y se continúa la ejecución de la aplicación, pasando esta forma a la siguiente pantalla del juego.

- **Robot Inteligente 2:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente para conectarse al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres “T2_Inteligente” al servidor. El servidor al recibir la cadena la desencripta y asigna al turno 2 el robot inteligente.

Al terminar de mandar la cadena “T2_Inteligente” se cierra el cliente socket y se continúa la ejecución de la aplicación, pasando esta forma a la siguiente pantalla del juego.

- **Robot Aleatorio 2:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente para conectarse al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres “T2_Aleatorio” al servidor. El servidor al recibir la cadena la desencripta y asigna al turno 2 el robot inteligente.

Al terminar de mandar la cadena “T2_Aleatorio” se cierra el cliente socket y se continúa la ejecución de la aplicación, pasando esta forma a la siguiente pantalla del juego.

- **Regresar:**

Al hacer clic en este botón, la aplicación regresa a la pantalla “Activity_cliente_android.xml” e inicializa todas las variables y objetos utilizados por la aplicación

- **Salir**

Al hacer clic en este botón la aplicación se detiene y fuerza su salida al menú principal del dispositivo Android que estemos utilizando.

El diagrama de flujo del layout "Pantalla_elegir_jugador2.xml" es el siguiente:

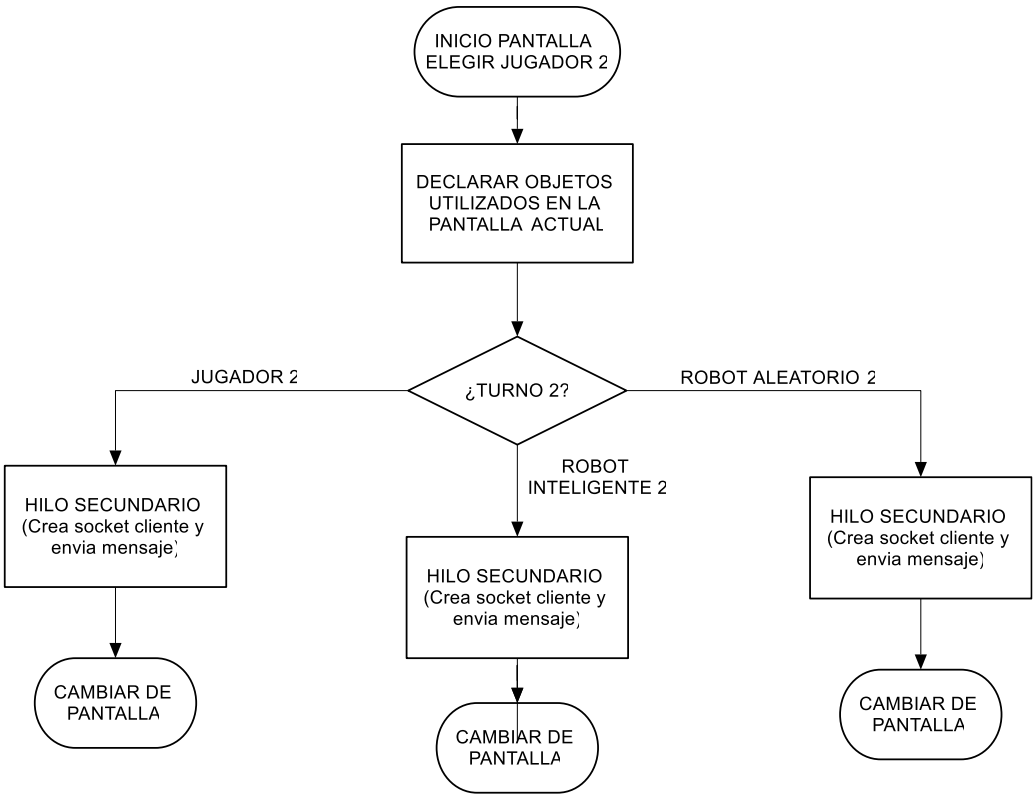


Diagrama 6.4: Pantalla_elegir_jugador2.xml

6.2.5 Pantalla_tres_raya_tablero.xml

La siguiente pantalla que nos aparece en la aplicación es el tablero de juego. Esta pantalla está compuesta por un cuadro de texto en la parte superior, que nos informa de la pantalla en la que estamos actualmente, por 9 botones con la imagen de números del 1 al 9 que representan las posiciones del tablero y los 2 botones para regresar a la pantalla principal y salir de la aplicación. En la Figura 6.5 se puede observar la distribución del layout "Pantalla_tres_raya_tablero.xml".

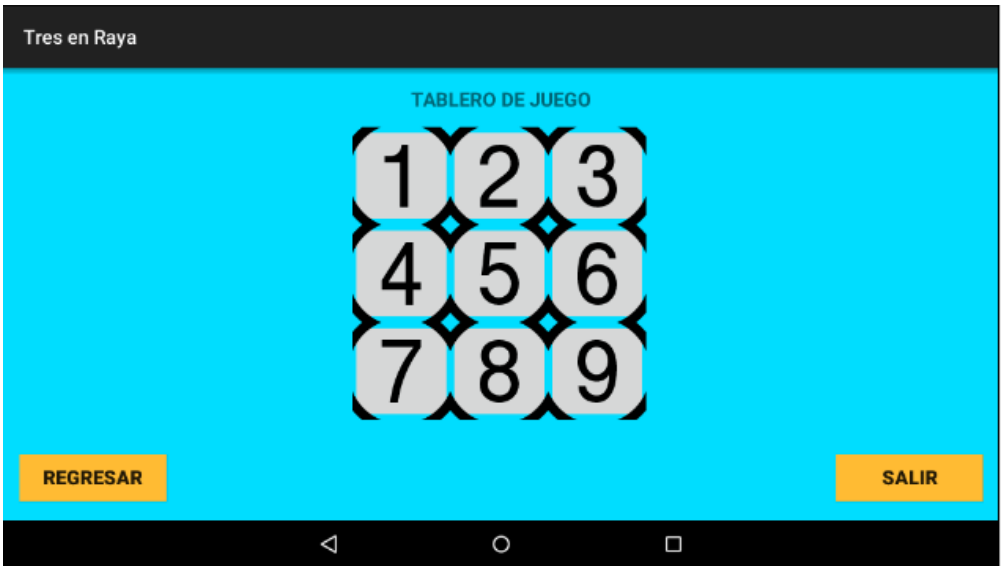


Figura 6.5: Pantalla_tres_raya_tablero.xml

A continuación se explicará la función de cada uno de estos botones.

- **Números:**

Los botones de los números solo reaccionarán al pulsarles sobre la pantalla del dispositivo Android si el turno lo tiene el usuario que maneja la aplicación. Al hacer clic en el botón de un número, cambia la imagen del botón a una cruz de color verde, para indicar la elección del usuario. El siguiente paso de la aplicación es crear un hilo secundario de ejecución que se encargará de enlazar un socket cliente al socket servidor del controlador.

Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "Jugada_X" al servidor, donde la X representa el número elegido en la pantalla. El servidor al recibir la cadena la desencripta y mueve los cubos que simulan las piezas del juego a la posición indicada por el tablero.

Al terminar de mandar la cadena "Jugada_X" se cierra el cliente socket, se espera un tiempo razonable para no saturar la comunicación y se vuelve a conectar un nuevo socket cliente al socket servidor.

Al terminar el robot IRB 120 el movimiento del turno 1, se comprueba si alguno de los jugadores ha ganado la partida, y en caso de que la partida continúe, el controlador envía al cliente socket la cadena de caracteres "Seguir". La aplicación Android en este momento muestra por pantalla un mensaje indicando que el turno del jugador 1 ha terminado.

Si el turno 2 pertenece al robot aleatorio o al robot inteligente, la ejecución de la aplicación Android y del programa en RobotStudio continúa automáticamente. La aplicación Android crea un hilo secundario de ejecución que se encargará de enlazar un socket cliente con el socket servidor del controlador. El programa de simulación continúa y el robot manipula la pieza a la posición elegida por el algoritmo del robot elegido. Una vez el robot coloca la pieza en la posición correspondiente, el socket servidor (controlador) envía la cadena de caracteres "Jugada_X" al cliente. La aplicación Android al recibir la cadena, la desencripta y cambia la imagen del botón correspondiente al número enviado por el servidor por un aspa de color rojo, que indica la elección del robot.

Al terminar el robot IRB 120 el movimiento del turno 2, se comprueba si alguno de los jugadores ha ganado la partida, y en caso de que la partida continúe, el controlador envía al cliente socket la cadena de caracteres "Seguir". La aplicación Android en este momento muestra por pantalla un mensaje indicando que el turno del jugador 2 ha terminado, cambiando de nuevo el turno de movimiento para el usuario 1. En la Figura 6.6 se puede observar una captura de imagen de la aplicación Android durante la ejecución de la simulación del juego de las "Tres en Raya".

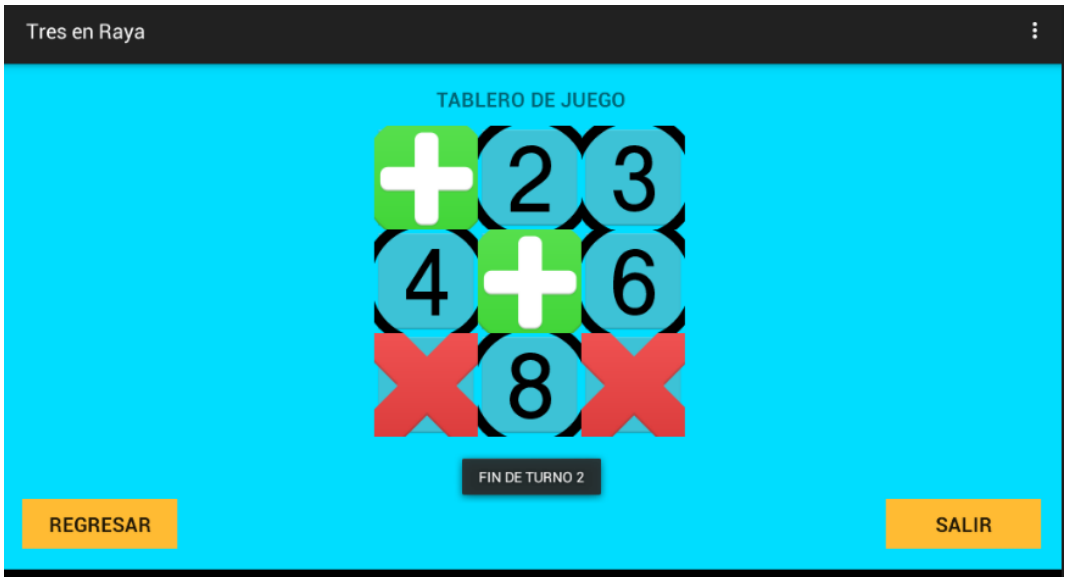


Figura 6.6: Pantalla simulación Tres en Raya

El ciclo completo de movimiento descrito para el movimiento de las fichas de los 2 jugadores se repite hasta el momento en el que el resultado de la partida sea empate, ganador el turno 1 o ganador el turno 2. Si se produce cualquiera de estas situaciones, el socket servidor (controlador) envía la cadena de caracteres “Empate” o “Ganador + (nombre del turno)” al cliente. La aplicación Android en este momento cambia de pantalla y en la nueva pantalla muestra los resultados finales de la partida.

El diagrama de flujo del layout “Pantalla_tres_raya_tablero.xml” es el siguiente:

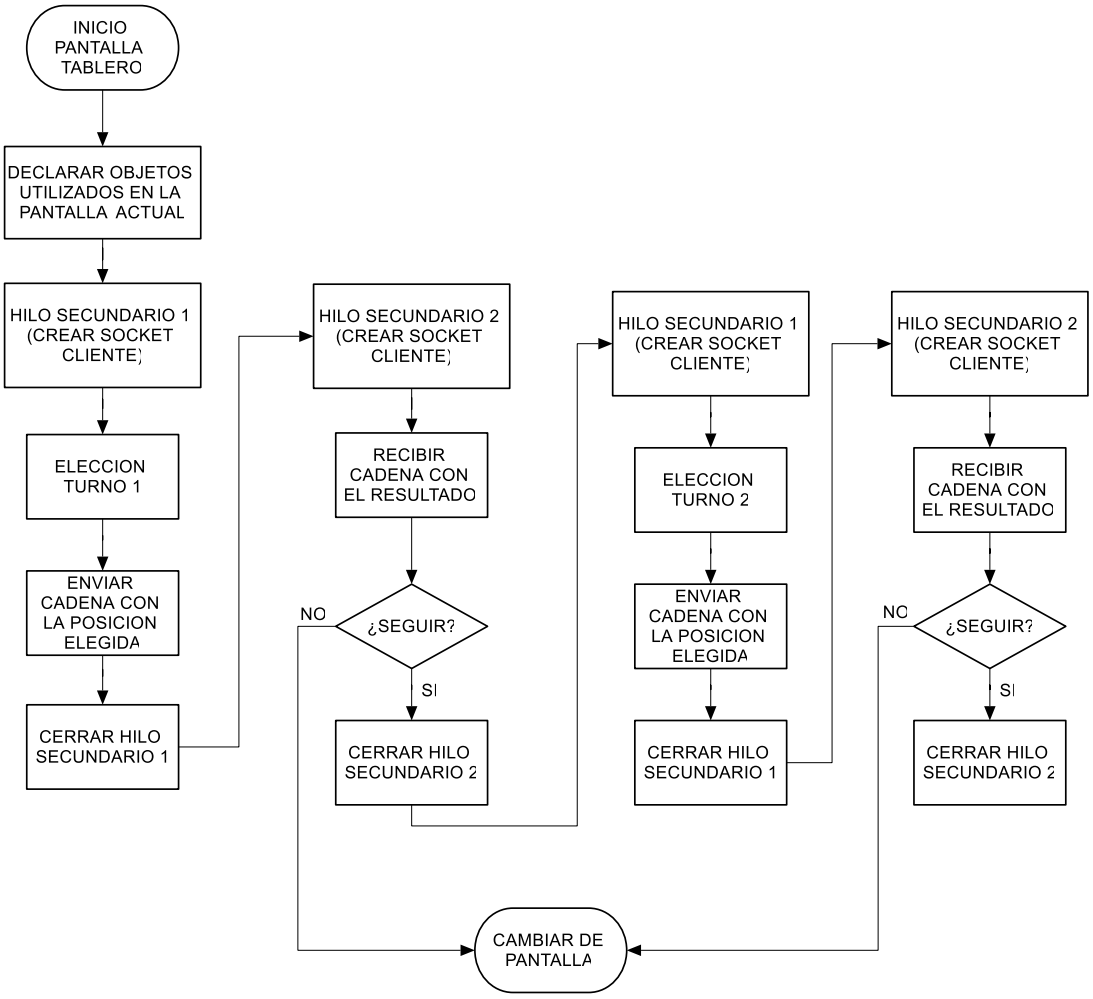


Diagrama 6.5: Pantalla_tres_raya_tablero.xml

6.2.6 Pantalla_tres_raya_fin.xml

La siguiente pantalla que nos aparece en la aplicación es la encargada de mostrar al usuario el resultado final de la partida, además de preguntar al usuario si desea que el robot IRB 120 recoja las piezas del tablero y las coloque en sus posiciones iniciales.

Esta pantalla está compuesta por tres cuadros de texto (TextView). El primero de ellos se encuentra en la parte superior y nos informa de la pantalla en la que estamos actualmente. El segundo cuadro de texto nos informa del ganador de la partida o muestra "Empate" en caso de que todas las piezas se hayan colocado en el tablero y ninguno de los jugadores haya resultado ganador. El tercer cuadro de texto pregunta al usuario si desea recoger las piezas del tablero. La aplicación está compuesta también por 5 botones. En la Figura 6.7 se puede observar la distribución de "Pantalla_tres_raya_fin.xml".

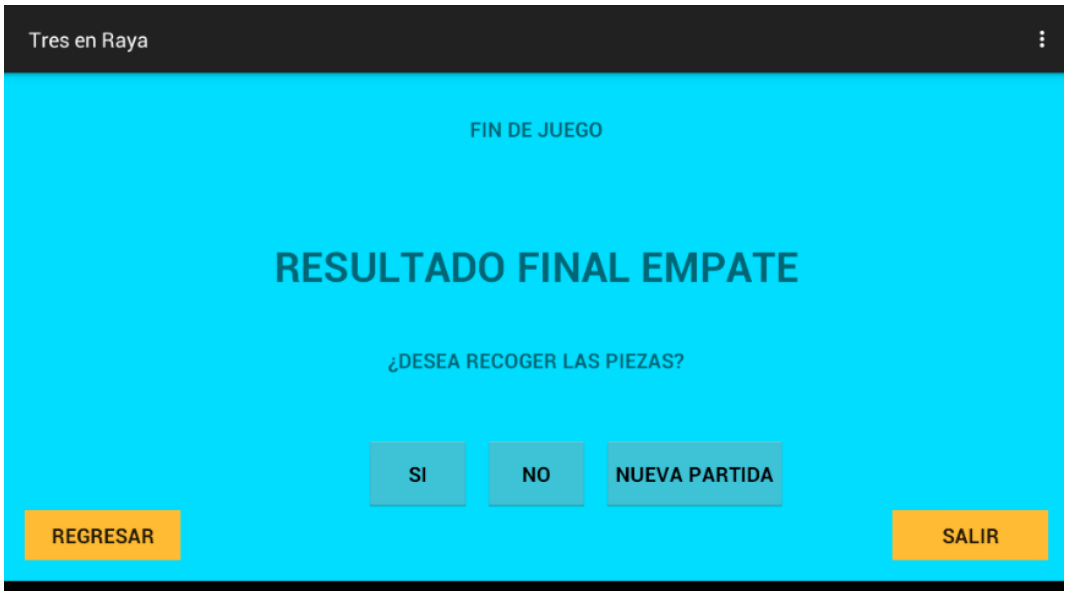


Figura 6.7: Pantalla_tres_raya_fin.xml

A continuación se explicará la función de cada uno de estos botones.

- **Recoger SI:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente al socket servidor del controlador, además de bloquear el resto de botones de la pantalla para que no interfieran las operaciones que selecciona cada uno de ellos cuando se ejecuta la recogida de las piezas.

Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "Recoge_si" al servidor. El servidor al recibir la cadena la desencripta y el robot IRB 120 comienza a recoger las piezas que se encuentra en el tablero, devolviéndolas a su posición inicial.

Al terminar de mandar la cadena "Recoge_si" se cierra el cliente socket, se espera un tiempo razonable para no saturar la comunicación y se vuelve a conectar un nuevo socket cliente al socket servidor.

Al terminar el robot IRB 120 el proceso recoger las piezas, el controlador envía al cliente socket la cadena de caracteres "Fin_recoger". La aplicación Android en este momento muestra por pantalla un mensaje indicando que las piezas se han recogido y se desbloquean el resto de botones de la pantalla.

- **Recoger NO:**

Al hacer clic en este botón la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente al socket servidor del controlador. Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "Recoge_no" al servidor. El servidor al recibir la cadena la desencripta y el programa en RAPID vuelve al inicio del "main", empezando de esta manera el programa de nuevo, esta vez sin recoger las piezas del tablero.

- **Nueva Partida:**

Al hacer clic en este botón, la aplicación regresa a la pantalla "Pantalla_calibrar.xml" e inicializa todas las variables y objetos utilizados por la aplicación, a excepción de la dirección IP y el puerto del controlador, que serán utilizados posteriormente y que no han cambiado durante toda la simulación del juego.

- **Regresar:**

Al hacer clic en este botón, la aplicación regresa a la pantalla "Activity_cliente_android.xml" e inicializa todas las variables y objetos utilizados por la aplicación

- **Salir**

Al hacer clic en este botón la aplicación se detiene y fuerza su salida al menú principal del dispositivo Android que estemos utilizando.

El diagrama de flujo del layout "Pantalla_tres_raya_fin.xml" es el siguiente:

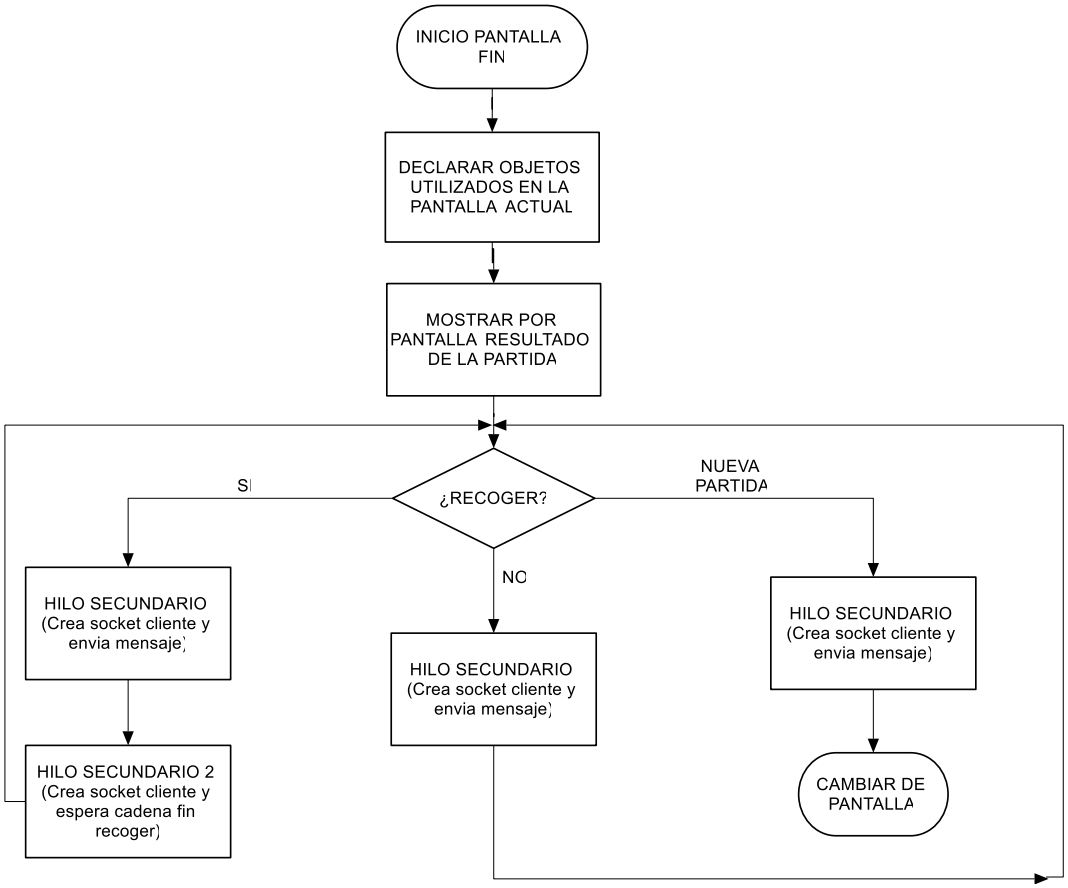


Diagrama 6.6: Pantalla_tres_raya_fin.xml

6.3 APLICACIÓN ANDROID CONTROL ROBOT

La aplicación Android "Control Robot.apk" desarrollada tiene como finalidad interactuar con la aplicación didáctica "Mover el robot" desarrollada en RobotStudio y que el usuario sea capaz de controlar por completo la ejecución del programa mediante la comunicación entre ambas aplicaciones. Para la correcta comunicación entre la aplicación Android y el controlador del robot, el dispositivo Android donde se ejecuta la aplicación y el controlador del robot deben de estar conectados al mismo punto de acceso de red Wi-Fi.

La aplicación Android "Control Robot" está compuesta por una serie de pantallas "layout" con diferentes objetos visuales que mediante la oportuna programación son capaces de comunicarse con el controlador IRC5 compact mediante el uso de Sockets.

Las pantallas desarrolladas en la aplicación Android son las siguientes:

- Activity_control_conectar.xml
- Activity_panel.xml
- Activity_movimiento.xml

6.3.1 Activity_control_conectar.xml

La pantalla correspondiente con "Activity_control_conectat.xml" es la pantalla principal de la aplicación, y por tanto, es la primera pantalla que nos aparece al ejecutar la aplicación "Control Robot" en nuestro dispositivo Android.

Esta pantalla está compuesta por dos cuadros de texto (TextView) que nos solicita que introduzcamos la dirección IP del servidor y el puerto al que nos conectaremos. La pantalla "Activity_cliente_android.xml" también dispone de dos cuadros de entrada de texto (EditText) donde podremos introducir la dirección IP del servidor y el puerto, además de dos botones de selección (RadioButton) que nos permitirán acceder a la pantalla "Activity_panel.xml" o a la pantalla "Activity_movimiento.xml" según la opción que marquemos y de un botón con el texto "Acceder Pantalla", que al hacerle clic nos llevará a la pantalla seleccionada en el RadioButton. En la Figura 6.8 se puede observar la estructura de esta pantalla.

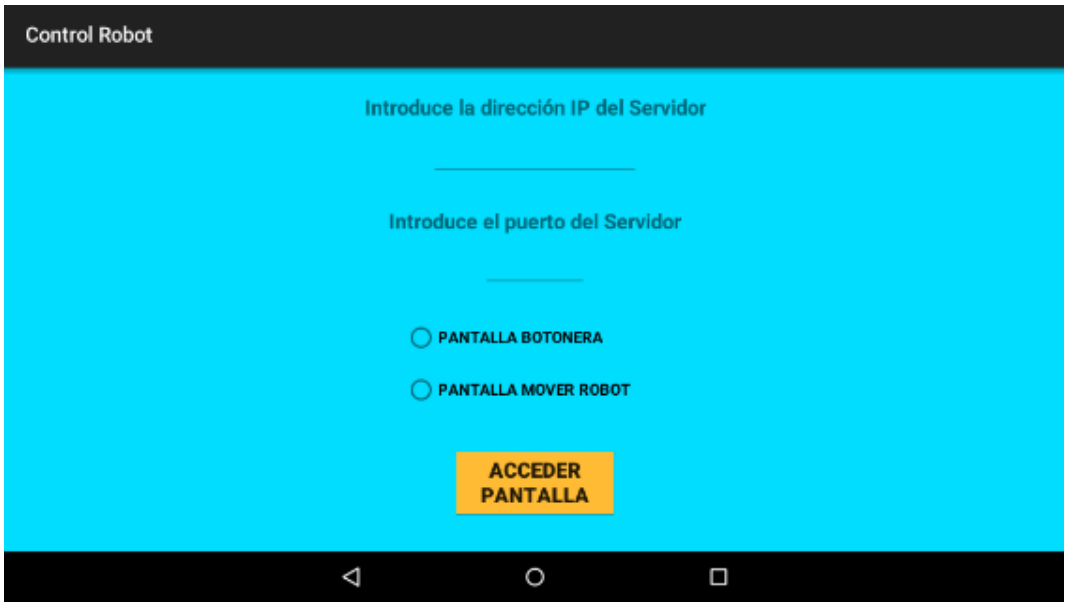


Figura 6.8: Activity_control_conectar.xml

La dirección IP del servidor que hay que introducir es la dirección IP estática del controlador del robot ABB IRB 120 que se encuentra en el laboratorio de prácticas, cuando trabajemos con la estación real; mientras que si trabajamos con la estación simulada en RobotStudio, la dirección IP del servidor que habrá que introducir es la dirección IP del ordenador personal que estemos utilizando para ejecutar el software RobotStudio.

Por defecto, al iniciar la aplicación la dirección IP que aparece en los EditText es la del controlador del laboratorio de prácticas ("157.88.201.130") y el puerto utilizado el 80, que es un puerto de libre acceso TCP/IP proporcionado por la red "eduroam" de la Escuela de Ingenierías Industriales. Si queremos trabajar con la estación virtual podemos cambiar esta dirección IP y el puerto utilizado haciendo clic en el EditText correspondiente.

Una vez que ya hemos configurado la dirección IP y el puerto que queremos utilizar podemos acceder a la siguiente pantalla a través del botón "Acceder Pantalla" que se encuentra en la parte inferior de la pantalla. Al hacer clic en este botón los datos contenidos en los EditText de la dirección IP y el puerto se almacenan en las variables "IP" y "puerto" que podrán ser utilizadas posteriormente.

El diagrama de flujo de la pantalla "Activity_cliente_android.xml" es el siguiente:

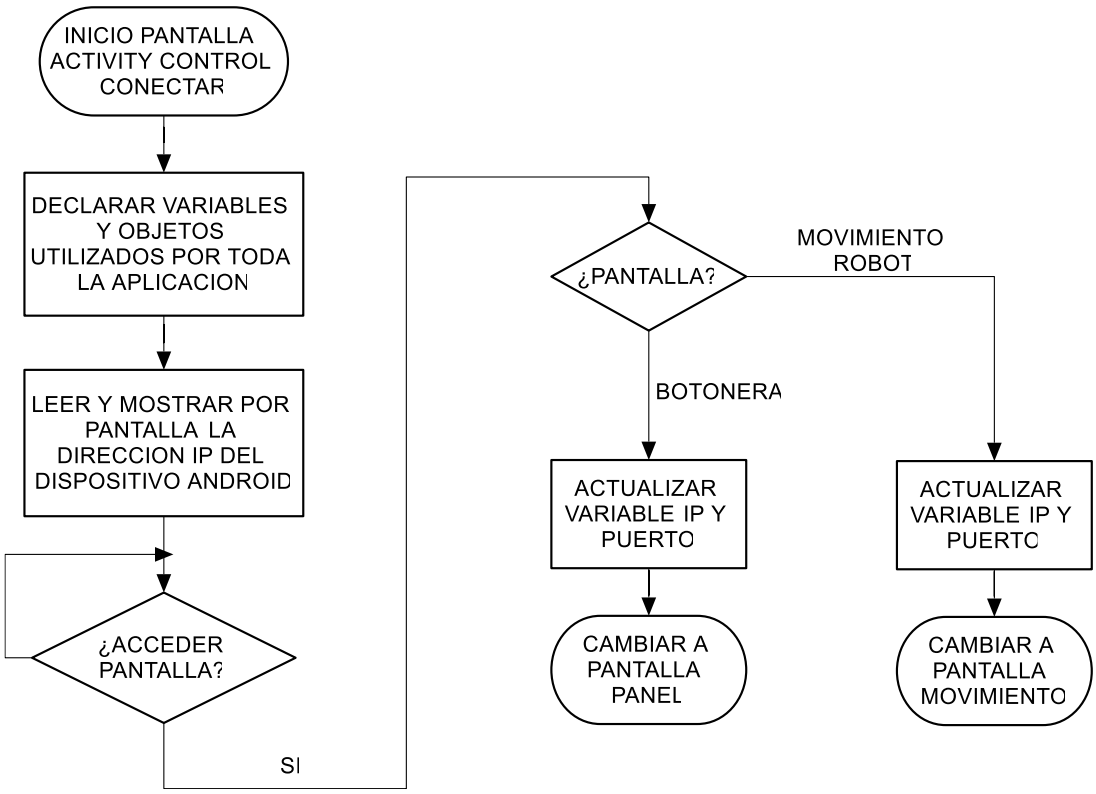


Diagrama 6.7: Activity_control_conectar.xml

6.3.2 Activity_panel.xml

En esta pantalla disponemos de 32 botones (ToogleButton) de dos posiciones ON/OFF, que simulan las 16 entradas digitales y las 16 salidas digitales disponibles en el panel externo de E/S del laboratorio de prácticas. En la parte superior de la pantalla aparece un cuadro de texto que nos informa en la pantalla que estamos y en la parte inferior se sitúa una barra deslizante (Seekbar) que nos permite regular la velocidad de movimiento del robot IRB 120. Desde esta pantalla podremos activar y desactivar las entradas y salidas digitales del controlador del robot, cambiando sus valores a través de unas variables persistentes que se han creado con la idea que el programa RAPID se pueda reciclar y ser de utilidad para una gran cantidad de situaciones.

Para simular el comportamiento del panel externo de E/S establecemos una comunicación entre la aplicación Android y el controlador del robot. En la Figura 6.9 se puede observar el layout "Activity_panel.xml"

Figura 6.9: Activity_panel.xml

A continuación se explicará la función de los botones de esta pantalla:

- **Botones Entradas Digitales:**

Al hacer clic en cualquiera de los botones que simulan las entradas digitales, la aplicación crea un hilo secundario de ejecución que se encarga de enlazar un socket cliente al socket servidor del controlador.

Una vez se ha conectado el cliente (hilo secundario), se envía la cadena de caracteres "do_X" al servidor, siendo la "X" el número correspondiente a la variable persistente cuyo valor deseemos modificar. El servidor al recibir la cadena la desencripta y el programa RAPID cambia el valor de la variable persistente correspondiente. Mediante la programación podremos realizar múltiples acciones al variar las variables persistentes, como en la aplicación que veremos posteriormente, donde podremos mover el robot IRB 120 mediante la programación con las 16 variables persistentes "pers_di".

- **Botones salidas digitales**

Estos botones se han incluido en la aplicación Android con el fin de un futuro poder programar la comunicación entre la aplicación Android y el controlador del robot, y poder simular los leds disponibles en el panel externo del laboratorio de prácticas.

- **Regresar:**

Al hacer clic en este botón, la aplicación regresa a la pantalla "Activity_control_conectar.xml" e inicializa todas las variables y objetos utilizados por la aplicación

- **Salir**

Al hacer clic en este botón la aplicación se detiene y fuerza su salida al menú principal del dispositivo Android que estemos utilizando.

El diagrama de flujo de la pantalla "Activity_panel.xml" es el siguiente:

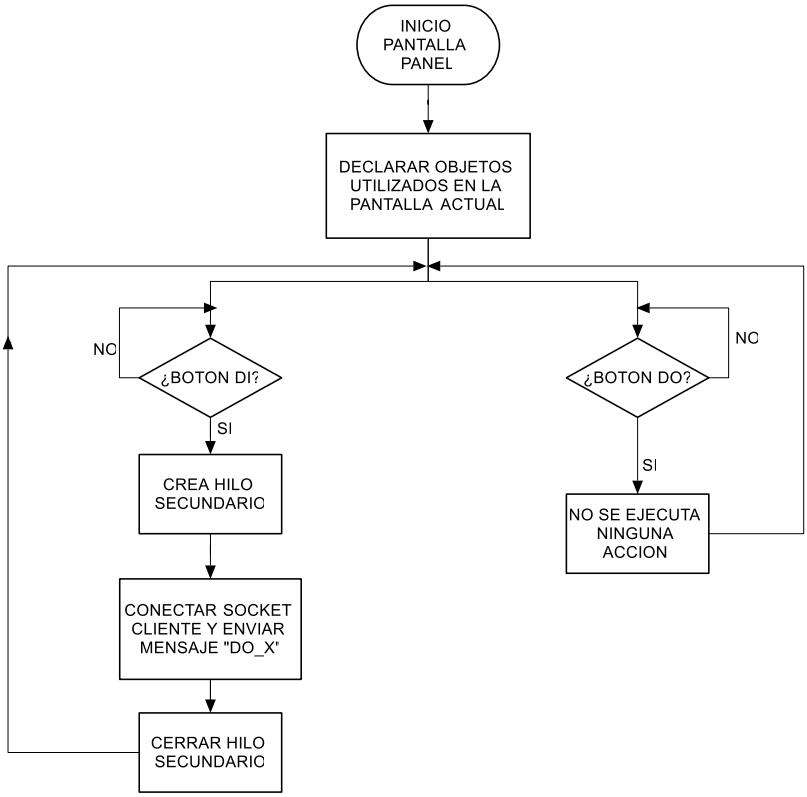


Diagrama 6.8: Activity_panel.xml

6.3.3 Activity_movimiento.xml

En esta pantalla disponemos de 11 botones (ToogleButton) de dos posiciones ON/OFF. Mediante el uso de 6 de estos ToogleButton se pretende simular las posiciones del joystick del FlexPendant para el movimiento del robot IRB 120 en las coordenadas WORLD y el movimiento de ejes. El control de la pinza de agarre se simula mediante el uso de 2 botones que permiten abrir/cerrar la pinza. Se puede cambiar el sistema de referencia del movimiento del robot mediante 2 botones, permitiendo la opción de movimiento respecto a las coordenadas WORLD, el movimiento de los ejes 1 al 3 y el movimiento de los ejes del 4 al 6. Mediante el uso de otro botón se puede acceder al menú, donde se puede observar la relación de variables persistentes que se controlan con los botones de la pantalla.

En esta pantalla también hay una lista desplegable que sirve para seleccionar la referencia de la herramienta que el robot IRB considerará en su movimiento. En la parte inferior se sitúa una barra deslizante (Seekbar) que nos permite regular la velocidad de movimiento del robot IRB 120. Desde esta pantalla podremos activar y desactivar las variables persistentes del programa RAPID que se encarga del control del robot IRB 120. En la Figura 6.10 se puede observar el layout "Activity_movimiento.xml"

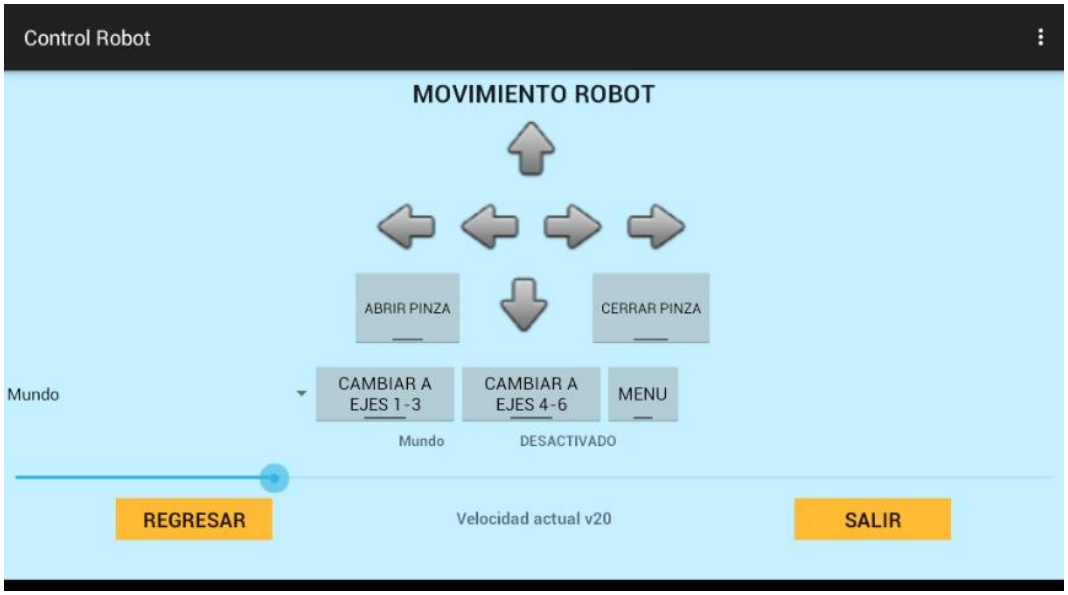


Figura 6.10: Activity_movimiento.xml

Los botones utilizados en la pantalla "Activity_movimiento.xml" son un caso concreto del panel de operaciones que se explicó anteriormente el layout "Activity_panel.xml", sólo que en este caso se ha cambiado la interfaz de la pantalla para adaptarla mejor al propósito de mover el robot y controlar la pinza de agarre. Por lo tanto, el Diagrama 6.8 mostrado en el apartado anterior sería también válido para explicar el funcionamiento de la aplicación Android en esta pantalla. La relación de botones y variables persistentes que se activan/desactivan para esta pantalla en particular es la siguiente:

- Botón izquierda: pers_di1
- Botón derecha: pers_di2
- Botón abajo: pers_di3
- Botón arriba: pers_di4
- Botón izquierda (interior): pers_di5
- Botón derecha (interior): pers_di6
- Botón "Abrir Pinza": pers_di9
- Botón "Cerrar Pinza": pers_di10
- Botón "Menú": pers_di12
- Botón cambiar referencia-ejes: pers_di13
- Botón cambiar ejes (1-3 a 4-6): pers_di14
- Barra velocidad: pers VelAct
- Referencia: pers ref_seleccion

- **Regresar:**

Al hacer clic en este botón, la aplicación regresa a la pantalla "Activity_control_conectar.xml" e inicializa todas las variables y objetos utilizados por la aplicación

- **Salir**

Al hacer clic en este botón la aplicación se detiene y fuerza su salida al menú principal del dispositivo Android que estemos utilizando

7 CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN

Las conclusiones que obtienen al realizar este proyecto son:

- La importancia de construir los modelos de una planta robotizada para su simulación, ya que permiten incrementar la rentabilidad del sistema robotizado mediante tareas como formación, programación y optimización, sin afectar a la producción. De esta manera se reducen los riesgos y se incrementa la productividad.

- El software RobotStudio de ABB es una herramienta muy útil que permite realizar simulaciones muy cercanas a la realidad. Este software es uno de los programas más utilizados en entornos reales para realizar la programación "offline" de células robóticas industriales.

- La importancia de un protocolo de comunicación robusto en la industria, permitiendo intercambiar libre y fácilmente información entre dispositivos y aplicaciones de cualquier tipo.

- El gran crecimiento que está teniendo en la actualidad es uso de aplicaciones Android, que nos permiten realizar interfaces personalizadas que den solución a nuestras necesidades, utilizando para ello el lenguaje de programación orientado a objetos Java.

Se proponen las siguientes líneas futuras de trabajo para aquellos interesados en la robótica:

- Introducir en la estación de trabajo sensores de distancia en la dirección Z de la pinza de agarre, lo cual nos daría información de la posición de los objetos cercanos al TCP de la herramienta.

- Estudiar la comunicación vía OPC entre el robot industrial y el software Matlab, comparando posteriormente los resultados con los obtenidos mediante la comunicación socket.

- Establecer un protocolo de comunicación entre el controlador del robot y la aplicación Android “Control Robot.apk” que nos permita activar y desactivar las botones de doble estado ON/OFF del panel de operaciones a partir del cambio de alguna salida digital del robot.

8 BIBLIOGRAFÍA

[1] ROBÓTICA

- https://es.wikipedia.org/wiki/Rob%C3%B3tica
- http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/industrial.htm
- https://robotica.wordpress.com/about/
- https://es.wikipedia.org/wiki/Robot_industrial
- http://www.ifr.org/industrial-robots/statistics/
- http://www.infoplc.net/actualidad-industrial/item/102567-crece-venta-robots-industriales-2014

[2] ROBOT ABB IRB 120

- http://new.abb.com/products/robotics/es/robots-industriales/irb-120
- http://www.axel-client.com/clients/abb/e-mail160310/irb120.html
- http://www.robotictools.it/robot-industriale-220-v-irb-120-abb.html
- http://new.abb.com/es/abb-in-spain/quienes-somos/historia
- Manual del operador IRC5 con FlexPendant. Doc. ID: 3HAC16590-5

[3] ROBOTSTUDIO

- http://new.abb.com/products/robotics/es/robotstudio
- Guía de Referencia RAPID online 3.0
- Manual del operador RobotStudio 6.0: ID de documento: 3HAC032104-005, Revisión: N
- Manual de referencia técnica Instrucciones, funciones y tipos de datos de RAPID: ID de documento: 3HAC16581-5 Revisión: J
- Curso RobotStudio (YouTube) Autor: AlonsoProfe

[4] JAVA

- https://www.java.com/es/download/faq/whatis_java.xml
- https://www.java.com/es/about/
- http://puntocomnoesunlenguaje.blogspot.com.es/2012/04/identificadores-y-palabras-reservadas.html
- http://www3.uji.es/~belfern/pdidoc/IX26/Documentos/introJava.pdf
- Curso de Java desde 0 (YouTube) Autor: Píldoras Informáticas

[5] ANDROID

- https://es.wikipedia.org/wiki/Android
- https://www.android.com/intl/es_es/
- https://www.wayerless.com/2015/08/android-sigue-dominando-el-mercado-de-smartphones/
- https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android

[6] ANDROID STUDIO

- El gran libro de Android. 3ª Edición 2013. Jesús Tomás Gironés. Editorial Marcombo. Enero-Julio/2014.
- Desarrollo de aplicaciones para Android. Edición 2013. Joan Ribas Lequerica. Editorial Anaya. Enero-Julio/2014.
- https://es.wikipedia.org/wiki/Android_Studio
- http://code.tutsplus.com/es/tutorials/getting-started-with-android-studio--mobile-22958
- http://academiaandroid.com/android-studio-v1-caracteristicas-comparativa-eclipse/
- Curso Programación Android (YouTube) Autor: Código Alonso
- http://www.androidcurso.com/index.php/99
- http://www.hermosaprogramacion.com/android/

[7] SOCKET

- http://androideity.com/2012/08/05/sockets-en-android/
- Sockets en Java - Aplicación Cliente/Servidor (YouTube) Autor: Luis Arias
- https://es.wikipedia.org/wiki/Socket_de_Internet

9 ANEXOS

9.1 DATASHEET ABB IRB 120

IRB 120

Specification

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

Features

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

Movement

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

Performance

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

Electrical connections

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

Physical

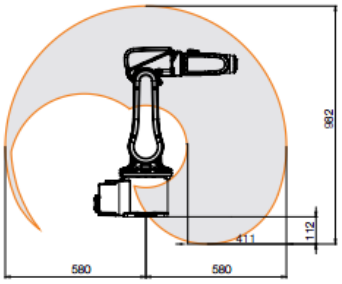
Dimension robot base	180 x 180 mm
Dimension robot height	700 mm
Weight	25 kg

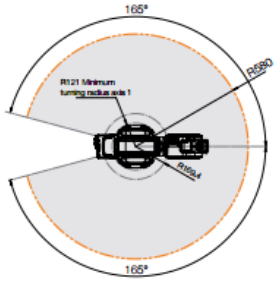
Environment

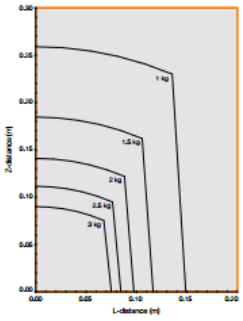
Ambient temperature for Robot manipulator:	
During operation	+5°C (41°F) to +45°C (122°F)
Relative transportation and storage	-25°C (-13°F) to +55°C (131°F)
For short periods	up to +70°C (158°F)
Relative humidity	Max 95%
Options	Clean Room ISO class 5 (certified by IPA)**
Noise level	Max 70 dB (A)
Safety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

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

Working range at wrist center & load diagram







9.2 DATASHEET IRC5 COMPACT CONTROLLER

Specification	
Control hardware:	Multi-processor system PCI bus Pentium® CPU Flash disk for mass memory Energy back-up power failure handling USB memory interface
Control software:	Object-oriented design High-level RAPID programming language Portable, open, expandable PC-DOS file format RobotWare software products Preloaded software, also available on DVD
Electrical Connections	
Supply voltage:	3 phase 200-600 V, 50-60 Hz Integrated transformer or direct mains connection 1 phase 220/230 V, 50-60 Hz (for Compact Controller only)

Physical	Size H x W x D	Weight
Single cabinet	970 x 725 x 710 mm	150 kg
Dual cabinet	1370 x 725 x 710 mm	180 kg
Control module	720 x 725 x 710 mm	50 kg
Drive module	720 x 725 x 710 mm	130 kg
Empty cabinet for customer equipment	- small 720 x 725 x 710 mm - large 970 x 725 x 710 mm	35 kg 42 kg
Panel Mounted *)		
Control module	375 x 498 x 271 mm	12 kg
Drive module small *1)	375 x 498 x 299 mm	24 kg
Drive module large *2)	658 x 498 x 425 mm	40 kg
Compact controller **)	258 x 450 x 580 mm	27.5 kg

*1) IRB 140, 340, 1600, 260

*2) IRB 2400, 2600, 4400, 4600, 6620, 6640, 6650, 7600, 660, 760

**) IRB 120, 140, 260, 360, 1410, 1600

Environment	
Ambient temperature:	0-45°C (32-113°F) option 0-52°C (32-125°F)
Relative humidity:	Max. 95% non condensing
Level of protection:	IP 54 (cooling ducts IP 33) Panel Mounted and Compact IP 20
Fulfilment of regulations:	Machine directive 98/37/EC regulations Annex II B EN 60204-1:2006 ISO 10218-1:2006 ANSI/RIA R 15.06 - 1999 UL 1740-1998
User Interfaces	
Control panel:	On cabinet or remote
FlexPendant:	Weight 1 kg Graphical color touch screen Joystick Emergency stop Hot plug Support for right and left-handed operators USB Memory support

User Interfaces continued	
Maintenance:	Status LEDs Diagnostic software Recovery procedures Logging with time stamp Remote Service enabled
Safety	
Basic:	Safety and emergency stops 2-channel safety circuits with supervision 3-position enabling device
Electronic Position Switches:	5 safe outputs monitoring axis 1-7
SafeMove:	Supervision of stand-still, speed, position and orientation (robot and additional axes) 8 safe inputs for function activation, 8 safe monitoring outputs
Machine Interfaces	
Inputs/outputs:	Up to 8192 signals
Digital:	24V DC or relay signals
Analogue:	2 x 0-10V , 3 x ± 10V, 1 x 4-20mA
Serial channel:	1 x RS 232/RS 422 with adapter
Network:	Ethernet(10/100 Mbits per second)
Two channels:	Service and LAN
Fieldbus Master:	DeviceNet™ PROFINET PROFIBUS DP Ethernet/IP™
Fieldbus Slave:	DeviceNet™ PROFINET PROFIBUS DP Ethernet/IP™ Allen-Bradley Remote I/O CC-link
Conveyor encoder	Up to 6 channels
Integrated PLC	AC500
Sensor Interfaces	
	Search stop with automatic program shift Seam/contour tracking Conveyor tracking Machine vision Force Control

Data and dimensions may be changed without notice.

Compact controller

Panel mounted controller