Escuela Politécnica Superior

Diseño y Simulación de una Célula Robotizada

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Luis Yared Delgado Soler

Tutor/es:
Francisco Andrés Candelas Herias

Junio 2019

Universitat d'Alacant
Universidad de Alicante

Justificación y Objetivos

Este proyecto encajaría dentro del ámbito industrial, en concreto en el ámbito de la robótica industrial donde se trata de realizar un "tracking" lineal, es decir, obtener la posición 2D o 3D del objeto a coger para que el robot pueda cogerlo en movimiento. Para ello, hay paquetes software propietarios de las principales marcas de robots que resuelven este problema, pero resultan bastante caros y poco flexibles. Implementar esta idea con herramientas mucho más baratas a una célula robotizada resulta interesante desde un punto de vista económico, donde una Raspberry Pi como core principal sea la que controla todos los procesos de la célula usando sensores, "encoders", motores, reductoras, robots y PLCs. Se pueden encontrar muchos proyectos para la Raspberry Pi que abordan comunicaciones utilizando librerías como SNAP7 de Siemens [1] o TCP/IP [2]. Se pretende utilizar una Raspberry Pi como controlador de alto nivel en la automatización de diversos procesos de la célula robotizada, teniendo en cuenta también la digitalización de esos procesos y un aumento de la flexibilidad en relación a la carga de trabajo del resto de procesos de la cadena productiva. De esta forma, añadimos las nuevas ideas de la Industria 4.0, flexibilidad y digitalización, sin prescindir de la máxima velocidad para disminuir tiempo de ciclo si se dieran todos los factores idóneos.

Agradecimientos

...

A mis Padres por haberme apoyado a nivel académico, económico y personal, a mi tutor por su disponibilidad y entrega ante cualquier problema o inquietud.
Por último a mis compañeros y amigos por su ayuda y apoyo.

...

No luchar por lo que quieres,
solo tiene un nombre y se llama perder.

Beret.

Contents

List of Figures

1 Introducción

A lo largo de mi etapa académica han habido varias asignaturas que me han suscitado especial interés, entre las cuales se podrían destacar Programación de Robots, Manipuladores, Comunicaciones o Procesadores Integrados y Automatización. Incluso cuando estaba en el instituto tenía inquietudes sobre los sistemas embebidos, y fue entonces cuando conocí Arduino. Recuerdo aquellas clases donde trabajábamos con este sistema, realizando programas sencillos para poder interactuar con muchos de los estímulos que recibía, pues siempre me ha resultado muy interesante poder entender el entorno, medirlo y actuar sobre él. Esta inquietud, que ha seguido creciendo dentro de mi cada vez más a medida que iba adquiriendo conocimientos en el grado, llegó a su auge en el tercer año, donde trabajé con la Raspberry Pi y con brazos manipuladores.

Durante ese curso adquirí mucha confianza en mí mismo, al ser capaz de entender y verme con posibilidades de realizar muchos de los proyectos abiertos que se pueden encontrar en Internet. También tome conciencia de la gran importancia a nivel laboral que tiene saber programar robots industriales de las principales marcas como FANUC, ABB, YASKAWA o KUKA. Por otro lado los PLCs son muy comunes en casi todas las células robotizadas, de ahí que el conocimiento sobre automatización también esté muy valorado a nivel laboral.

Todos estos razonamientos me llevaron a querer integrar una Raspberry Pi, un PLC y un brazo antropomórfico. Además, también quería ampliar mi conocimiento sobre otros marcas de estos equipos y utilizar otros estilos de programación que no hubiera visto

en la carrera. Esta fue la razón por la que decidí utilizar Python con programación en multiprocesos y una interfaz gráfica de usuario. Por otro lado, también quería utilizar los robots ABB para ampliar conocimiento tratando de utilizar los equipos reales, no únicamente simulación, así como un PLC de marca Siemens, ya que es una de las marcas líderes en la automatización industrial, por lo que sería interesante conocer el software y sus principales utilidades.

Una vez elegidos los equipos, el siguiente paso fue buscar una aplicación realista y útil a nivel industrial. Esto me llevo a pensar en resolver el problema del Line Tracking, que consiste en que un robot sea capaz de saber las coordenadas a las que tiene que moverse para poder coger un objeto que ha sido detectado previamente mediante un sistema de visión artificial. El Line Tracking se puede resolver sabiendo una posición inicial y el incremento de pulsos del "encoder" desde el momento en el que fue detectado, hasta el momento actual. Pero no solo eso, nosotros también queremos ser capaces de cogerlo en movimiento y el lenguaje de programación de ABB, RAPID, en su control cinemático, permite enviar la velocidad máxima que debe de tener el movimiento o bien, el tiempo en el que este puede ser realizado. A partir de este tiempo, el planificador de trayectorias ajustará las velocidades de cada eje para que el movimiento se pueda llevar a cabo de forma exitosa. En el caso de especificar un tiempo demasiado pequeño, en el cual el planificador de trayectorias resuelva que se tiene que utilizar una aceleración y velocidad incompatibles con los motores del robot, saturará la velocidad y aceleración máxima del robot y no podremos cumplir el tiempo enviado.

La Raspberry Pi será el corazón de nuestro proyecto, puesto que recibirá los pulsos del "encoder", controlará la posición de cada objeto en la cinta, evaluará el estado de los sensores y recibirá comandos mediante una interfaz gráfica de usuario. Por último utilizaremos el puerto Ethernet para enviar al robot, vía TCP/IP, cadenas de caracteres que contengan el resultado de todo lo anterior en forma de instrucción de movimiento.

1.1 Objetivos

El objetivo general es controlar la velocidad de un robot antropomórfico y de la cinta de entrada de piezas en función del nivel de carga de trabajo de todas las cintas de la célula, teniendo como entrada las piezas que circulan sobre la cinta y sus coordenadas que nos proporcione una cámara . Para ello, se diseñará una célula robotizada donde intervengan dispositivos de sensorización, motores, robots, así como los dispositivos de control de estos elementos como la Raspberry Pi y el controlador del Robot, centrándonos en la interconexión de estos equipos para conseguir una comunicación fluida.

Para conseguir este objetivo, tenemos que completar las siguientes tareas a nivel simulación y posteriormente con los equipos físicos:

- Hacer un diseño 3D de la célula.
- Comprobar mediante tests simples las comunicaciones entre todos los equipos.
- Crear un programa con la Raspeberry que tenga control total sobre la célula.
- Crear un proyecto en RobotStudio, el software de ABB.
- Desarrollar un codigo que permita la simulación conjunta en RobotStudio y Raspberry Pi.
- Realizar la simulación completa.
- Análisis de Resultados.
- Implementación en los equipos reales.

1.2 Estructura del trabajo

Después de este capítulo de introducción, el resto del contenido de este documento se encuentra organizado en los siguientes capítulos:

- ***Capítulo 1: Introducción*** ⇒ Capítulo introductorio que abarca los fundamentos del proyecto y describe los objetivos que se pretenden alcanzar.

- ***Capítulo 2: Estado del arte*** ⇒ Dedicado a proporcionar una base teórica general del ámbito en el que se mueve el problema a resolver.

- ***Capítulo 3: Metodología*** ⇒ En este capítulo se describen las principales etapas de las que ha constado el proyecto y las diferentes subtareas realizadas y se presenta el diseño 3D de la aplicación.

- ***Capítulo 4: Diseño de la aplicación*** ⇒ En este apartado abordaremos la solución propuesta a la introducción previamente descrita.

- ***Capítulo 5: Simulación de la aplicación*** ⇒ Dedicado a explicar cómo se realizó la simulación en RobotStudio.

- ***Capítulo 6: Resultados*** ⇒ Aquí se analizan en profundidad los resultados obtenidos, y se evalúa el rendimiento de la solución planteada.

- ***Capítulo 7: Conclusiones y trabajos futuros*** ⇒ Este capítulo expone las reflexiones sobre los resultados, así como de posibles trabajos futuros a abordar.

2 Estado del arte

En el mercado hay bastantes herramientas que resuelven el problema del "conveyor tracking" utilizando "encoders", pero la mayoría de estas herramientas no son abiertas y obligan a adquirir equipos, como los propios codificadores, y el software de una marca concreta. En lo referente al control de alto nivel de la planta, existen aplicaciones SCADA, proporcionadas habitualmente por fabricantes de automatización o compañías de software, que facilitan el desarrollo rápido de una solución. Pero esto normalmente es a costa de perder flexibilidad en el diseño, tener problemas compatibilidad con equipos o protocolos de otras marcas, y requerir una inversión mucho mayor que si se usa una alternativa de hardware de bajo coste y software libre, como es el caso de Raspberry Pi.

2.1 Conveyor Tracking

El "conveyor tracking" que se representa en la figura 2.1 es una técnica muy utilizada a nivel industrial puesto que permite que el robot sea capaz de coger lo elementos presentes en la cinta sin que esta tenga que parar. Un sistema que implementa "conveyor tracking" suele ir acompañado de un sistema de visión que permite detectar los elementos sobre la cinta cuando estos pasar por una determinada zona, determinando aspectos como su tipo, tamaño u orientación. Además, mediante un codificador colocado en la cinta, se puede saber el desplazamiento de un elemento desde su identificación con el sistema de visión, comparando la diferencia de los pulsos codificador entre el momento

de la detección y el instante actual. Estos sistemas suelen establecer 2 umbrales, de tal forma que si el elemento a coger se encuentra dentro de estos umbrales, lo cogería a la velocidad indicada. En caso de no estar dentro de este umbral porque todavía no ha llegado o porque ya lo ha pasado el robot se quedaría parado y la cinta continuará moviéndose.

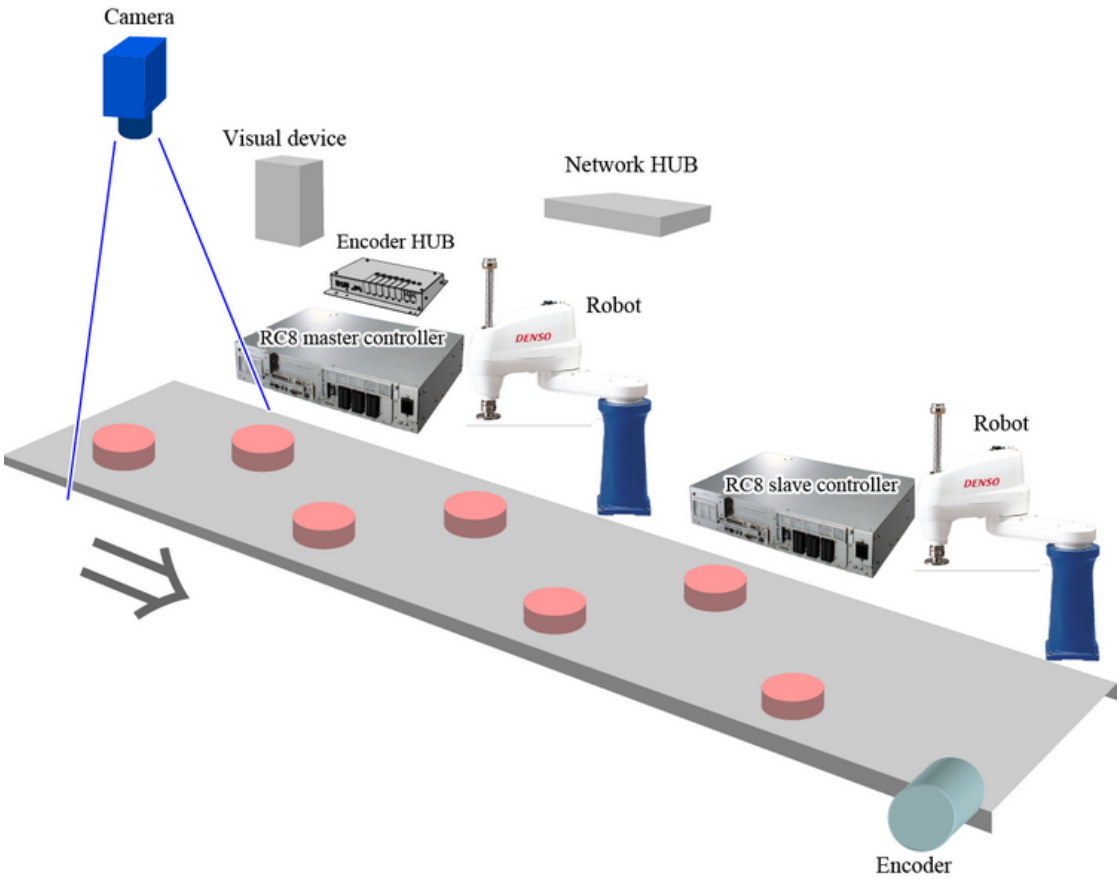


Figure 2.1: Esquema del funcionamiento del "Conveyor Tracking" (imágen de Denso Robotics: https://densorobotics.com/content/user_manuals/19/002219.html).

La mayor parte de las grandes empresas dedicadas a la creación de robots industriales ya han resuelto el problema que abordamos en este trabajo de final de grado ofreciendo sus soluciones en forma de paquetes de software bastante caros, entre los cuales destacan:

- Conveyor Tracking de ABB (PVP: 1700 euros).
- IR pick Tool de FANUC (PVP 1280 euros).

2.2 HMI y SCADA

Los sistemas HMI(Human-Machine Interface) y más concretamente las pantallas industriales, se suelen colocar en la célula robotizada por fuera de las barreras de seguridad, de esta forma, el operario podrá seleccionar el proceso que se va a realizar entre todos los que estén programados. Suele ser una interfaz hombre máquina bastante sencilla, con unos pocos botones o táctil como la que aparece en la figura 2.2. Entre sus desventajas están el uso de pantallas HMI se restringe a utilizar el software de la misma marca, que es caro y poco flexible. Aunque como ventajas podemos destacar que el software de diseño es bastante fácil y éste permite desarrollar rápidamente la interfaz.

Mientras que el HMI instalado en planta solo ofrece información local de la misma y un conjunto limitado de acciones sobre esta, los sistemas SCADA (Supervisory Control And Data Acquisition)[3] suelen estar aislados del entorno de fabricación, y controlan no solo el PLC dentro de la célula robotizada sino todos los PLCs, sensores, motores que influyen dentro de una cadena de fabricación. Con este sistema se puede programar la lectura de los datos para obtener tablas o diagramas de la productividad en cada instante del tiempo, obtener una visión global del estado de toda la cadena de fabricación, mantener un historial de las variables críticas para la optimización del proceso y tener un servidor web para poder actuar y controlar la planta de forma remota y segura. El precio de un software SCADA de gama media, que permita gestionar un buen número de E/S y protocolos industriales (estas suelen ser necesidades comunes en una aplicación SCADA), puede sobrepasa fácilmente los 1000.

Figure 2.2: Pantalla HMI SIMATIC KTP900 MOBILE, 9.0" TFT (imagen de PLC-CITY en https://www.plc-city.com/shop/es/siemens-simatic-hmi-mobile-panels-2nd-generation/6av2125-2jb03-0ax0.html).

2.3 Raspberry Pi

Como alternativa a los sistemas HMI y SCADA utilizaremos una Raspberry Pi (ver la figura 2.3), que es bastante más barata y más flexible. No obstante, utilizar una Raspberry Pi implica la necesidad de mayores conocimientos de informática y programación, aunque si el diseñador tiene estos conocimientos, entonces es una buena opción en proyectos pequeños.

La Raspberry Pi es un SoC(System On chip), con interfaces de comunicación típicas (USB, Ethernet) y salida de video.

El motivo principal para utilizar la Raspberry, frente a otros sistemas embebidos, es la infinidad de proyectos accesibles que hay internet para utilizar como base tanto en contenido como en estilo. Este sistema ha triunfado tanto por todas las posibilidades que da con sus 27 pines GPIO para utilizar sensores u otros dispositivos, pines para

Figure 2.3: Raspberry PI 3.

comunicaciones SPI e I2C, además de tener posibilidad de comunicaciones via USB y Ethernet, 4 núcleos de 1GHz y 1 GB de RAM por un precio alrededor a los 30 euros. Esto convierte a la Raspberry en mucho más interesante para este proyecto que otros sistemas empotrados como Arduino o MPS430.

Además, una Raspberry Pi utiliza como sistema operativo Raspbian, que es una variante de la distribución Debian de Linux, por lo que puede ejecutar muchas aplicaciones de Linux, incluidas una gran variedad de herramientas de programación. En concreto, es posible utilizar el lenguaje de programación Python (en este proyecto se emplea Python 2.7 por compatibilidades con algunas librerías usadas), muy popular últimamente por su sencillez a la hora de programar, además de ser un lenguaje multiplataforma, cosa que lo hace más atractivo todavía.

2.3.1 Python

El lenguaje de programación Python nació el año 1991 diseñado por Guido van Rossum. Entre sus principales características destacan que es un lenguaje multiplataforma, multiparadigma, orientado a objetos, de tipado dinámico, similar a pseudocódigo y con una enorme comunidad de usuarios.

El programa principal que corra en la Raspberry intentará optimizar los 4 núcleos de los que dispone la Raspberry Pi; de esta forma dispondremos de unos 4 GHz útiles de trabajo para utilizará la librería de python llamada multiprocessing

Además utilizaremos la librería “time” para poder contar el tiempo a la hora de simular el “encoder”

Todos los valores relativos a la carga de trabajo de las cintas destino, como las constantes para cambiar la velocidad del extremo del robot en función de la carga de la cinta origen, se gestionarán mediante una sencilla interfaz gráfica creada utilizando la librería “tkinter”.

2.4 Robot Antropomórfico IRB120

Figure 2.4: Robot ABB IRB 120 3kg.

Como protagonista principal de la célula robotizada tendremos un robot IRB 120 de la marca ABB figura 2.4 capaz de levantar 3kg en muñeca, que además está disponible en el laboratorio de robótica de la Escuela Politécnica Superior. Se utilizará el software de simulación Robot Studio de la marca ABB, del que la Universidad de Alicante dispone de licencias. ABB es una de las 3 principales marcas de robótica industrial, además de dedicarse a una enorme variedad de aplicaciones de electricidad y automatización. Tanto

ABB como FANUC y KUKA tienen alrededor del 50 % del mercado actual de robots figura 2.5.

Estos robots vienen con el paquete de software de comunicaciones comprado, en concreto el que nos interesa es el PC Interface para poder establecer una comunicación TCP/IP

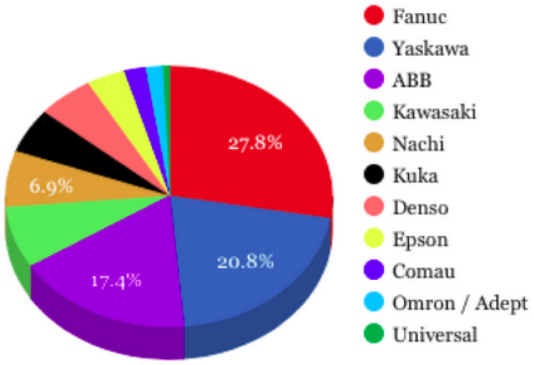


Figure 2.5: Porcentaje de robots instalados según su marca a nivel mundial.

Previamente se realizará un simulación utilizando RobotStudio y posteriormente varias pruebas con el robot real

3 Planificación

En este capítulo se presenta un diseño 3D de la aplicación describen las principales etapas de las que ha constado el proyecto y las diferentes subtareas realizadas.

3.1 Diseño 3D

La célula robotizada constará de los siguientes elementos:

- 3 barreras físicas rodeando al robot.
- 1 barrera fotoeléctrica de seguridad que permitirá pasar los objetos. utilizando "muting" [4]
- 1 robot IRB 120 de la marca ABB.
- 1 Raspberry Pi.
- 1 sistema de visión.
- 1 cinta de entrada.
- 2 cintas de salida.

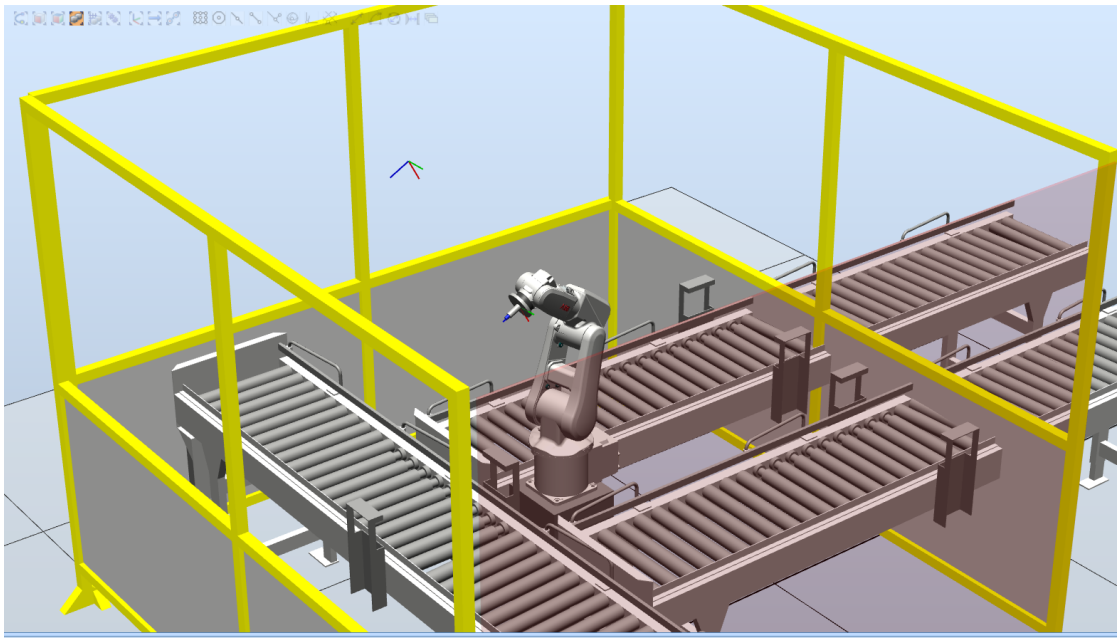


Figure 3.1: Diseño de la célula hecho con RobotStudio 6.08.

3.2 Metodología

La metodología para el desarrollo de este proyecto descrita en la sección 1.1 se ha llevado a cabo tal y como se muestra en la figura 3.2, donde se puede ver el desarrollo temporal de la distintas tareas de las que consta el proyecto.

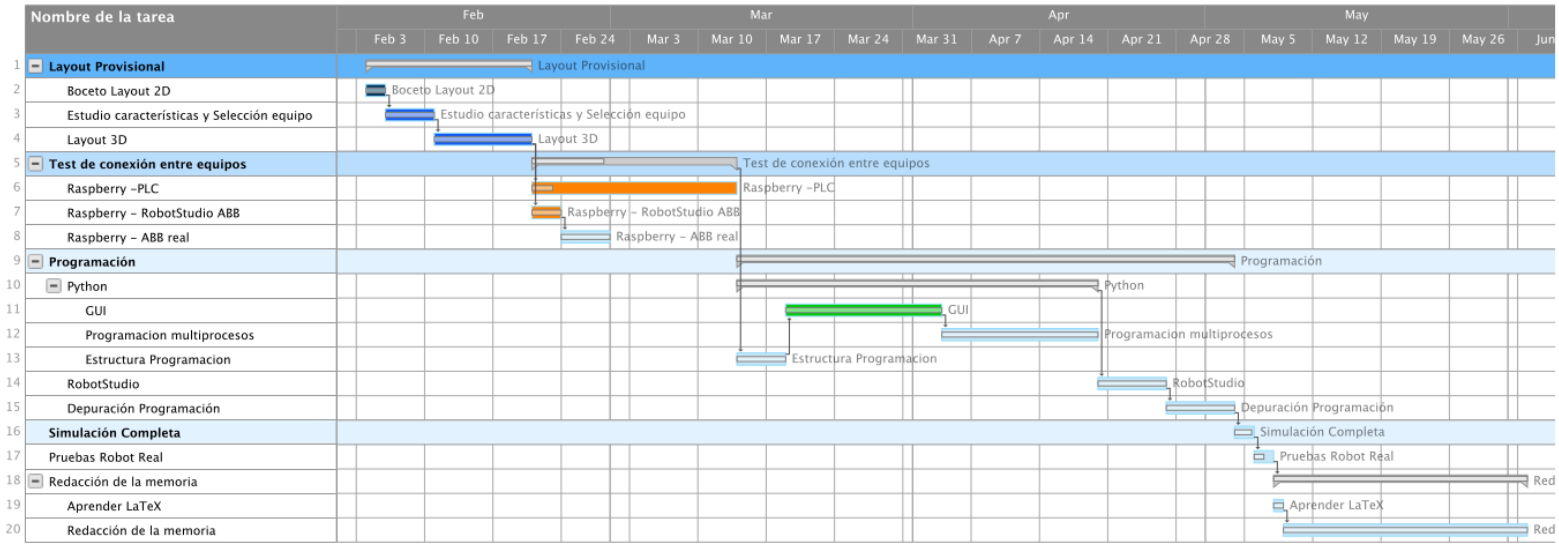


Figure 3.2: Diagrama de Gantt del desarrollo del proyeto

4 Diseño de la Aplicación

4.1 Estructura de la solución

En primer lugar, vamos a especificar claramente de que versará el proyecto. Para comenzar, crearemos una lista de objetos complejos de la clase Part en el código fuente . La clase Part representa un objeto que está sobre la cinta, describiendo sus características principales que habrán sido detectadas previamente por algún sistema de visión. En la solución aquí presentada no se considera el diseño del sistema de visión, ni se elige un sistema específico. Estas características serán: el **tipo de objeto**, sus coordenadas **x**, **y**(eje en el que se moverán las objetos), **orientación** y el **número de pulsos** del codificador de la cinta en el momento de la detección. No guardaremos la coordenada z porque en el diseño de la aplicación hemos establecido que ambos tipos de objetos tendrán la misma altura. Es muy importante que, al definir las coordenadas de los objetos, coincidan con el sistema de coordenadas que utilice el robot como referencia para moverse. En la figura 4.1 podemos observar los tipos de Parts posibles. El objeto de la izquierda es el tipo 0, cuyas, medidas son a=b=80, c=40 mm. A la derecha se muestra el objeto del tipo 1, cuyas medidas son a=30, b=50, c=40 mm.

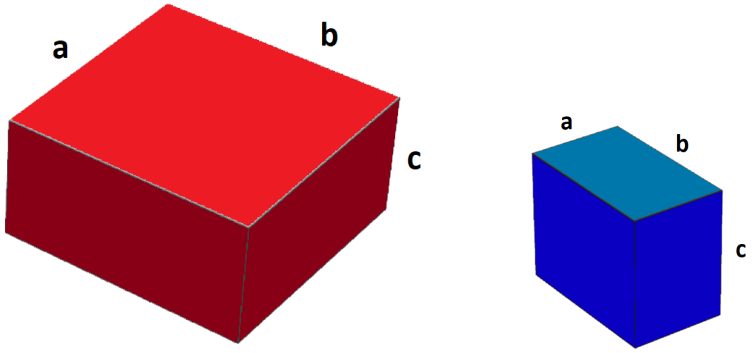


Figure 4.1: Tipos de objetos de la clase utilizados, pertenencientes a la clase Part

Estas regiones A,B y C representadas en la figura 4.2 están elegidas en base al alcance del robot. Estos objetos podrán estar en cualquier parte de la cinta de entrada e irán en la dirección C⇒B⇒A.

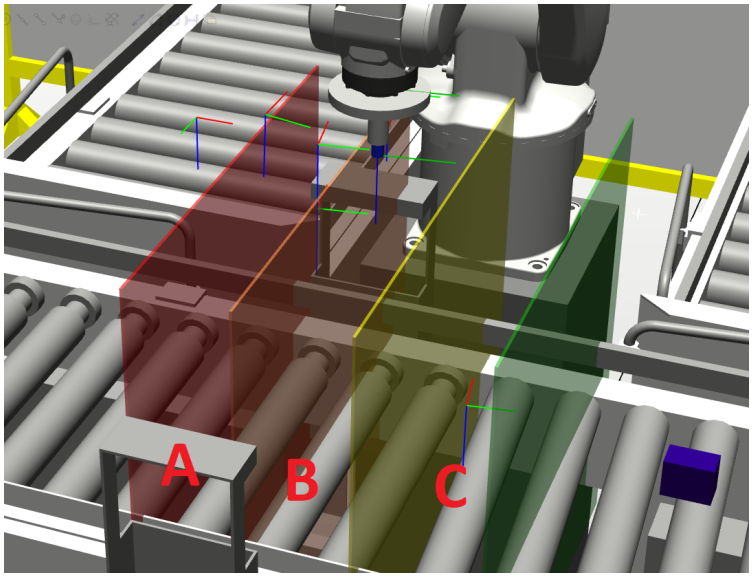


Figure 4.2: Regiones de alcanzabilidad.

Mediante la diferencia entre los pulsos actuales y los pulsos que tenía la cinta cuando se detectaron estos objetos, podremos saber las coordenadas exactas de estas objetos en cada momento. Estas coordenadas, junto con la velocidad de cinta que dependerá

de los pulsos por segundo simulados y la equivalencia establecida entre la relación pulsos/milímetro, serán enviadas vía TCP/IP al controlador del robot. Además, también tendremos una interfaz gráfica en la cual simularemos el nivel de saturación de las cintas destino mediante el uso de unos sliders.

4.2 Raspberry:Python

El programa donde se ha desarrollado lo que a continuación se expondrá se llama **tfg.py**. Al haber utilizado Python como lenguaje y ser este un lenguaje multiplataforma, se puede desarrollar el programa en cualquier equipo con Phyton instalado, independientemente de que el objetivo sea ejecutarlo en en la Rasbberry Pi.

Como se comentó en el apartado 3.2 sobre la metodología seguida en el proyecto, en desarrollo del programa de control comenzó por la interfaz gráfica de usuario (GUI). Así, esta parte se describirá primero a continuación.

4.2.1 GUI

El motivo principal de hacer esta interfaz gráfica de usuario es poder cambiar de forma sencilla y dinámica los valores de las constantes de la ecuación del control de la velocidad del robot utilizando controles del tipo “sliders” . También interesa poder mostrar por pantalla el resultado obtenido tras aplicar la función de control que se muestra en la ecuación 4.1, la cual devolverá un valor entre 0 y 100, que lo interpretaremos como un porcentaje de velocidad máxima del robot, que predefiniremos como 1000 milímetros por segundo . Para eso, tras hacer el análisis de las diferentes librerías en Python, se optó por utilizar “tkinter”, donde encontramos un pequeño ejemplo con cada una de sus utilidades por separado en [5].

$$vel = k * (k_a * A + k_b * B + k_c * C) \tag{4.1}$$

La ecuación 4.1 también aparecen las variables A, B y C, que representan el número de objetos en cada una de las tres áreas delimitadas en la cinta de entrada, según lo explicado en el apartado 4.1, estableciéndose de este modo un nivel de preocupación tal que las objetos que se encuentren en la zona A serán las más preocupantes que las de la zona B y estas más preocupantes que las de la zona C. El objetivo de la ecuación es que un mismo número de objetos situado en una zona de mayor preocupación provocará un aumento de la velocidad del robot. Estos niveles crecientes de preocupación o peligrosidad se conseguirán forzando las constantes presentes en la ecuación 4.1 de forma que $k_a > k_b > k_c$.

De este modo podremos variar el valor de las constantes que se ven en la ecuación 4.1 y obtener la velocidad a la que se moverá el robot. Tanto el nivel de estrés de la cintas destino como el valor de estas constantes determinarán la velocidad a la que se tiene que mover el robot en la simulación y este valor será enviado vía TCP/IP.

Tkinter necesita un hilo de ejecución para mostrar la interfaz gráfica que se puede ver en la figura 4.3. Esto nos obligará a tener que utilizar programación en multiprocesos, que veremos en la siguiente sección 4.2.2.

Como se puede ver en la figura 4.3, con los controles de tipo"slider" podemos variar las constantes vistas en la ecuación 4.1. Tal y como aparece en la figura 4.3, también podemos variar % Conveyor 1 y %c Conveyor 2, haciendo referencia al nivel de carga de trabajo que tienen las cintas destino. Esta carga de trabajo se medirá como número de objetos minuto, detectados por cualquier tipo de sensor de presencia en la célula robótica real.Además, en la parte inferior de la interfaz de la figura 4.3, aparece aparece el resultado de la ecuación de control 4.1, bajo la etiqueta llamada SpeedSent. Este valor se enviará al proceso que establece y controla la comunicación con el robot. Las

etiquetas Parts in C, Parts in B y Parts in A hacen referencia la número de objetos que hay en cada una de las zonas de trabajo definidas para la cinta de entrada.. Si las etiquetas están en verde, significa que esa parte de la interfaz está habilitada.

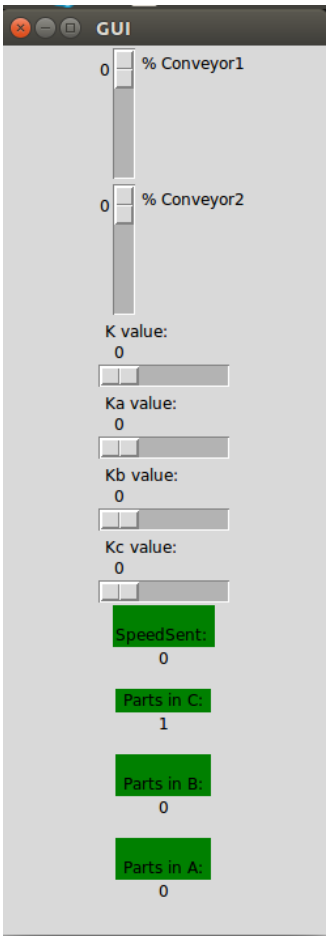


Figure 4.3: Interfaz gráfica de usuario.

4.2.2 Multiprocessing

De esta librería depende el éxito o el fracaso de este proyecto, pues es necesario utilizar diferentes procesos compartiendo algunas variables entre ellas. Sin duda, esta parte ha sido la más complicada por la limitación de tipos de datos que pueden ser compartidos en-

tre procesos cuando se usa el lenguaje Python. Tal y como aparece en la documentación de esta librería [6], para la comunicación entre procesos recomiendan utilizar objetos de la clase Queue y Pipe, que en definitiva ambos se comportan como un tipo de datos de cola FIFO (First In, First Out), pero con la limitación de que no se pueden enviar objetos objetos complejos entre procesos. La diferencia entre Queue y Pipe es la cantidad de procesos que pueden compartir los datos, ya que mientras Queue permite compartir un dato entre todos los procesos que queramos, Pipe solo permite entre 2 procesos.

El principal problema de utilizar tanto Queue o Pipe es que ambos generan un error cuando se acumulan mas de 278 datos enviados, y esto puede pasar con mucha facilidad si la frecuencia de ejecución del proceso que lee es menor que la del proceso que envía. Además en el caso contrario, de intentar escuchar algo que todavía no ha sido enviado, ambos métodos bloquearán el proceso hasta recibir el dato que necesita. Me decanté por la última idea, en tanto que al menos no salía un error de ejecución. En consecuencia, la suma de todos los tiempos en los que el programa se quedaba bloqueado durante cada iteración provocaba que si, a nivel teórico, la cinta se tuviera que mover 10 mm/s, lo hacia alrededor de 5 mm/s.

El problema de enviar una lista de objetos complejos entre procesos se solucionó finalmente escribiendo en un fichero llamado ConveyorParts.txt todas los objetos creados de la clase Part, que tras ordenarse en función de la coordenada Y, serían escritos en ese fichero para que el resto de procesos tuvieran acceso de lectura a él.

Cuando ya tenía casi acabado el proyecto descubrí por casualidad que también era posible enviar, entre procesos, objetos del tipo Array y Value, tal y como aparece en este código fuente [7] a pesar de que la documentación oficial recomiende usar Pipe y Queue. Tras realizar unos breves experimentos observé que Array tampoco permite enviar un Array de objetos complejos, mientras que el tipo Value tenía la gran ventaja de que podía ser leída y escrita por cualquier proceso en cualquier momento sin tener la estructura FIFO. En consecuencia, todas esas instrucciones bloqueantes por cada

iteración se eliminaron, consiguiendo tiempos muy parecidos a los reales[1] tal y como se puede ver en la tabla 6.1.

Método	**Velocidad Teórica (mm/s)**	**Velocidad Media Real (mm/s)**
Pipes	10	4.8
Value	10	9.8

Tabla 4.1: Valores obtenidos en mm/s tras la ejecución de la simulación

La estructura principal del código en python utilizando multiprocesos quedaría así:

[1]Cálculo realizado teniendo en cuenta únicamente los primeros 500 pulsos, podría haber variaciones dependiendo del nivel de carga de los núcleos en cada instante

```
if __name__ == '__main__':

    #Definimos las variables que se compartirán en memoria, nótese
        el uso de Pipe y Value

        misPulsos=Value('d',0)
        pulsSeg,pulsSeg2=Pipe()         #encoder mas tcpip
        startIn,startOut=Pipe()
        ProcessedParts=Value('d',0)
        FinalSpeed=Value('d',0)
        stopGuiTCP=Value('d',0)

        createParts()
    #Llamada a los 3 procesos utilizando las variables compartidas

        Process(target=encoderSIM, args=(misPulsos,pulsSeg,
            startOut)).start()
        Process(target=GUI,args=(misPulsos,FinalSpeed,
            ProcessedParts,stopGuiTCP)).start()
        Process(target=TCP_IP_Comm,args=(pulsSeg2,startIn,
            FinalSpeed,ProcessedParts,misPulsos,stopGuiTCP)).start
            ()
```

4.2.3 Simulación del enconder

Dado que no se pudo conseguir utilizar el PLC con su entrada de contador rápido, apropiada para utilización de enconders reales, se optó por simular estos pulsos, de tal manera que uno de los procesos que se ejecutan en la Raspberry Pi se encarga de, una vez establecida la conexión con RobotStudio, incrementar 1 pulso cada 0.01 segundos, o lo que es lo mismo, 100 pulsos por segundo y lo guarda en un objeto del tipo Value, de

este modo tendrán acceso al contenido de los pulsos actuales los procesos que se encargan de visualizar la interfaz gráfica y de enviar los resultados via TCP/IP, que se explicará en el apartado 4.2.4. El control del tiempo para incrementar 1 pulso a una frecuencia determinada se consigue mediante la librería de Python "time"

4.2.4 Comunicación via TCP/IP

Para conseguir la comunicación via TCP/IP entre la Raspberry Pi y el simulador del robot utilizaremos la libreria "socket" de Python, donde la Raspberry Pi se comportará como cliente comunicandose con un servidor que será el simulador. Así, la Raspberry Pi necesita conocer la IP del servidor en la red local, que en mi caso será 192.168.1.44 y se utilizará el puerto 55000. El proceso que ejecuta la comunicación en la Rasberry Pi se encargará de recibir los datos de los otros procesos mostrados en la tabla 4.2

Proceso	**Input**
GUI	Velocidad del robot o "stop" por saturación de las cintas salida
encoderSIM	pulsos/s reales, pulsos actuales
ConveyorParts.txt	(solo lectura) Características de los objetos que hay en la cinta

Tabla 4.2: Variables compartidas que utiliza como entrada.

Con los datos procedentes de la tabla 4.2 esperaremos a que el primer objeto pase la frontera C, que será la primera que se puede traspasar tal y como aparece en la figura 4.2. Una vez traspasada esta frontera, se empaquetarán los datos recibidos del siguiente modo:

" p__x___y___o__t__s__ "

Siendo:

- p: Tipo de objeto [0,1].
- x: Coordenada en x.
- y: Coordenada en y.
- o: Orientación del objeto (º tomando como referencia el eje de la cinta).
- t: Tiempo para realizar el movimiento del Pick.
- s: Velocidad a la que se moverá el robot de 0 a 1.000 mm/s.

Esta cadena será enviada al simulador de RobotStudio, que posteriormente la interpretará. Las simulaciones del robot y el resto de la celda mediante RobotStudio se aborda en el siguiente capítulo.

A continuación podemos ver los diagramas de flujo los programas de RobotStudio figura 4.4 y de la Raspberry Pi figura 4.5

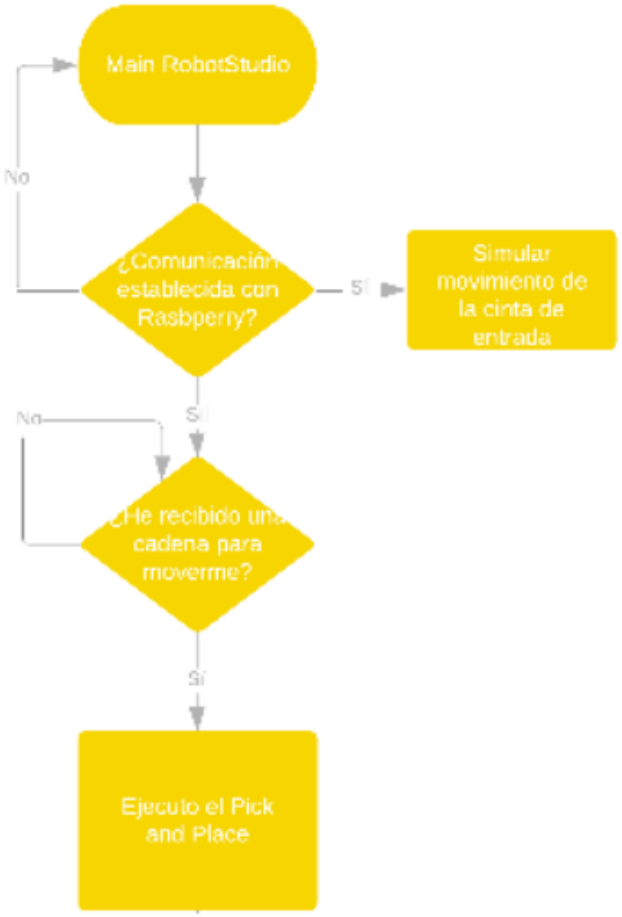


Figure 4.4: Diagrama de flujo del programa de RobotStudio.

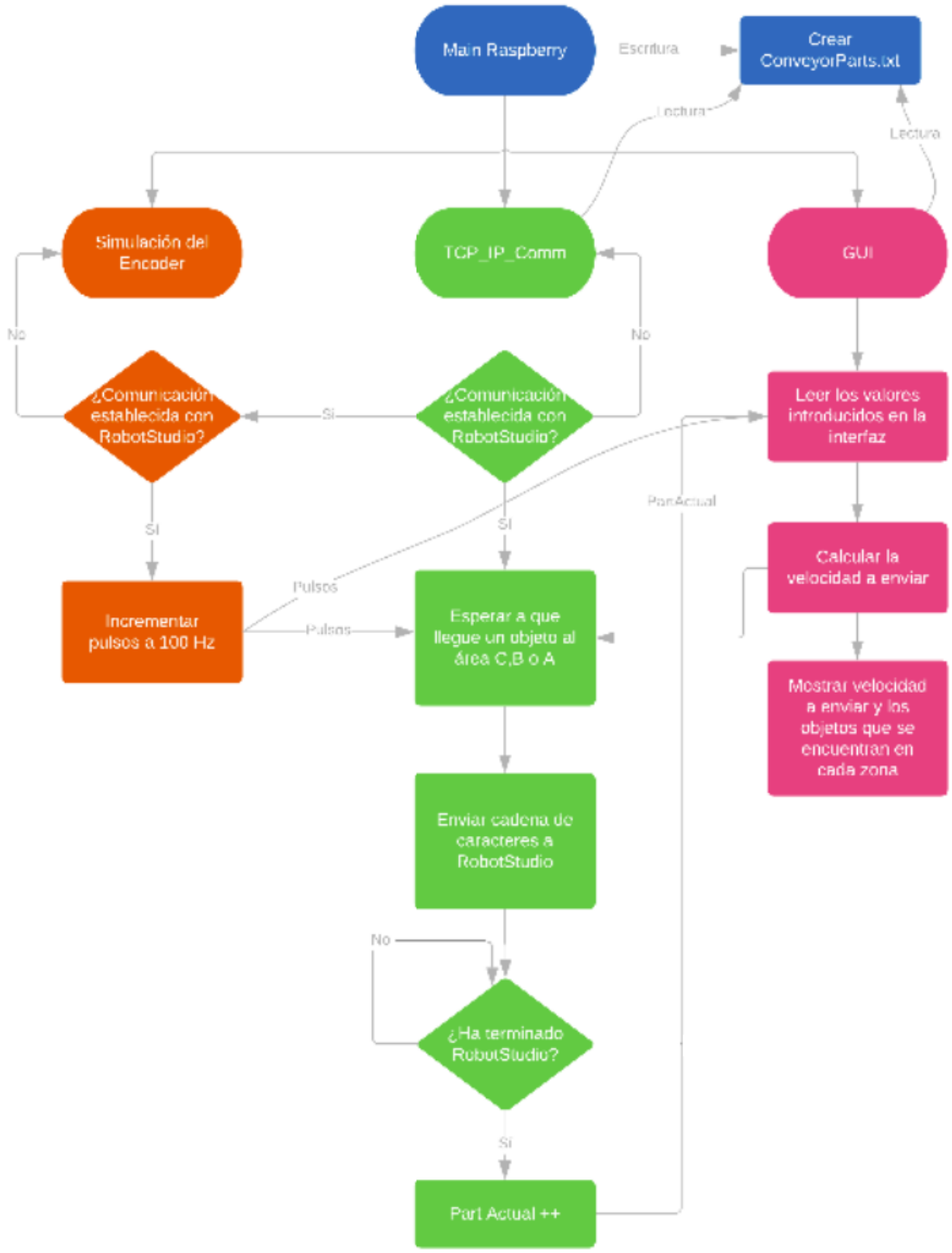


Figure 4.5: Diagrama de flujo del programa de la Raspberry Pi.

5 Simulación de la Aplicación

5.1 Diseño de la herramienta del robot

Para realizar una simulación, en primer lugar se tomaron medidas de la herramienta que tienen los robots ABB IRB 120 del laboratorio, que es un ventosa simple de la marca Festo, con el fin de reproducir esta herramienta mediante un software de construcción 3D. Para ello, se utilizará Autodesk Inventor, obteniendo el resultado de la figura 5.2. En la figura 5.1 observamos como quedaría el montaje con el robot tanto en RobotStudio como en la realidad.

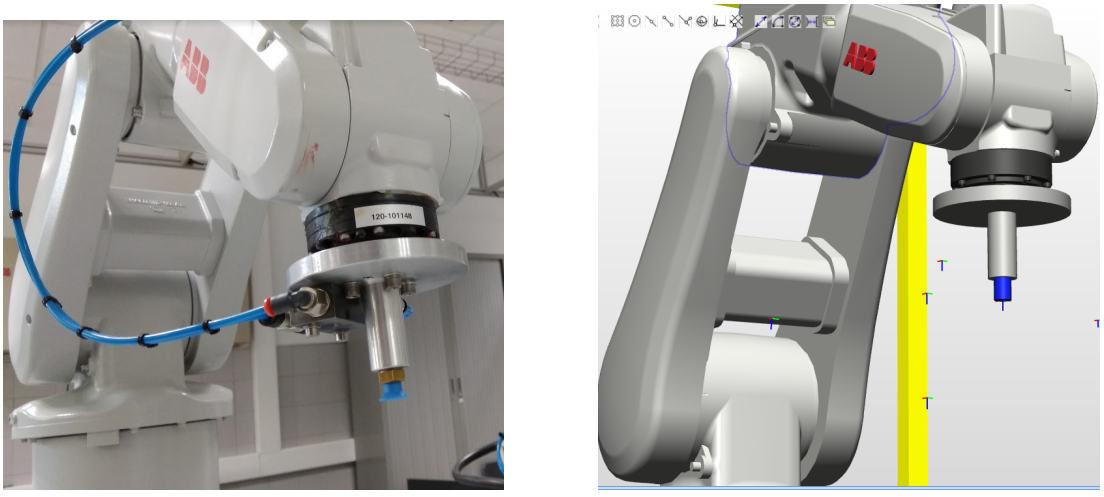


Figure 5.1: Ensamblaje de la herramienta en el robot real y en el simulado

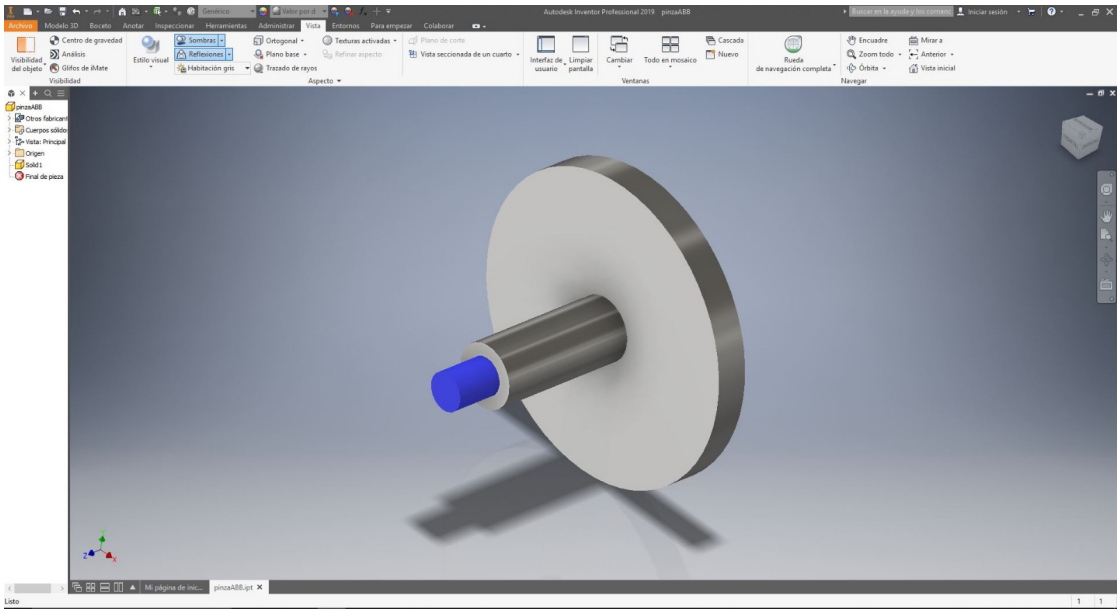


Figure 5.2: Diseño de la herramienta utilizando Autodesk Inventor.

5.2 RobotStudio

La simulación de la celda con el robot IRB120 y las cintas se ha realizado mediante la versión 6.08.01 de RobotStudio, como se describe en los siguientes apartados.

5.2.1 RAPID

A medida que a continuación se describa el desarrollo de la simulación, se comprenderá como el diseño del proyecto cobra más sentido. Hasta ahora solo teníamos la posición actual de las cajas en la cinta, pero las instrucciones de movimiento requieren un tiempo para ser finalizadas. Afortunadamente los planificadores de trayectorias de FANUC y ABB permiten especificar el tiempo en el que se tiene que realizar un movimiento cuya instrucción es:

MoveL *, v1000\T:=5, fine, grip3;

El TCP("Tool Center Point") de la herramienta, grip3, se mueve linealmente hacia un punto fino almacenado en la instrucción (marcado con un asterisco). Todo el movimiento requiere 5 segundos.

En segundo lugar es necesario establecer la comunicación con el cliente; la Rasberry Pi. Como se explicó en el apartado 4.2.4, RobotStudio actúa como servidor de la comunicación. Para ello tenemos que crear el código del servidor, teniendo un ejemplo en la página 478 del manual de RAPID [8], donde tendremos que poner la IP del PC en el que correrá la simulación. En mi caso utilizaré la IP: 192.168.1.44 y el puerto 55000. Esta comunicación es posible porque hemos instalado en la simulación el paquete PC Interface, y en el robot del laboratorio este paquete esta instalado también. Las principales funciones de este paquete son:

- SocketCreate server: Arranca un servidor.
- SocketBind server, "192.168.1.44", 55000: Hay que poner la IP del controlador y un puerto libre. Al utilizar el robot en simulación, la IP del controlador es la IP del PC donde corre la simulación.
- SocketListen server: Permitimos que el servidor pueda escuchar
- SocketAccept server, client: Se utiliza para aceptar peticiones de conexión entrantes.
- SocketReceive client\RawData:=data\Time:=WAIT_MAX: Esta instrucción bloquea la ejecución hasta recibir un mensaje y lo guarda los bytes recibidos en la variable data.
- UnpackRawBytes data, 1, message\ASCII:=50: Esta instrucción transforma los primeros 50 bytes recibidos en una cadena de caracteres ASCII.

- SocketSend client,\Str:=message: Con esta instrucción enviamos una cadena de caracteres al cliente.

Después de que la Rasberry Pi genere y envíe la cadena con el comando para el robot, según se indica en la sección 4.2.4, esta es recibida por RobotStudio, donde se descompone para finalmente construir la instrucción de movimiento. Por regla general una instrucción de movimiento de RAPID consta de los siguientes elementos:

MoveJ miPunto, v1000, z50, tool;

Siendo:

- MoveJ: Tipo de movimiento. Los principales son: [MoveJ, MoveL, MoveC].
- miPunto: es una instancia de la clase "robtarget", que contiene las coordenadas de un punto.
- v1000: es un objeto del tipo "speeddata" y contiene la velocidad consigna del TCP en mm/s durante el movimiento.
- z50: es un objeto del tipo "zonedata", que especifica el grado de distancia de aproximación al punto.
- tool: Es un objeto del tipo "tooldata" que contiene las coordendas del TCP(*Tool Center Point*).

En nuestro caso necesitamos guardar las coordenadas recibidas en un punto. Esto lo podemos hacer entendiendo la estructura de un objeto de la clase robtarget figura 5.3.

Luego, accederemos a cambiar las coordenadas de un punto de esta forma:

miPunto.trans.x := Coordenadas recibidas;

```
< dataobject of robtarget >
    < trans of pos >
        < x of num >
        < y of num >
        < z of num >
    < rot of orient >
        < q1 of num >
        < q2 of num >
        < q3 of num >
        < q4 of num >
    < robconf of confdata >
        < cf1 of num >
        < cf4 of num >
        < cf6 of num >
        < cfx of num >
```

Figure 5.3: Estructura de un objeto de la clase robtarget.

Lo siguiente que recibíamos en la cadena de caracteres era la orientación, respecto al eje longitudinal de la cinta de entrada, para variar la rotación del TCP de la herramienta. Se puede utilizar la instrucción siguiente, la cual aplica un desplazamiento en x,y (y) z y una rotación en grados en cualquier de estos ejes escribiendo \Rx/Ry/Rz:

MoveJ RelTool(miPunto,0,0,0\Rz:=Orientación), v1000, z50, tool;

Por último, una vez que el robot realiza la operación de "pick" o agarre sobre el objeto correctamente con su herramienta, realizaremos la operación de "place" o soltado de modo que los objetos del tipo 0 quedarán a la derecha del robot, mientras que las cajas del tipo 1 quedarán a las izquierda del robot tal y como se muestra en la figura

5.4. El TCP del robot se moverá a la velocidad que decidió la Raspberry Pi, siendo ésta resultado de la ecuación 4.1, pudiendo ir entre 0 mm/s o 1000 mm/s. Para establecer esa velocidad debemos crear un objeto de la clase speeddata y acceder al campo de la velocidad del tcp del siguiente modo:

miVelocidad.v_tcp := ***velocidad enviada***

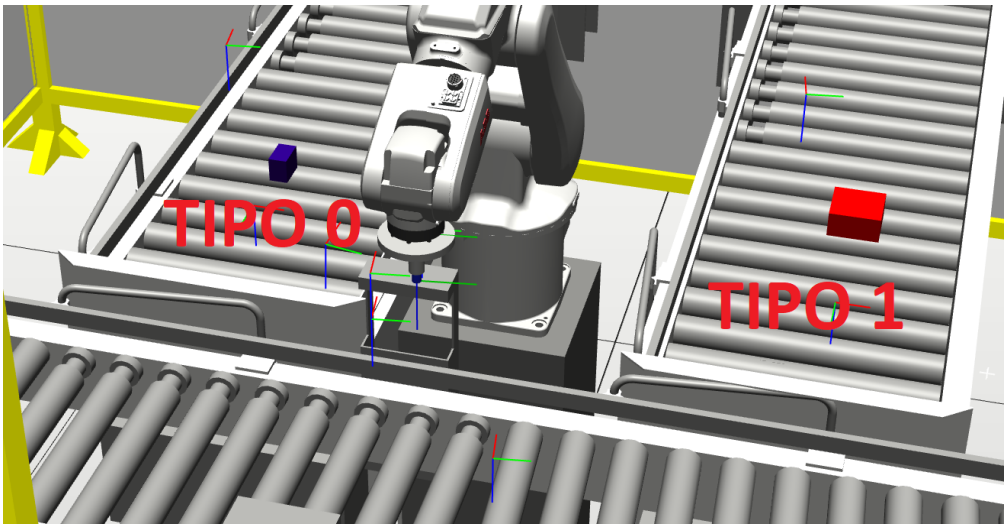


Figure 5.4: Tipos de objetos que llevarán las cintas de salida.

5.2.2 Simulación

Mediante lo visto en la sección 5.2.1, somos capaces de generar los movimientos del robot. El siguiente paso es realizar de forma visual el "pick and place" para así poder ver las cajas en movimiento en la cinta de entrada. Para realizar esta simulación utilizaremos 2 objetos, uno del tipo 0 y otro del tipo 1.

En primer lugar posicionaremos los objetos de forma arbitraria en el mismo plano XY, posteriormente, debemos identificar las coordenadas de inicio de los objetos para que la Raspberry Pi sepa donde estaban en el instante de tiempo T_0. RobotStudio per-

mite obtener cualquier punto respecto al sistema de coordenadas World (Preestablecido cuando se arranca la simulación, por defecto el robot se posicionará ahí cuando se especifica que robot se va a utilizar), Workobject (Sistema de coordenadas en la base del robot) o UCS (Cualquier sistema de referencia creado por el usuario). Al haber movido el robot 180º sobre el eje Z y haber elevado el robot 800mm en el eje Z el sistema World no es igual que el WorkObject. Así que a partir de este momento trabajaremos con el sistema de referencia WorkObject.

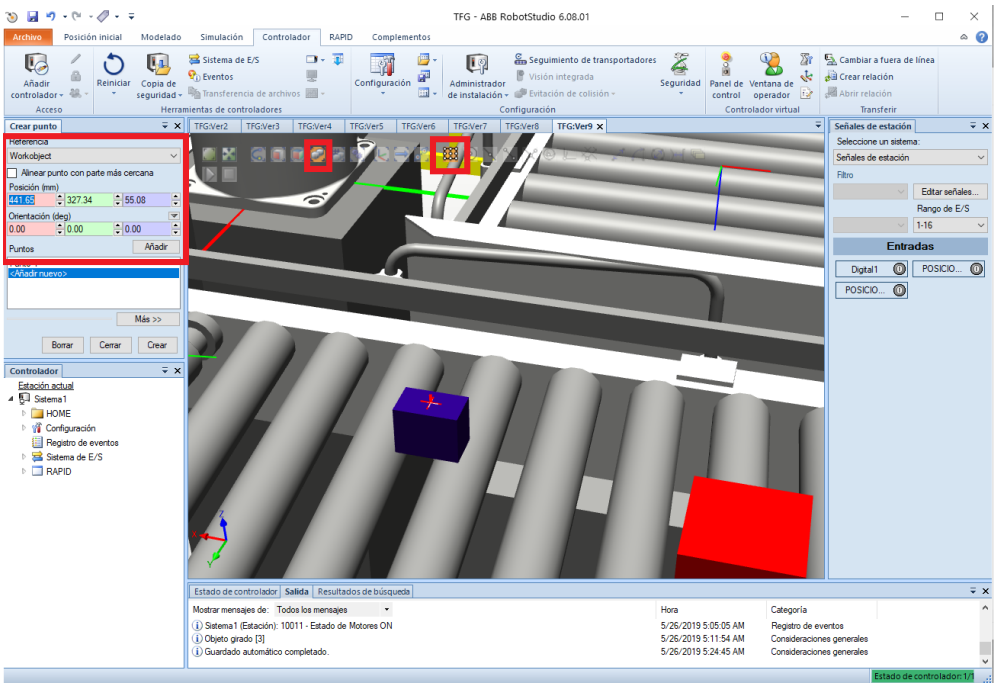


Figure 5.5: Esta imagen muestra obtener las coordenadas de los objetos.

La simulación de los movimientos de los objetos o la manipulación de los mismos se consigue mediante el uso de componentes inteligentes, en el video [9] podemos ver como utilizar algunos de ellos. En nuestro caso utilizaremos lo siguientes:

- Positioner x(N[1]): Este elemento es útil para restablecer la posición de los obje-

[1]Número de objetos en la cinta de entrada simuladas

tos que han sido manipulados. Rellenaremos sus campos con la posición en la que se encuentra la caja antes de ser movida respecto al sistema de coordenadas global. Este bloque se puede ejecutar manualmente o conectando una salida de la estación al bloque, de modo que activando una salida, se activará el bloque. Así lo hice usando la opción en Simulación > Simulador de E/S> Seleccione un sistema: Estación > añadir una señal digital de RobotStudio.

- Colision Sensor x(N+2):Este bloque se activa cuando existe una colisión o un acercamiento predefinido suficiente como para considerar que han colisionado entre 2 objetos. En nuestro caso, la herramienta y cada una de los objetos de entrada. Además habrá 2 objetos cuya función será detectar la colisión con la caja del place, y estos objetos estarán invisibles en la simulación, de forma que, en el momento en el que la ventosa choque con la caja invisible, se activará la señal de soltar el objeto que lleva en la herramienta.

- Attacher x(N): Este bloque nos permite unir 2 objetos estableciendo una relación "Parent and Child", de forma que el "Child" (el objeto) se moverá del mismo modo que el "Parent" (la herramienta) hasta que se ejecute un bloque "Deattacher" (ver siguiente párrafo), de modo que en cuanto se activa el bloque collision sensor con el objeto del pick, se activará el attacher.

- Deattacher x(2N): Este bloque separa el "Parent del Child" creados previamente, posicionando al "Child" en la parte del espacio en la que se activa el "Deattacher", de ahí que sea importante el bloque "Positioner" comentando previamente, para poder simular varias veces. El "Deattacher" se activará cuando salte el collision sensor entre la herramienta y la caja del place. Además también habrá un "Deattacher" por cada "Positioner", de modo que, siempre que se active la señal de la estación que activa el "Positioner", también desaparezcan las relaciones "Parent Child".

- LinearMover(2N): Simula un movimiento lineal de los objetos en las cintas de entrada y de salida y para ello hay que indicarle un velocidad en mm/s, la cual será la establecida en la tabla 6.1.

En la figura 5.6 se observa como se realiza la programación para la simulación de estos componentes inteligentes solo para coger un objeto, para que se vea más claro. En el caso de querer añadir mas de un objeto en la cinta de entrada,bastaría con duplicar el diagrama. La simulación real esta realizada con 2 objetos diferentes en la cinta de entrada.

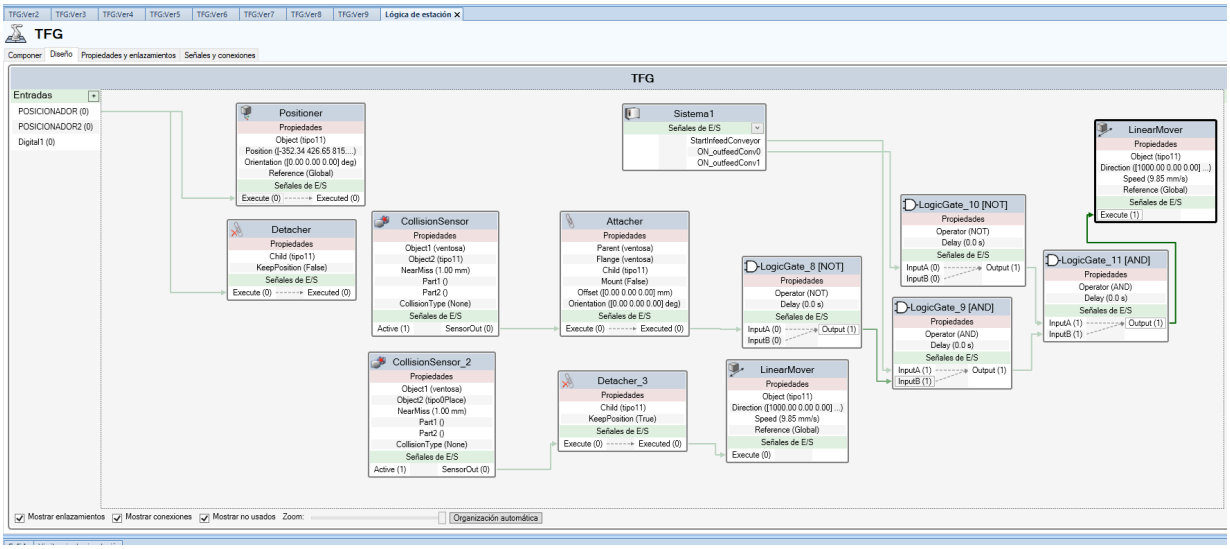


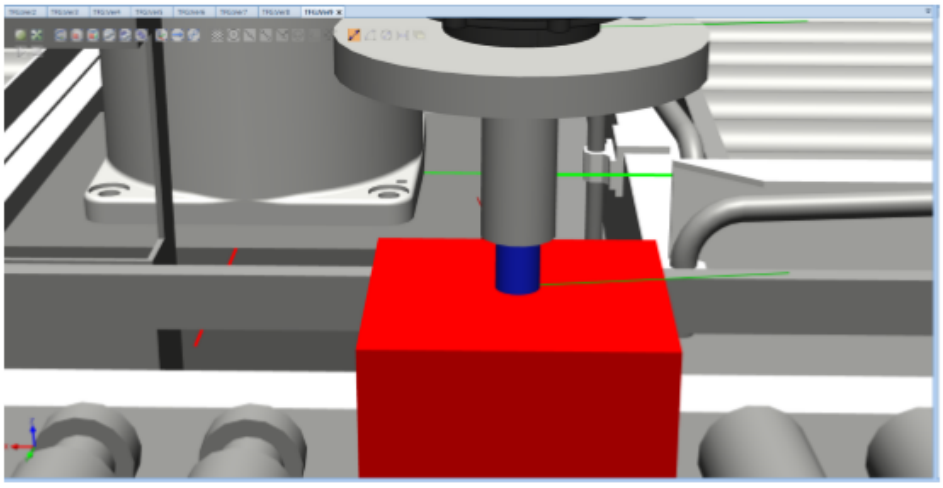
Figure 5.6: Diagrama de bloques para realizar la simulación.

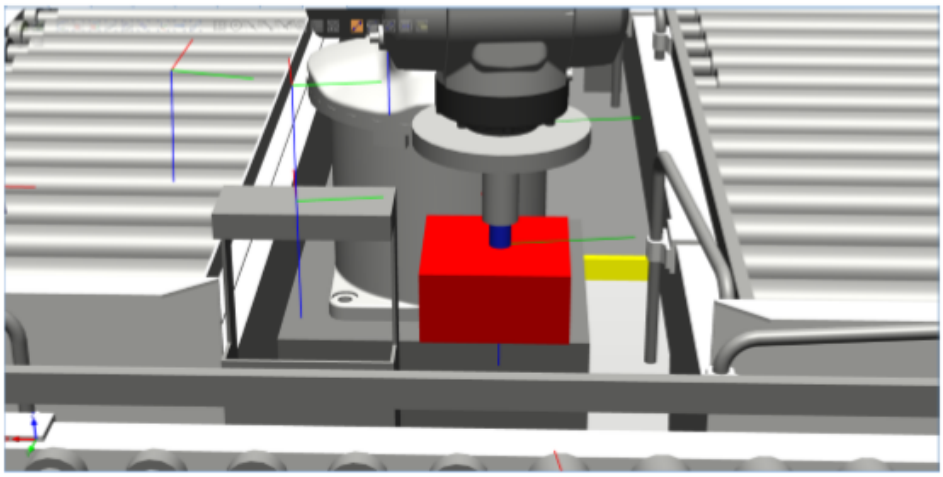
Para arrancar el proyecto, habrá que pulsar la opción “play” de la simulación en RobotStudio y luego ejecutar el programa de python llamado tfg.py

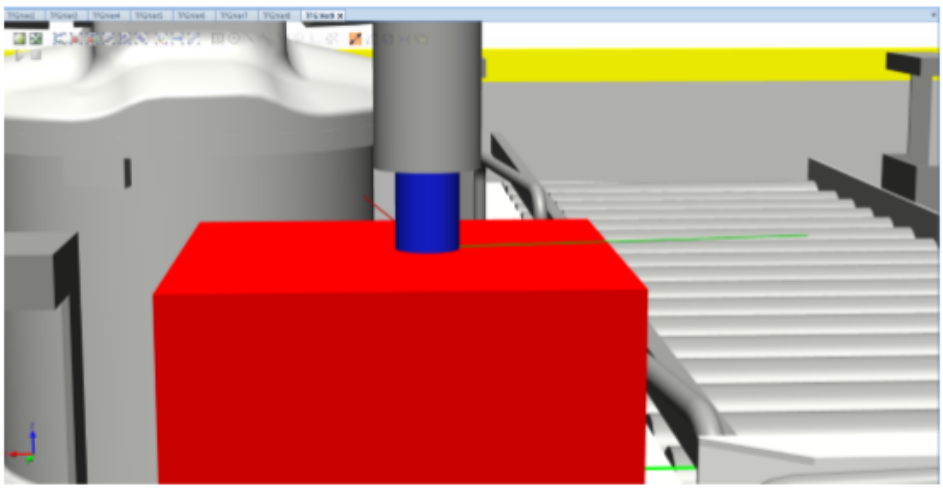
6 Resultados

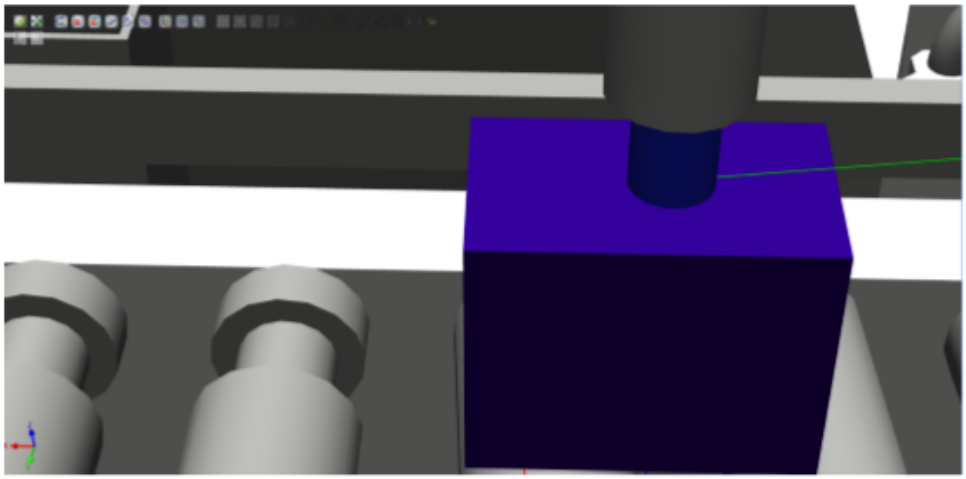
A continuación se describirán una serie de experimentos realizados para comprobar el funcionamiento de la aplicación desarrollada, en los que el objetivo será coger el objeto por el centro geométrico de la cara superior. En concreto, se han considerando diferentes tareas de agarre de los objetos de la cinta por parte del robot. Al final de este capítulo, se presentarán los datos mas relevantes de esos experimentos.

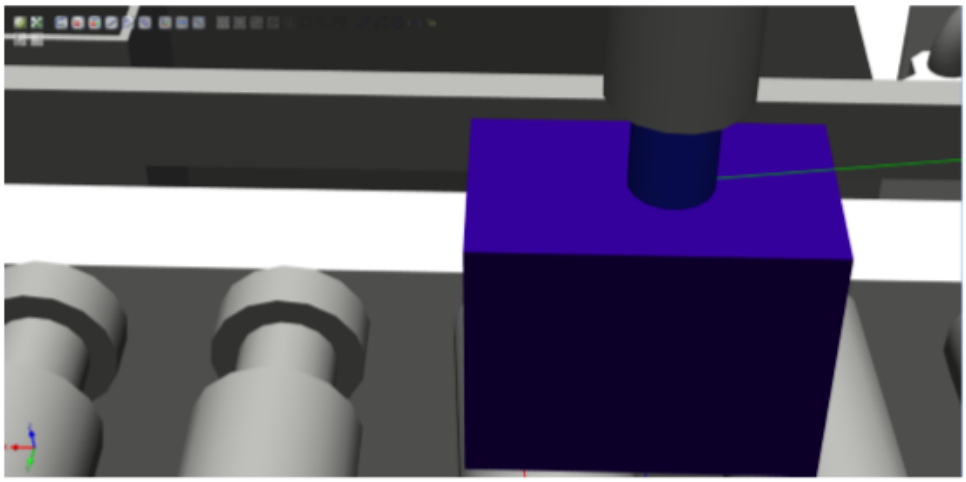
Las imágenes que aparecerán a continuación corresponden a la realización de la simulación en la que se coge primero la caja del tipo 1 y después la caja del tipo 2. Cada imagen es el resultado del "pick" de una ejecución distinta, para así posteriormente estudiar el error medio.

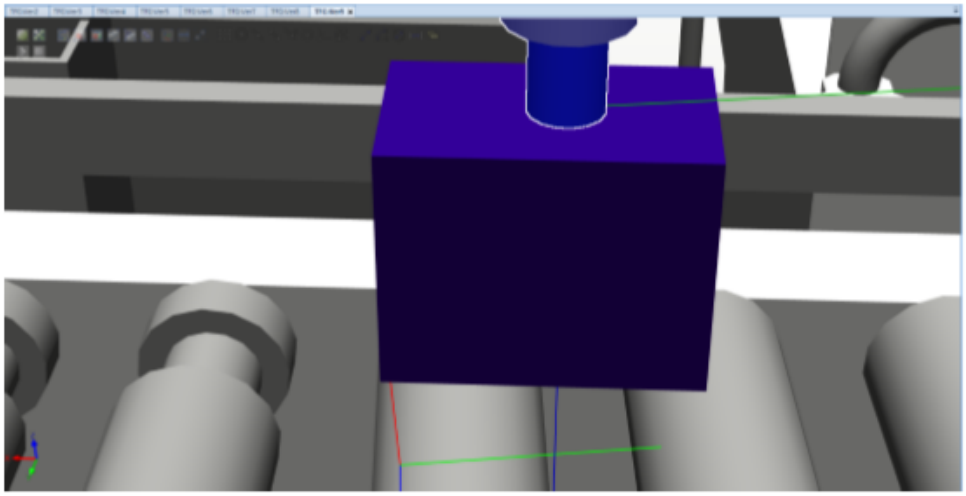












6.1 Conversión de pulsos a velocidad

En la figura 6.1 podemos observar los pulsos/s o mm/s reales para diferentes ejecuciones. Cuyo resultado teórico es 10 milímetros por segundo, al generarse 100 pulsos por segundo y tener una resolución de 10 pulsos por milímetro. Además también se observa las cadenas de caracteres enviadas a RobotStudio y las respuestas de este ("done") cuando ha realizado "place" del objeto en la cinta de salida

6.2 Tabla de resultados

A continuación se mostrará una tabla de resultados, donde el error está calculado utilizando la regla de RobotStudio, entre el centro de la ventosa y el centro de la cara superior de los objetos. En la columna del error, un error positivo implica que el robot lo ha cogido mas tarde de lo esperado. Por el contrario, un error negativo será que el robot se ha adelantando en coger el objeto.

Número de Ejecución	**Tipo de Caja**	**Velocidad Teórica (mm/s)**	**Velocidad Real (mm/s)**	**Error (mm)**
1	0	10	9.84	+3
2	0	10	9.81	+5
3	0	10	9.91	+4
4	1	10	9.85	+3
5	1	10	9.8	+2
6	1	10	9.77	+3

Tabla 6.1: Resultados tras distintas ejecuciones.

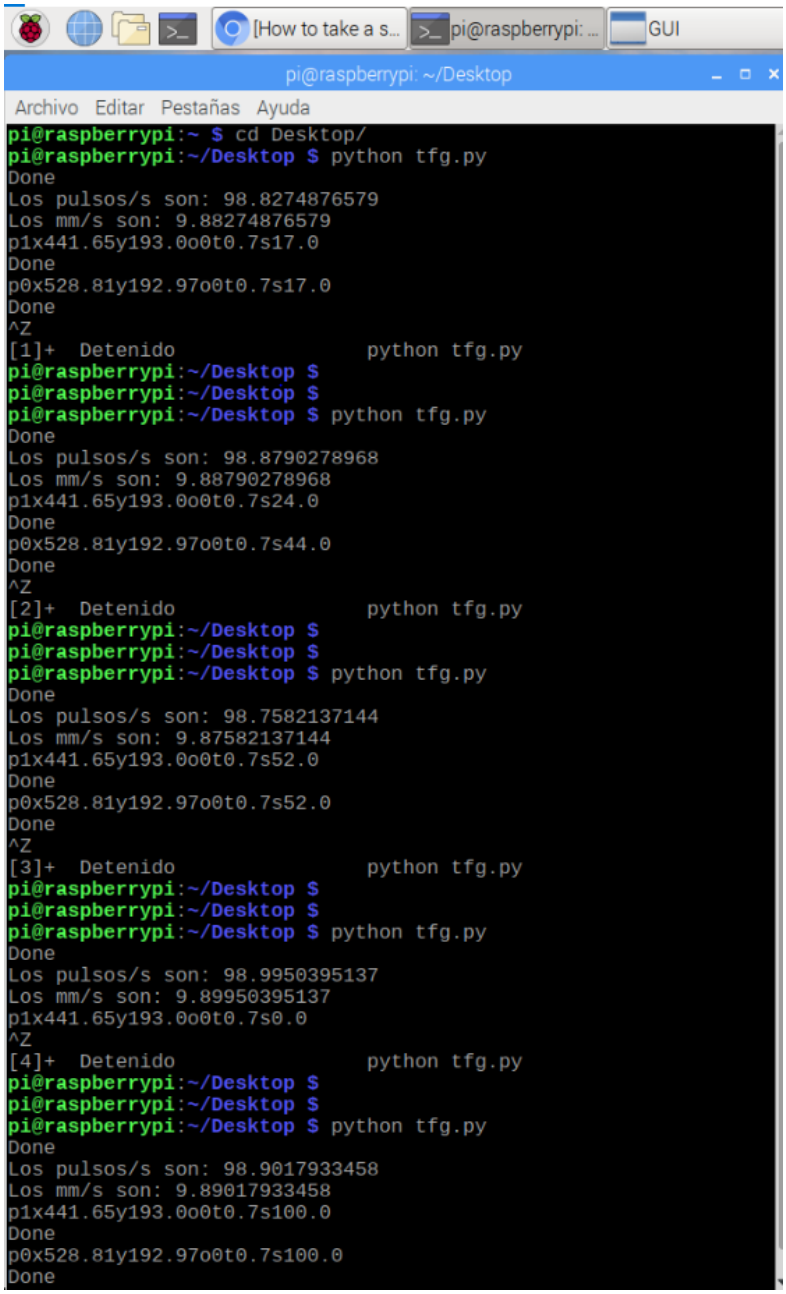


Figure 6.1: Salida obtenida al ejecutar el programa de planificación en la Raspberry Pi varias veces.

7 Conclusiones y trabajos futuros

En lo que se refiere a los objetivos que planteábamos al inicio de este documento, podemos concluir que la mayoría de los mismos han sido cumplidos, en tanto que hemos conseguido realizar el "Conveyor tracking" de una cinta gobernada con un sistema embebido de bajo coste utilizando herramientas gratuitas. No obstante los resultados no han sido lo suficientemente buenos como para poder ser aplicado directamente a la industria, donde la precisión juega un papel fundamental requiriéndose tolerancias del orden de décimas de milímetro, mientras que el sistema planteado tiene una precisión máxima de alrededor de 3 mm. Esto es debido principalmente a que ni Raspbian, el sistema operativo de la Raspberry Pi, ni el protocolo TCP/IP son sistemas de tiempo real. Raspbian está usando diferentes hilos de ejecución, lectura y escritura en disco, interfaz gráfica de usuario y simulación del encoder. Todo esto provoca una carga de trabajo en el microcontrolador que hace que resuelva un poco más lento de lo esperado a nivel teórico y, además, no podemos garantizar que se tarde el mismo tiempo en realizar cada iteración al no ser un sistema operativo determinista.

Además el hardware de la Rasberry Pi no es suficientemente robusto para un entorno industrial. Estos problemas del hardware y el software se podrían solucionar en gran parte optando por una versión más industrial de la Raspberry Pi (como por ejemplo Raspberry Pi Compute Module 3+ o Strato Pi [10] y [11] respectivamente), y por otro sistema operativo alternativo, aunque con un mayor coste de compra y de tiempo de puesta en marcha.

Como proyectos futuros me gustaría poder resolver la problemática de las comunicaciones entre las Raspberry Pi y el PLC S7-1200 de siemens, ya que tras muchos intentos no lo pude lograr. De este modo, sería el PLC el encargado de recibir los pulsos del encoder y no lo tendría que simular, mejorando el rendimiento de la aplicación y dotando a la célula de más robustez al constar de un PLC, un Robot y un microcontrolador de bajo coste, además de posibilitar la utilización de equipos y protocolos que garanticen tiempo real.

En cualquier caso, estoy realmente orgulloso del trabajo que he conseguido hacer, ya que he podido profundizar en los temas que más me han interesado (Raspberry Pi, RobotStudio, TIA-Portal) y lograr un resultado mejor de lo que esperaba. Siempre he sentido curiosidad por la programación en forma de multiprocesos, la generación de una interfaz gráfica de usuario y por la integración de equipos diferentes para diseñar tareas complejas. En este proyecto he encontrado una excusa para poder satisfacer todas estas inquietudes.

Bibliografía

[1] Anónimo. Raspberry Pi getting data from a s7-1200 plc. Simply Automationized. 2014. Consultado en mayo de 2019: http://simplyautomationized.blogspot.com/2014/12/raspberry-pi-getting-data-from-s7-1200.html.

[2] Kishlay Verma. Socket programming in python. Geeks for geeks. 2018. Consultado en marzo de 2019: https://www.geeksforgeeks.org/socket-programming-python/.

[3] Siemens. Scada system SIMATIC WinCC. Youtube. Consultado en abril de 2019: https://www.youtube.com/watch?v=gMFjV3kG5H4.

[4] Pilz GmbH & Co. Consultado en Mayo de 2019: https://www.pilz.com/es-ES/lexicon/muting.

[5] Mokhtar Ebrahim. Ejemplos de la GUI de python (tutorial de tkinter). Like Geeks. 2005. Consultado en mayo de 2019: https://likegeeks.com/es/ejemplos-de-la-gui-de-python/.

[6] Python Software foundation. 2019, Consultado en Marzo de 2019: https://docs.python.org/3/library/multiprocessing.html.

[7] Tim Peters. Python: How to use value and array in multiprocessing pool. StackOverflow. 2016. Consultado en mayo de 2019:

https://stackoverflow.com/questions/39322677/python-how-to-use-value-and-array-in-multiprocessing-pool.

[8] ABB. Instrucciones, funciones y tipos de datos de rapid. Manual de Referencia Técnica. 2010.

[9] Agustin Cañete López. Robotstudio - práctica de robótica nº 16 movimientos de objetos mediante componentes inteligentes. Youtube. Consultado en Mayo de 2019: https://www.youtube.com/watch?v=UmDubIZ$_ejIlist = LLAfFPeRxjF9ITHx7ceH6Ylw$.

[10] Raspberry Pi Foundation. Consultado en Mayo de 2019: https://www.raspberrypi.org/products/compute-module-3/.

[11] Sfera Labs. Consultado en Mayo de 2019: https://www.sferalabs.cc/strato-pi/.