UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Control distribuido del robot colaborativo UR3

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Alfonso Safont, Raúl

Tutor/a: Zotovic Stanisic, Ranko

CURSO ACADÉMICO: 2021/2022

Índice general

- **Agradecimientos** **4**
- **Resumen** **5**
- **Summary** **6**

1.Memoria **7**

Anexo 1 ..... **108**

2.Planos **130**

3.Presupuesto **138**

- ## Agradecimientos

Antes de comenzar a desarrollar este proyecto, quería dar las gracias a todas las personas que han contribuido directa o indirectamente en la realización de este trabajo:

A los alumnos que han compartido conmigo el laboratorio y hemos pasado muchas horas juntos, a los técnicos de laboratorio del ISA que siempre han estado en completa disposición para ayudar a solucionar los problemas.
A mi tutor Ranko Zotovic Stanisic por ayudarme en todo lo posible en la realización de este TFM.
Y por último, pero no por ello menos especial, dar las gracias a mi familia, pareja y amigos que sin su apoyo durante todos los años, pero en especial en este año último, no hubiese sido posible.

- **Resumen**

Los controladores de los robots colaborativos habitualmente son cerrados. Permiten al usuario enviar al robot un conjunto de comandos predeterminados, pero no permite implementar formas más complejas de control.
Por lo que el propósito de este proyecto, es realizar un control distribuido con robots UR3, donde parte del control se realice desde el propio robot, utilizando su interfaz y sus funciones, y la otra parte se realice mediante un PC externo, en este caso con lenguaje Python. De esta forma se aprovechan las funciones ya implementadas en el robot (cinematica, dinámica, etc) pudiéndolas ampliar con funciones propias del desarrollador.

Se estudiara la manera de poder conectarse al sensor externo de fuerza y par del robot, sin necesidad de utilizar la interfaz del robot, y poder así leer sus valores.

La parte fundamental del trabajo consistirá en la realización de una seria de experimentos para realizar un control coordinado de dos robots colaborativos UR3. El objetivo principal será comparar los 3 movimientos básicos del robot (MoveL , MoveJ y MoveP) para que en todos se pueda realizar un movimiento coordinado y posteriormente que ambos puedan transportar un objeto.

Con el Movimiento MoveJ, se puede producir una mayor desviación para la coordinación de ambos robots, por lo que se va a realizar un control por bucle interior-exterior, el lazo interior que esta implementado por el fabricante del robot, es un control cerrado y limitado, y el lazo exterior será un control por velocidad con un regulador proporcional. Para el control por velocidad se utilizara el comando SpeedL del robot UR3, donde el robot maestro se moverá haciendo una trayectoria y el robot esclavo lo seguirá.

Se estudiara el tiempo óptimo para conseguir una buena recepción y envió de datos entre los robots y el PC externo. Al ser este un tiempo mayor que el periodo de muestreo de los robots, puede causar una mayor desviación entre ellos, ya que este envío de datos se realizaría una menor cantidad de veces, por lo que se realiza un experimento donde se guardan las posiciones del maestro en un archivo de datos previamente, para así poder saber el robot esclavo la posición actual del maestro y su posición en el instante siguiente.

Por último, se realiza un estudio del tipo de generador de trayectoria que tiene el robot UR3, así como la viabilidad de realizar un experimento basado en la inversa del generador de trayectoria, observando los tiempos entre las fases de velocidad, su velocidad y su aceleración.

Para finalizar, se realizaran los planos de robot UR3, así como un presupuesto del proyecto.

Palabras clave:

Control distribuido. Robot Colaborativo. Robot UR3. Python.

- **Summary**

The collaborative robot controllers are usually closed source. It allow the user to send default commands to the robot, but not enable to implement more complex ways of control.

Thus, the purpose of this project is to perform a distributed control using UR3 robots, where a part of the control be carried out by the robot itself, using its interface and functions, and the other side is realised using an external PC, in this case, in Python language. This way, we can benefit from the functions implemented at the robot (kinematic, dynamic, etc) and will extend it with developer own functions.

Will be consider the way to connect to the external robot sensors of power and torque, without the need to use the robot interface, and this way can read its values.

The essential part of the work will consist on the completion of a set of experiments to make a coordinated control of two collaborative UR3 robots. The main objective will be compare the tree basic robot movements (MoveL, MoveJ and MoveP) so that they can perform a coordinated movement and subsequently, that both may transport an object.

The MoveJ movement may cause a greater deviation to the coordination of both robots. To avoid this, an internal-external loop control will be performed. The internal link implemented by the robot manufacturer is a closed and limited control. The external link will be a speed control with a proportional regulator. For the speed control will be use the SpeedL command of the UR3 robot, where the master robot will move performing a path, and the slave will follow it.

I will study what is the optimal time to achieve a good sending and receiving data between the robots and the external PC. As a bigger time that the sampling period, this may result in a greater deviation between the robots, as the data sending would be performed a smaller number of times. Therefore, a experiment is realized, where we save the master robot positions in a data file, so that the slave robot may know the current position of the master robot and its position and the next moment.

Finally, the path generator type of the UR3 robot is studied, just as the viability of doing an experiment base on the reverse of the path generator, monitoring the time between the speed stages, the current speed and its acceleration.

To conclude, the UR3 robot drawings will be made, as well as a budget of the project.

Keywords:

Distributed control. Collaborative robot. UR3 robot. Phyton.

UNIVERSIDAD POLITECNICA DE VALENCIA

Dpto. de Ingeniería de Sistemas y Automática

Control distribuido del robot colaborativo UR3

1. MEMORIA

TRABAJO DE FIN DE MASTER

Master Universitario en Automática e Informática Industrial

AUTOR/A: Alfonso Safont, Raúl

Tutor/a: Zotovic Stanisic, Ranko

Curso Académico: 2021-2022

Índice

1. Memoria 7
1.1. Objetivos 7
1.2. Antecedentes 7
1.3. Factores a considerar 10
1.3.1. Robots Colaborativos 10
1.3.2. Universal Robots 12
1.3.3. Protocolos Nivel de Transporte 13
1.3.4. Gestión. Normas ISO 16
1.3.5. Generador de trayectorias 17
1.3.6. Tipos de trayectorias 19
1.3.7. Planificación 20
1.4. Planteamiento de soluciones alternativas y justificación de la solución adoptada 21
1.5. Descripción detallada de la solución adoptada 24
1.6. Justificación detallada de los elementos o componentes de la solución adoptada. 25
1.6.1. Sensor On robot 25
1.6.2. Compute Box 26
1.6.3. Polyscope 27
1.6.4. Parámetros Denavit-Hatenberg 31
1.6.5. Limitaciones de velocidad y posición 31
1.6.6. Limitaciones de movimiento 32
1.7. Experimentos 33
- Entorno del laboratorio 33
- Cambio del sistema de coordenadas 34
1.7.1. Lectura de datos del Sensor 41
1.7.2. Experimento 1: Movimiento del Robot UR3 48
1.7.3. Experimento 2: Movimiento coordinado de dos robots colaborativos. 58
1.7.4. Experimento 3. Movimiento coordinado de dos robots cogiendo un objeto 65
1.7.5. Experimento 4. Funcion get_target_tcp_pose 71
1.7.6. Experimento 5: Comunicación por Modbus 76
1.7.7. Experimento 6: Control bucle interior, bucle exterior 80
1.7.8. Inversa del generador de trayectoria 89
1.8. Conclusiones 99
1.9. Bibliografía 100

Índice de Figuras

Fig1. Robot Yumi 7
Fig2. Robot PhantomX Pincher 8
Fig3. Robot Siscom 9
Fig4. Grafica trayectoria trapezoide 17
Fig5. Grafica trayectoria perfil en S 18
Fig6. Grafica trayectoria Perfil en S parcial 18
Fig7. Robot UR3e de perfil 25
Fig8. Sensor On robot 25
Fig9. Sensor On robot instalado en el robot UR3 25
Fig10. Compute Box 26
Fig11. Conexiones Compute Box 27
Fig12. Pantalla principal Polyscope 28
Fig13. Pantalla inicio UR3 28
Fig14. Pestaña inicialización del robot 29
Fig15. Primera pestaña de inicialización 30
Fig16. Segunda pestaña de inicialización 30
Fig17. Límites de velocidad de cada articulación del robot UR3 31
Fig18. Límites de posición de cada articulación del robot UR3 32
Fig19. Ejemplo de Singularidad 32
Fig20. Ejemplo de una articulación fuera de rango 33
Fig21. Ejemplo de puntos alcanzables con diferente configuración 33
Fig22. Espació de trabajo con UR3 y UR3e 34
Fig23. Espació de trabajo con dos UR3e 34
Fig24. Configuración de un plano de referencia 36
Fig25. Pantalla con un plano definido 37
Fig26. Pantalla de Mover 38
Fig27. Ecuación transformada 38
Fig28. Matriz de Rotación en x,y,z 39
Fig29. Espacio de trabajo con representación de ejes 40
Fig30. Espacio de trabajo de los dos robots UR3e con indicación de ejes 40
Fig31. Pantalla Move UR3e 41
Fig32. Página web del sensor 42
Fig33. Estructura comandos UDP 43
Fig34. Estructura datos de salida UDP 43
Fig35. Estructura de comandos TCP 44
Fig36. Datos de salida TCP 44
Fig37. Pantalla del Hercules para UDP 45
Fig38. Pantalla del Hercules para UDP con los comandos a utilizar 46
Fig39. Pantalla Hercules para TCP 47
Fig40. Programa movimiento individual UR3 50
Fig41. Respuesta del codigo Python al movimiento x 50
Fig42. Respuesta del codigo Python al movimiento y 51
Fig43. Respuesta del codigo Python al movimiento z 51
Fig43. Codigo de Python del recibiento de datos 52
Fig44. Codigo de Polyscope movimientoo individual UR3e 53

Fig45. Instrucción de Python con la nueva posición 54
Fig46. Respuesta del codigo de Python al movimiento en x 55
Fig47. Respuesta del codigo de Python al movimiento en y 56
Fig48. Instrucción de Python con la nueva posición variando z 56
Fig50. Codigo Polyscope con Path_offset_set 58
Fig51. Codigo Polyscope UR3e y UR3 para movimiento coordinado 59
Fig52. Primera toma de datos de las posciones con MoveJ 60
Fig53. Segunda toma de datos de las posciones con MoveJ 60
Fig54. Tercera toma de datos de las posciones con MoveJ 61
Fig55. Cuarta toma de datos de las posciones con MoveJ 61
Fig56. Primera toma de datos de las posciones con MoveL 62
Fig57. Segunda toma de datos de las posciones con MoveL 62
Fig58. Tercera toma de datos de las posciones con MoveL 63
Fig59. Cuarta toma de datos de las posciones con MoveL 63
Fig60. Primera toma de datos de las posciones con MoveP 64
Fig61. Segunda toma de datos de las posciones con MoveP 64
Fig62. Tercera toma de datos de las posciones con MoveP 64
Fig63. Cuarta toma de datos de las posciones con MoveP 65
Fig64. Codigo devuelto por Python con movimiento MoveL 67
Fig65. Codigo devuelto por Python con movimiento MoveJ 68
Fig66. Codigo devuelto por Python con movimiento MoveP y 5mm de radio 69
Fig67. Codigo devuelto por Python con movimiento MoveP y 2mm de radio 70
Fig68. Codigo devuelto por Python con movimiento MoveL 72
Fig69. Codigo devuelto por Python con movimiento MoveJ 73
Fig70. Codigo devuelto por Python con movimiento MoveP 74
Fig71. Codigo devuelto por Python con movimiento MoveP 75
Fig72. Pantalla configurtación Modbus UR3 76
Fig73. Pantalla configuración Modbus UR3e 77
Fig74. Codigo Polyscope programa basico en UR3e 78
Fig75. Codigo Polyscope programa basico en UR3 78
Fig76. Pantalla configuración Modbus UR3e maestro 79
Fig77. Pantalla configuración Modbus UR3e esclavo 79
Fig78. Codigo Polyscope para movimiento coordinado con Modbus 80
Fig79. Esquema control por bucle interior-exterior 81
Fig80. Codigo Pyhon matriz de transformación 82
Fig81. Codigo Pyhon calculo de errores 82
Fig82. Codigo Pyhon calculo velocidad de referencia 82
Fig83. Codigo Pyhon dato a enviar 83
Fig84. Codigo Polyscope para el control por bucle interior-exterior 84
Fig85. Respuesta a la prueba con ninguna espera 85
Fig86. Respuesta a la prueba con 0.05s de espera 86
Fig87. Respuesta a la prueba con 0.02s de espera 87
Fig88. Codigo Polyscope para guardar las posiciones 88
Fig89. Posición de las articulaciones 89
Fig90. Velocidad de las articulaciones 90
Fig91. Aceleración de las articulaciones 90
Fig92. Velocidad de las articulaciones 91

Fig93. Velocidad de las articulaciones 91
Fig94. Velocidad de las articulaciones 92
Fig95. Velocidad de las articulaciones 93
Fig96. Posición de las articulaciones 93
Fig97. Velocidad de las articulaciones 94
Fig98. Velocidad de las articulaciones 94
Fig99. Posición de las articulaciones 95
Fig100. Velocidad de las articulaciones 95
Fig101. Velocidad de las articulaciones 96
Fig102. Posición de las articulaciones 96
Fig103. Velocidad de las articulaciones 97
Fig104. Velocidad de las articulaciones 97
Fig105. Posición de las articulaciones 98

Índice de Tablas

Tabla1. Especificaciones robot Yumi 8
Tabla2. Especificaciones robot PhantomX 9
Tabla3. Diagrama de Gantt 20
Tabla4. Especificaciones robot UR3/UR3e 22
Tabla5. Especificaciones robot UR5 22
Tabla6. Especificaciones robot UR10 23
Tabla7. Especificaciones robot UR16 23
Tabla8. Especificaciones robot UR20 23
Tabla9. Especificaciones sensor On robot 26
Tabla10. Parametros Denavit-Hatenberg robot UR3 31
Tabla11. Lista de comandos para la comunicación con el sensor 42
Tabla 12. Comparación de posición del UR3 con MoveJ 61
Tabla13. Comparación de las posciones del UR3 con MoveL 63
Tabla14. Comparación de la posición de UR3 con MoveP 65
Tabla 15. Comparación desviaciones de las dos funciones anteriores 76
Tabla 16. Comparación del tiempo 1, del tiempo 2 y de la aceleración con sus desviaciones 98

1. Memoria

1.1. Objetivos

En este trabajo académico se va a promover la realización de la comunicación entre un ordenador y robots UR3. Se realizaran diferentes experimentos para comprobar con que tecnología se consigue una mejor comunicación en tiempo real.

- Realización de la comunicación entre el UR3 y el ordenador como con el UR3e y el ordenador mediante Python.
- Medición en tiempo real de la fuerza realizada por el robot UR3
- Envío y recepción de los datos de posición al robot UR3 y UR3e
- Movimiento coordinado de ambos robots mediante cambio de sistema de coordenadas
- Movimiento coordinado de ambos robots realizando un trabajo de pick and place
- Movimiento coordinado mediante MODBUS
- Movimiento coordinado mediante control bucle interior y exterior.

1.2. Antecedentes

Se ha observado otros proyectos similares donde realizaban un control con 2 brazos robóticos. Sin embargo en ninguno de ellos se ha utilizado o bien dos robots separados, sin tener una parte en común, o bien no se ha utilizado ningún lenguaje para su programación.

- **Robot Yumi**

Fig1. Robot Yumi

[1]En 2015, ABB presentó YuMi, el primer robot verdaderamente colaborativo del mundo YuMi cambió las reglas del juego y anunció una nueva era en la que las personas y los robots trabajan de manera segura y productiva uno al lado del otro, sin barreras.

Los robots colaborativos son expertos en agregar flexibilidad a los procesos de ensamblaje que necesitan hacer pequeños lotes de productos altamente individualizados, en ciclos cortos.

Al combinar la capacidad única de las personas para adaptarse al cambio con la resistencia incansable del robot para tareas precisas y repetitivas, es posible automatizar el ensamblaje de muchos tipos de productos en la misma línea.

[2]YuMi ha establecido el estándar de seguridad en aplicaciones colaborativas donde los robots y los trabajadores manuales necesitan trabajar de manera conjunta. Con su destreza y diseño de doble eje y 14 ejes, YuMi ha desempeñado un papel clave en la mejora de la productividad y la calidad en las líneas de producción en todo el mundo.

Tipo	Articulado, colaborativo
Numero de ejes	14 ejes
Carga máxima	500 g
Radio de acción	559 mm
Velocidad	25 mm/s
Repetibilidad	0.02 mm
Peso	38 kg

[3] **Tabla1.** Especificaciones robot Yumi

- **PhantomX Pincher** de 4 grados de libertad utilizando matlab como método de programación.

Fig2. Robot PhantomX Pincher

[4]El brazo PhantomX Pincher AX-12 es un brazo robótico de 5 grados de libertad y fácil adición a la plataforma del robot TurtleBot ROS. No dispone de un software compatible. Los ejemplos de ROS están disponibles y son de código abierto.

Peso	550 g
Alcance vertical	35 cm
Alcance horizontal	31 cm
Fuerza	25 cm/ 40g; 20cm/70g 10 cm/100 g
Fuerza pinza	500 g
Fuerza elevación de la muñeca	250 g

Tabla2. Especificaciones robot PhantomX

[5]Mediante 2 robots PhantomX se realiza un control con los dos brazos robóticos para trabajar sincronizados utilizando Matlab.

- **Agarre bimanual de objetos asistido por visión**

[6]La plataforma robótica consta de un torso metálico con articulación rotacional en la cadera y de dos manipuladores industriales, de 7 grados de libertad, que actúan como brazos.

- **SISCOM**

[7]Sistema cooperativo multi-robot basado en 2 robots colaborativos UR para la impresión 3D de muros, columnas, etc…

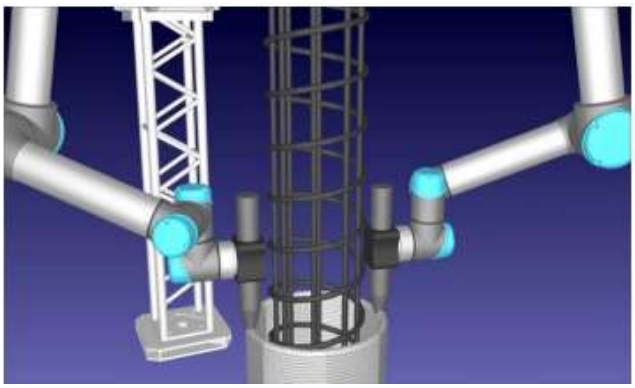


Fig3. Robot Siscom

El sistema paralelo de construcción impresa, también conocido como Delta, está compuesto fundamentalmente por tres brazos articulados concurrentes o cables en tensión

concurrentes. Los eslabones de la cadena cinemática de cada brazo se conectan mediante, ya sea una unión prismática (traslacional) en la dirección del eje vertical Z, o una unión de revolución (rotacional) en torno a uno de los ejes horizontales X o Y y dos uniones universales (rotacionales) en torno a uno de los ejes horizontales X o Y y en torno al eje vertical Z.

La boquilla convencional por donde se extruye el material se desplaza con tres grados de libertad y con una única orientación fija.

1.3. Factores a considerar

1.3.1. Robots Colaborativos

Los robots colaborativos o Cobots son robots articulados industriales capaces de trabajar junto a los humanos sin necesidad de un sistema de seguridad externo, caracterizados por ser ligeros, flexibles y fáciles de instalar.
[8]Los cobots liberan a los trabajadores de las tareas más peligrosas, repetitivas y pesadas reduciendo las bajas y las enfermedades laborales. Además, cuentan con sensores capaces de detectar la presencia humana y actuar en consecuencia.

[9]Estos también se pueden aplicar a las acciones o puestos de trabajo que requieran posturas más incomodas o que pongan en riesgo la salud de los trabajadores. Igualmente, puedes redirigir su trabajo a operaciones más pequeñas que no ocupen el tiempo de los funcionarios de la empresa. También se pueden utilizar en otro tipo de operaciones donde sean expuestos a altas temperaturas, trabajen con materiales tóxicos o empleen utensilios punzantes o cortantes.

Principales características frente a los robots industriales clásicos:

- ✓ Mas portátiles
- ✓ Más versátiles
- ✓ Más colaborativos
- ✓ Más rentables
- ✓ Más seguros

El mercado mundial de los robots industriales está experimentando un crecimiento sin precedentes en los últimos años. A medida que las tecnologías de la robotización evolucionan, su uso se extiende cada vez más en sectores como la automoción y la electrónica, actualmente considerados como los principales impulsores de su crecimiento. Lo más importante es que el coste de estas soluciones de automatización ya no representa una barrera para las pequeñas o medianas empresas, dado que un cobot se puede adquirir por menos de 25.000 €. Ante este horizonte, los cobots son una excelente herramienta para optimizar los sistemas productivos.

Los cobots son, en la actualidad, el segmento de mayor crecimiento de la automatización industrial. Según datos de la Robotic Industries Association (RIA), se espera que en 2025 se multipliquen por diez hasta alcanzar el 34% de todas las ventas de robots industriales.

[10]La ISO define 4 tipos de modos de funcionamiento de los robots colaborativos:

- Limitación de potencia y fuerza

Según la especificación técnica *ISO/TS 15066:2016,* si ocurre un contacto intencionado o accidental entre robots y personas, dicho contacto no puede provocar lesiones ni dolor. Esta norma define más concretamente los modos de funcionamiento y complementa a la norma *UNE-EN ISO 10218-2:2011.*

En caso de contacto, los robots deben ejercer una fuerza y/o potencia limitada que garantice la seguridad de las personas. Estos límites o valores fueron establecidos en un estudio realizado por el Instituto para la Seguridad y la Salud en el Trabajo del Seguro Social Alemán de Accidentes (IFA), en cooperación con la Universidad de Maguncia. En el estudio se analizan diferentes partes del cuerpo y la cabeza, teniendo también en cuenta si el contacto es cuasi estático o transitorio.

Para aplicar este modo de operación es necesario evaluar el alcance y la posición de los robots, estimar las fuerzas de contacto y validar con mediciones que los límites de fuerza y/o potencia de la aplicación no produzca dolor o lesión.

- Parada segura motorizada

En este modo de funcionamiento, cuando una persona entra en la zona de trabajo colaborativo, el cobot se detiene y permanece inmóvil**.** Según el campo de detección (interno, del área de carga o de entrada y salida), el robot podrá reiniciar sus tareas de forma manual o automática, cuando la persona abandone la zona de trabajo.

Para aplicar una parada segura monitorizada, debe calcularse la correspondiente distancia de seguridad, teniendo en cuenta el tiempo de parada del robot, la velocidad de aproximación del operario y las necesidades de la aplicación.

- Guiado Manual

En el modo de guiado manual, el cobot también se paraliza cuando alguien entra en la zona de trabajo colaborativo y puede reiniciarse automáticamente cuando la persona abandona esta zona. Lo que diferencia a este modo de operación es que, mientras está paralizado, el cobot puede moverse si es guiado por una persona (mediante contacto directo).
El guiado manual exige una velocidad reducida de trabajo y el uso de un mando de validación (sensitivo) que la persona deberá mantener pulsado en todo momento.

- Control de velocidad y separación

Con este modo de operación, en cualquier momento una persona puede acercarse a un cobot de forma segura, ya que la distancia entre la persona y el cobot está controlada (si es necesario, el cobot puede modificar su trayectoria para evitar el contacto directo). Además, la velocidad se adapta a la distancia de separación**,** llegando a paralizarse si la persona se acerca demasiado.

Son varios los fabricantes que ya han creado su propia marca de robots colaborativos debido a su importante papel hacia una Industrial 4.0:

- Omron
- Kuka
- ABB
- Universal Robots

1.3.2. Universal Robots

Fue fundada en 2005, su único objetivo era que los robots fueran accesible para la pequeña y mediana empresa. En 2008 vendió su primer robot, este significo un cambio en la automatización con robots, y puso sobre la mesa la robótica colaborativa.
Consagrados en la fabricación de robots colaborativos, poseen tres cobots UR3 que se integran de manera muy sencilla en cualquier entorno de producción. Todos ellos, gozan de 6 ejes articulados y de un brazo robótico que emula los movimientos de un brazo humano.

UR ofrece una serie de ventajas frente al resto de fabricantes como su facilidad de programación, su facilidad de montajes y el entorno UR, donde el cliente dispondrá de análisis remoto del cobot, análisis in situ, plan de disponibilidad, formación e inventario de las piezas de repuesto.

Dentro de UR la gama e-Series. La gama e-Series ha sido diseñada pensando en el futuro, en la conocida como industria 5.0 que pone a las personas en el centro de los procesos de producción, y en la que los cobots interactúan con los humanos.

[11]Ventajas de la gama e-Series:

1. **Colaborativos y seguros**
   Los robots colaborativos de la gama e-Series están preparados para realizar tareas repetitivas, tediosas y peligrosas. El tiempo y la distancia de parada son ajustables, y su capacidad de detención por colisión garantiza una óptima seguridad cuando trabajan junto con los operadores.

2. **Rápida instalación**
   No necesita instalaciones eléctricas especiales permite que se puedan conectar a cualquier toma de corriente**.** A ello hay que sumar que su intuitiva interfaz garantiza una sencilla configuración, programación y una perfecta integración en la línea de producción.

3. **Flexibilidad**
   Gracias a su ligereza y pequeño tamaño los robots colaborativos pueden reubicarse fácilmente para nuevas tareas sin modificar los esquemas de producción. Además, se pueden reutilizar programas, lo que permite una gran flexibilidad de automatizar múltiples tareas manuales**.**

4. **Facilidad de programación**
   La tableta de programación intuitiva de los cobots de Universal Robots es tan fácil de usar que cualquier operario puede aprender a programarlo moviendo los ejes del robot manualmente o utilizando los las funciones de “arrastrar y colocar” de la pantalla táctil.

5. **Rápida amortización**
   La última de las ventajas de la nueva gama es que cuenta con la amortización más rápida de la industria, con un retorno de la inversión incomparable a las costosas soluciones tradicionales.

1.3.3. Protocolos Nivel de Transporte

- **TCP**[12]

Es un servicio con conexión. Los datos son una secuencia o trama de bytes. Cada comunicación secciona la trama mediante segmentos. Las tramas de TCP se les llaman segmentos.
La conexión requiere tres etapas: Establecimiento de conexión, transferencia de datos y liberación de conexión.
El campo FLAGS de 6 bits es utilizado para determinar el propósito y contenido del segmento. Estos bits indican cómo interpretar otros campos en el encabezado.

TCP usa una transmisión dúplex integral (full-duplex), es decir, pueden enviarse dos flujos de datos simultáneamente en direcciones opuestas. En consecuencia, la aplicación de destino puede enviar información de control o datos de vuelta a la aplicación emisora mientras esta continua enviando datos.

TCP permite al emisor tener varios segmentos pendientes antes de que el receptor envié un reconocimiento. Cuando el nodo emisor recibe el reconocimiento, indica a la aplicación que los últimos datos se enviaron satisfactoriamente, si el nodo emisor no recibe el reconocimiento de un segmento, en un periodo de tiempo determinado, volverá a retransmitir este segmento. Este esquema, llamado retransmisión con acuse de recibo, asegura que la entrega de flujo sea correcta.

Funciones de TCP:

- Establecer y terminar conexiones
- Gestionar los buffers y ejercer control de flujo de forma eficiente.
- Multiplexar el nivel de aplicación (port) e intercambiar datos con las aplicaciones
- Controlar errores, retransmitir segmentos perdidos o erróneos y eliminar duplicados
- Efectuar control de congestión

- **UDP**

Es un protocolo sencillo, en el que no hay conexión, basado en el intercambio de datagramas. Los mensajes se denominan datagramas UDP. UDP multiplexa los datos de las aplicaciones y efectúa opcionalmente una comprobación de errores, pero no realiza:

- Control de flujo
- Control de congestión
- Retransmisión de datos perdidos
- Conexión/desconexión

Se utiliza en los siguientes entornos:

- Si el intercambio de mensajes es muy escaso
- Si una aplicación es de tiempo real y no puede esperar los ACKs
- Cuando los mensajes se producen regularmente y no importa si se pierde alguno
- Cuando el medio de transmisión es altamente fiable y sin congestion (LANs).
- Cuando se envía tráfico broadcast/multicast

- **Sockets**

Un Socket es una terminal de una línea de comunicación bidireccional además de un punto final de un enlace de comunicación de dos vías entre dos programas que se ejecutan a través de la red.

Siguen el patrón de comunicación cliente – servidor. El cliente y el servidor deben ponerse de acuerdo sobre el protocolo que utilizarán.

Es una combinación de dirección IP y el número de puerto. Un socket identifica un proceso de red de manera única en internet. Una conexión TCP queda especificada por los dos sockets que se comunican.

Pueden vincularse (bind) con una dirección. Si se utiliza TCP/IP está formada por una tripleta: (familia-prot, dir-ip, port). Existen dos estilos de comunicación: Socket orientados a la conexión y Socket no orientados a la conexión.

- **Modbus**

[13]Modbus es un protocolo de comunicación abierto, utilizado para transmitir información a través de redes en serie entre dispositivos electrónicos. El dispositivo que solicita la información se llama maestro Modbus y los dispositivos que suministran la información son los esclavos Modbus.
En realidad, esto significa que un dispositivo esclavo no puede ofrecer información; debe esperar a que se le pida. El maestro escribirá datos en los registros de un dispositivo esclavo y leerá los datos de los registros de un dispositivo esclavo.

Esta red de comunicación industrial usa los protocolos RS232/RS485/RS422**.** Su simplicidad y el hecho de que los fabricantes pueden incorporarlo en sus productos sin cargo alguno han ayudado a que se convierta en el método más popular de conexión de dispositivos electrónicos industriales.

Existen varios tipos de versiones en el protocolo Modbus para el puerto serie y Ethernet, que se utilizan para atender las necesidades específicas de los sistemas de automatización industrial en las empresas.

[14]La consulta de un maestro consistirá en una dirección de esclavo (o la dirección de difusión), un código de función que define la acción solicitada, todos los datos necesarios, y un campo de comprobación de errores. La respuesta de un esclavo se compone de campos confirman la acción realizada, los datos que se devolverán, y la comprobación de un error de campo.

Las más comunes son:

- Modbus RTU
- Modbus TCP
- Modbus ASCII
- Modbus Plus

- **Modbus TCP**

Es simplemente el protocolo Modbus RTU con una interfaz TCP que se ejecuta en Ethernet. Modbus TCP / IP utiliza TCP / IP y Ethernet para transportar los datos de la estructura del mensaje Modbus entre dispositivos compatibles. Es decir, Modbus TCP / IP combina una red física (Ethernet), con un estándar de red (TCP / IP), y un método estándar de representación de datos (Modbus como el protocolo de aplicación). En esencia, el mensaje Modbus TCP / IP es simplemente una comunicación Modbus encapsulado en una envoltura de Ethernet TCP / IP.

La completa / unidad de datos de aplicaciones de IP Modbus TCP se incrusta en el campo de datos de una trama TCP estándar y se envía a través de TCP al conocido puerto del sistema 502, que está reservado específicamente para aplicaciones Modbus, clientes Modbus TCP / IP y servidores de escuchar y recibir datos a través del puerto Modbus 502.

El Modbus/TCP se utiliza frecuentemente en:

- PLC (controladores industriales)
- Sistemas SCADA (visualizaciones y control básico de procesos industriales)
- Sensores y actuadores

1.3.4. Gestión. Normas ISO

[15]La norma ***UNE-EN ISO 10218-1:2011****("Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 1: Robots."),* que contempla el robot como una "cuasi máquina" y especifica los requisitos de seguridad para robots industriales.

La norma ***UNE-EN ISO 10218-2:2011*** *("Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 2: Sistemas robot e integración"),* que contempla los sistemas de robots como máquinas completas y especifica los requisitos para dichos sistemas de robots y las integraciones (combinación de un robot con otros equipamientos o máquinas).

1.3.5. Generador de trayectorias

El generador de trayectorias, calcula la trayectoria para cada movimiento, esta debe ser físicamente realizable y más o menos suaves, según la aplicación. El generador debe coordinar los ejes para un robot con más articulaciones. Por cada periodo de muestreo, debe calcular los valores de referencia de posición y velocidad.

1. Trayectoria Trapezoide

Esta trayectoria es la más utilizada por los robots. Es muy reconocible visualmente ya que respecto a la posición su progresión es ascendente realizando la forma de una rampa, respecto la velocidad se puede observar tres fases reconocibles: la fase de aceleración, la fase constante y la fase de desaceleración. En cuanto a la aceleración esta comienza en un su aceleración máxima, y va descendiendo progresivamente, pasando por una fase de aceleración máxima, una aceleración de 0 y una aceleración máxima en sentido contrario.

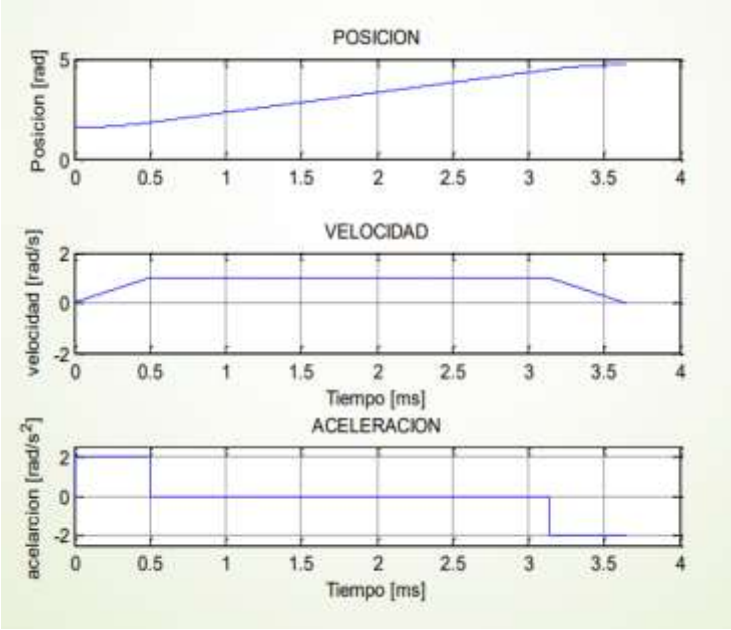


Fig4. Grafica trayectoria trapezoide

2. Perfil en S

Esta trayectoria en cuanto a posición y velocidad es igual que respecto a la trapezoidal, aunque se puede observar que los cambios entre las fases no son tan brusco como en la anterior. Respecto a la velocidad si varía, ya que se tienen 3 fases bien diferenciadas, la primera y última fase tiene una parte de aceleración y una parte de desaceleración, por lo que forman esa pirámide tan característica.

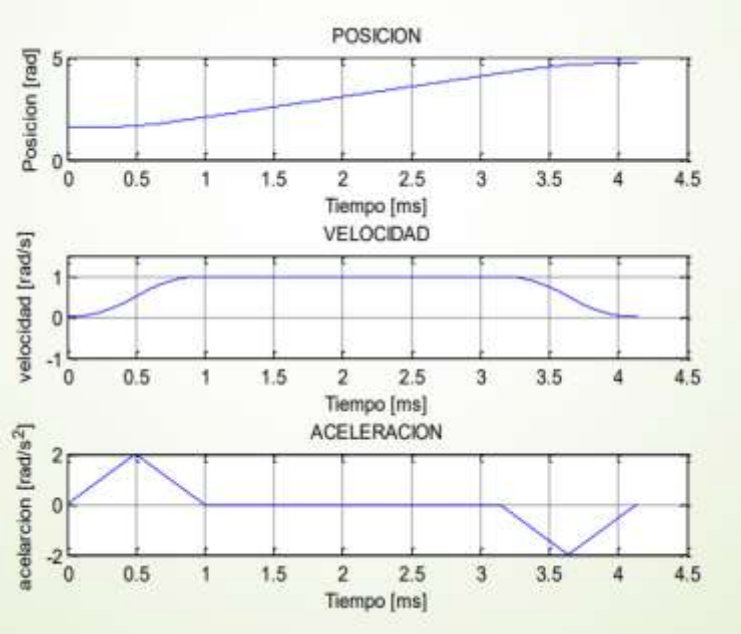


Fig5. Grafica trayectoria perfil en S

3. Perfil en S parcial

Esta trayectoria es similar al Perfil en S, la única variación es en cuanto a la aceleración, donde los cambios no son tan bruscos, ya que en las fases 1 y 3, ambas tienen su fase de aceleración, la fase constante y la fase de desaceleración, lo que hace que el robot no alcance valores máximos de manera tan rápida.

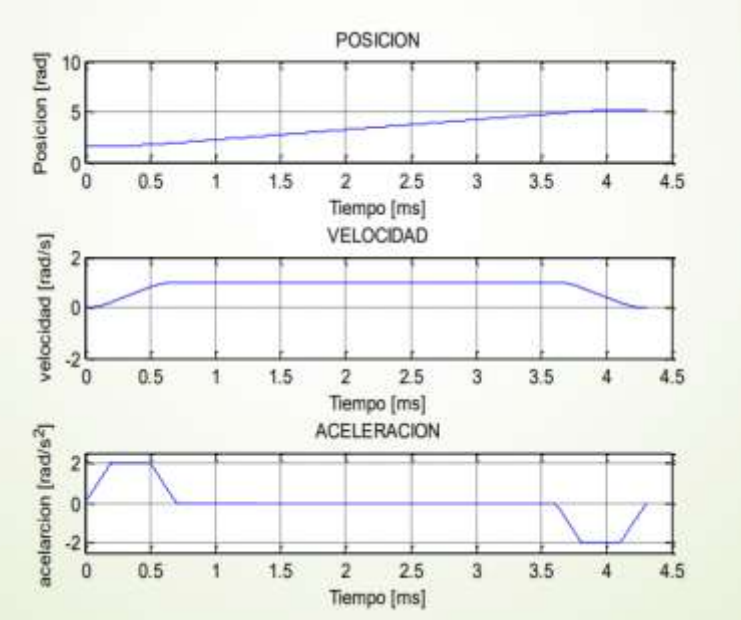


Fig6. Grafica trayectoria Perfil en S parcial

1.3.6. Tipos de trayectorias

[16]A continuación se cita brevemente una clasificación de tipos de trayectorias de robots comerciales clásicos.

- **Trayectorias continuas.** En este tipo de trayectorias se pretende que el camino seguido por el extremo del robot sea conocido. Para ello las trayectorias articulares deben acomodarse conjuntamente. Cada articulación por separado parece tener un movimiento desordenado, sin embargo el resultado es que el extremo se mueve siguiendo el camino previsto.

  [17]En este tipo de trayectorias se encunetran los movimientos MoveL y Move P:

  Con MoveL, se mueve el punto central de la herramienta linealmente entre puntos de referencia. Esto significa que cada articulación realiza un movimiento más complicado para mantener la herramienta en línea recta. Los parámetros compartidos que se pueden configurar para este tipo de movimientos son la velocidad de la herramienta deseada y la aceleración de la herramienta especificada en mm/s y mm/s^2 respectivamente.

  Con MoveP, se mueve la herramienta linealmente con velocidad constante con mezclas circulares. El tamaño del radio de mezcla es por defecto un valor compartido entre los puntos de paso. Un valor más pequeño hará que la ruta se vuelva más nítida, mientras que un valor más alto hará que la ruta sea más suave.

- **Trayectorias coordinadas o isocronas**. En este tipo de trayectorias se procura que el movimiento de todos los actuadores sea coordinado e isocrona. Esto quiere decir que el actuador que tarda más tiempo en alcanzar la posición requerida ralentiza al resto, de manera que ningún movimiento acaba antes que el de otra articulación. El tiempo total invertido en el movimiento es el menor posible, y los requerimientos de velocidad y aceleración de los motores son menores que en otro tipo de movimiento. El inconveniente de este tipo de planificadores es que la trayectoria que describe el extremo del robot es desconocida a priori.

  En este tipo se encuentra el movimiento de MoveJ, en él se realizan movimientos eje a eje, se calculan en el espacio articular del brazo del robot. Las articulaciones se controlan para terminar sus movimientos al mismo tiempo. Este tipo de movimiento da como resultado una trayectoria curva para la herramienta. Los parámetros compartidos que se aplican a este tipo de trayectoria son la velocidad máxima de la articulación y la aceleración de la articulación, especificadas en grados/s y grados/s^2 respectivamente.

- **Trayectorias punto a punto**. En este tipo de trayectorias cada articulación se mueve independientemente, sin considerar el efecto del resto de las articulaciones. Dentro de esta tipo se engloban las trayectorias con movimiento eje a eje y las de movimiento simultáneo de ejes. En las trayectorias con movimiento eje a eje en primer lugar se actúa

sobre un motor, y cuando este ha finalizado su recorrido, se activa el siguiente motor. Este tipo de movimiento tiene como única ventaja el ahorro energético.

1.3.7. Planificación

A continuación, se muestra de manera global el orden de las tareas realizadas y el tiempo que requieren cada una. Posteriormente se representa un diagrama de Gantt correspondiente a la planificación.

- Planteamiento inicial del proyecto: Que se va a realizar, como, cuando.
- Estado del arte: se realiza una búsqueda de información para analizar que se ha hecho hasta ahora respecto a movimientos coordinados entre dos robots.
- Estudio y planteamiento de alternativas: incluye la comparación y análisis de tres alternativas en cuanto a lenguaje y robot a utilizar.
- Alternativa adoptada.
- Estudio sobre la conectividad con el sensor externo de fuerza y par.
- Experimento 1: movimiento individual de los dos robots.
- Experimento 2: movimiento coordinado.
- Experimento 3: movimiento coordinado con objeto.
- Experimento 4: Prueba con otras funciones.
- Experimento 5: Movimiento utilizando conexión por Modbus.
- Experimento 6: Control por bucle interior-exterior.
- Generador de trayectorias
- Redacción del proyecto.
- Revisión final.

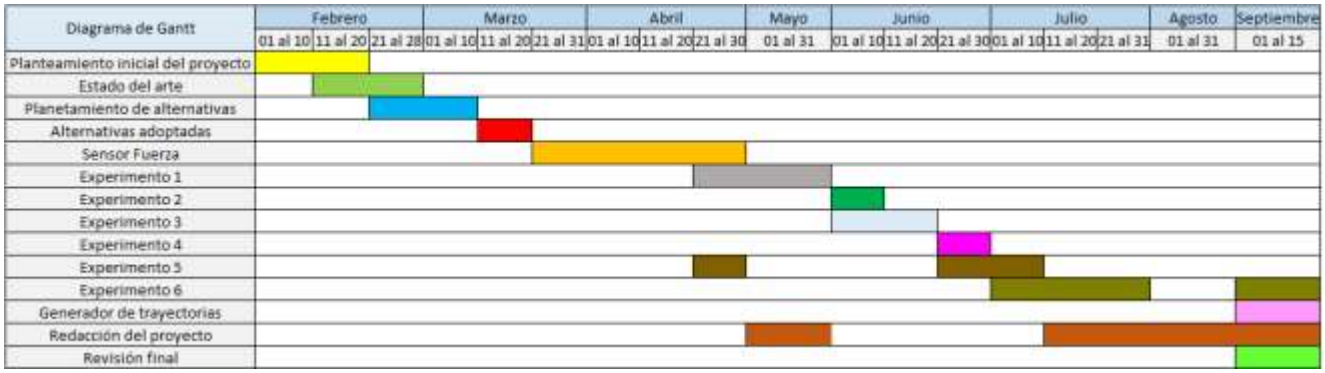


Tabla3. Diagrama de Gantt

1.4. Planteamiento de soluciones alternativas y justificación de la solución adoptada

Para la realización de un control de dos brazos robots de forma simultánea se plantean varias alternativas, la primera en cuanto al lenguaje de programación para la realización de los diferentes scripts, como también de los robots que se pueden llegar a utilizar.

- **Lenguajes de programación**

[18]Es el conjunto de instrucciones a través del cual los humanos interactúan con las computadoras. Un lenguaje de programación permite comunicar a las personas con las computadoras a través de algoritmos e instrucciones escritas en una sintaxis que la computadora entiende e interpreta en lenguaje de máquina.

- **Python**

Es un lenguaje de programación de código abierto, una sintaxis sencilla y fácil de entender, por lo que ahorra tiempo y recursos. Python es un lenguaje versátil que puede tener multiples aplicaciones.
[19]Encuentra usos en la creación de aplicaciones web, el análisis de datos y el desarrollo de algoritmos. Tiene una sintaxis fácil de usar y se centra en la legibilidad y la simplicidad del código convirtiéndolo en un candidato ideal para los desarrolladores de nivel de entrada, especialmente para aquellos que persiguen una carrera en inteligencia artificial, big data, robótica o ciberseguridad.

- **C++**

[20]**C**C++ es un lenguaje de programación compilado, multiparadigma, principamente de tipo imperativo y orientado a objetos, incluyendo también programación genérica y funcional.
Es un lenguaje ampliamente utilizado en bases de datos, compiladores, navegadores web y videojuegos.
[21]Las principales ventajas de utilizar este lenguaje es el alto rendimiento, el lenguaje actualizado, es multiplataforma y está bastante extendido. Como desventaja es que se trata de un lenguaje muy amplio, tiene que tener una compilación por plataforma y su depuración se complica debido a los errores

- **JAVA**

[22]Java es un lenguaje multiplataforma, orientado a objetos y centrado en la red que se puede utilizar como una plataforma en sí mismo. Es un lenguaje de programación rápido, seguro y fiable para codificar todo, desde aplicaciones móviles y software empresarial hasta aplicaciones de macrodatos y tecnologías del lado del servidor.
Las principales ventajas que presenta son su simplicidad, está orientado a objetos, es distribuido e independiente de la plataforma y que es seguro y multihilo.

- **Robots UR**

  - **UR3/UR3-e**

El UR3e de Universal Robots es un robot industrial colaborativo ultraligero y compacto, ideal para la aplicación sobre mesas de trabajo. Su tamaño reducido lo convierte en el más adecuado para implementarse directamente dentro de maquinaria o en otros espacios de trabajo pequeños. Entre sus principales características destaca que puede rotar en todos sus ejes 360 grados y es muy interesante en procesos de atornillado y montajes de componentes.

Alcance	500 mm
Carga útil	3 Kg
Espacio requerido	128 mm
Peso	11.2 kg
Repetibilidad	+/- 0.03
Velocidad máxima	1000 mm/s

Tabla4. Especificaciones robot UR3/UR3e

  - **UR5**

El UR5e de Universal Robots es un robot colaborativo industrial ligero construido para aplicaciones de servicio medio (hasta 5 kg). Los propósitos generales del desarrollo de este robot son la versatilidad y la adaptabilidad. El UR5e está diseñado para integrarse perfectamente en una amplia gama de aplicaciones. El flexible robot UR5e es perfecto para la optimización de procesos colaborativos de poco peso, como pick & place y pruebas de producto.

Alcance	850 mm
Carga útil	5 Kg
Espacio requerido	149 mm
Peso	20.6 kg
Repetibilidad	+/- 0.03
Velocidad máxima	1000 mm/s

Tabla5. Especificaciones robot UR5

  - **UR10**

El UR10e es un robot industrial colaborativo extraordinariamente versátil que ofrece una gran carga útil (12,5 kg) de elevación y un largo alcance (1300 mm), por lo que es ideal para una amplia gama de aplicaciones, como el mantenimiento de máquinas, el paletizado y el embalaje.

Alcance	1300 mm
Carga útil	12.5 Kg
Espacio requerido	190 mm
Peso	33.5 kg
Repetibilidad	+/- 0.05
Velocidad máxima	1000 mm/s

Tabla6. Especificaciones robot UR10

- **UR16**

El UR16e de Universal Robots ofrece una impresionante capacidad de carga útil de 16 kg en un espacio reducido. Por ello, es ideal para el uso en máquinas pesadas, manipulación de materiales, packaging, embalaje y aplicaciones de atornillado y tuercas. Este potente robot admite herramientas de final de brazo pesadas y múltiples manejos, y es especialmente útil para lograr tiempos de ciclo más cortos.

Alcance	900 mm
Carga útil	16 Kg
Espacio requerido	190 mm
Peso	33.1 kg
Repetibilidad	+/- 0.05
Velocidad máxima	1000 mm/s

Tabla7. Especificaciones robot UR16

- **UR20**

EL UR20 se ha replanteado, rediseñado y reconstruido desde cero. Cada detalle, desde el software hasta todo el hardware, ha sido diseñado estratégicamente para ofrecer un rendimiento y una calidad de última generación para que pueda aumentar la producción y el tiempo de actividad y sacar más productos al mercado más rápidamente que nunca.

El UR20 está diseñado para mayores cargas útiles, velocidades más elevadas y un control de movimientos superior, y todo dentro de un sistema ligero y de tamaño reducido, para una versatilidad óptima en su espacio de producción existente.

Alcance	1750 mm
Carga útil	20 Kg
Espacio requerido	245 mm
Peso	64 kg
Repetibilidad	+/- 0.05
Velocidad máxima	1000 mm/s

Tabla8. Especificaciones robot UR20

1.5. Descripción detallada de la solución adoptada

- **Lenguajes de programación**

Entre los lenguajes de programación expuestos en el anterior punto, se ha escogido Python como la mejor opción para la realización de los diferentes scripts para realizar la conexión entre PC y robot.
Este lenguaje de programación es fácil de leer, y también ayuda a desarrollar un estilo de programación limpio sin necesidad de ser muy estricto con la sintaxis. Se puede utilizar para Robótica en los campos como Machine Learning, donde los algoritmos aprenden de los datos sin que nadie codifique explícitamente ninguna regla, y Big Data ya que Python es conocido por su capacidad para manejar grandes conjuntos de datos.
A demás se ha elegido este lenguaje ya que se ha afrontado como una oportunidad para aprender un nuevo lenguaje de programación hasta ahora visto, y poder realizar un código propio para el proyecto.

- **Robot UR**

En cuanto al robot UR utilizado para realizar un movimiento coordinado va a depender de la zona de trabajo que se tenga disponible y para el caso de pick and place dependerá del peso de la pieza que se quiera utilizar.
En este caso la zona de trabajo está limitada a una mesa de trabajo de laboratorio y se quiere mover mediante pick and place una caja de cartón, por lo que con el robot más pequeño y con la menor carga útil serviría. Por lo que el robot finalmente escogido es el **robot UR3/UR3e.**

Puede montarse encima de una mesa creando y optimizando estaciones de trabajo independientes. Su acabado compacto y su fácil sistema de programación, característico de los robots Universal Robots, permite que sea muy sencillo de configurar para realizar diferentes tareas en un entorno de fabricación dinámico, donde se necesiten soluciones flexibles y cambiantes, minimizando los costes de puesta en marcha y ofreciendo el retorno de inversión más rápido de la industria.
El brazo robótico está provisto con 6 ejes y es por esto capaz de colocarse en casi cualquier posición.
Además cada UR-3 incorpora una pantalla tactil de 12" que es el panel de control para la configuración y le permite opcionalmente instalar paquetes de software adicionales. El ambiente gráfico polyscope le permite integrar sensores o pinzas adicionales en poco tiempo.

Fig7. Robot UR3e de perfil

1.6. Justificación detallada de los elementos o componentes de la solución adoptada.

1.6.1. Sensor On robot

Fig8. Sensor On robot

[23]El robot UR3 lleva un sensor de fuerza/par **HEX 6-Axis Force/TorqueNs** conectado externamente. Este proporciona mediciones precisas de fuerza a lo largo de los 6 ejes. Esto te proporciona un control preciso cuando se trata de tareas difíciles de montaje, pulido, lijado o desbardado. El software HEX incluye registro de trayectoria, control de fuerza y características especiales para tareas de inserción. El sensor asegura que se mantenga una fuerza y velocidad constantes.

Fig9. Sensor On robot instalado en el robot UR3

Fuerzas/Par nominal	Fxy: 200 N Fz:200N Txy:10 Nm Tz:6.5 Nm
Peso	0.347 kg
Dimensiones	50 x 71 x 93 mm
Voltaje	7 – 24 V

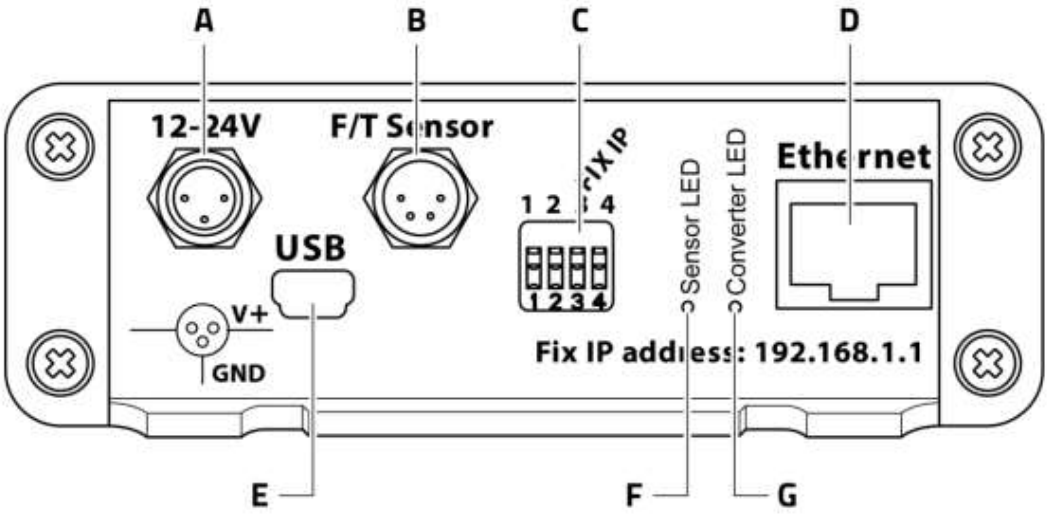
Tabla9. Especificaciones sensor On robot

1.6.2. Compute Box

El compute box está diseñado para trabajar con el sensor de OnRobot de 6 ejes que mide las fuerzas y los pares. Es usado para leer y configurar el sensor mediante una interfaz Ethernet. La compute box proprociona los datos recibidos del sensor a cualquier dispositivo a través de la interfaz de Ethernet. Se proporciona un cable para conectar la compute Box a una PC u ordenador portátil.

Fig10. Compute Box

En la siguiente imagen se puede observar el compute box y las diferentes conexiones para conectarlo al sensor de OnRobot.



[24]**Fig11.** Conexiones Compute Box

A. Conector de alimentación
B. Conector del sensor Fuerza / Par
C. DIP Conmutador
D. Cable Ethernet
E. Conector USB
F. Indicador del estado del sensor
G. Indicador del estado del convertidor

1.6.3. Polyscope

PolyScope o la interfaz de usuario de robot es la pantalla táctil en su panel consola portátil. Es la interfaz gráfica de usuario (IGU) que maneja el brazo robótico y la caja de control, ejecuta y crea programas del robot. PolyScope se compone de tres zonas:

A: **Encabezado** con pestañas/iconos que ponen a su disposición pantallas interactivas.
B: **Pie de página** con botones que controlan su o sus programas cargados.
C: **Pantalla** con campos que gestionan y supervisan las acciones del robot.

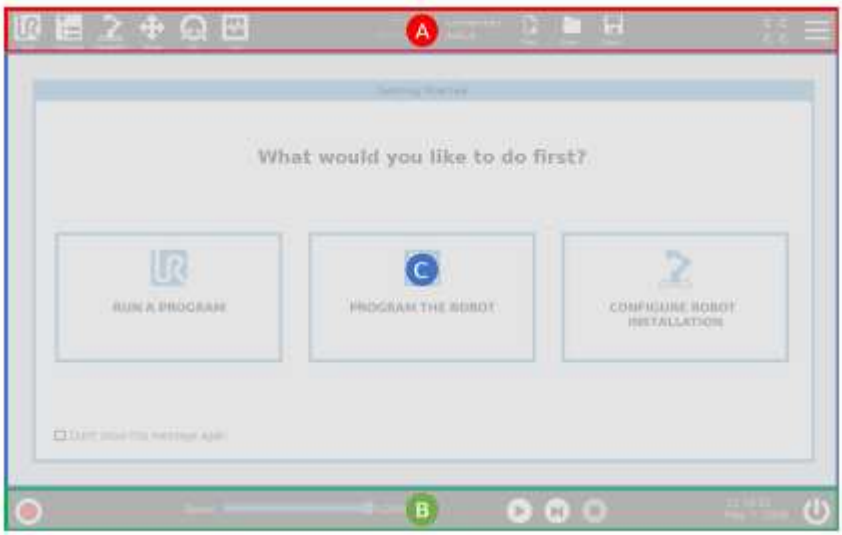


Fig12. Pantalla principal Polyscope

La interfaz de usuario puede variar según la versión instalada en el robot. En el laboratorio se ha tenido un UR3 con la versión 3.14 hasta tener un UR3e con la versión 5.12.

Para encender el robot se debe pulsar el botón de encendido localizado en la parte exterior de la pantalla, junto a la seta de emergencia.

Una vez pulsado el botón aparecerá la pantalla inicial que indicara la versión actual del robot, en este caso la 3.14.

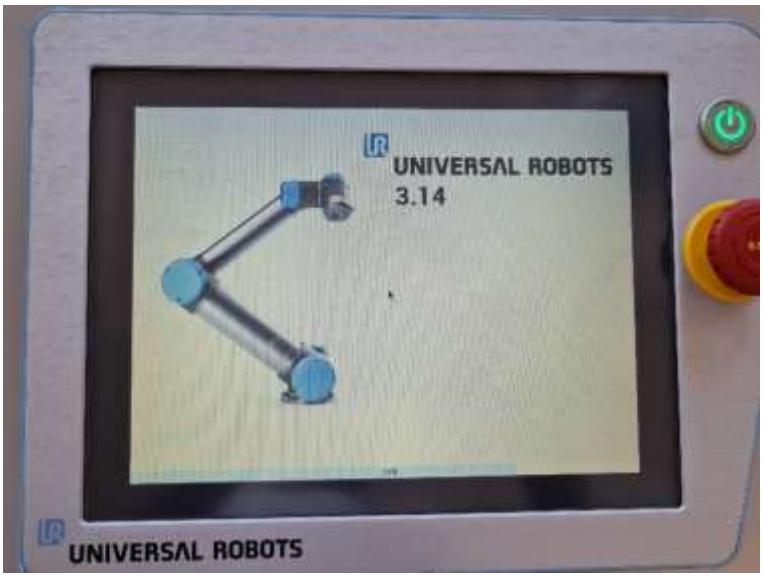


Fig13. Pantalla inicio UR3

Una vez pasada la pantalla de inicio se abrirá una nueva pantalla donde saldrá una ventana emergente indicando si se quiere inicializar el robot. Lo más conveniente es indicar en la ventana el botón de **Ahora no**, para primero realizar la programación del robot y posteriormente inicializarlo, ya que pueden producirse paradas de emergencia por protección del robot al cargar el programa y su correspondiente nueva puesta en marcha.

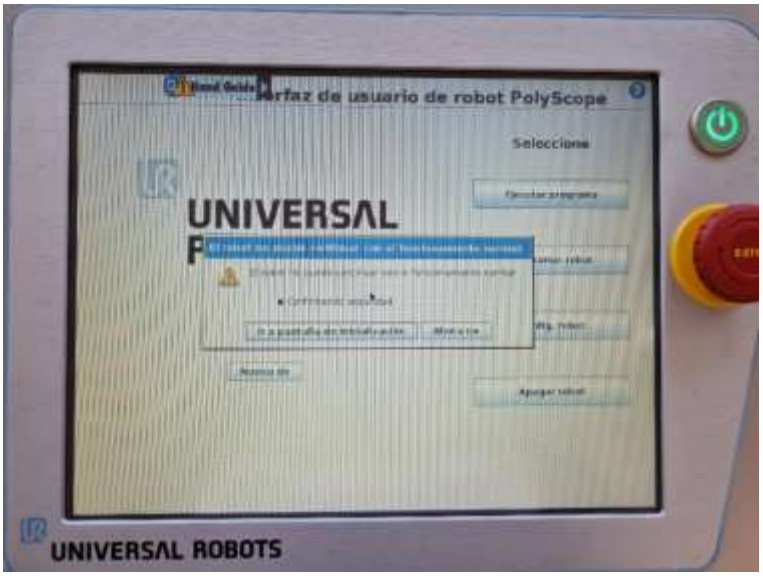


Fig14. Pestaña inicialización del robot

La inicialización del robot si varía más según la versión que se tenga. Por lo que se va a exponer tanto la versión 3.14 como la 5.3.

En la versión 3.14 se pulsa el botón de **Encender**, posteriormente aparecerá un nuevo botón de **Iniciar** sustituyendo el botón de Encender. Una vez pulsado este botón, indicara encima del botón el estado del robot, si está correctamente indicara con un luz verde y al lado aparecerá el estado de Normal, posteriormente para volver a la pantalla de programación se deberá pulsar el botón **OK** situado en la parte posterior derecha.

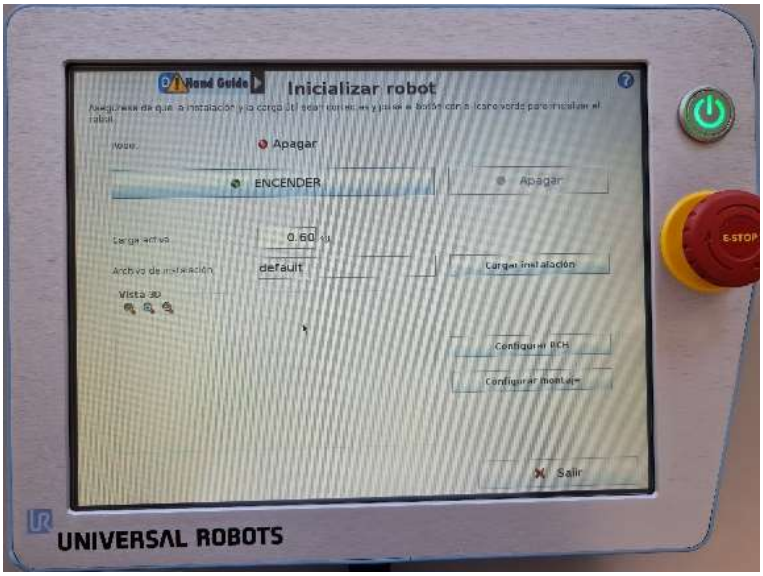


Fig15. Primera pestaña de inicialización

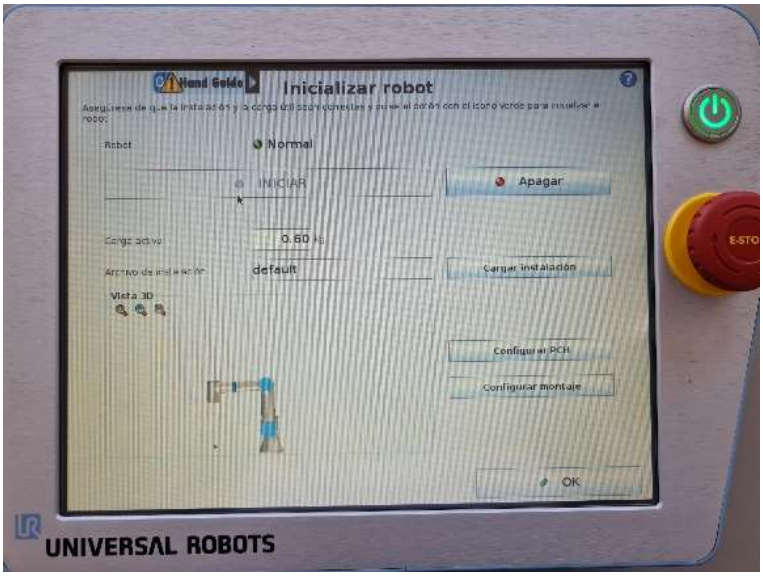


Fig16. Segunda pestaña de inicialización

En cambio en la versión 5.12 del UR3e varía, observando 3 tipos de leds que indican el estado del robot.

Rojo indica que el brazo robótico se encuentra en estado de parada, por diferentes razones posibles.
Amarillo indica que el brazo robótico está encendido, pero no está listo para funcionar con normalidad.
Verde indica que el brazo robótico está encendido y listo para funcionar con normalidad.

Cuando se arranca el PC del controlador, pulse el botón ENCENDIDO una vez para encender el brazo robótico. El estado del robot pasa a amarillo para indicar que el suministro eléctrico está activado y en **Inactivo**.

Cuando el estado del robot es **Inactivo**, pulse el botón INICIAR para iniciar el brazo robótico. En este momento, los datos del sensor se comparan con el montaje configurado del brazo robótico. Si no coinciden (con una tolerancia de 30◦), se deshabilita el botón y aparece un mensaje de error bajo él. Si se comprueba el montaje, pulsar el botón libera todos los frenos de junta (la liberación de frenos se acompaña por clics y un ligero movimiento) y el brazo robótico está listo para un funcionamiento normal. Si el brazo robótico supera uno de los límites de seguridad después de ponerse en marcha, funcionará en un modo de Recuperación. En este modo, al tocar el botón se abrirá una ventana de modo de recuperación en la que podrá devolver el brazo robótico dentro de los límites de seguridad.

La pantalla prinipal de programación estara compuesta:

- **Programa**, crea o modifica los programas de robot.
- **Instalación**, configura los ajustes del brazo robótico y el equipo externo,
- **Mover**, controla o regula el movimiento del robot.
- **E/S**, supervisa y ajusta las señales de Entrada/Salida hacia y desde la caja de control del robot.
- **Registro**, indica la salud del robot así como cualquier mensaje de advertencia error.

1.6.4. Parámetros Denavit-Hatenberg

UR3e							
Cinemática	theta [rad]	soy]	re [m]	alfa [rad]	**Dinámica**	Masa [kg]	Centro de masa [m]
Articulación 1	0	0	0.15185	π/2	Enlace 1	1.98	[0, -0.02, 0]
Articulación 2	0	-0.24355	0	0	Enlace 2	3.4445	[0,13, 0, 0,1157]
Articulación 3	0	-0.2132	0	0	Enlace 3	1.437	[0,05, 0, 0,0238]
Articulación 4	0	0	0.13105	π/2	Enlace 4	0.871	[0, 0, 0,01]
Conjunto 5	0	0	0.08535	-π/2	Enlace 5	0.805	[0, 0, 0,01]
Articulación 6	0	0	0.0921	0	Enlace 6	0.261	[0, 0, -0,02]

Tabla10. Parametros Denavit-Hatenberg robot UR3

1.6.5. Limitaciones de velocidad y posición

Juntas	Máximo	Modo normal	Modo reducido	
Base	máx.: 191 °/s	**191**	**191**	-11 °/s
Hombro	máx.: 191 °/s	**191**	**191**	-11 °/s
Codo	máx.: 191 °/s	**191**	**191**	-11 °/s
Muñeca 1	máx.: 191 °/s	**191**	**191**	-11 °/s
Muñeca 2	máx.: 191 °/s	**191**	**191**	-11 °/s
Muñeca 3	máx.: 191 °/s	**191**	**191**	-11 °/s

Fig17. Límites de velocidad de cada articulación del robot UR3

Position range

Joints	Range	Normal Mode		Reduced Mode		
		Minimum	Maximum	Minimum	Maximum	
Base	-363 – 363 °	-363	363	-363	363	-2 °/-2 °
Shoulder	-363 – 363 °	-363	363	-363	363	+2 °/-2 °
Elbow	-363 – 363 °	-363	363	-363	363	+2 °/-2 °
Wrist 1	-363 – 363 °	-363	363	-363	363	-2 °/-2 °
Wrist 2	-363 – 363 °	-363	363	-363	363	+2 °/-2 °
Wrist 3	Unlimited	-Unlimited	+Unlimited	-Unlimited	+Unlimited	+2 °/-2 °

Fig18. Límites de posición de cada articulación del robot UR3

1.6.6. Limitaciones de movimiento

El robot puede no alcanzar la posición final indicada, esto puede deberse a diferentes motivos:

- **Singularidad**

Si un movimiento objetivo requiere menos de los GDL del robot, se origina redundancia cinemática robot-tarea. Los GDL que exceden a los necesarios para la realización de la tarea, es decir una tarea puede realizarse de diversas maneras, aparece el concepto de singularidad.

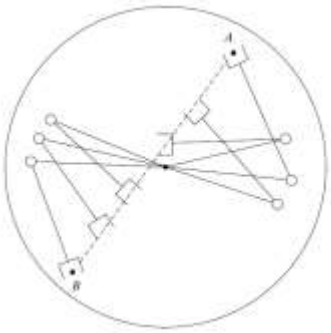


Fig19. Ejemplo de Singularidad

- **Articulación sale de rango**

También si la posición de la tarea a realizar no puede ser alcanzada por el robot, ya que este se encuentra fuera de rango de movimiento, este llegará hasta su ángulo máximo en la articulación y aparecerá un error de posición.

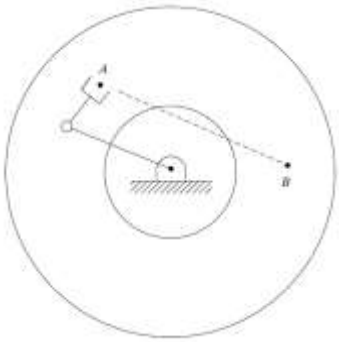


Fig20. Ejemplo de una articulación fuera de rango

- **Punto inicial y final alcanzables con diferentes configuraciones**

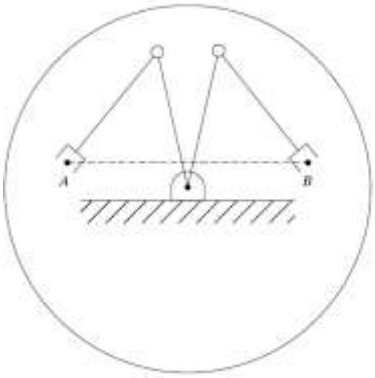


Fig21. Ejemplo de puntos alcanzables con diferente configuración

1.7. Experimentos

- **Entorno del laboratorio**

En la imagen se puede observar el robot UR3 a la parte izquierda con el sensor de Fuerza y Par, este robot esta sobre una base fija. A la derecha se puede observar el robot UR3e, este lleva incorporados los sensores de fuerza internamente y su base es portátil.

El UR3 tiene un tiempo de muestreo de 8ms mientras que el UR3e, el robot de la derecha, tiene un tiempo de muestreo de 2 ms.

Fig22. Espació de trabajo con UR3 y UR3e

En los últimos experimentos se han utilizado dos robots UR3e, ambos con un tiempo de respuesta de 2ms.

Fig23. Espació de trabajo con dos UR3e

- **Cambio del sistema de coordenadas**

Durante este trabajo se va a realizar experimentos basados en la sincronización maestro esclavo, donde el robot de la izquierda, será el robot Maestro y el robot de la derecha el Esclavo. El esclavo deberá conocer en qué posición está el maestro para así hacer movimientos sincronizados entre los dos.

En cada robot, su sistema de coordenadas está indicado respecto de la base del mismo, por lo que se debe tener o un punto común que sea el sistema de coordenadas de ambos o bien realizar una transformación de uno de los sistemas de coordenadas de un robot.

- **Nuevo sistema de coordenadas**

Primero, se define un sistema de coordenadas nuevo. En el menú de polyscope, en la pestaña de instalación y en el apartado Funciones se define el sistema de coordenadas o bien mediante un Punto o mediante un Plano.

Mediante un punto

Seleccionando el botón de Punto, y se observa cómo se crea en el apartado Funciones un nuevo punto llamado por defecto Punto_1, se selecciona en la pantalla polyscope. Al pulsar, se cambiara a la pantalla del Punto_1 donde se podrá cambiar el nombre de dicho punto con el botón de Renombrar, eliminarlo mediante el botón de Eliminar o fijar la posición del punto mediante el botón de Enseñar este punto.

Al fijar la posición del punto, se pulsa el botón de Enseñar este punto y cambiara a la pestaña de Mover, donde se podrá fijar la posición de este nuevo punto de coordenadas moviendo el robot hasta la posición deseada del sistema. Para mover el robot hasta el punto de coordenadas deseado se utilizan los botones de posición y orientación de PCH donde se puede editar la X, Y, Z, RX, RY y RZ. Una vez el robot este posicionado donde se desea colocar el nuevo sistema de coordenadas, se debe pulsar al botón de OK.

Mediante un plano

Seleccionando el botón de Plano y se abrirá la pestaña de Plane, donde se indican los pasos a seguir para crear el nuevo Plano. Se pulsa el botón de siguiente donde primero se define el origen del plano pulsando el botón de establecer posición, una vez fijada se pulsa OK.
En la siguiente imagen se puede observar los pasos que hay que seguir para realizar correctamente el plano.

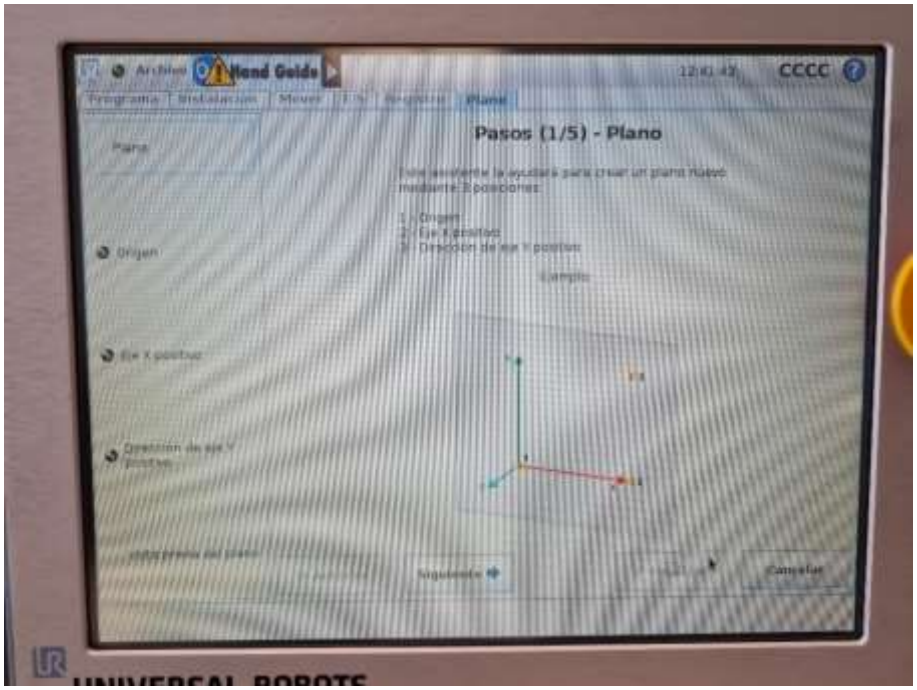


Fig24. Configuración de un plano de referencia

El segundo paso el fijar el Eje X. Se establece el eje positivo, pulsando en establecer la posición y colocar el robot en una nueva posición. Esta nueva posición y la del origen definirán la trayectoria del eje X.

El tercer paso es definir el eje Y positivo, para ello se pulsa de nuevo en establecer posición y mover así el robot hacia una nueva posición. Esta nueva posición, con la posición del origen define el eje Y positivo. Para finalizar la configuración se pulsa sobre el botón siguiente y mostrara un esquema de la posición actual del robot y la posición del plano, a continuación se pulsa el botón de Finalizar y ya se tendría el plano definido como Plano_1. Para poder cambiar el nombre al plano, en el apartado de Funciones, a Plano_1 y pulsando sobre Renombrar se puede cambiar el nombre al plano. Se define el plano como Plano_ref.

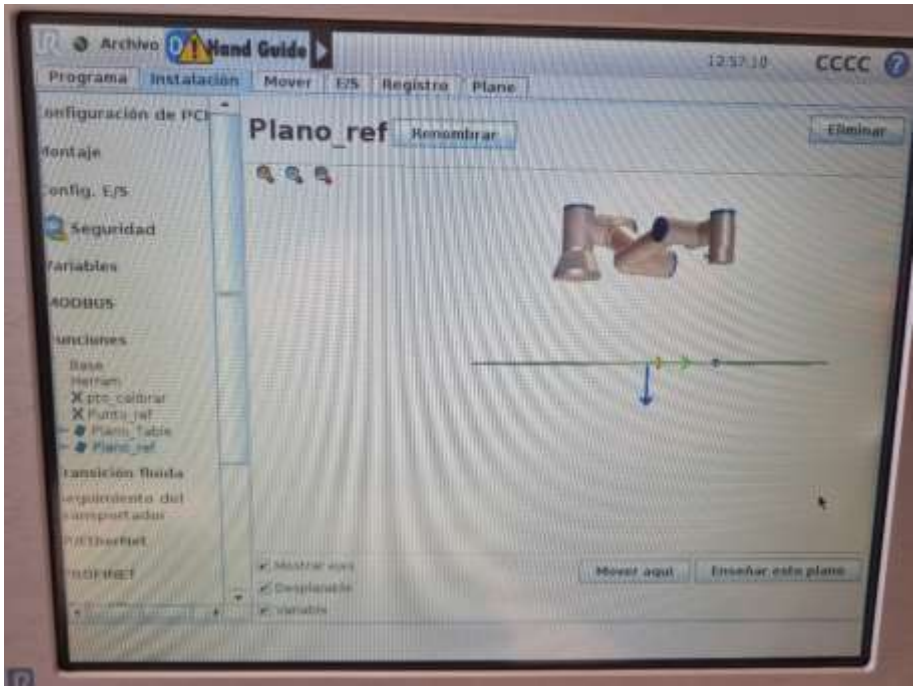


Fig25. Pantalla con un plano definido

Una vez definido el sistema de referencia nuevo mediante un Punto o mediante un Plano, se puede mover el robot hasta un punto según el nuevo punto de referencia.

Para saber las coordenadas del nuevo sistema de referencia respecto la Base, primero se mueve el robot hasta donde está el sistema de referencias, para ello se selecciona la pestaña de Instalación->Funciones y posteriormente se elige el punto o plano de referencia y se selecciona el botón de Mover aquí, así el robot se posicionara en el centro del sistema de referencia. A continuación se abre la pestaña de Mover y se selecciona en el apartado Función la opción de Base, así se puede obtener la posición del nuevo sistema de referencia en función del sistema de referencia de la base del robot.

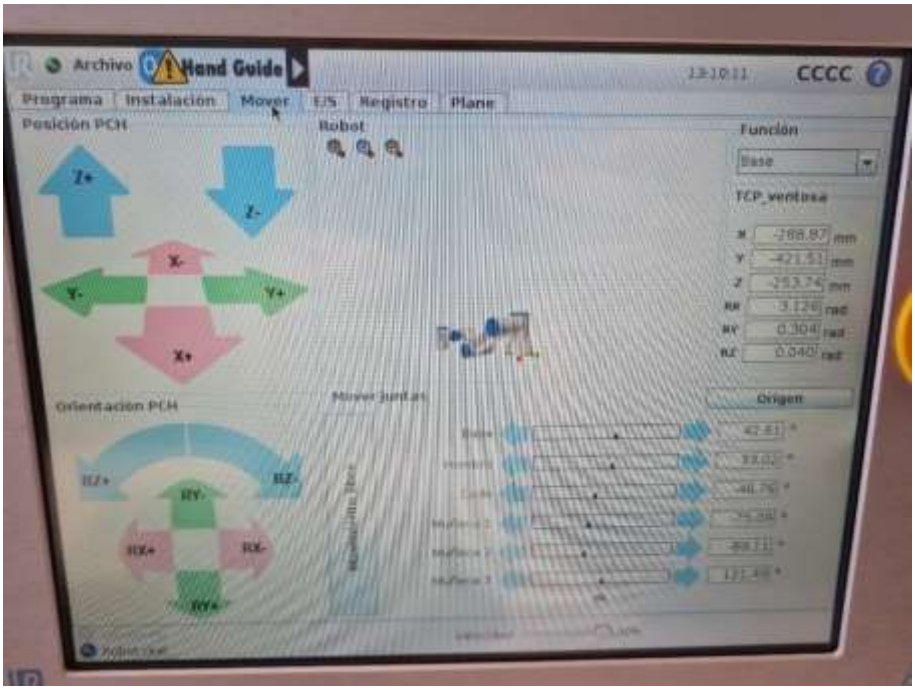


Fig26. Pantalla de Mover

- **Transformación sistema de coordenadas**

Para realizar un cambio de sistema de coordenadas se debe conocer las matrices de transformación, rotación, perspectiva y escalado.

En este trabajo no es necesario realizar ningún cambio de perspectiva, por lo que serán ceros y no es necesario un escalado, por lo que será factor 1.

$$T = \begin{bmatrix} R_{3\times3} & p_{3\times1} \\ f_{1\times3} & w_{1\times1} \end{bmatrix} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ \text{Perspectiva} & \text{Escalado} \end{bmatrix}$$

Fig27. Ecuación transformada

La matriz de Rotación pueden ser respecto el eje x, respecto el eje y o respecto el eje z. Estas matrices se representan de la siguiente manera:

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

[25]**Fig28.** Matriz de Rotación en x,y,z

A demás de conocer los tipos de transformaciones que existen, se debe conocer el tipo de sistema de coordenadas que se tiene, este puede ser o dextrógiro, o bien levógiro.

Para el espacio de trabajo del UR3 y URe se ha realizado el mismo cambio de coordenadas al realizando con los dos UR3e, ya que presentan los mismos sistemas de coordenadas respecto de la base del robot.

Se realiza una transformación mediante la matriz de rotación y transformación. Se debe conocer la distancia entre los dos sistemas de coordenadas, en X, Y y Z y además, como están colocadas las direcciones de los ejes de ambos sistema de coordenadas, para saber que giros de rotación hay que realizar.

Observando las coordenadas de ambos robots, el giro que se tendría que realizar para cambiar del sistema de coordenadas del maestro al esclavo seria realizando un giro de -90 grados, y del esclavo al maestro de 90 grados, siguiendo la regla de la mano derecha, considerando que el eje Z es positivo hacia arriba.

Pero para realizar un cambio de coordenadas basándonos en un punto conocido, varía. En el caso del cambio del maestro al esclavo conociendo el punto común en coordenadas del maestro, para hallar respecto el del esclavo se tendría que realizar:

$$P_1^S = rot(z, -90)^{-1} \cdot P_1^0 = rot(z, 90) \cdot P_1^0$$

Para el caso del esclavo al maestro sería similar, pero en sentido contrario, por lo que finalmente su giro tendría que ser el de -90 grados.

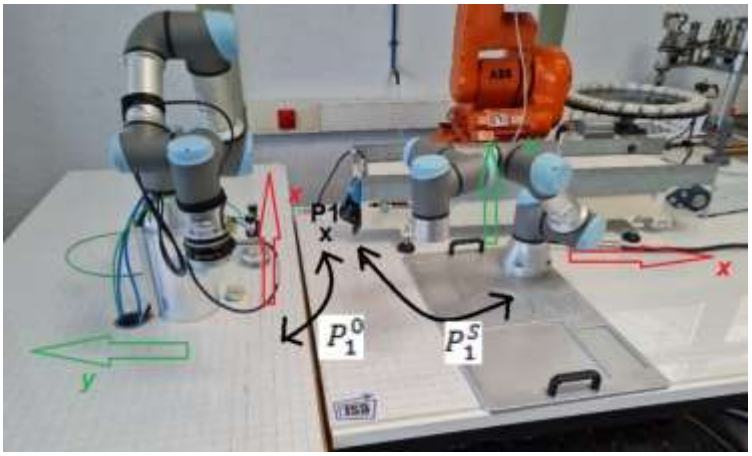


Fig29. Espacio de trabajo con representación de ejes

Con los dos UR3e, es similar al caso anterior, ya que se tiene el mismo sistema de coordenadas:

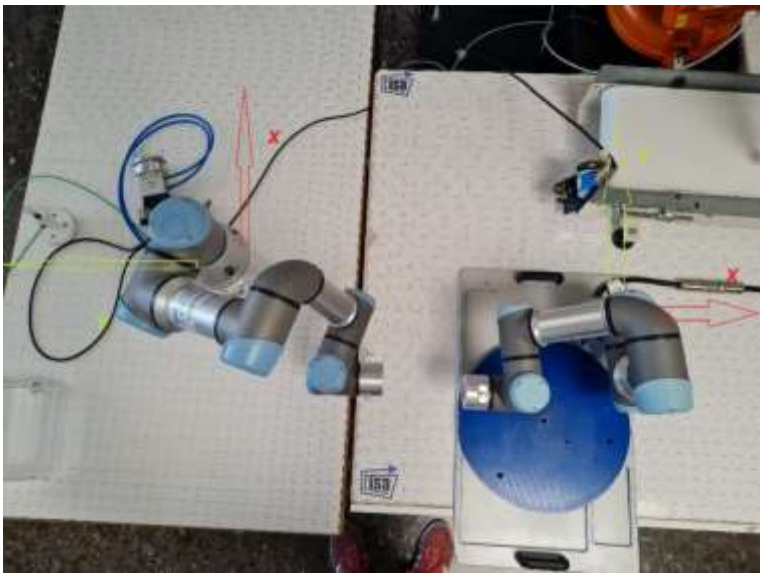


Fig30. Espacio de trabajo de los dos robots UR3e con indicación de ejes

Las bases de ambos robots están separadas unos 94 mm en X y 800 mm en Y, respecto el sistema de coordenadas de maestro. También hay una diferencia en Z ya que el maestro está colocado sobre una base de 200 mm de altura.

El sistema de coordenadas de ambos robots respecto de la base se puede observar en el apartado Move de la pantalla del robot.

Fig31. Pantalla Move UR3e

1.7.1. Lectura de datos del Sensor

Uno de los experimentos realizados en el laboratorio fue la conexión entre el sensor de fuerza externo del UR3 y el PC para poder leer los datos del sensor sin necesidad de utilizar el Polyscope del robot.

La interfaz de Ethernet admite tres modos de funcionamiento:

1. La web del cliente: fácil lectura de datos del sensor en tiempo real
2. TCP: lectura de datos del sensor de alta velocidad (hasta 500 Hz)
3. UDP: lectura de datos de sensor individual o iterada

Mediante la web se puede leer los datos del sensor, para acceder a ella se debe abrir un explorador de internet e introducir la dirección IP asociada al compute box.

Fig32. Página web del sensor

Los valor de fuerza y par (Fx,Fy,Fz y Tx,Ty,Tz) se muestran en Newton y Nm.
El interruptor ZERO se puede usar para poner a cero la lectura de fuerza y par.

- **Configuración para recibir datos del sensor mediante conexión UDP**

La conexión del Protocolo de datagramas de usuario (UDP) se puede utilizar para leer la salida del sensor en una frecuencia máxima de 500 Hz (2ms). En UDP también se puede utilizar para configurar la lectura, la frecuencia de corte y polarizar la salida del sensor. El protocolo UDP tiene cinco comandos. Para que el dispositivo inicie la salida de mensajes UDP, enviar una solicitud la dirección IP del dispositivo. El dispositivo escucha solicitudes UDP en el puerto 49152. Este puerto también se utiliza para los mensajes de salida

Se implementan los siguientes cinco comandos:

Command	Name	Data	Response
0x0000	Stop sending the output	Any value	none
0x0002	Start sending the output	Sample count	UDP record(s)
0x0042	Set software bias	0 or 255 decimal	none
0x0081	Set internal filtering	0-6 decimal	none
0x0082	Set read-out speed	Period in ms	none

Tabla11. Lista de comandos para la comunicación con el sensor

El único comando con respuesta es 0x0002, que inicia el envío de la salida. Los otros comandos no se reconocen, por lo tanto no tienen respuesta.

- **Petición**

Los comandos deben enviarse al dispositivo como una solicitud con la siguiente estructura:

```
UINT16    Header;        // Must be 0x1234
UINT16    Command;       // Value according to the command table
UINT32    Data;          // data according to the actual command
```

Fig33. Estructura comandos UDP

El recuento de bytes de la solicitud debe ser de 9 bytes y los valores de varios bytes deben enviarse como el primer byte alto.

- **Respuesta**

El dispositivo envía la salida como un registro UDP que tiene la siguiente estructura:

```
UINT32    HS_sequence;    // The sequence number of the current UDP record
UINT32    FT_sequence;    // The internal sample counter of the Compute Box
UINT32    Status;         // Status word of the sensor and Compute Box
UINT32    Fx;             // X-axis force in 32 bit Counts*
UINT32    Fy;             // Y-axis force in 32 bit Counts*
UINT32    Fz;             // Z-axis force in 32 bit Counts*

UINT32    Tx;            // X-axis torque in 32 bit Counts* (0 if not available)
UINT32    Ty;            // Y-axis torque in 32 bit Counts* (0 if not available)
UINT32    Tz;            // Z-axis torque in 32 bit Counts* (0 if not available)
```

Fig34. Estructura datos de salida UDP

El recuento de bytes de la salida es siempre de 36 bytes. Si se reciben menos de 36 bytes, se ignoran.

HS_sequence muestra el número actual de la salida. Si la solicitud de inicio fue enviada con datos (recuento de muestras) =1000, entonces HS_sequence comenzara desde 1 y finalizara con 1000. Si los datos eran 0, entonces la salida se produce hasta que se detiene la solicitud que se ha enviado.

Los valores de Fx, Fy, Fz, Tx, Ty,Tz se puede convertir a Newton/Newton-metro dividiendo los valores de fuerza por 10000 y los valores de torque por 100000.

La velocidad de lectura es la velocidad a la que hay nuevas muestras disponibles. Este valor se puede establecer en el rango de 254 ms a 2ms, que son 4Hz a 500Hz respectivamente.

- **Configuración para recibir datos del sensor mediante conexión TCP**

El modo Protocolo de Control de Transmisión (TCP) se utiliza para leer la salida y la información del estado del sensor. Las conexiones TCP son generalmente más lentas en comparación con las conexiones UDP, y varios son los factores de software y hardware que pueden afectar a la velocidad de respuesta como el firewall de software, el enrutador, etc…

Para una velocidad de lectura más rápida, se recomienda el uso de UDP. En el protocolo TCP, el dispositivo es el servidor y los clientes pueden conectarse a él.

La conexión se establece de la siguiente manera:

- El dispositivo escucha la conexión en el puerto TCP 49151.
- Una vez que un cliente ha establecido con éxito la conexión con el dispositivo, el cliente puede solicitar datos al dispositivo.
- Después de recibir la solicitud, el dispositivo responde con la respuesta adecuada.
- Después de que el usuario haya recibido la respuesta, se puede enviar una nueva solicitud sin reestablecer la conexión TCP. Si el dispositivo no recibe una solicitud durante más de 1 segundo, el dispositivo cierra la conexión (tiempo de espera). En este caso, el usuario necesita reestablecer la conexión TCP para poder solicitar más datos.

Solo una conexión TCP puede estar activa en cualquier momento.

Se debe enviar un comando simple al dispositivo mediante una solicitud que tiene la siguiente estructura.

```
UINT8     Command;          // Must be decimal 0 (0x00)
UINT8     Reserved[19];     // All the 19 value should be 0s.
```

Fig35. Estructura de comandos TCP

El recuento de bytes de la solicitud debe de ser de 20 bytes.

El dispositivo envía la salida como un registro TCP que tiene la siguiente estructura:

```
UINT16    Header;      // Fixed 0x1234
UINT16    Status;      // Status word of the sensor and Compute Box
 INT16    Fx;          // X-axis force in 16bit Counts*
 INT16    Fy;          // Y-axis force in 16bit Counts*
 INT16    Fz;          // Z-axis force in 16bit Counts*
 INT16    Tx;          // X-axis torque in 16bit Counts* (0 if not available)
 INT16    Ty;          // Y-axis torque in 16bit Counts*(0 if not available)
 INT16    Tz;          // Z-axis torque in 16bit Counts* (0 if not available)
```

Fig36. Datos de salida TCP

El recuento de bytes de la respuesta es siempre de 16 bytes con valores de varios bytes enviados como el primer byte alto.

Para establecer la conexión mediante UDP o TCP se puede utilizar el software Hercules.

- **Hercules para UDP**

Una de las aplicaciones para poder comprobar el correcto funcionamiento de la conexión con el sensor, mediante la lectura de los datos, es utilizar Hercules.

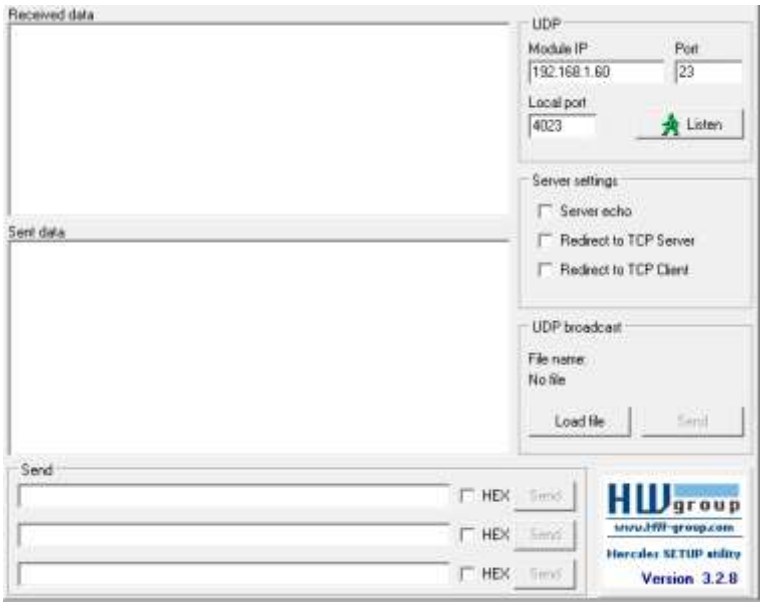


Fig37. Pantalla del Hercules para UDP

La pestaña UDP es un "terminal" simple basado en los datagramas UDP. El propósito principal de esta pestaña es enviar paquetes UDP desde su computadora al destino. Si desea utilizar el modo de comunicación UDP, es bastante complicado encontrar alguna utilidad para la depuración en Windows.

Parámetros:

- Asigne el puerto y la IP para escuchar
- Enviar transmisiones UDP
- Envíe y muestre mensajes o comandos hacia y desde un dispositivo remoto

Otros parámetros (en el menú contextual)

- Mostrar caracteres especiales (en ASCII, HEX o DEC)
- Registrar la comunicación en un archivo
- Enviar un archivo al dispositivo remoto

Módulo IP:

- La dirección IP del dispositivo remoto.

Puerto:

- El puerto del dispositivo remoto donde se envían los paquetes UDP.

Puerto local:

- Escuche en este puerto local los paquetes UDP recibidos.

Configuración del servidor - Eco del servidor:

- Al marcar esta casilla de verificación, el "terminal" UDP inicia la función ECHO = envía todos los datos recibidos al destino UDP definido.

Difusión UDP:

- Al cargar un archivo de difusión especial .brf, puede enviar paquetes de difusión a la red.

Botón Enviar:

- Para enviar datos al dispositivo, se presiona el botón Enviar a la derecha del campo respectivo o use F1, F2, F3 respectivamente para cada campo.
  Se usa la casilla HEX para enviar caracteres hexadecimales (usados para enviar comandos NVT). Los caracteres ASCII se envían escribiendo #xxx (donde x es un número del 0 al 9), $yy (donde y es un número hexadecimal)

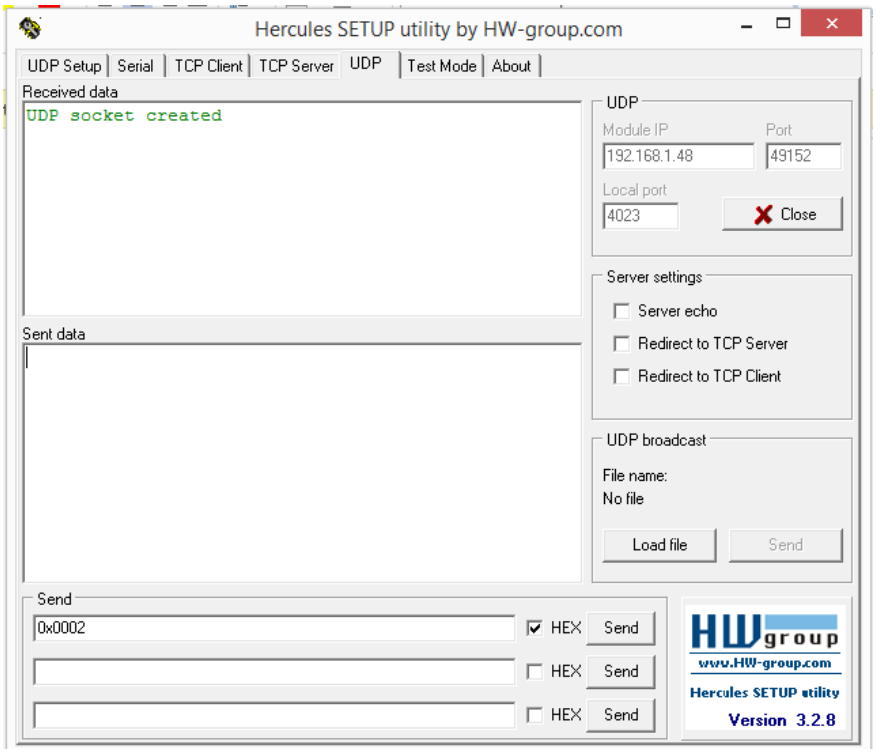


Fig38. Pantalla del Hercules para UDP con los comandos a utilizar

- **Hercules para TCP**

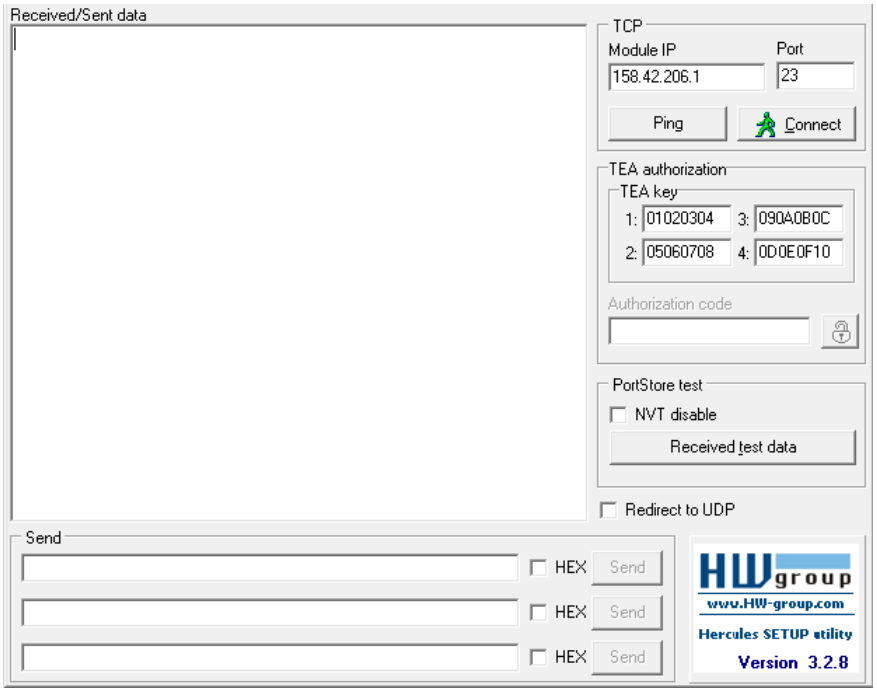


Fig39. Pantalla Hercules para TCP

La pestaña Cliente TCP de la utilidad Hercules se puede utilizar como un terminal de cliente TCP/IP simple similar a Telnet.

Parámetros:

- Asignar los parámetros de la conexión (IP, puerto)
- Establecer la clave TEA y el código de autorización
- Envíe y muestre mensajes o comandos hacia y desde un dispositivo remoto
- Recibir datos de prueba: utilizados con dispositivos HWg

Otros parámetros (en el menú contextual):

- Mostrar caracteres especiales (en ASCII, HEX o DEC)
- Registrar la comunicación en un archivo
- Enviar un archivo al dispositivo remoto

Modulo IP:

- La dirección IP del dispositivo remoto

Puerto:

- El puerto del dispositivo remoto: 23 para Telnet, 99 para la configuración TCP de dispositivos HWg.

Botón de ping:

- Facilidad para hacer ping al dispositivo remoto para comprobar si hay una conexión. Resultados mostrados en la ventana de datos Recibidos/Enviados.

Botón Conectar/Desconectar:

- Abre y cierra la conexión TCP/IP con el dispositivo remoto.

Clave TEA Contraseña:

- Clave segura de 16 bytes. Está configurado en ambos lados, nunca va a pensar en la red.
  Está definido en la mayoría de nuestros dispositivos como 4 grupos con 4 bytes definidos en forma HEX.

Código de autorización:

- Para abrir la comunicación con el dispositivo seguro de TEA, debe cortar 12 dígitos en su portapapeles y pegarlos en el formulario "Código de autorización". Luego haga clic en el botón con la imagen del candado y el resultado se enviará a la conexión.
  Si su TEA Key es igual a la Key del lado opuesto, la conexión TCP está habilitada. De lo contrario, la conexión está con algún tiempo de espera cerrado por el Cliente TCP (dispositivo).

Finalmente se puede resolver la conexión con el sensor de manera directa sin utilización del polyscope mediante el software Hercules o bien mediante Python. No se puede comprobar el correcto funcionamiento debido a la rotura de dicho dispositivo.

1.7.2. Experimento 1: Movimiento del Robot UR3

Se prueba el movimiento individualizado de un robot, probando que se mueve correctamente indicándole un desplazamiento en x, y o z según su sistema de coordenadas.

Se realiza un código independiente para el UR3 y para el URe.

- Primero se realizan los experimentos en el **robot UR3: (Conexión_Ure22)**

Mediante la función bind se ata al ServerSocket a la dirección especificada (dirección IP y puerto). La dirección IP es la 192.168.1.103, que corresponde al ordenador utilizado, y el puerto 3000. El ServerSocket no debe estar atado previamente.

Se realiza el accept, para establecer la conexión con el cliente. Escucha una solicitud de conexión y la acepta cuando se recibe. En ese momento, se crea nuevo Socket que es el que realmente se conecta con el cliente. Tras esto, el ServerSocket sigue disponible para realizar nuevos accept ().

Durante el código se realizan envió de datos y recibimiento de datos a través de las funciones receive (recv) y send. La función receive, recibe un datagrama en el socket. Esta llamada es bloqueante. Cuando este método retorna, los datos se devuelven en el buffer del DatagramPacket. El DatagramPacket también contiene la dirección IP y el puerto del emisor.

La función send, envía un datagrama desde el socket. El DatagramPacket contiene los datos, su longitud, la dirección IP del receptor y su número de puerto.

En este programa se recibe la posición inicial del robot, posteriormente se envia el robot a una nueva posición a 260 mm en Y del punto en el que se encuentra. Se recibe la posición que se le ha enviado para ver si es correcta esa llegada de datos y por último se reciben las conversiones realizadas en el Polyscope, recibiendo finalmente la posición final.

Respecto al programa en Polyscope para abrir la conexión mediante socket, se utiliza la función **socket_open(),** en ella se especifica la IP del ordenador y el puerto especificado. La obtención de la posición actual del robot se obtiene mediante la función **get_actual_tcp_pose()**, se envía la posición como una string mediante la función **socket_send_string().** Para leer unos datos enviados a través del socket se utiliza la función **socket_read_ascii_float().**

Por último se utilliza la función **pose_trans(),** ya que el movimiento que se envía es relativo a la posición actual del robot se utiliza esta función, esta devolverá la posición en coordenadas absolutas respecto de la base a donde se debe mover el robot. Recibe como entradas una posición inicial (p_from) y una posición final respecto la inicial (p_from_to).

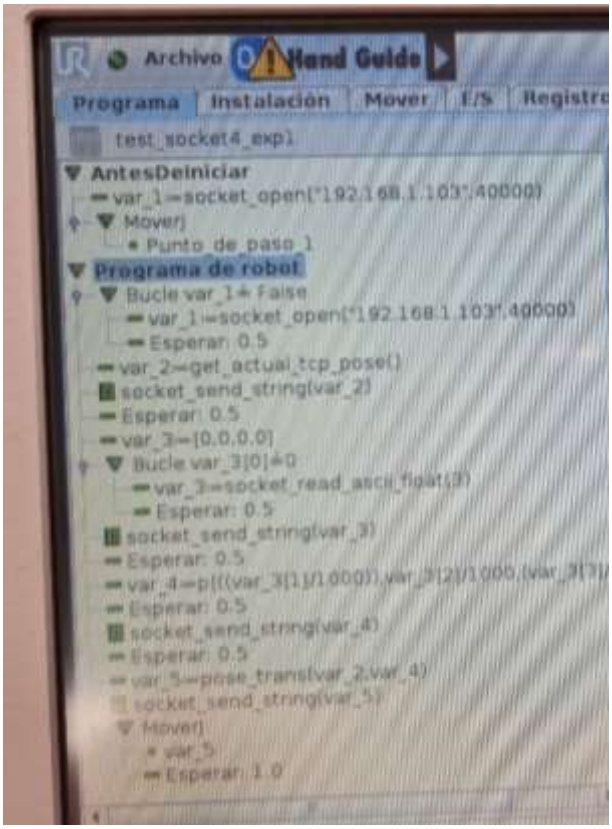


Fig40. Programa movimiento individual UR3

Se realizan pruebas con movimientos tanto en x, como en y, como en z. Y comprobando si realizando el movimiento indicado, se alcanza la posición final deseada y se mueve correctamente por el eje indicado.

- Movimiento en x:

```
Starting Program
Port binded
Client connected
Connection with client
Posición Inicial
b'p[-0.312096,-0.134968,0.0681184,2.23626,2.18544,-0.0390463]'
Data sent
Posicion enviada al UR3
b'[3,0,200,0]'
Posicion recibida en el UR3, var_4
b'p[0,0.26,0,0,0,0]'
Posicion Final del UR3 (var_5)
b'p[-0.0521716,-0.140969,0.0582822,2.23626,2.18544,-0.0390463]'
Program finish

Process finished with exit code 0
```

Fig41. Respuesta del codigo Python al movimiento x

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento en x:

https://youtube.com/shorts/z9Vzdezajcc

- Movimiento en y:

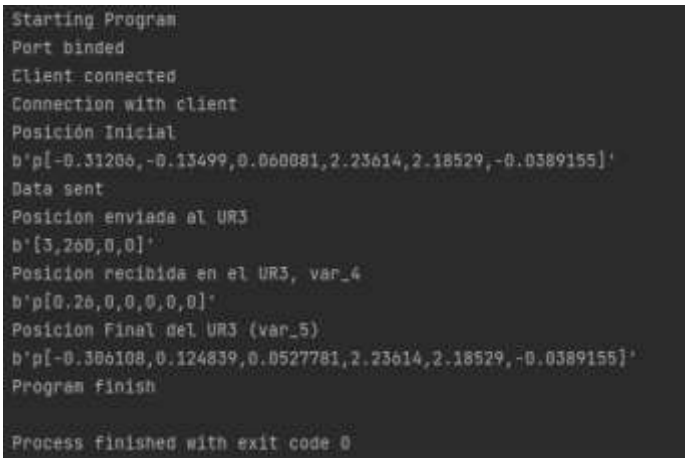


Fig42. Respuesta del codigo Python al movimiento y

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento en y:

https://youtube.com/shorts/touuZF43BBc

- Movimiento en z:

Se realiza un movimiento negativo respecto al eje, por lo que se movera -260mm en Z.

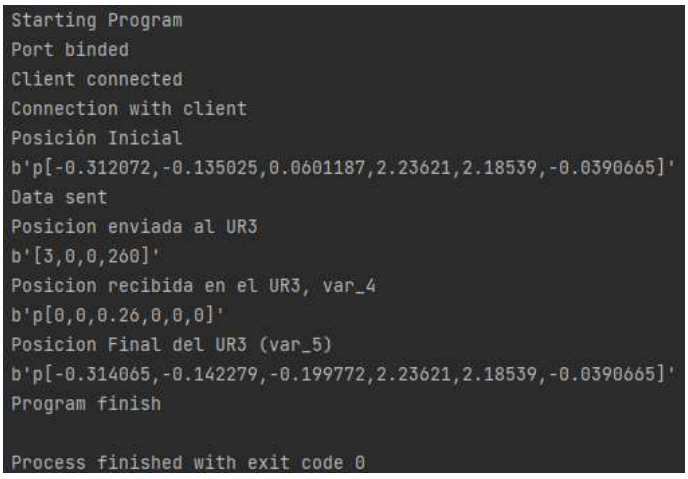


Fig43. Respuesta del codigo Python al movimiento z

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento en z:

https://youtube.com/shorts/ysY-uphbsFA

- Se realizan los experimentos en el **robot URe:**

Se utilizan las mismas funciones para el UR3 que para el URe como son las funciones bind, accept, receive y send.

En este código se va a cambiar el sistema de coordenadas de referencia del robot para que ambos tengan un sistema común, para ello primero se recibirá la posición actual del mismo. Como se ha podido comprobar en el experimento anterior las posiciones al enviarse llevan consigo el formato de vector acompañado previamente de las letras ***b'p***, esto dificulta la transformación de coordenadas al no poder asignar un valor a una variable de forma directa, por lo que se remplazan estos caracteres y los corchetes para una correcta interpretación de los datos mediante los siguiente comandos:

```
#UR3
pos1 = msg.decode()
pos1 = pos1[2:len(pos1)]
a = pos1.replace("b'p[", "").replace("]", "")
lst1 = [x for x in a.split(',')]
print(lst1)

p01 = (float(lst1[0]))
p02 = (float(lst1[1]))
p03 = (float(lst1[2]))
```

Fig43. Codigo de Python del recibiento de datos

Posteriormente estos datos se transforman mediante una matriz de rotación y transformación. El nuevo sistema de coordenadas será el sistema de coordenadas de la base del UR3, el maestro.

El código desarrollado se puede observar en el **Anexo 1.**

Del sistema de la base del esclavo, al sistema de la base del maestro, se realiza un giro en z de -90 grados.

```
T =

         0    1.0000         0   -0.0994
   -1.0000         0         0   -0.8780
         0         0    1.0000   -0.2060
         0         0         0    1.0000
```

Posteriormente a esta posición actual del robot en el nuevo sistema de referencia, se le sumara el desplazamiento que se quiere realizar para obtener la nueva posición final. A esta posición final expresada en el sistema de coordenadas del maestro, se volverá a pasar al sistema de coordenadas de la base del esclavo para poder enviarle esta nueva posición.

Del sistema de la base del maestro, al sistema de la base del esclavo, se realiza un giro en z de 90 grados.

```
T2 =

         0   -1.0000         0   -0.8780
    1.0000         0         0    0.0994
         0         0    1.0000    0.2060
         0         0         0    1.0000
```

Para cambiar las coordenadas de rotación también aplicamos una matriz de rotación.

```
T3 =

     0     1     0     0
    -1     0     0     0
     0     0     1     0
     0     0     0     1
```

A diferencia del código anterior, al sumar las posiciones, se envía directamente la posición final respecto la base del esclavo y no es necesario la función pose_trans ().

Código realizado en el Polyscope:

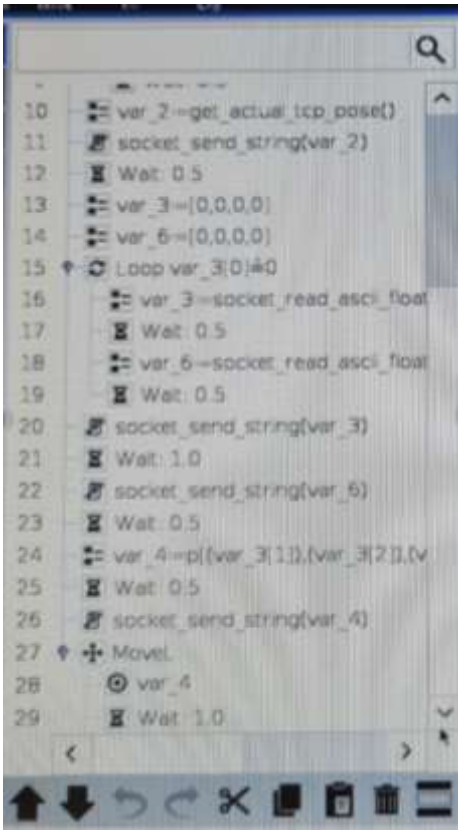


Fig44. Codigo de Polyscope movimientoo individual UR3e

Se realizan pruebas con movimiento tanto en x, como en y, como en z. Y comprobando si realizando el movimiento indicado, se alcanza la posición final deseada y se mueve correctamente por el eje indicado.

- Movimiento en x

Se envía el robot a la nueva posición de 200 mm respecto el nuevo sistema de coordenadas.

```
pos = np.array([[0.200], [0], [0], [0]])
# REALIZAR  #X,Y,Z,-;
```

Fig45. Instrucción de Python con la nueva posición

Se puede observar cómo se ha desplazado el robot 200 mm en x observado la posición inicial y posición final. La posición inicial era de -0.2794 m y la final ha sido de -0.0794 m.

A traves del siguiente enlace, se puede ver el video de otra prueba con un movimiento en x negativo:

https://youtube.com/shorts/mOUumdrNVko

```
Starting Program
Port binded
Client connected
Connection with client
Posición Inicial
b'p[0.134057,-0.279407,0.144673,-2.355,-2.06974,-0.00895055]'
['0.134057', '-0.279407', '0.144673', '-2.355', '-2.06974', '-0.00895055']
Posiciones recibidas
0.134057
-0.279407
0.144673
Posicion respecto el nuevo sistema de coordenadas
[[-0.378807]
 [-1.012057]
 [-0.061327]
 [ 1.      ]]
Nueva posicion respecto el nuevo sistema de coordenadas
[[-0.178807]
 [-1.012057]
 [-0.061327]
 [ 1.      ]]
Nueva posicion respecto el  sistema de coordenadas de la base
[[ 0.134057]
 [-0.079407]
 [ 0.144673]
 [ 1.      ]]
```

```
POSICION A ENVIAR
(0.13405699999999998, -0.07940699999999999, 0.144673)
(-2.355, -2.06974, -0.00895055)
Data sent
Data sent 2
Posicion enviada al UR3
b'[3,0.134057,-0.079407,0.144673]'
Posicion 2 enviada al UR3
b'[3,-2.355,-2.06974,-0.00895055]'
Posicion Final UR3
b'p[0.134057,-0.079407,0.144673,-2.355,-2.06974,-0.00895055]'

Program finish

Process finished with exit code 0
```

Fig46. Respuesta del codigo de Python al movimiento en x

- Movimiento en y

Se envía el robot a la nueva posición de 200 mm en Y respecto el nuevo sistema de coordenadas. Se puede observar cómo se ha desplazado el robot 200 mm en Y observado la posición inicial y posición final. La posición inicial era de 0.1340 m y la final ha sido de -0.0659 m.

A traves del siguiente enlace, se puede ver el video de otra prueba del movimiento en Y en sentido positivo:

https://youtube.com/shorts/sOoHXksAPaE

```
Starting Program
Port binded
Client connected
Connection with client
Posición Inicial
b'p[0.134046,-0.279426,0.144686,-2.35511,-2.06974,-0.00892]'
['0.134046', '-0.279426', '0.144686', '-2.35511', '-2.06974', '-0.00892']
Posiciones recibidas
0.134046
-0.279426
0.144686
Posicion respecto el nuevo sistema de coordenadas
[[-0.378826]
 [-1.012046]
 [-0.061314]
 [ 1.      ]]
Nueva posicion respecto el nuevo sistema de coordenadas
[[-0.378826]
 [-0.812046]
 [-0.061314]
 [ 1.      ]]
Nueva posicion respecto el  sistema de coordenadas de la base
[[-0.865954]
 [-0.279426]
 [ 0.144686]
 [ 1.      ]]
```

```
POSICION A ENVIAR
(-0.0659539999999999, -0.279426, 0.144686)
(-2.35511, -2.06974, -0.00892)
Data sent
Data sent 2
Posicion enviada al UR3
b'[3,-0.065954,-0.279426,0.144686]'
Posicion 2 enviada al UR3
b'[3,-2.35511,-2.06974,-0.00892]'
Posicion Final UR3
b'p[-0.065954,-0.279426,0.144686,-2.35511,-2.06974,-0.00892]'

Program finish

Process finished with exit code 0
```

Fig47. Respuesta del codigo de Python al movimiento en y

- Movimiento en z

Se envía el robot a la nueva posición de -200 mm en Z respecto el nuevo sistema de coordenadas.

```
pos = np.array([[0], [0], [-0.200], [0]])
# REALIZAR  #X,Y,Z,-;
```

Fig48. Instrucción de Python con la nueva posición variando z

Se puede observar cómo se ha desplazado el robot 200 mm en Z, observado la posición inicial y posición final. La posición inicial era de 0.144 m y la final ha sido de -0.0553 m.

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento en z negativo:

https://youtube.com/shorts/odpp3dPjU9U

```
Starting Program
Port binded
Client connected
Connection with client
Posición Inicial
b'p[0.134059,-0.279396,0.144664,-2.35504,-2.06968,-0.00905631]'
['0.134059', '-0.279396', '0.144664', '-2.35504', '-2.06968', '-0.00905631']
Posiciones recibidas
0.134059
-0.279396
0.144664
Posicion respecto el nuevo sistema de coordenadas
[[-0.378796]
 [-1.012059]
 [-0.061336]
 [ 1.      ]]
Nueva posicion respecto el nuevo sistema de coordenadas
[[-0.378796]
 [-1.012059]
 [-0.261336]
 [ 1.      ]]
Nueva posicion respecto el  sistema de coordenadas de la base
[[ 0.134059]
 [-0.279396]
 [-0.055336]
 [ 1.      ]]
POSICION A ENVIAR
(0.13405900000000004, -0.279396, -0.055336000000000024)
(-2.35504, -2.06968, -0.00905631)
Data sent
Data sent 2
Posicion enviada al UR3
b'[3,0.134059,-0.279396,-0.055336]'
Posicion 2 enviada al UR3
b'[3,-2.35504,-2.06968,-0.00905631]'
Posicion Final UR3
b'p[0.134059,-0.279396,-0.055336,-2.35504,-2.06968,-0.00905631]'

Program finish
```

Fig49. Respuesta del codigo de Python al movimiento en z

- **Prueba con Path_offset_set**

Se realiza una prueba con las funciones path_offset_set (offset, type) y path_offset_enable() para conseguir reducir el offset a 0 entre los dos sistemas de coordenadas. Esto sería una alternativa a lo realizado mediante Python.

Path_offset_enable, sirve para habilitar la compensación de ruta. Esta se utiliza para superponer una compensación cartesiana en el movimiento del robot a medida que sigue una trayectoria.

Path_offset_set, se especifica el desplazamiento de la ruta cartesiana que se aplicara. La compensación se aplica durante cada ciclo a 500Hz. Es probable que las compensaciones discontinuas o entrecortadas causen paradas de protección.

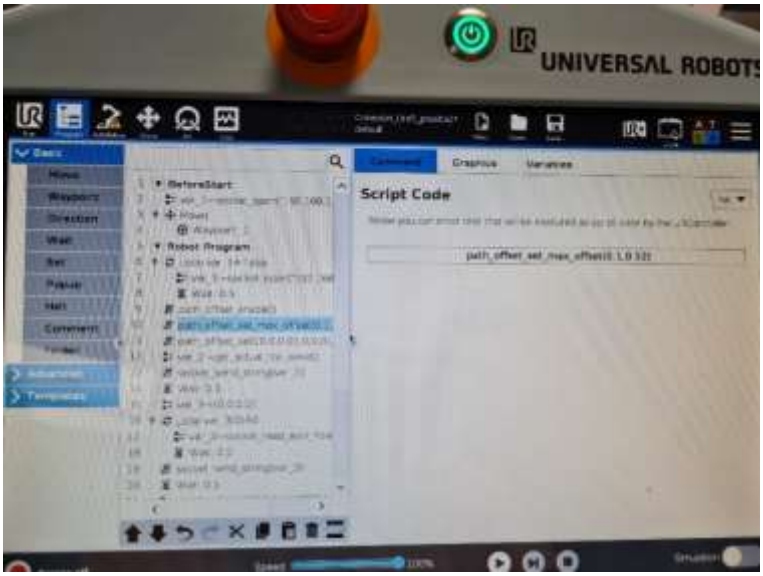


Fig50. Codigo Polyscope con Path_offset_set

Durante la realización de la prueba se observa que no consigue realizar correctamente el offset y que además aparecen errores de posición, donde el robot es incapaz de alcanzar la posición por dinámica inversa

1.7.3. Experimento 2: Movimiento coordinado de dos robots colaborativos.

Para la realización de este experimento se ha realizado dos programas en la pantalla de cada robot, una en el UR3e (izquierda) y otra en el UR3 (derecha):

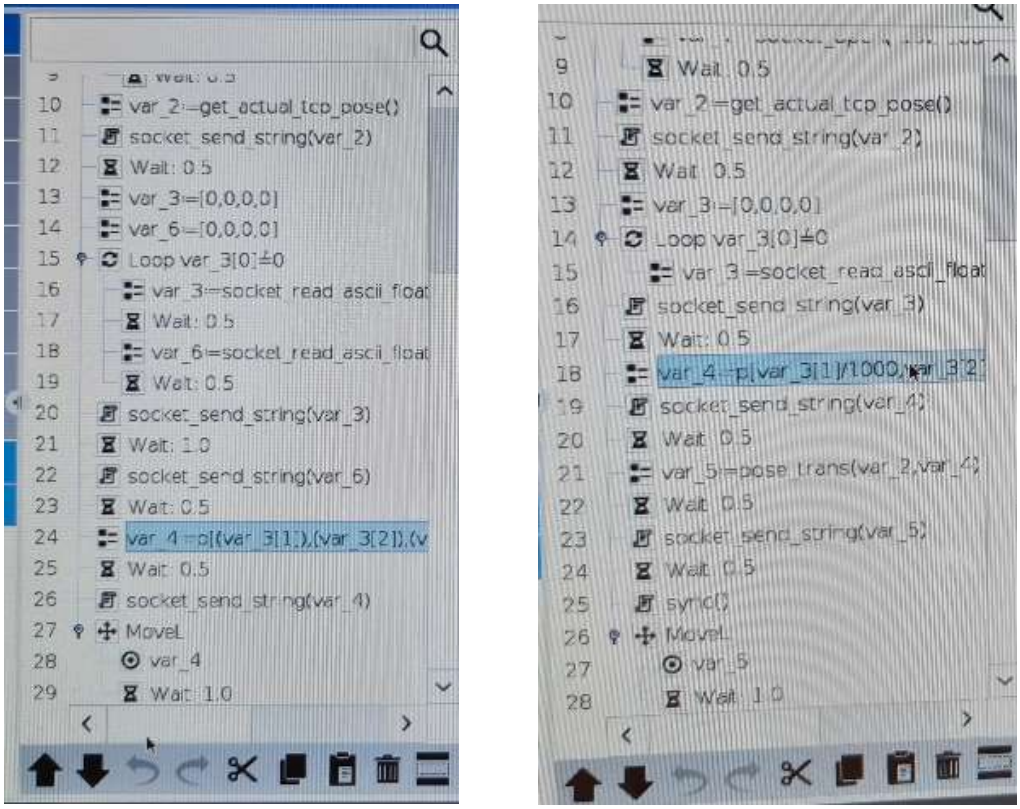


Fig51. Codigo Polyscope UR3e y UR3 para movimiento coordinado

En el UR3e se envia por separado la posición y la orientación final del robot, existe la limitación de no poder enviarse y recibir los 6 datos a la vez por las funciones del propio robot. La función **socket_read_ascii_float** limita en 30 los valores que se pueden leer con el mismo comando.

También se ha realizado un programa en Python que nos permite comunicarnos con ambos robots. En este programa primero mediante la función c.recv y d.recv se recibe la posición de cada robot enviada por el comando get_actual_tcp_pose().

El código realizado en Python se puede encontrar en el Anexo 1.

En Python se realiza dicha conversión de las coordenadas actuales basadas en el sistema de coordenadas de la base del URe al nuevo sistema de coordenadas.

Una vez realizada la conversión y teniendo ambas coordenadas en el mismo sistema de referencia, se realiza un movimiento en X de 260 mm en cada uno de los robots.
Así una vez se tiene la nueva posición que se quiere que se desplace el robot, se envia la posición nueva al UR3 y al URe. Antes de enviar al URe se realiza nuevamente una conversión del sistema de coordenadas de la base del maestro al sistema de coordenadas de la base del robot.

Cabe destacar que en el UR3 se expresa las coordenadas como XYZ mientras que en el UR3e se expresa como YXZ.

- **Move J**

Con MoveJ se observa que en el desplazamiento del UR3 se mueve también en z, mientras que el URe realiza el movimiento sin tanta desviación. Esto es debido al movimiento con MoveJ, ya que no realiza un movimiento lineal.

Se realizan 4 experimentos para observar cuanto desviamiento hay en z al realizar un movimiento en x.

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveJ:

https://youtube.com/shorts/69i3E62kfGE?feature=share

- **Datos 1**

```
Posicion Inicial UR3
b'p[-0.375911,-0.425258,-0.0275252,1.57015,0.0678737,-0.0630615]'
Posicion Inicial URe
b'p[-0.610032,-0.26763,0.158751,0.0677632,-1.57011,-0.0631828]'
Posicion Final URe
b'p[-0.610032,-0.00763,0.158751,0.0677632,-1.57011,-0.0631828]'
Posicion Final del UR3 (var_5)
b'p[-0.116826,-0.424446,-0.049173,1.57013,0.0677649,-0.0630846]'
```

Fig52. Primera toma de datos de las posciones con MoveJ

- **Datos 2**

```
Posicion Inicial UR3
b'p[-0.375927,-0.425284,-0.0275103,1.57015,0.0679299,-0.0631551]'
Posicion Inicial URe
b'p[-0.610022,-0.26762,0.158694,0.0677651,-1.57024,-0.0632152]'
['-0.610022', '-0.26762', '0.158694', '0.0677651', '-1.57024', '-0.0632152']
Posicion Final URe
b'p[-0.610022,-0.00762,0.158694,0.0677651,-1.57024,-0.0632152]'
Posicion Final del UR3 (var_5)
b'p[-0.116835,-0.424484,-0.049196,1.57014,0.067917,-0.0631018]'
```

Fig53. Segunda toma de datos de las posciones con MoveJ

- **Datos 3**

```
Posicion Inicial UR3
b'p[-0.375918,-0.425276,-0.0275155,1.57014,0.0679966,-0.0631648]'
Posicion Inicial URe
b'p[-0.610019,-0.267631,0.1587,0.0677774,-1.57022,-0.063228]'
Posicion Final URe
b'p[-0.610019,-0.007631,0.1587,0.0677774,-1.57022,-0.063228]'
Posicion Final del UR3 (var_5)
b'p[-0.116846,-0.424464,-0.049202,1.57014,0.0680284,-0.0632324]'
```

Fig54. Tercera toma de datos de las posciones con MoveJ

- **Datos 4**

```
Posicion Inicial UR3
b'p[-0.375916,-0.425276,-0.0275102,1.57014,0.0678834,-0.06309]'
Posicion Inicial URe
b'p[-0.610025,-0.267627,0.158716,0.0677661,-1.57018,-0.0632195]'
Posicion Final URe
b'p[-0.610025,-0.007627,0.158716,0.0677661,-1.57018,-0.0632195]'
Posicion Final del UR3 (var_5)
b'p[-0.116824,-0.42447,-0.0491695,1.57014,0.0678593,-0.0630458]'
```

Fig55. Cuarta toma de datos de las posciones con MoveJ

Posición Z UR3	Inicial	Final	Desviación
1	**-0.0275252**	**-0.049173**	**0.0216478**
2	**-0.0275103**	**-0.049196**	**0.0216857**
3	**-0.0275155**	**-0.049202**	**0.0216865**
4	**-0.0275102**	**-0.0491695**	**0.0216593**
Promedio	**x**	**x**	**0.0216698**

Tabla 12. Comparación de posición del UR3 con MoveJ

- **Move L**

Con MoveL se observa que el movimiento de ambos es correcto, aunque se sigue observando un pequeño desplazamiento en z en el UR3.

- **Datos 1**

```
Posicion Inicial UR3
b'p[-0.375847,-0.425308,-0.0275616,1.57014,0.0679305,-0.0629485]'
Posicion Inicial URe
b'p[-0.610021,-0.267639,0.158735,0.0677742,-1.57015,-0.0632104]'
Posicion Final URe
b'p[-0.610021,-0.007639,0.158735,0.0677742,-1.57015,-0.0632104]'
Posicion Final del UR3 (var_5)
b'p[-0.116778,-0.424469,-0.049207,1.57015,0.0679913,-0.0630453]'
```

Fig56. Primera toma de datos de las posciones con MoveL

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveL:

https://youtube.com/shorts/mm3R1QdFmv0?feature=share

- **Datos 2**

```
Posicion Inicial UR3
b'p[-0.37587,-0.425303,-0.0275647,1.57014,0.0678174,-0.0629672]'
Posicion Inicial URe
b'p[-0.610026,-0.267619,0.158705,0.067777,-1.57022,-0.0632473]'
Posicion Final URe
b'p[-0.610026,-0.007619,0.158705,0.067777,-1.57022,-0.0632473]'
Posicion Final del UR3 (var_5)
b'p[-0.116811,-0.424488,-0.0491882,1.57015,0.0678786,-0.0630817]'
```

Fig57. Segunda toma de datos de las posciones con MoveL

- **Datos 3**

```
Posicion Inicial UR3
b'p[-0.375902,-0.425267,-0.0275252,1.57014,0.0679018,-0.0630521]'
Posicion Inicial URe
b'p[-0.61002,-0.267639,0.158735,0.0677686,-1.57016,-0.0632043]'
Posicion Final URe
b'p[-0.61002,-0.007639,0.158735,0.0677686,-1.57016,-0.0632043]'
```

```
Posicion Final del UR3 (var_5)
b'p[-0.116853,-0.424447,-0.0491871,1.57016,0.0680007,-0.063186]'
```

Fig58. Tercera toma de datos de las posciones con MoveL

- **Datos 4**

```
Posicion Inicial UR3
b'p[-0.375893,-0.42529,-0.0275323,1.57014,0.0678838,-0.0630522]'
Posicion Inicial URe
b'p[-0.61002,-0.267622,0.158689,0.0677654,-1.57024,-0.063234]'
Posicion Final URe
b'p[-0.61002,-0.007622,0.158689,0.0677654,-1.57024,-0.063234]'
Posicion Final del UR3 (var_5)
b'p[-0.116805,-0.424478,-0.0491605,1.57014,0.0678975,-0.0631199]'
```

Fig59. Cuarta toma de datos de las posciones con MoveL

Posición Z UR3	Inicial	Final	Desviación
1	**-0.0275616**	**-0.049207**	**0.0216454**
2	**-0.0275647**	**-0.0491882**	**0.0216235**
3	**-0.0275252**	**-0.0491871**	**0.0216619**
4	**-0.0275323**	**-0.0491605**	**0.0216282**
Promedio	**x**	**x**	**0.0216397**

Tabla13. Comparación de las posciones del UR3 con MoveL

- **Move P**

Con MoveP, se observa que es el movimiento que mejor realiza la trayectoria, no hay prácticamente desplazamiento en z y es un movimiento suave y coordinado. Este se puede conseguir con ambos robots a las mismas velocidades y aceleraciones.

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveP:

https://youtube.com/shorts/sNR6Kd-5QXE?feature=share

- **Datos 1**

```
Posicion Inicial UR3
b'p[-0.375872,-0.42529,-0.027505,1.57014,0.0680905,-0.06307]'
Posicion Inicial URe
b'p[-0.610014,-0.267658,0.158721,0.0678229,-1.57018,-0.0631627]'
Posicion Final URe
b'p[-0.610014,-0.007658,0.158721,0.0678229,-1.57018,-0.0631627]'
Posicion Final del UR3 (var_5)
b'p[-0.1417,-0.424538,-0.0471114,1.57014,0.0682091,-0.0631833]'
```

Fig60. Primera toma de datos de las posciones con MoveP

- **Datos 2**

```
Posicion Inicial UR3
b'p[-0.37588,-0.425294,-0.0275523,1.57014,0.0679965,-0.0631079]'
Posicion Inicial URe
b'p[-0.610018,-0.26763,0.158724,0.0677415,-1.57019,-0.0631939]'
Posicion Final URe
b'p[-0.610018,-0.00763,0.158724,0.0677415,-1.57019,-0.0631939]'
Posicion Final del UR3 (var_5)
b'p[-0.141703,-0.424556,-0.0471387,1.57014,0.0682001,-0.0631922]'
```

Fig61. Segunda toma de datos de las posciones con MoveP

- **Datos 3**

```
Posicion Inicial UR3
b'p[-0.375896,-0.425263,-0.0275573,1.57014,0.067883,-0.0630896]'
Posicion Inicial URe
b'p[-0.610029,-0.267634,0.158735,0.0677512,-1.57014,-0.06317]'
Posicion Final URe
b'p[-0.610029,-0.007634,0.158735,0.0677512,-1.57014,-0.06317]'
Posicion Final del UR3 (var_5)
b'p[-0.141692,-0.424545,-0.047128,1.57014,0.0680115,-0.0630612]'
```

Fig62. Tercera toma de datos de las posciones con MoveP

- **Datos 4**

```
Posicion Inicial UR3
b'p[-0.375946,-0.42524,-0.0275292,1.57015,0.0679307,-0.0631932]'
Posicion Inicial URe
b'p[-0.610026,-0.267628,0.158732,0.0677682,-1.57016,-0.0631848]'
Posicion Final URe
b'p[-0.610026,-0.007628,0.158732,0.0677682,-1.57016,-0.0631848]'
Posicion Final del UR3 (var_5)
b'p[-0.141809,-0.424525,-0.0471266,1.57015,0.0680022,-0.0633334]'
```

Fig63. Cuarta toma de datos de las posciones con MoveP

Posición Z UR3	Inicial	Final	Desviación
1	**-0.027505**	**-0.0471114**	**0.0196064**
2	**-0.0275523**	**-0.0471387**	**0.0195864**
3	**-0.0275573**	**-0.047128**	**0.0195707**
4	**-0.0275292**	**-0.0471266**	**0.0195974**
Promedio	**x**	**x**	**0.0195902**

Tabla14. Comparación de la posición de UR3 con MoveP

El movimiento que menos desviación provoca en la coordenada Z es el MoveP. Esto se puede deber a que MoveP combina movimientos lineales y circulares y tiene una velocidad contante.

1.7.4. Experimento 3. Movimiento coordinado de dos robots cogiendo un objeto

En esta parte del trabajo se realizan dos experimentos de movimiento coordinado de dos robots colaborativos cogiendo un objeto, uno con dos robots colaborativos UR3e y otro con un robot colaborativo UR3 y UR3e. El objeto en cuestión es una caja de cartón de 200x150x100 mm.

Se realiza el experimento para conocer con qué movimiento se consigue una mejor coordinación de ambos robots, entre MoveL, MoveJ y MoveP, sin que se caiga el objeto y no se produzcan movimientos bruscos.

Para esta parte se ha utilizado el código de Python de movimiento coordinado utilizado en el experimento anterior.

Se ha realizado dos nuevos programas en el Polyscope de cada robot. Ambos se encuentran en el Anexo1.

Cabe recordar que en el UR3 se expresa las coordenadas como XYZ mientras que en el UR3e se expresa como YXZ.

- **Move L**

Se realiza un movimiento en X de 200 mm para desplazar los robots hasta una nueva posición. Con MoveL se realiza correctamente el desplazamiento con la caja, ya que esta no se cae durante el recorrido y llega hasta la posición final.

Al ser el mismo robot, con la misma velocidad en MoveL (60 mm/s de tool speed y 60 mm/s2 de tool acceleration) el movimiento es coordinado.

En las siguientes imágenes se puede observar las posiciones de ambos robots mediante el código devuelto de Python. El primer robot UR3 se observa una pequeña desviación tanto en Y como en Z, ya que se debería mantener la posición inicial en ambas coordenadas y se observa un pequeño incremento. En Y se desvía un 0.01m y en Z un 0.02m.

```
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.393888,-0.426969,-0.0516368,1.56564,0.0461657,-0.111243]'
Posicion Inicial URe
b'p[-0.553243,-0.294075,0.136496,1.45724,-1.04326,-1.45418]'
['-0.553243', '-0.294075', '0.136496', '1.45724', '-1.04326', '-1.45418']
Posiciones recibidas del URe
-0.553243
-0.294075
0.136496
Posicion del URe respecto el nuevo sistema de coordenadas
[[-0.307757]
 [-0.378925]
 [-0.044504]
 [ 1.      ]]
Nueva posicion del URe  respecto el nuevo sistema de coordenadas
[[-0.307757]
 [-0.578925]
 [-0.044504]
 [ 1.      ]]
Nueva posicion del URe respecto el  sistema de coordenadas de la base
[[-0.553243]
 [-0.094075]
 [ 0.136496]
 [ 1.      ]]
POSICION A ENVIAR AL URe
(-0.553243, -0.09407499999999991, 0.136496)
(1.45724, -1.04326, -1.45418)
Data sent 1 URe
Data sent 2 URe
Data sent UR3
```

```
Posicion enviada al URe
b'[3,-0.553243,-0.094075,0.136496][3,1.45724,-1.04326,-1.45418]'
Posicion 2 enviada al URe
b'p[-0.553243,-0.094075,0.136496,1.45724,-1.04326,-1.45418]'
Posicion enviada al UR3
b'[3,200,0,0]p[0.2,0,0,0,0,0]'
Posicion Final URe
b'p[-0.537143,-0.294016,0.0893965,1.45684,-1.04316,-1.4543]'
Posicion recibida en el UR3, var_4
b'p[-0.195064,-0.435278,-0.0716362,1.56564,0.0461657,-0.111243]'
Posicion Final del UR3 (var_5)
b'p[-0.393902,-0.422165,-0.100382,1.56561,0.0463028,-0.111335]'
Program finish
```

Fig64. Codigo devuelto por Python con movimiento MoveL

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveL:

https://youtube.com/shorts/kvctmvElzZI

- **Move J**

Se utiliza un joint speed de 30 º/s y un joint acceleration de 30º/s2. Se observa una pequeña desviación tanto en Y con un valor 0.008 como en Z con 0.02 en el UR3.

Con Move J se observa que no es el movimiento ideal para transportar el objeto, ya que con este tipo de movimiento da como resultado una trayectoria curva para la herramienta.

```
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.393864,-0.426973,-0.0516203,1.56565,0.0461934,-0.111188]'
Posicion Inicial URe
b'p[-0.553251,-0.29407,0.136618,1.45711,-1.04319,-1.45422]'
['-0.553251', '-0.29407', '0.136618', '1.45711', '-1.04319', '-1.45422']
Posiciones recibidas del URe
-0.553251
-0.29407
0.136618
Posicion del URe respecto el nuevo sistema de coordenadas
[[-0.307749]
 [-0.37893 ]
 [-0.044382]
 [ 1.      ]]
Nueva posicion del URe  respecto el nuevo sistema de coordenadas
[[-0.307749]
 [-0.57893 ]
 [-0.044382]
 [ 1.      ]]
Nueva posicion del URe respecto el  sistema de coordenadas de la base
[[-0.553251]
 [-0.09407 ]
 [ 0.136618]
 [ 1.      ]]
POSICION A ENVIAR AL URe
(-0.553251, -0.09406999999999999, 0.136618)
(1.45711, -1.04319, -1.45422)
Data sent 1 URe
Data sent 2 URe
Data sent UR3
Posicion enviada al URe
b'[3,-0.553251,-0.09407,0.136618][3,1.45711,-1.04319,-1.45422]'
Posicion 2 enviada al URe
b'p[-0.553251,-0.09407,0.136618,1.45711,-1.04319,-1.45422]'
Posicion enviada al UR3
b'[3,200,0,0]p[0.2,0,0,0,0,0]'
Posicion Final URe
b'p[-0.537114,-0.29405,0.0893806,1.45687,-1.04312,-1.45432]'
Posicion recibida en el UR3, var_4
b'p[-0.195039,-0.435271,-0.0716163,1.56565,0.0461934,-0.111188]'
Posicion Final del UR3 (var_5)
b'p[-0.393877,-0.422204,-0.100368,1.5656,0.0463814,-0.111319]'
Program finish
```

Fig65. Codigo devuelto por Python con movimiento MoveJ

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveJ:

https://youtube.com/shorts/dw7RB7nF4U8

- **Move P**

Se nota que al llegar al punto final de un movimiento, el cambio de velocidad es más brusco en comparación con los otros movimientos, y se puede escuchar el freno de la articulación por haber llegado al punto. Se puede observar en los videos, como con MoveP si se puede realizar el movimiento coordinado con un objeto, ya que no realiza tanta desviación y el objeto llega hasta la posición final.

Se realizan dos experimentos, con el *Blend with radius* de 5 mm y de 2 mm.

```
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.393917,-0.426461,-0.0566917,1.56563,0.0462874,-0.111379]'
Posicion Inicial URe
b'p[-0.553257,-0.29404,0.131559,1.45714,-1.04328,-1.45425]'
['-0.553257', '-0.29404', '0.131559', '1.45714', '-1.04328', '-1.45425']
Posiciones recibidas del URe
-0.553257
-0.29404
0.131559
Posicion del URe respecto el nuevo sistema de coordenadas
[[-0.307743]
 [-0.37896 ]
 [-0.049441]
 [ 1.      ]]
Nueva posicion del URe  respecto el nuevo sistema de coordenadas
[[-0.307743]
 [-0.57896 ]
 [-0.049441]
 [ 1.      ]]
Nueva posicion del URe respecto el  sistema de coordenadas de la base
[[-0.553257]
 [-0.09404 ]
 [ 0.131559]
 [ 1.      ]]
POSICION A ENVIAR AL URe
(-0.553257, -0.09404000000000001, 0.131559)
(1.45714, -1.04328, -1.45425)
Data sent 1 URe
Data sent 2 URe
Data sent UR3
Posicion enviada al URe
b'[3,-0.553257,-0.09404,0.131559][3,1.45714,-1.04328,-1.45425]'
Posicion 2 enviada al URe
b'p[-0.553257,-0.09404,0.131559,1.45714,-1.04328,-1.45425]'
Posicion enviada al UR3
b'[3,200,0,0]p[0.2,0,0,0,0,0]'
Posicion Final URe
b'p[-0.535759,-0.289302,0.0905244,1.45696,-1.0432,-1.45436]'
Posicion recibida en el UR3, var_4
b'p[-0.195097,-0.434772,-0.0767238,1.56563,0.0462874,-0.111379]'
Posicion Final del UR3 (var_5)
b'p[-0.388897,-0.422476,-0.0996884,1.56564,0.0461376,-0.111269]'
Program finish
```

Fig66. Codigo devuelto por Python con movimiento MoveP y 5mm de radio

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveP y radio 5mm:

https://youtube.com/shorts/c6mQWAQJ1Wg

- **Con 2 mm**

```
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.39387,-0.426968,-0.0537698,1.56565,0.0462791,-0.111238]'
Posicion Inicial URe
b'p[-0.55327,-0.294047,0.134515,1.45706,-1.04323,-1.4543]'
['-0.55327', '-0.294047', '0.134515', '1.45706', '-1.04323', '-1.4543']
Posiciones recibidas del URe
-0.55327
-0.294047
0.134515
Posicion del URe respecto el nuevo sistema de coordenadas
[[-0.30773 ]
 [-0.378953]
 [-0.046485]
 [ 1.      ]]
Nueva posicion del URe  respecto el nuevo sistema de coordenadas
[[-0.30773 ]
 [-0.578953]
 [-0.046485]
 [ 1.      ]]
Nueva posicion del URe respecto el  sistema de coordenadas de la base
[[-0.55327 ]
 [-0.094047]
 [ 0.134515]
 [ 1.      ]]
POSICION A ENVIAR AL URe
(-0.55327, -0.09404699999999999, 0.134515)
(1.45706, -1.04323, -1.4543)
Data sent 1 URe
Data sent 2 URe
Data sent UR3
Posicion enviada al URe
b'[3,-0.55327,-0.094047,0.134515][3,1.45706,-1.04323,-1.4543]'
Posicion 2 enviada al URe
b'p[-0.55327,-0.094047,0.134515,1.45706,-1.04323,-1.4543]'
Posicion enviada al UR3
b'[3,200,0,0]p[0.2,0,0,0,0,0]'
Posicion Final URe
b'p[-0.536537,-0.292177,0.0898204,1.45691,-1.04314,-1.45428]'
Posicion recibida en el UR3, var_4
b'p[-0.195047,-0.435262,-0.073783,1.56565,0.0462791,-0.111238]'
Posicion Final del UR3 (var_5)
b'p[-0.391765,-0.4223,-0.100221,1.56568,0.0459438,-0.110941]'
Program finish
```

Fig67. Codigo devuelto por Python con movimiento MoveP y 2mm de radio

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveP y radio de 2 mm:

https://youtube.com/shorts/3fWKTHWSyNI

Con 5mm de blend se observa una desviación en el UR3 en Y de 0.008 m y en Z de 0.02 m.

Con 2mm de blend se observa una desviación en el UR3 en Y de 0.009 m y en Z de 0.02 m.

1.7.5. Experimento 4. Funcion get_target_tcp_pose

En este experimento se va a utilizar la función *get_target_tcp_pose()* para obtener la posición del robot y comparar así con la función utilizada en el resto de programas que es *get_actual_tcp_pose().* En esta comparación de ambas funciones se quiere comprobar la exactitud al enviar la posición con cada función.

Esta función devuelve las 6 dimensiones que representan la posición y la orientación de la herramienta, especificadas en el marco de la base. El cálculo de esta *pose* se basa en las posiciones articulares objetivo actuales. Mientras que con la función de get_actual_tcp_pose el cálculo de la pose se basa en las lecturas reales del codificador del robot.

Para ello se va a realizar 3 pruebas, una por cada movimiento de MoveL, MoveJ y MoveP.

- **Move L**

```
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.393879,-0.426972,-0.0516088,1.56564,0.0461828,-0.111231]'
Posicion Inicial URe
b'p[-0.553249,-0.294078,0.136623,1.45709,-1.04321,-1.45422]'
['-0.553249', '-0.294078', '0.136623', '1.45709', '-1.04321', '-1.45422']
Posiciones recibidas del URe
-0.553249
-0.294078
0.136623
Posicion del URe respecto el nuevo sistema de coordenadas
[[-0.307751]
 [-0.378922]
 [-0.044377]
 [ 1.      ]]
Nueva posicion del URe  respecto el nuevo sistema de coordenadas
[[-0.307751]
 [-0.578922]
 [-0.044377]
 [ 1.      ]]
Nueva posicion del URe respecto el  sistema de coordenadas de la base
[[-0.553249]
 [-0.094078]
 [ 0.136623]
 [ 1.      ]]
POSICION A ENVIAR AL URe
(-0.553249, -0.094078, 0.136623)
(1.45709, -1.04321, -1.45422)
Data sent 1 URe
Data sent 2 URe
Data sent UR3
Posicion enviada al URe
b'[3,-0.553249,-0.094078,0.136623][3,1.45709,-1.04321,-1.45422]'
Posicion 2 enviada al URe
b'p[-0.553249,-0.094078,0.136623,1.45709,-1.04321,-1.45422]'
Posicion enviada al UR3
b'[3,200,0,0]p[0.2,0,0,0,0,0]'
Posicion Final URe
b'p[-0.537086,-0.294067,0.0893065,1.45708,-1.04319,-1.45426]'
Posicion recibida en el UR3, var_4
b'p[-0.195055,-0.435277,-0.0716089,1.56564,0.0461828,-0.111231]'
Posicion Final del UR3 (var_5)
b'p[-0.393863,-0.422151,-0.100453,1.56566,0.0460854,-0.111118]'
Program finish
```

Fig68. Codigo devuelto por Python con movimiento MoveL

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveL:

https://youtube.com/shorts/7iyB9WSmAKs

- **Move J**

```
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.393879,-0.426972,-0.0516088,1.56564,0.0461828,-0.111231]'
Posicion Inicial URe
b'p[-0.553249,-0.294078,0.136623,1.45709,-1.04321,-1.45422]'
['-0.553249', '-0.294078', '0.136623', '1.45709', '-1.04321', '-1.45422']
Posiciones recibidas del URe
-0.553249
-0.294078
0.136623
Posicion del URe respecto el nuevo sistema de coordenadas
[[-0.307751]
 [-0.378922]
 [-0.044377]
 [ 1.      ]]
Nueva posicion del URe  respecto el nuevo sistema de coordenadas
[[-0.307751]
 [-0.578922]
 [-0.044377]
 [ 1.      ]]
Nueva posicion del URe respecto el  sistema de coordenadas de la base
[[-0.553249]
 [-0.094078]
 [ 0.136623]
 [ 1.      ]]
POSICION A ENVIAR AL URe
(-0.553249, -0.094078, 0.136623)
(1.45709, -1.04321, -1.45422)
Data sent 1 URe
Data sent 2 URe
Data sent UR3
Posicion enviada al URe
b'[3,-0.553249,-0.094078,0.136623][3,1.45709,-1.04321,-1.45422]'
Posicion 2 enviada al URe
b'p[-0.553249,-0.094078,0.136623,1.45709,-1.04321,-1.45422]'
Posicion enviada al UR3
b'[3,200,0,0]p[0.2,0,0,0,0,0]'
Posicion Final URe
b'p[-0.537086,-0.294067,0.0893066,1.45708,-1.04319,-1.45426]'
Posicion recibida en el UR3, var_4
b'p[-0.195055,-0.435277,-0.0716089,1.56564,0.0461828,-0.111231]'
Posicion Final del UR3 (var_5)
b'p[-0.393863,-0.422151,-0.100453,1.56566,0.0460854,-0.111118]'
Program finish
```

Fig69. Codigo devuelto por Python con movimiento MoveJ

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveJ:

https://youtube.com/shorts/4leg2tpXhUg

- **Move P 5mm**

```
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.393877,-0.426471,-0.0566865,1.56565,0.0461755,-0.111223]'
Posicion Inicial URe
b'p[-0.55325,-0.294078,0.131622,1.45709,-1.04321,-1.45422]'
['-0.55325', '-0.294078', '0.131622', '1.45709', '-1.04321', '-1.45422']
Posiciones recibidas del URe
-0.55325
-0.294078
0.131622
Posicion del URe respecto el nuevo sistema de coordenadas
[[-0.30775 ]
 [-0.378922]
 [-0.049378]
 [ 1.      ]]
Nueva posicion del URe  respecto el nuevo sistema de coordenadas
[[-0.30775 ]
 [-0.578922]
 [-0.049378]
 [ 1.      ]]
Nueva posicion del URe respecto el  sistema de coordenadas de la base
[[-0.55325 ]
 [-0.094078]
 [ 0.131622]
 [ 1.      ]]
POSICION A ENVIAR AL URe
(-0.55325, -0.094078, 0.131622)
(1.45709, -1.04321, -1.45422)
Data sent 1 URe
Data sent 2 URe
Data sent UR3
Posicion enviada al URe
b'[3,-0.55325,-0.094078,0.131622][3,1.45709,-1.04321,-1.45422]'
Posicion 2 enviada al URe
b'p[-0.55325,-0.094078,0.131622,1.45709,-1.04321,-1.45422]'
Posicion enviada al UR3
b'[3,200,0,0]p[0.2,0,0,0,0,0]'
Posicion Final URe
b'p[-0.535752,-0.289373,0.0904153,1.45709,-1.04319,-1.45426]'
Posicion recibida en el UR3, var_4
b'p[-0.195053,-0.434776,-0.0766846,1.56565,0.0461755,-0.111223]'
Posicion Final del UR3 (var_5)
b'p[-0.388855,-0.422472,-0.0998266,1.56566,0.0460877,-0.111121]'
Program finish
```

Fig70. Codigo devuelto por Python con movimiento MoveP

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveP y radio de 5 mm:

https://youtube.com/shorts/f8AF7f6Pih4

- **Move P 2 mm**

```
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.393878,-0.426972,-0.0537248,1.56565,0.0461803,-0.111228]'
Posicion Inicial URe
b'p[-0.55325,-0.294078,0.134533,1.45709,-1.04321,-1.45422]'
['-0.55325', '-0.294078', '0.134533', '1.45709', '-1.04321', '-1.45422']
Posiciones recibidas del URe
-0.55325
-0.294078
0.134533
Posicion del URe respecto el nuevo sistema de coordenadas
[[-0.30775 ]
 [-0.378922]
 [-0.046467]
 [ 1.      ]]
Nueva posicion del URe  respecto el nuevo sistema de coordenadas
[[-0.30775 ]
 [-0.578922]
 [-0.046467]
 [ 1.      ]]
Nueva posicion del URe respecto el  sistema de coordenadas de la base
[[-0.55325 ]
 [-0.094078]
 [ 0.134533]
 [ 1.      ]]
POSICION A ENVIAR AL URe
(-0.55325, -0.094078, 0.134533)
(1.45709, -1.04321, -1.45422)
Data sent 1 URe
Data sent 2 URe
Data sent UR3
Posicion enviada al URe
b'[3,-0.55325,-0.094078,0.134533][3,1.45709,-1.04321,-1.45422]'
Posicion 2 enviada al URe
b'p[-0.55325,-0.094078,0.134533,1.45709,-1.04321,-1.45422]'
Posicion enviada al UR3
b'[3,200,0,0]p[0.2,0,0,0,0,0]'
Posicion Final URe
b'p[-0.536529,-0.292105,0.0897699,1.45709,-1.04319,-1.45426]'
Posicion recibida en el UR3, var_4
b'p[-0.195054,-0.435277,-0.0737241,1.56565,0.0461803,-0.111228]'
Posicion Final del UR3 (var_5)
b'p[-0.391852,-0.422284,-0.100178,1.56566,0.0460863,-0.111119]'
Program finish
```

Fig71. Codigo devuelto por Python con movimiento MoveP

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveP y radio de 2mm:

https://youtube.com/shorts/HQC7-ldxxCI

Desviaciones	Get_actual_tcp_pose		Get_target_tcp_pose	
	Y	Z	Y	Z
MoveL	0.01	0.02	0.0083	0.02
MoveJ	0.008	0.02	0.0083	0.02
MoveP (5mm)	0.008	0.02	0.0083	0.02
MoveP (2mm)	0.008	0.02	0.0083	0.02

Tabla 15. Comparación desviaciones de las dos funciones anteriores

Se observa en la tabla comparativa como ambas funciones son similares ya que realizan el mismo movimiento provocando la misma desviación.

1.7.6. Experimento 5: Comunicación por Modbus

En este experimento se ha probado la comunicación por Modbus propia de los UR. Primero se ha probado un movimiento coordinado con un robot UR3 y un robot UR3e. Y posteriormente un movimiento coordinado con dos robots UR3e

Para realizar la comunicación por Modbus primero se tiene que configurar el Modbus en cada robot. En el UR3 se configura el MODBUS_5 con la IP del UR3e que es la 192.168.1.50 y la dirección 128.

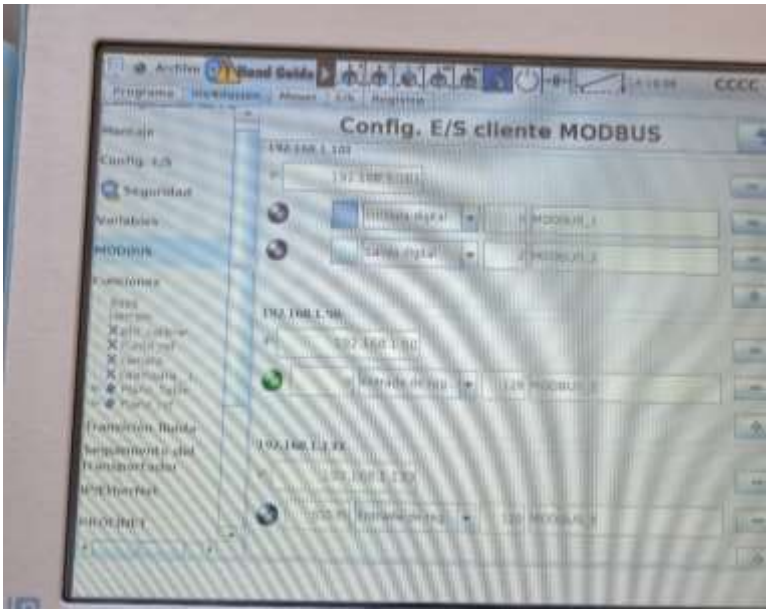


Fig72. Pantalla configurtación Modbus UR3

En el UR3e se configura el MODBUS_4 con la IP del UR3 que es la 192.168.1.152 y la dirección 128.

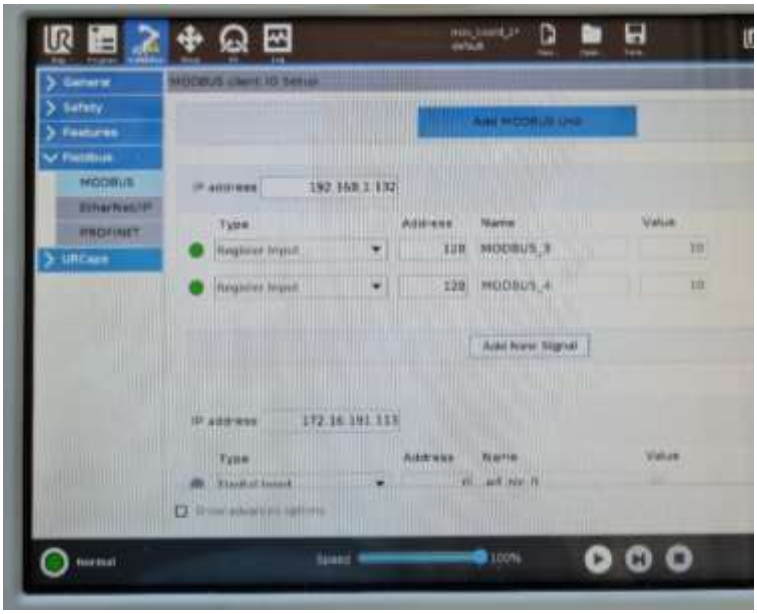


Fig73. Pantalla configuración Modbus UR3e

Posteriormente se realiza el código en el Polyscope de cada robot, las funciones que se van a utilizar son write_port_register () y la función wait.

Write_port_register escribe en uno de los puertos, a lo que también pueden acceder los clientes Modbus, tiene dos argumentos, la dirección del puerto y el valor a configurar en el puerto (del 0 al 65536 o del -32768 al 32768).

En el código del robot UR3e se programa el robot para que antes de iniciar se escriba en el puerto 128 el valor de 0, así saber el valor de ambos puertos. Una vez iniciado el programa se escribe en el puerto el valor 50, se realiza un movimiento pero antes se espera a que el Modbus4 sea de valor 10. Posteriormente se produce una espera de 1 segundo, se escribe en el puerto el valor 0 y se espera hasta que el Modbus4, es decir el otro robot también escriba un 0, el cual indicara que ha realizado el movimiento. Por último se realiza un segundo movimiento y su correspondiente espera.

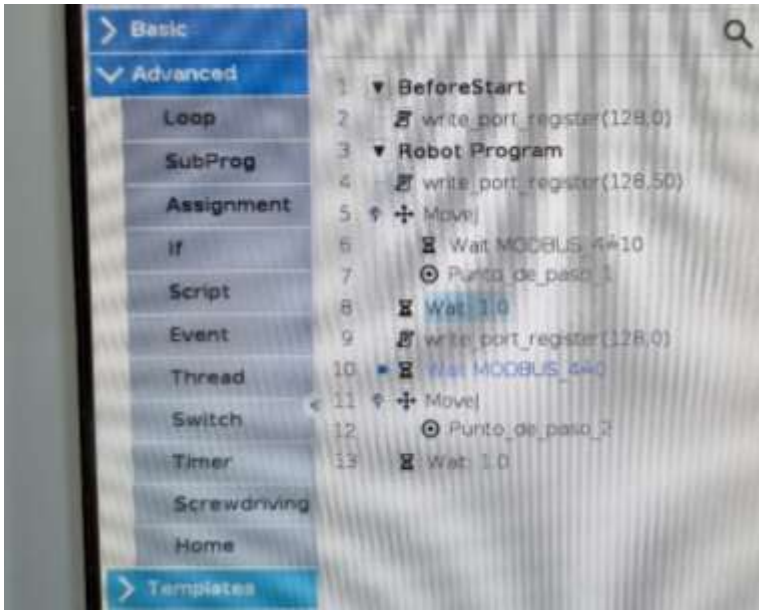


Fig74. Codigo Polyscope programa basico en UR3e

En el UR3 se realiza la misma estructura y funciones anteriores. Primero se escribe en el puerto un 0 antes de iniciar el programa. En el programa se escribe en el puerto un 10 y se realiza un movimiento antes se espera a que el Modbus5, es decir el otro robot, haya escrito un 50 en el puerto. Despues se realiza una espera y se vuelve a escribir 0 en el puerto y se espera hasta que el Modbus5 tambien valga 0, asi por ultimo se realiza un segundo movimiento.

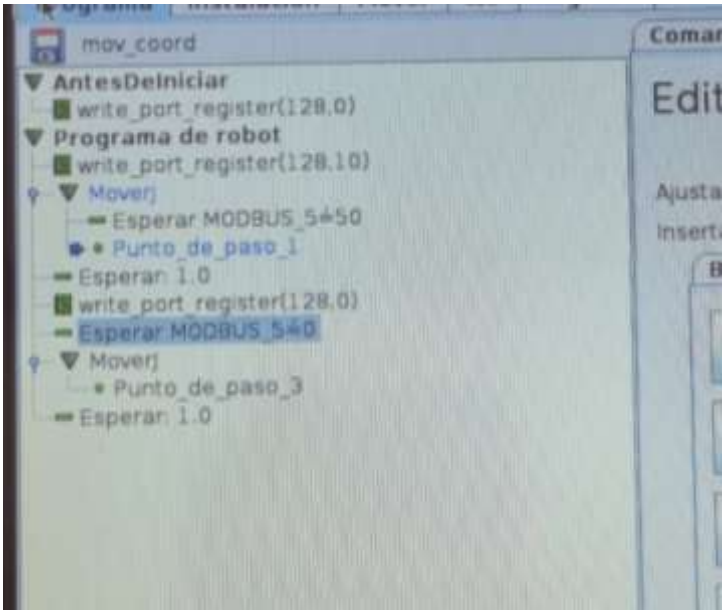


Fig75. Codigo Polyscope programa basico en UR3

En la segunda parte del experimento se a probado con dos robots UR3e, en este caso ya se realiza un codigo mas completo que implica mas movimientos.

Se configura el Modbus en el primer robot UR3e con la dirección IP del otro robot, que es la 192.168.1.49 y dirección 128.

Fig76. Pantalla configuración Modbus UR3e maestro

Se procede de la misma manera para el segundo robot UR3e donde la dirección IP del otro robot es la 192.168.1.50, y la dirección la 128.

Fig77. Pantalla configuración Modbus UR3e esclavo

Se realiza un código en ambos robots, este será un poco más complejo ya que se realizara un movimiento coordinado con un objeto. Por lo que el primer movimiento será un acercamiento a la caja colocándose en medio de esta, el segundo movimiento será el de realizar una pequeño movimiento para sujetar la caja, el tercer movimiento será el de elevación de la misma y el cuarto será el transporte de esta. El UR3e de la derecha tendrá un movimiento adicional que será el de separarse y así dejar caer la caja en la posición final. Finalmente ambos robots volverán a su posición inicial.

En la siguiente imagen se pude observar el código de polyscope desarrollado en cada robot, en el robot maestro se ha realizado el código situado a la izquierda y el esclavo el situado a la derecha.

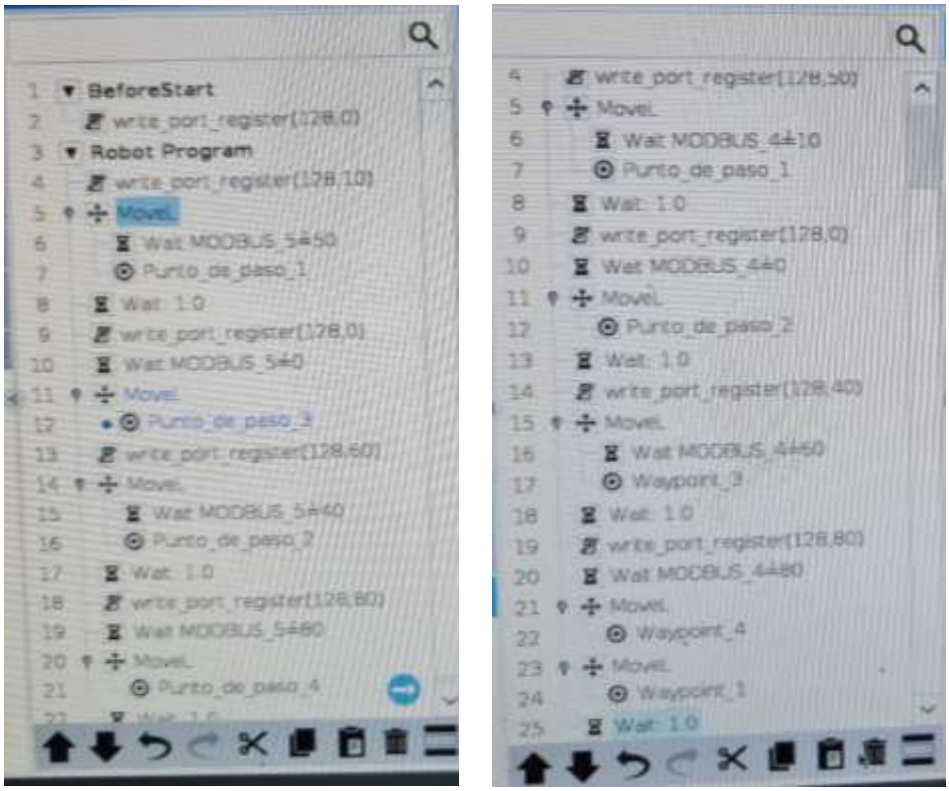


Fig78. Codigo Polyscope para movimiento coordinado con Modbus

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento utilizando la comuicación por Modbus:

https://youtube.com/shorts/PRHIVh1XYZc?feature=share

Se puede observar como realiza correctamente la trayectoria en ambos, esperando cada robot que el otro llegue a la posición indicada, para así llevar esa coordinación.

1.7.7. Experimento 6: Control bucle interior, bucle exterior

El control por bucle interior- exterior permite diseñar el sistema de control en dos etapas diferenciadas y con objetivos distintos, primero se utiliza el lazo interior que esta implementado por el fabricante del robot , es un control cerrado y limitado y hay muchas cosas que no se pueden realizar se utiliza para mover el robot siguiendo una posición o velocidad de referencia y corrigiendo los efectos de las fricciones y posteriormente se diseña el lazo exterior de control por velocidad con un regulador proporcional.

Una cosa buena es que utiliza todo lo que ha hecho el fabricante, como la dinámica y la cinemática.

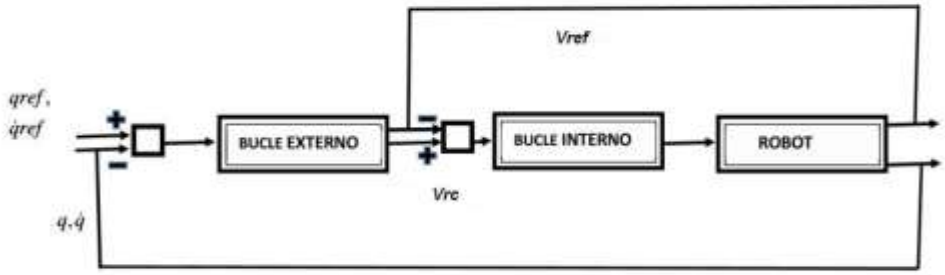


Fig79. Esquema control por bucle interior-exterior

El bucle interior, el implementado en el propio robot, no se sabe cómo puede ser, para ello se realizara posteriormente una toma de datos de un movimiento, se guardan tanto las posiciones como las velocidades de las articulaciones y así observar que generador de trayectoria utiliza.

- **Speed L**

Se realiza el siguiente experimento probando la función de speedL.

La función **speedL (xd, a, t, aRot='a'),** sirve para definir la velocidad de la herramienta. Acelera linealmente en el espacio cartesiano y continuo con una velocidad de herramienta constante. El tiempo t es opcional, si se proporciona, la función volverá después del tiempo t independientemente de que se haya alcanzado la velocidad objetivo. Si no se proporciona t, la función volverá cuando se alcance la velocidad objetivo.
xd -> velocidad de la herramienta [m/s] (vector espacial)
a -> aceleración de la posición de la herramienta [m/s^2]
t -> tiempo [s] antes de que regrese a la función (opcional)
aRot -> aceleración de la herramienta, si no se define, se utiliza la aceleración de la posición.

- **Posición**

Se realiza un primer experimento donde se tiene el código anterior realizado en Python. En él, se recibe las posiciones actuales del maestro y del esclavo después para tener el mismo sistema de coordenadas de referencia se pasa la posición del esclavo al sistema de referencia del maestro. Así se calcula el error de posición en el mismo sistema de referencia. Una vez calculado el error este es multiplicado por una matriz de ganancias obteniendo así las velocidades de referencia.
Estas velocidades, se convierten al sistema de coordenadas del esclavo para que pueda realizar el seguimiento a través de un speedL. Se comprueba que al realizar el experimento el esclavo no puede alcanzar la posición final debido a las limitaciones que puede presentar el comando speedL y que puede aparecer mayor error al realizar una doble transformación del sistema de coordenadas.

Se efectúa un segundo experimento donde se hace una sola conversión del sistema de coordenadas. Una vez recibidas las posiciones de los robots, se transforman la posición del maestro a las coordenadas de la base del esclavo.

```
A = np.array([[0, -1, 0, -0.769], [1, 0, 0, 0.094], [0, 0, 1, 0.186], [0, 0, 0, 1]])
B = np.array([[p01], [p02], [p03], [1]])

T = np.dot(A, B)
print("Posicion del UR3 respecto el nuevo sistema de coordenadas")
print(T)
p11 = (float(T[0]))
p22 = (float(T[1]))
p33 = (float(T[2]))
```

Fig80. Codigo Pyhon matriz de transformación

Se recibe las posiciones esclavo y se calcula el error:

```
print("Posiciones recibidas del URe")
print(p1)
print(p2)
print(p3)

print("Errores")
e_x = p11 - p1
e_y = p22 - p2
e_z = p33 - p3
C = np.array([[e_x], [e_y], [e_z]])
print(C)
```

Fig81. Codigo Pyhon calculo de errores

Se calcula la velocidad de referencia multiplicando el error previamente calculado por una matriz de ganancias:

```
kpx = 1
kpy = 1
kpz = 1
Kp = np.array([[kpx, 0, 0], [0, kpy, 0], [0, 0, kpz]])
Vref = np.dot(Kp, C)
print("Vref")
print(Vref)

v1 = (float(Vref[0]))
v2 = (float(Vref[1]))
v3 = (float(Vref[2]))
```

Fig82. Codigo Pyhon calculo velocidad de referencia

Por ultimo estas velocidades son pasadas a una string y luego se envían mediante la función send:

```
VEL = (v1, v2, v3)
VEL1 = str(VEL)
print("POSICION A ENVIAR AL URe")
print(VEL1)

d.send(VEL1.encode())
print("Data sent URe")
```

Fig83. Codigo Pyhon dato a enviar

Se comprueba que al fijar el Maestro en una posición, el esclavo se coloca en la misma posición al cabo de unos segundos.
También se realiza un seguimiento, donde se lleva a cabo un movimiento del maestro y se verifica que el esclavo lo sigue. Al realizar el experimento se observa que el movimiento es correcto pero hay mucho tiempo entre el movimiento del maestro y el movimiento del esclavo.

Para ello se realiza un nuevo código en Polyscope donde se utiliza los Threads para realizar un envío de datos y un recibimiento de datos continuo.
Los threads o hilos son partes del código que se ejecutan cada periodo de muestreo, estos hilos también se puede programar como subprogramas del código principal.
Cada vez que se programa un subproceso, puede usar una parte de su intervalo de tiempo ejecutando instrucciones que controlan el robot o puede ejecutar instrucciones que no controlan el robot y por lo tanto no utiliza ningún tiempo físico.

En el Polyscope del maestro se realiza un Thread el cual envía la posición cada periodo de muestreo. En el polyscope del esclavo se realizan dos Threads, uno para enviar la posición del robot y otro para recibir la velocidad de referencia que se le debe pasar al comando SpeedL.

El código del UR3e maestro seria el código de la derecha, mientras que el código del esclavo es el de la izquierda.

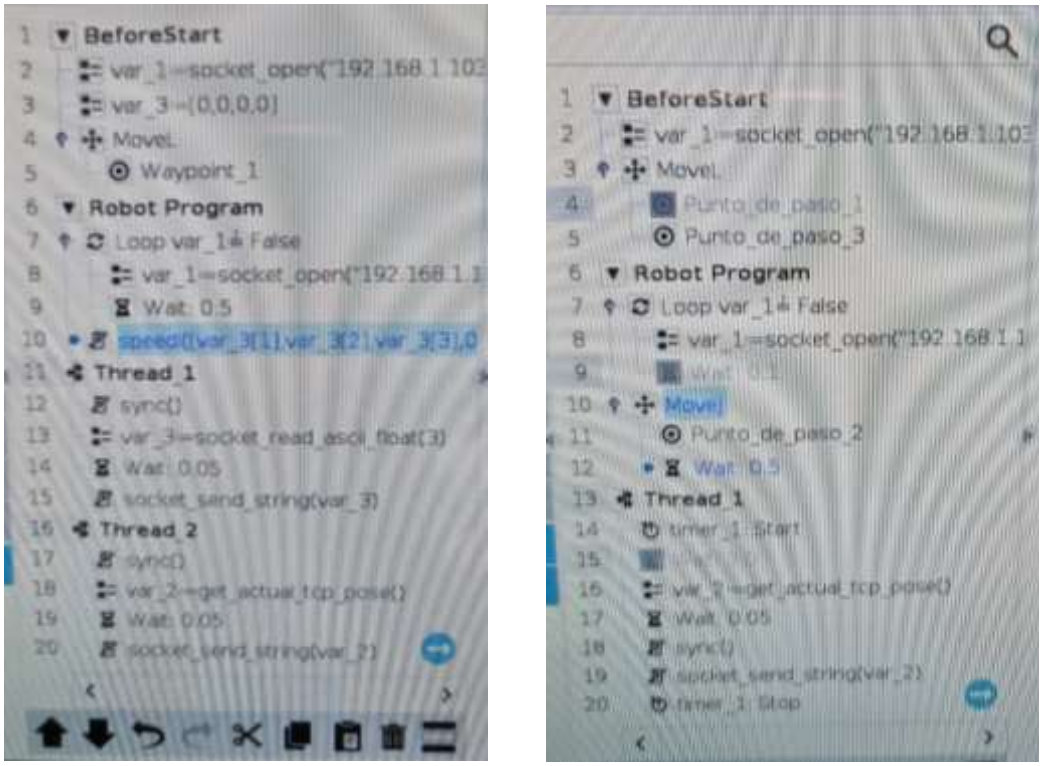


Fig84. Codigo Polyscope para el control por bucle interior-exterior

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveJ y SpeedL

https://youtube.com/shorts/ciuz5cfi8yE?feature=share

Se puede observar como realiza correctamente el seguimiento el esclavo sobre el maestro, realizando la misma trayectoria, pero tambien se observa un pequeño retraso en el. Esto es debido a que el tiempo de muestreo es alto.

Por otra parte se comprueba que el tiempo de muestreo de 50ms elegido es el correcto, ya que con un tiempo menos da problemas de comunicación
Se cerciora que se puede enviar cada 50ms ya que en el Polyscope de los robots hace falta un pequeño Wait y/o el comando sync(). Al probar con un tiempo menor se observa que hay pérdidas y solapamiento de datos al recibirlos por Python.
Se ejecuta una pequeña prueba, primero se realiza un código con solo el comando sync() **sin ningún wait (espera)**, posteriormente con 0.05s de espera y por ultimo con 0.02s de espera.

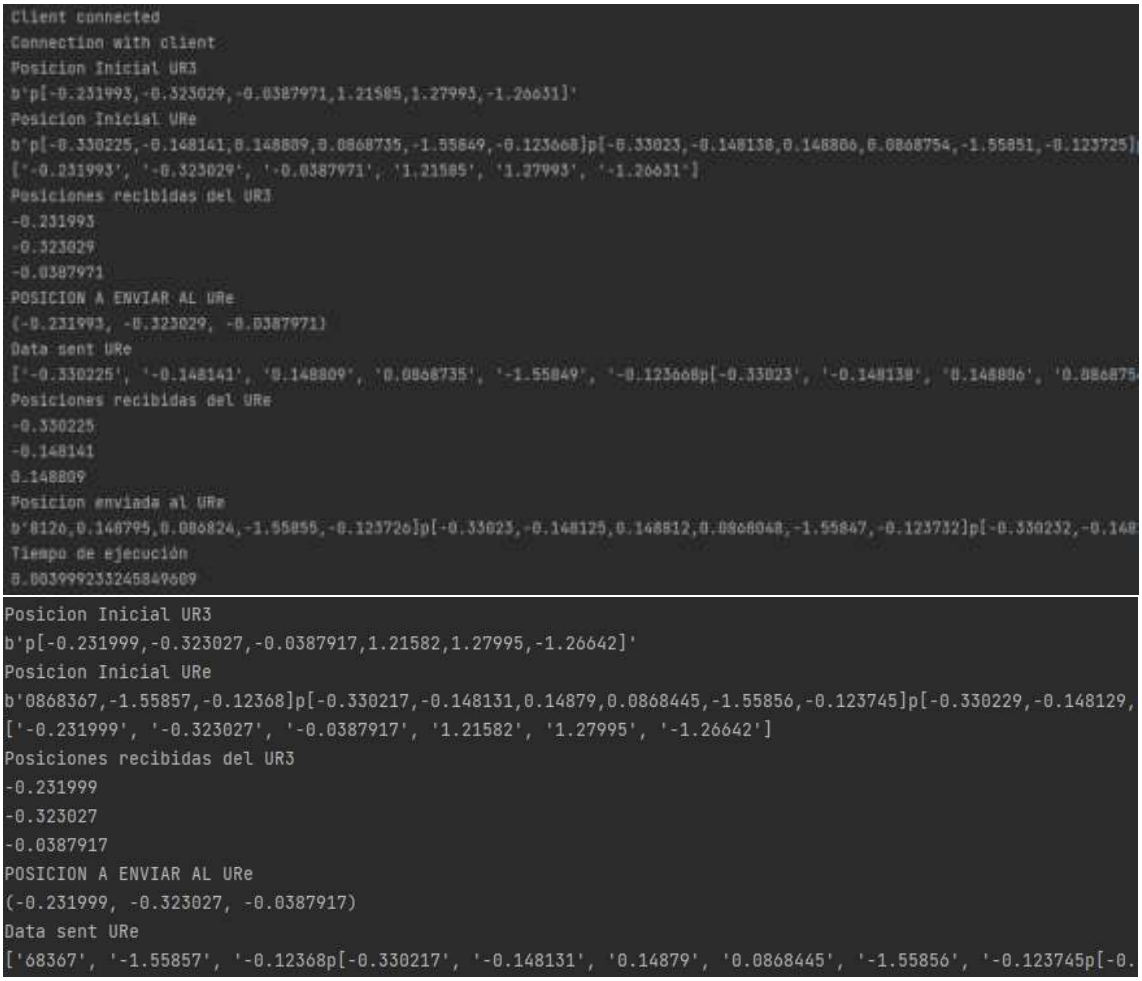


Fig85. Respuesta a la prueba con ninguna espera

Con ningún wait se puede observar como hay solapamiento y una pérdida de datos al recibir las posiciones de los robots.

Con 0.05s de espera, se puede observar como no hay solapamiento de datos ni ninguna perdida.

```
C:\RoboDK\Python37\python.exe C:/Users/RAUALSA/Desktop/Ultimo_Experimento/Prueba_tiempo.py
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.231991,-0.323019,-0.0387958,1.21582,1.27995,-1.26634]p[-0.231988,-0.323019,-0.0387975,
Posicion Inicial URe
b'p[-0.330225,-0.148151,0.148798,0.0868904,-1.55848,-0.123666]'
['-0.231991', '-0.323019', '-0.0387958', '1.21582', '1.27995', '-1.26634p[-0.231988', '-0.323
Posiciones recibidas del UR3
-0.231991
-0.323019
-0.0387958
POSICION A ENVIAR AL URe
(-0.231991, -0.323019, -0.0387958)
Data sent URe
['-0.330225', '-0.148151', '0.148798', '0.0868904', '-1.55848', '-0.123666']
Posiciones recibidas del URe
-0.330225
-0.148151
0.148798
Posicion enviada al URe
b'[3,-0.231991,-0.323019,-0.0387958]'
Tiempo de ejecución
0.05501055717468262
```

Fig86. Respuesta a la prueba con 0.05s de espera

Con 0.02 segundos de espera tampoco se observa ninguna pérdida de datos, como se puede ver en la siguiente imagen, pero este ha sido un código más reducido por lo que al tener el resto más tiempo de ejecución se ha optado por usar un tiempo de 0.05 segundos.

```
C:\RoboDK\Python37\python.exe C:/Users/RAUALSA/Desktop/Ultimo_Experimento/Prueba_tiempo.py
Starting Program
Port binded
Client connected
Connection with client
Posicion Inicial UR3
b'p[-0.231981,-0.323028,-0.038795,1.21575,1.28,-1.26632]'
Posicion Inicial URe
b'p[-0.330227,-0.148131,0.148803,0.0868437,-1.55848,-0.123714]p[-0.330221,-0.148138,0.148783
['-0.231981', '-0.323028', '-0.038795', '1.21575', '1.28', '-1.26632']
Posiciones recibidas del UR3
-0.231981
-0.323028
-0.038795
POSICION A ENVIAR AL URe
(-0.231981, -0.323028, -0.038795)
Data sent URe
['-0.330227', '-0.148131', '0.148803', '0.0868437', '-1.55848', '-0.123714p[-0.330221', '-0.
Posiciones recibidas del URe
-0.330227
-0.148131
0.148803
Posicion enviada al URe
b'[3,-0.231981,-0.323028,-0.038795]'
Tiempo de ejecución
0.01699042320251465
```

Fig87. Respuesta a la prueba con 0.02s de espera

- **Fuerza**

Se realiza una lectura de fuerzas, mediante la función get_tcp_force.
Devuelve la torsión (vector Fuerza/Par) en el TCP. La función devuelve **p [Fx(N), Fy(N), Fz(N), TRx(Nm), TRy(Nm), TRz(Nm)]** donde Fx, Fy y Fz son las fuerzas en los ejes del sistema de coordenadas de la base del robot medidas en Newton y TRx, TRy y Trz son los pares alrededor de los ejes medidos en Newton-metro. La torsión medida se compensa por las fuerzas y pares causados por la carga útil.

Esta fuerza se envía al ordenador (Python) mediante la comunicación por socket y se muestra por pantalla.

Se realiza además un control de Fuerza en el esclavo para poder sujetar el objeto, para ello se mide la fuerza en x y se establece una fuerza de referencia entre 0.5 y 1N, se calcula el error de la Fuerza en x y se multiplica por la matriz proporcional para hallar la velocidad de referencia.

Se utiliza el mismo código que en el apartado de posición, menos por una base de cambios. Dicho código, se encuentra en el Anexo1.
A traves del siguiente enlace, se puede ver el video del control de fuerza en x:
https://youtube.com/shorts/1dUjpsfcoD4?feature=share

- **Con base de datos**

La ejecución del programa no se puede realizar más rápido que unos 40-50 ms, esto puede provocar una mayor diferencia entre el maestro y el esclavo al realizar un movimiento coordinado.
Para ello se ha realizado un nuevo experimento, donde primero se guardaran las posiciones de un determinado movimiento en un archivo csv.

A continuación se encuentra el código realizado para guardar las posiciones del maestro en Polyscope:

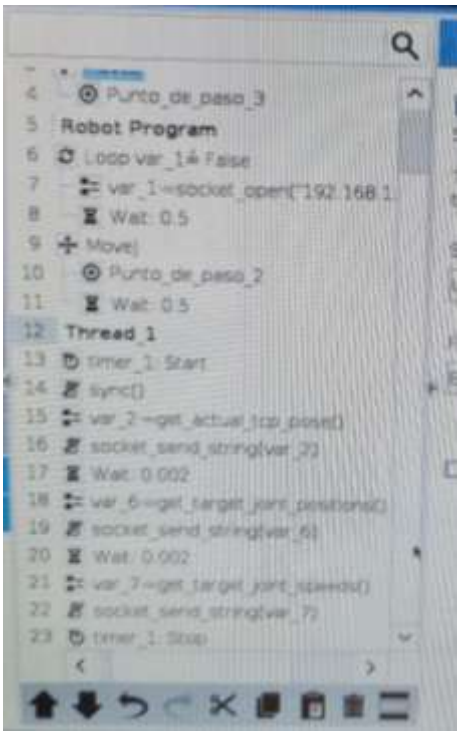


Fig88. Codigo Polyscope para guardar las posiciones

En el código anterior se envían las posiciones del robot con el **get_actual_tcp_pose**, así como también se envía las 6 posiciones y las 6 velocidad de las articulaciones con **get_Target_joint_positions()** y **get_target_joint_speed().**

El código de Python se puede ver en el Anexo 1. Se utiliza el comando with y open para abrir un fichero nuevo y con **as** se le asigna un nombre al fichero para poder escribir luego durante el programa, con el comando **archivo.write()** .

Previamente a escribir los datos en el archivo con dicha función, se les da un formato a estos mediante la función **format**. Esta función es un método que formate los valores especificados y los inserta dentro del marcador de posición de una cadena, el marcador de posición se define mediante corchetes: { }.

Una vez guardado los datos del movimiento se realiza un código de Python que tenga en cuenta tanto el valor en tiempo real del maestro como el valor guardado en la base de datos

en el instante siguiente. Con esas dos posiciones se calculara un error de ambas, dependiendo de la posición actual del esclavo y se realizara una ponderación con una K proporcional, que se podrá ir variando para cambiar el peso que se le quiere dar a cada error. Posteriormente con la suma de los dos errores multiplicada por la ganancia se podrá hallar la velocidad de referencia y pasársela al esclavo y realizar un movimiento mediante SpeedL.

El código de Polyscope realizado en cada uno de los robots es el mismo que el realizado en el apartado anterior de **Posición.**

El maestro es el que realiza un movimiento (MoveJ) y envía la posición al ordenador mediante la función ya vista, get_actual_tcp_pose. Mientras que el esclavo envía también su posición y recibe una string de datos con la velocidad de referencia que debe seguir, por lo que este dato es pasado a la función SpeedL.

A traves del siguiente enlace, se puede ver el video de la prueba del movimiento con MoveJ utilizando SpeedL y la base de datos:

https://youtube.com/shorts/ISmuM30m3E0?feature=share

1.7.8. Inversa del generador de trayectoria

Se prende realizar un experimento basándonos en la inversa del generador de trayectorias, donde primero se hallaran la **q** de las articulaciones en cada instante, teniendo en cuenta la posición en el instante actual, la aceleración y el tiempo, así como el tipo de generador de trayectoria. Posteriormente mediante las matrices de Denavit-Hartenberg hallar la posición cartesiana del efector final.

Primero se identifica el tipo de generador de trayectoria que se utiliza, para ello se realiza una prueba y se representan las posiciones, velocidades y las aceleraciones de las articulaciones.

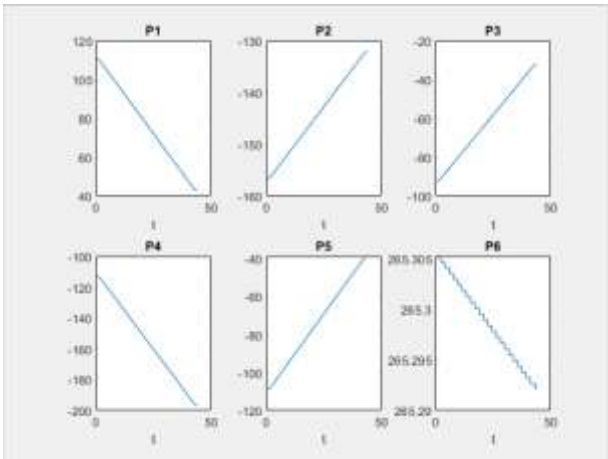


Fig89. Posición de las articulaciones

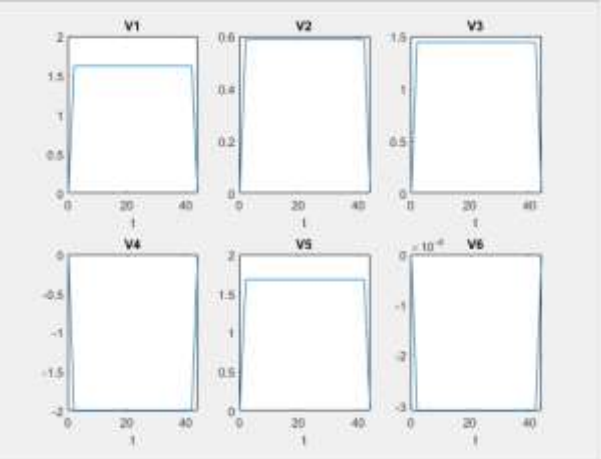


Fig90. Velocidad de las articulaciones

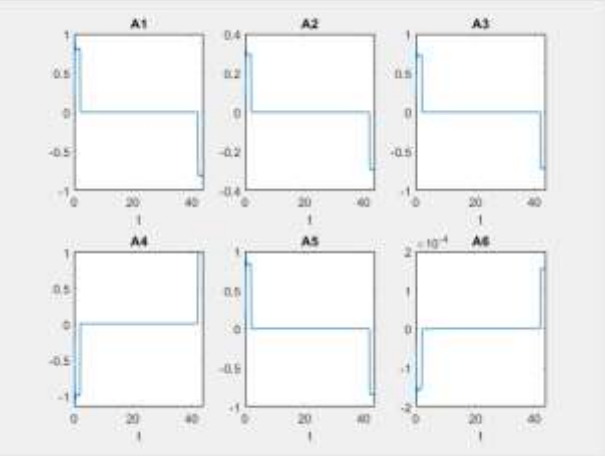


Fig91. Aceleración de las articulaciones

Como se puede comprobar, se tiene una trayectoria trapezoidal, su velocidad tiene 3 etapas diferenciadas con la fase de aceleración, la constante y la desaceleración.

La aceleración se observa como empieza en su aceleración máxima, después de vuelve 0 y por ultimo una aceleración máxima negativa.

Se realiza un experimento para poder hallar los tiempos en los que la trayectoria cambia de fase de aceleración a constante (t1) y de constante a la fase de desaceleración (t2), así como observar que articulación alcanza la velocidad máxima establecida.

- Velocidad de 2°/s y Aceleración de 2°/s^2

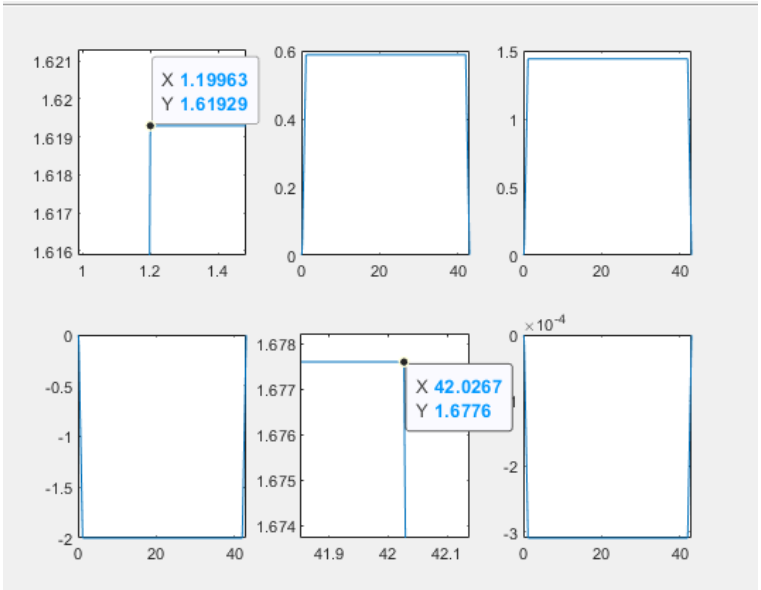


Fig92. Velocidad de las articulaciones

En este experimento como se puede observar ninguna articulación alcanza la velocidad de 2°/s, las que más se aproximan son la primera y la quinta con una velocidad de 1.6 °/s En la imagen se puede observar como el valor de t1 es de 1.99 segundos y el valor de t2 de 42.02 segundos.

- Velocidad de 5°/s y Aceleración de 5°/s

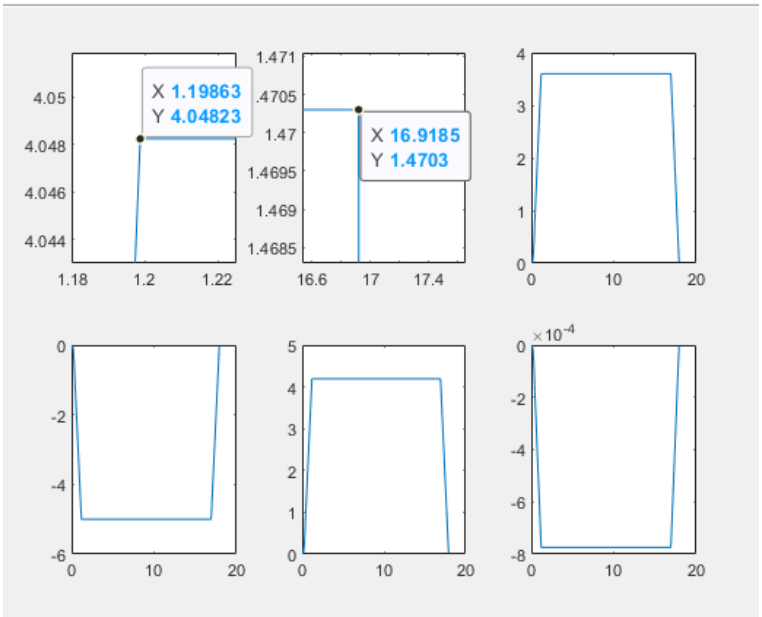


Fig93. Velocidad de las articulaciones

En este experimento como se puede observar ninguna articulación alcanza la velocidad de 5°/s, las que más se aproximan son la primera y la quinta con una velocidad de 4.04°/s En la imagen se puede observar como el valor de t1 es de 1.98 segundos y el valor de t2 de 16.91 segundos.

En estos dos experimentos se ha podido ver que la aceleración no se corresponde ya que el tiempo es diferente. El tiempo de la fase de aceleración a la fase de reposo debería de dar lo mismo que dividir entre la velocidad y la aceleración, en estos dos casos el valor debería haber dado 1.

Por ello se calcula la aceleración real, para el primer caso $a = \frac{V}{t} = \frac{2}{1.199} = 1.66\ °/s^2$

Para el segundo caso $a = \frac{V}{t} = \frac{5}{1.198} = 4.173\ °/s^2$

A continuación se ejecuta otra prueba para comprobar el tiempo 2 con la estimación, así como también el tiempo 1 haciendo cada vez más grande su valor y observando si existe una correlación.

- Velocidad de 2°/s y Aceleración de 1°/s^2

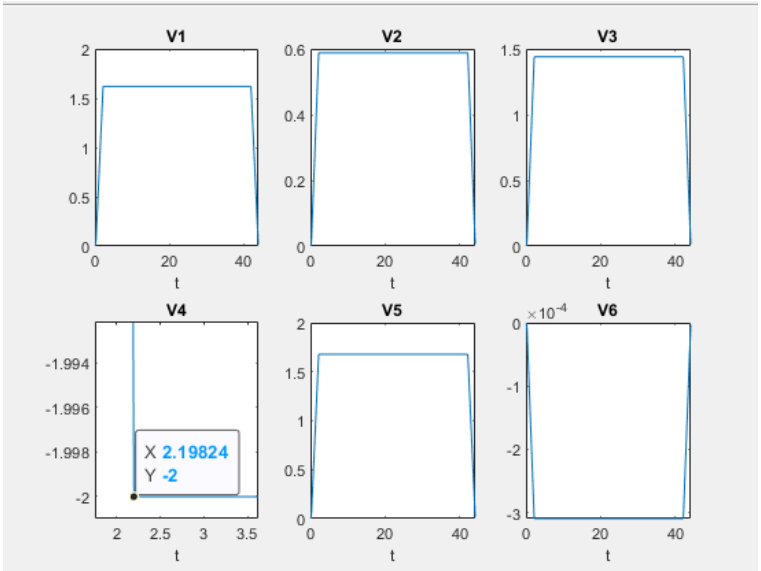


Fig94. Velocidad de las articulaciones

En este caso el valor de velocidad máxima es alcanzado por la articulación 4, por lo que se observa en ella el valor de los tiempos. El tiempo t1 alcanza un valor de 2.198 segundos.

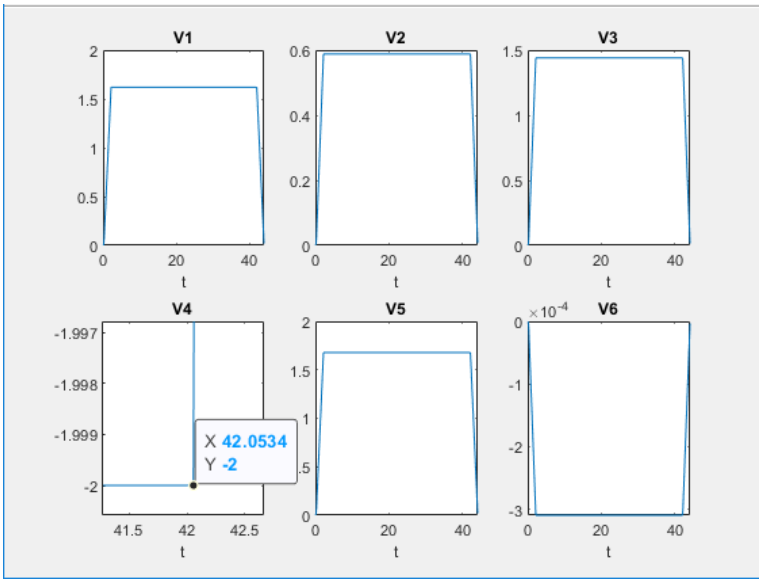


Fig95. Velocidad de las articulaciones

El tiempo t2 que se alcanza es un valor de 42.0534 segundos.

A continuación se representan las posiciones para poder calcular el tiempo 2 de forma analítica. El tiempo t2 se calcula realizando la división entre la resta de la posición final e inicial y la velocidad.

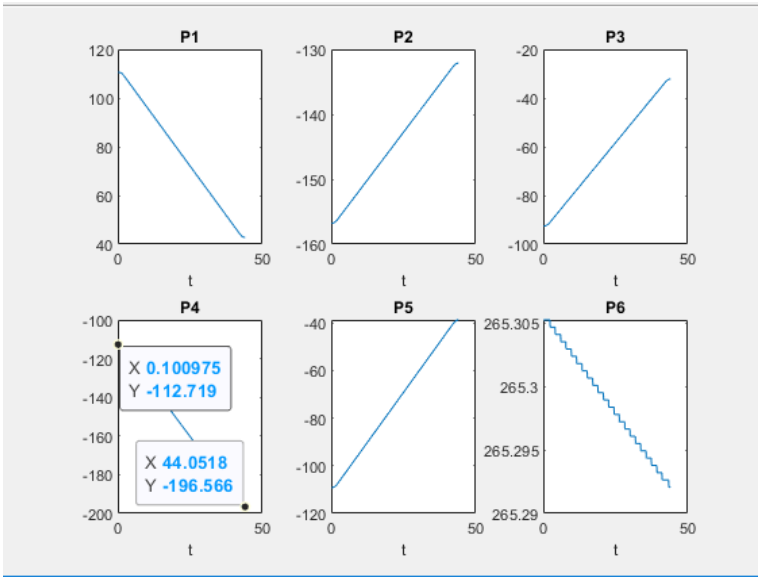


Fig96. Posición de las articulaciones

$$|t2| = \left|\frac{P2 - P1}{V}\right| = \frac{-196.566 - (-112.719)}{2} = 41.923\ s$$

$$a = \frac{V}{t} = \frac{2}{2.1982} = 0.909\ °/s^2$$

- Velocidad de 6°/s y Aceleración de 2°/s^2

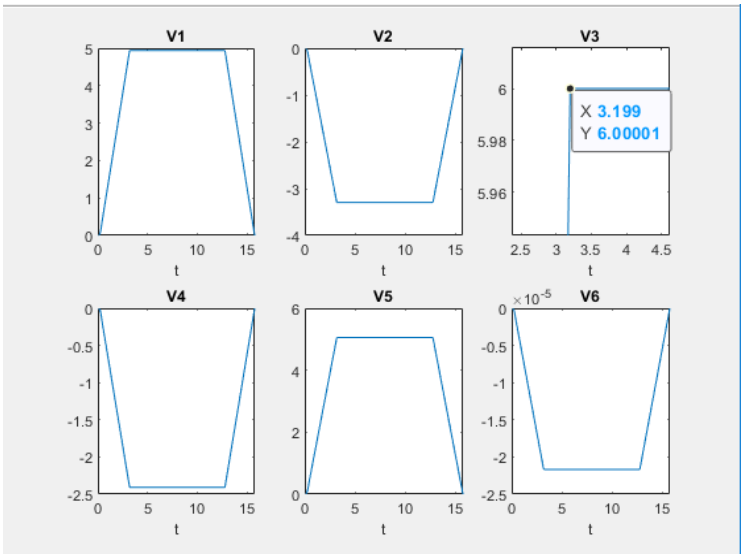


Fig97. Velocidad de las articulaciones

En este caso el valor de velocidad máxima es alcanzado por la articulación 3, por lo que se observa en ella el valor de los tiempos. El tiempo t1 alcanza un valor de 3.199 segundos.

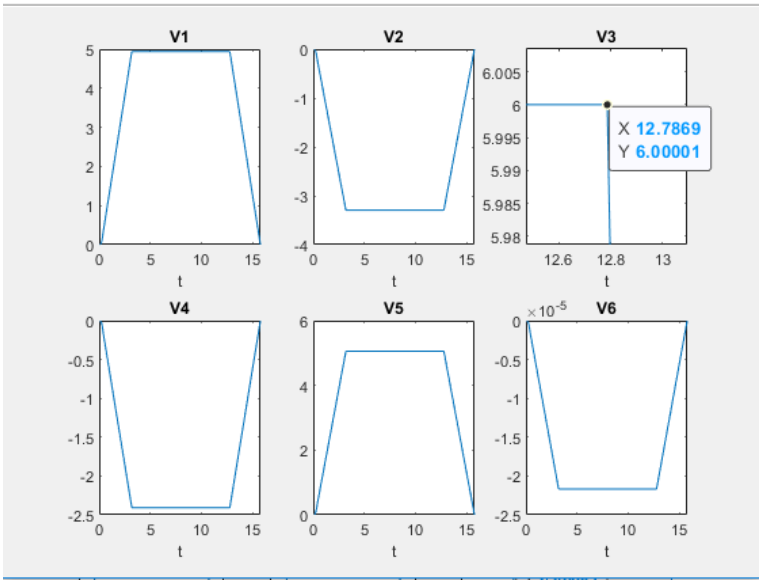


Fig98. Velocidad de las articulaciones

El tiempo t2 que se alcanza es un valor de 12.786 segundos.

A continuación se representan las posiciones para poder calcular el tiempo 2 de forma analítica:

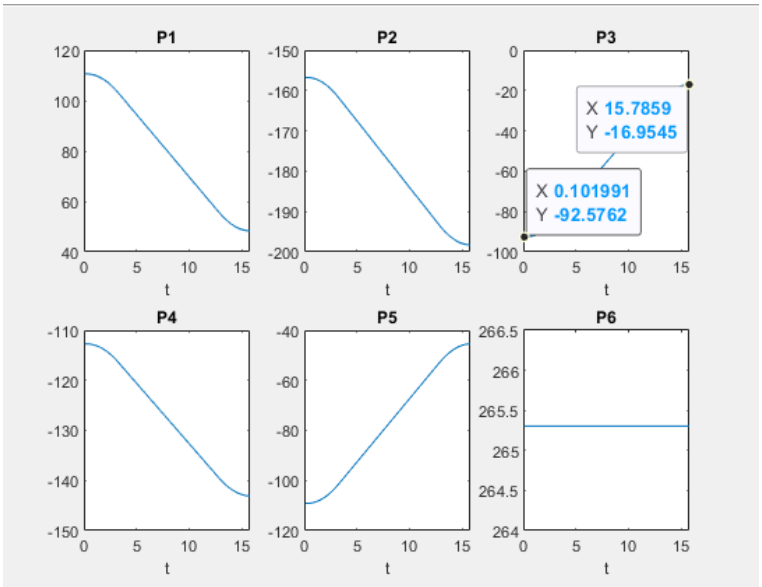


Fig99. Posición de las articulaciones

$$|t2| = \left|\frac{P2 - P1}{V}\right| = \frac{-16.9545 - (-92.5762)}{6} = 12.603\ s$$

$$a = \frac{V}{t} = \frac{6}{3.199} = 1.875\ °/s^2$$

- Velocidad de 8°/s y Aceleración de 2°/s²

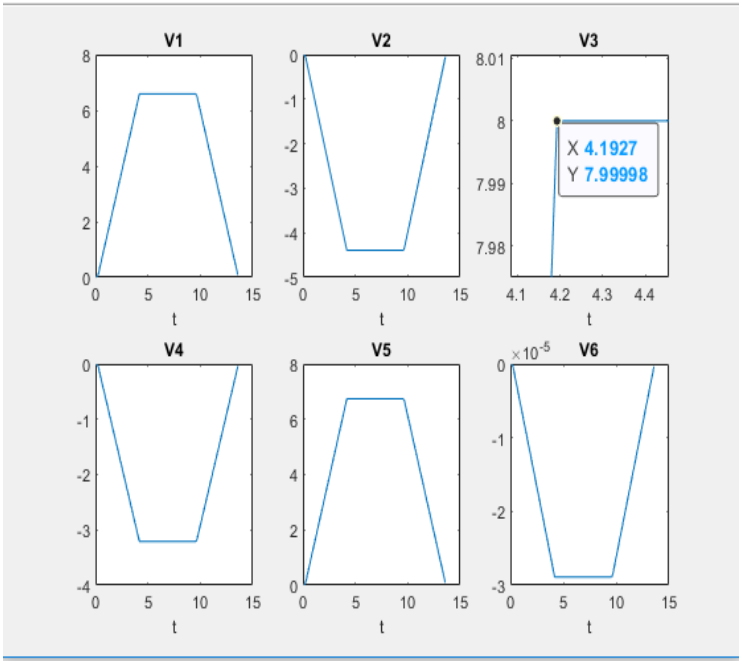


Fig100. Velocidad de las articulaciones

En este caso el valor de velocidad máxima es alcanzado por la articulación 3, por lo que se observa en ella el valor de los tiempos. El tiempo t1 alcanza un valor de 4.1927 segundos.

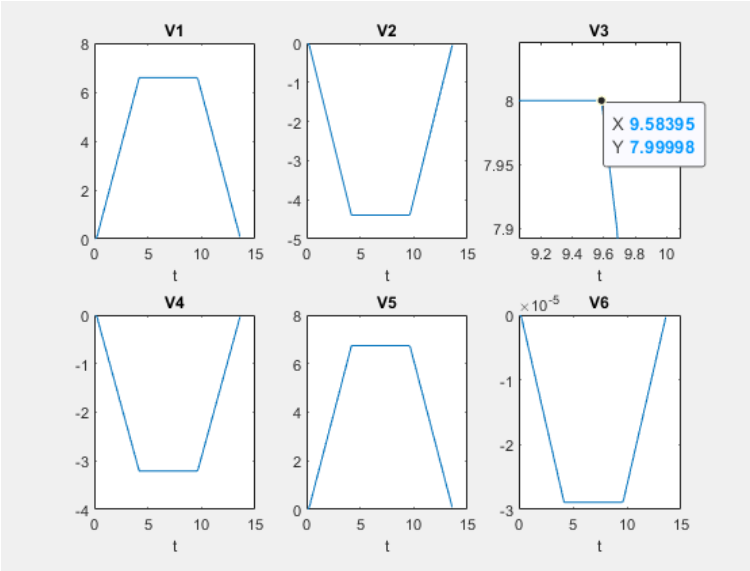


Fig101. Velocidad de las articulaciones

El tiempo t2 que se alcanza es un valor de 9.5839 segundos.

A continuación se representan las posiciones para poder calcular el tiempo 2 de forma analítica:

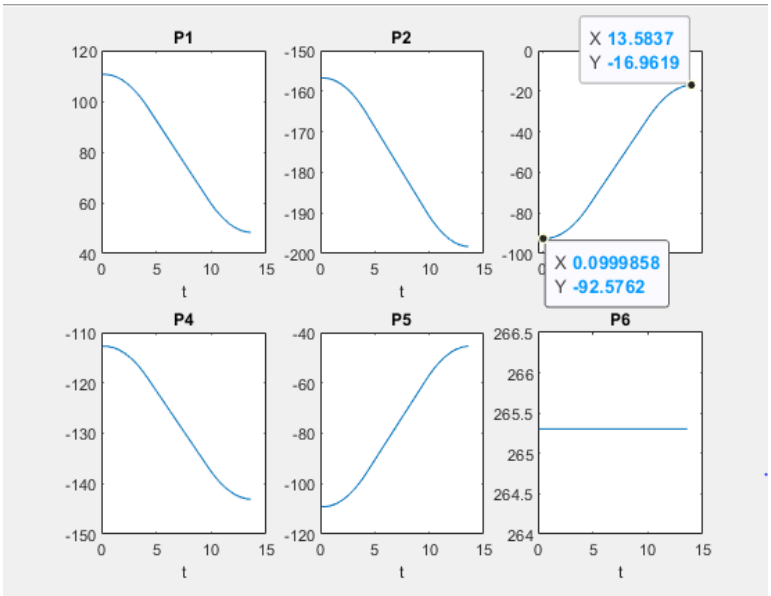


Fig102. Posición de las articulaciones

$$|t2| = \left|\frac{P2 - P1}{V}\right| = \frac{-16.9619 - (-92.5762)}{8} = 9.451\ s$$

$$a = \frac{V}{t} = \frac{8}{4.1927} = 1.908\ °/s^2$$

- Velocidad de 10°/s y Aceleración de 2°/s²

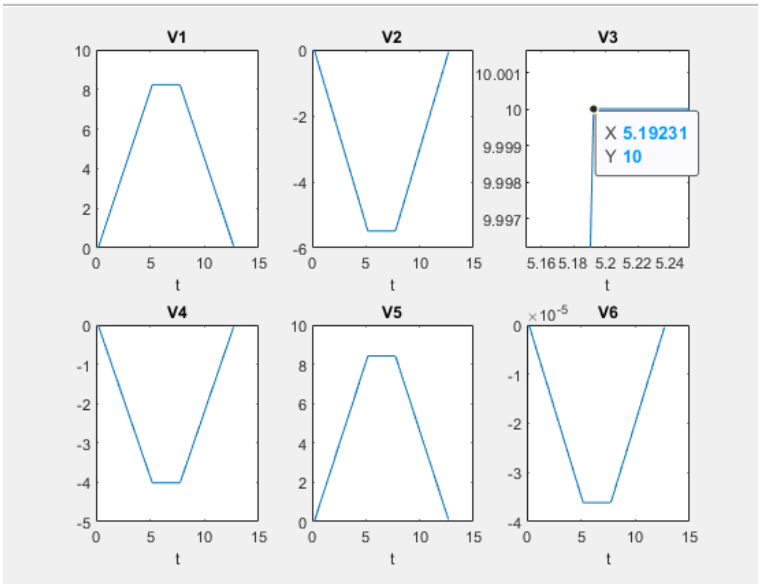


Fig103. Velocidad de las articulaciones

En este caso el valor de velocidad máxima es alcanzado por la articulación 3, por lo que se observa en ella el valor de los tiempos. El tiempo t1 alcanza un valor de 5.1923 segundos.

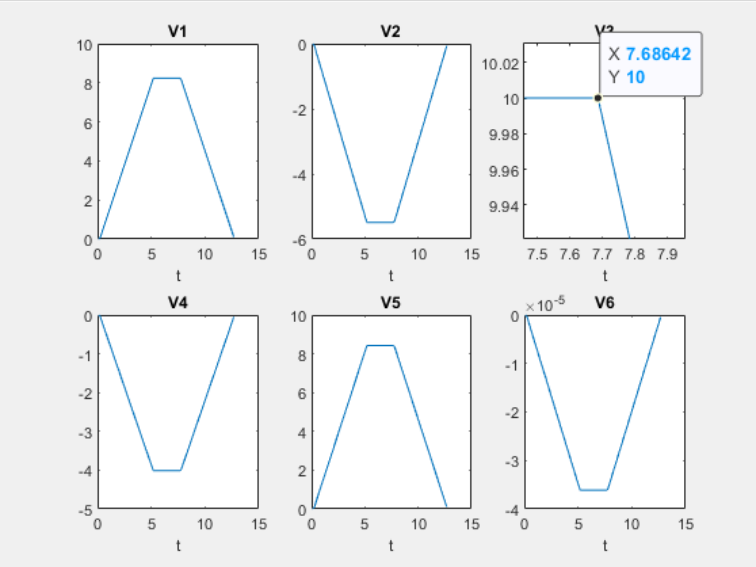


Fig104. Velocidad de las articulaciones

El tiempo t2 que se alcanza es un valor de 7.686 segundos.

A continuación se representan las posiciones para poder calcular el tiempo 2 de forma analítica:

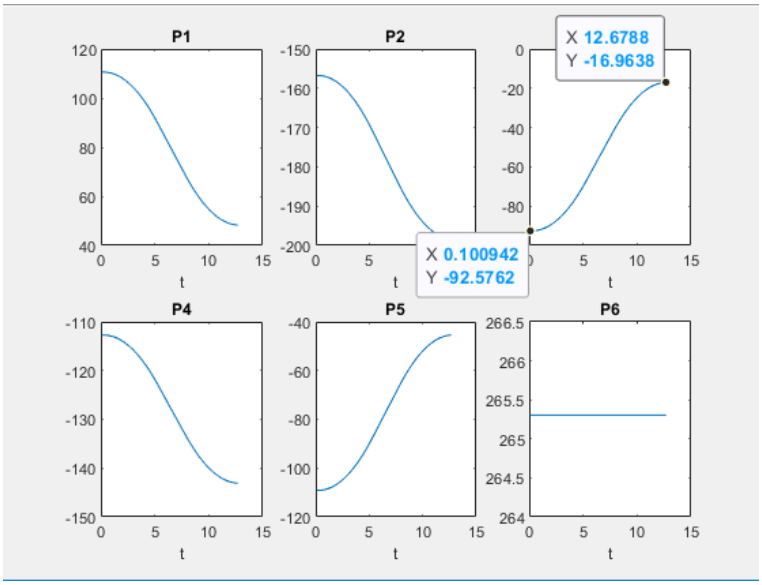


Fig105. Posición de las articulaciones

$$|t2| = \left|\frac{P2 - P1}{V}\right| = \frac{-16.9638 - (-92.5762)}{10} = 7.561\ s$$

$$a = \frac{V}{t} = \frac{10}{5.1923} = 1.925\ °/s^2$$

	T1 calculado (s)	T1 prueba (s)	**Desv** (s)	T2 calculado (s)	T2 prueba (s)	**Desv** (s)	Ac teórica (°/s²)	Ac calculada (°/s²)	**Desv** (°/s²)
Vel 2°/s Ac 1°/s²	2	2.198	**0.198**	41.923	42.053	**0.13**	1	0.909	**0.091**
Vel 6°/s Ac 2°/s²	3	3.199	**0.199**	12.603	12.786	**0.183**	2	1.875	**0.125**
Vel 8°/s Ac 2°/s²	4	4.193	**0.193**	9.451	9.583	**0.132**	2	1.908	**0.092**
Vel10°/s Ac 2°/s²	5	5.192	**0.192**	7.561	7.686	**0.125**	2	1.925	**0.075**
Promedio			**0.1955**			**0.1425**			**0.0957**

Tabla 16. Comparación del tiempo 1, del tiempo 2 y de la aceleración con sus desviaciones

Se ha visto que variando las velocidades y aceleraciones, la aceleración es menor que se da en la asignación, por lo que se ha calculado la aceleración real ejercida.

Además t1 siempre tardar entre 0.19 y 0.20 segundos más de lo que debería y que el tiempo t2 también tarda una media de 0.1425 segundos más.

Se ha deducido que hacer la inversa del generador de trayectorias no es viable ya que la aceleración no es la misma que la fijada y es diferente para diferentes movimientos.

1.8. Conclusiones

En este trabajo se ha aprendido a utilizar la interfaz de los robots UR, configurando los robots y programándolos. Así como un nuevo lenguaje de programación hasta ahora visto como es Python, para realiza la comunicación por el PC, realizando un código desde cero. Así como a solucionar o adaptarse a los problemas que han ido surgiendo durante el desarrollo del trabajo como el fallo por rotura del sensor externo de fuerza y par, lo que supuso que ya no se pudiera realizar más experimentos con él, los problemas con el tiempo de muestreo a la hora de enviar y recibir datos, el cambio del robot UR3 por el robot UR3e en el laboratorio.

En conclusión, se ha podido comprobar primero el correcto funcionamiento de la comunicación por Sockets realizando envió y recibimiento de datos entre los robots y el PC. Después se ha realizado diferentes movimientos con MoveL, MoveJ y MoveP comprobado que con MoveL y MoveP se puede realizar correctamente movimientos coordinados con ambos robots, aunque hay desviación en z. Con MoveJ se ha observado que este movimiento realiza trayectorias curvas diferentes en cada uno de los robots, por lo que por sí solo no es correcto utilizarlo para la coordinación de dos robots.

Para poder realizar una coordinación de ambos robots con MoveJ se ha realizado el control por bucle interior-exterior, donde utilizando el comando SpeedL se realiza el seguimiento del esclavo sobre el maestro. Así al realizar el experimento se comprueba que el tiempo de muestreo a la hora de recibir y enviar datos es demasiado grande, 50 ms, por lo que hay una mayor diferencia entre los dos robots.

Para solucionar eso se realiza un experimento con una base de datos, donde el esclavo recibe la posición actual del maestro y la del instante siguiente, por lo que se observa realizando el experimento como esta diferencia entre los dos ya no existe, y se realiza correctamente el seguimiento.

Además de realizar un experimento de comunicación por Modbus de ambos robots, sin necesidad de utilizar el PC, donde se ha comprobado el buen seguimiento de trayectoria de los robots.

Así se puede concluir que con todos los movimientos se ha podido realizar un control de ambos robots que funcionen de manera coordinada, realizando pruebas con objetos, y con diferentes tipos de comunicación.

1.9. Bibliografía

[1] «IRB 14000 YUMI - ROBOT COLABORATIVO», *Robotics*. https://new.abb.com/products/robotics/es/robots-colaborativos/yumi (accedido 8 de agosto de 2022).
[2] «El innovador robot YuMi® de ABB ha establecido nuevos estándares para la robótica colaborativa durante cinco años», *News*. https://new.abb.com/news/es/detail/63931/el-innovador-robot-yumir-de-abb-ha-establecido-nuevos-estandares-para-la-robotica-colaborativa-durante-cinco-anos (accedido 8 de agosto de 2022).
[3] «YuMi® - IRB 14000 - Robot articulado by ABB Robotics | DirectIndustry». https://www.directindustry.es/prod/abb-robotics/product-30265-1635187.html (accedido 8 de agosto de 2022).
[4] «PhantomX Pincher», *ROS Components*. http://www.roscomponents.com/es/brazos-roboticos/99-phantomx-pincher.html (accedido 8 de agosto de 2022).
[5] «TAZ-TFG-2019-3511.pdf». Accedido: 8 de agosto de 2022. [En línea]. Disponible en: https://zaguan.unizar.es/record/84981/files/TAZ-TFG-2019-3511.pdf
[6] J. A. Castro-Vargas, B. S. Zapata-Impata, P. Gil, y J. Pomares, «AGARRE BIMANUAL DE OBJETOS ASISTIDO POR VISION», p. 8.
[7] L. F. González-Böhme, R. García-Alvarado, F. J. Quitral-Zapata, y E. A. Valenzuela-Astudillo, «SISCOM: Cooperative Multi-Robot Systems in Construction», en *Blucher Design Proceedings*, Medellín, Colombia, dic. 2020, pp. 349-356. doi: 10.5151/sigradi2020-48.
[8] «"Cobots". los nuevos robots colaborativos - Iberdrola». https://www.iberdrola.com/innovacion/cobots-robots-colaborativos (accedido 16 de agosto de 2022).
[9] «▷Todo sobre los Robots Colaborativos | Marcas y Precios». https://www.neobotik.com/robots-colaborativos/#Que_son_los_robots_colaborativos (accedido 16 de agosto de 2022).
[10] «UN COBOT ES SEGURO POR SÍ MISMO, PERO ¿Y LA APLICACIÓN DONDE SE INTEGRA? | CADE Cobots», 12 de noviembre de 2019. https://cadecobots.com/un-cobot-es-seguro-por-si-mismo-pero-y-la-aplicacion-donde-se-integra/ (accedido 16 de agosto de 2022).
[11] «Conozca las cinco ventajas clave de la gama e-Series». https://blog.universal-robots.com/es/ventajas-e-series (accedido 16 de agosto de 2022).
[12] E. Universitat Politècnica de València, «Universitat Politècnica de València», *Ing. Agua*, vol. 18, n.º 1, p. ix, sep. 2014, doi: 10.4995/ia.2014.3293.
[13] «Modbus: Qué es y cómo funciona | Comunicaciones Industriales», *aula21 | Formación para la Industria*, 5 de mayo de 2020. https://www.cursosaula21.com/modbus-que-es-y-como-funciona/ (accedido 16 de agosto de 2022).
[14] «Modbus TCP/IP – Automation Networks». http://automation-networks.es/glossary/modbus-tcpip (accedido 16 de agosto de 2022).
[15] «UN COBOT ES SEGURO POR SÍ MISMO, PERO ¿Y LA APLICACIÓN DONDE SE INTEGRA? | CADE Cobots», 12 de noviembre de 2019. https://cadecobots.com/un-cobot-es-seguro-por-si-mismo-pero-y-la-aplicacion-donde-se-integra/ (accedido 14 de septiembre de 2022).

[16] «practica5.pdf». Accedido: 14 de septiembre de 2022. [En línea]. Disponible en: https://nbio.umh.es/files/2012/04/practica5.pdf
[17] «Move». http://help.universal-robots.com/SW_5_11/UR5e/Content/from-tex/software_manual/programming/commandtab_move_en.htm (accedido 14 de septiembre de 2022).
[18] «Qué es un lenguaje de programación», *OpenWebinars.net*, 16 de julio de 2020. https://openwebinars.net/blog/que-es-un-lenguaje-de-programacion/ (accedido 18 de agosto de 2022).
[19] «Los lenguajes de programación más usados en la actualidad». https://www.universia.net/es/actualidad/empleo/lenguajes-programacion-mas-usados-actualidad-1136443.html (accedido 18 de agosto de 2022).
[20] «El lenguaje C++ — Fundamentos de Programación en C++». https://www2.eii.uva.es/fund_inf/cpp/temas/1_introduccion/introduccion.html (accedido 18 de agosto de 2022).
[21] «Qué es C++: Características y aplicaciones», *OpenWebinars.net*, 22 de julio de 2019. https://openwebinars.net/blog/que-es-cpp/ (accedido 18 de agosto de 2022).
[22] «¿Que es Java? - Guía de Java empresarial para principiantes - AWS», *Amazon Web Services, Inc.* https://aws.amazon.com/es/what-is/java/ (accedido 18 de agosto de 2022).
[23] «Nuestro sensor de fuerza/par HEX para brazos robóticos hace que la automatización sea sencilla | OnRobot». https://onrobot.com/es/productos/sensor-de-fuerza-par-hex-de-6-ejes (accedido 14 de septiembre de 2022).
[24] «onrobot-compute-box-description_e10_en.pdf». Accedido: 14 de septiembre de 2022. [En línea]. Disponible en: https://onrobot.com/sites/default/files/documents/onrobot-compute-box-description_e10_en.pdf
[25] A. M. Téllez, «La matriz generadora de rotación», *La Mecánica Cuántica*, 11 de agosto de 2009. http://la-mecanica-cuantica.blogspot.com/2009/08/la-matriz-generadora-de-rotacion.html (accedido 14 de septiembre de 2022).

UNIVERSIDAD POLITECNICA DE VALENCIA

Dpto. de Ingeniería de Sistemas y Automática

Control distribuido del robot colaborativo UR3

Anexo 1

TRABAJO DE FIN DE MASTER

Master Universitario en Automática e Informática Industrial

AUTOR/A: Alfonso Safont, Raúl

Tutor/a: Zotovic Stanisic, Ranko

Curso Académico: 2021-2022

- Código movimiento individual UR3, experimento 1.

```
HOST = "192.168.1.103"# The remote host
PORT = 30000 # The same port as used by the server
print("Starting Program")
i = 0

while i<1:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT)) # Bind to the port
    print("Port binded")
    s.listen(5) # Now wait for client connection.
    print("Client connected")
    c, addr = s.accept() # Establish connection with client.
    print("Connection with client")
    try:
        msg = c.recv(1024)
        print("Posición Inicial")
        print(msg)
        time.sleep(2)
        i = i + 1


        pos = "(260,0,0)" #(Y,X,Z,0,0,0)
        c.send(pos.encode())
        print("Data sent")

        msg2 = c.recv(1024)
        print("Posicion enviada al UR3")
        print(msg2)
        time.sleep(1)

        msg3 = c.recv(1024)
        print("Posicion recibida en el UR3, var_4")
        print(msg3)
        time.sleep(1)
            msg4 = c.recv(1024)
            print("Posicion Final del UR3 (var_5)")
            print(msg4)
            time.sleep(1)


    except socket.error as socketerror:
            print("Error")

c.close()
s.close()
print("Program finish")
```

- Código movimiento individual UR3e, experimento 1.

```
HOST = "192.168.1.103"# The remote host
PORT = 30000 # The same port as used by the server
print("Starting Program")
i = 0

while i<1:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT)) # Bind to the port
    print("Port binded")
    s.listen(5) # Now wait for client connection.
    print("Client connected")
    c, addr = s.accept() # Establish connection with client.
    print("Connection with client")
    try:
        msg = c.recv(1024)
        print("Posición Inicial")
        print(msg)

        pos2 = msg.decode()
        pos2 = pos2[2:len(pos2)]
        b = pos2.replace("b'p[", "").replace("]", "")
        lst = [x for x in b.split(',')]
        print(lst)

        p1 = (float(lst[0]))
        p2 = (float(lst[1]))
        p3 = (float(lst[2]))
        r1 = (float(lst[3]))
        r2 = (float(lst[4]))
        r3 = (float(lst[5]))
        print("Posiciones recibidas")
        print(p1)
        print(p2)
        print(p3)
A = np.array([[0, 1, 0, -0.0994], [-1, 0, 0, -0.878], [0, 0, 1, -0.206], [0, 0, 0, 1]])
B = np.array([[p1], [p2], [p3], [1]])
T = np.dot(A,B)
print("Posicion respecto el nuevo sistema de coordenadas")
print(T)

time.sleep(2)
i = i + 1
pos = np.array([[0.200], [0], [0], [0]]) #SE INDICA EL DESPLAZAMIENTO QUE SE QUIERE
# REALIZAR  #X,Y,Z,-;
P1=T+pos
print("Nueva posicion respecto el nuevo sistema de coordenadas")
print(P1)


C = np.array([[0, -1, 0, -0.878], [1, 0, 0, 0.0994], [0, 0, 1, 0.206], [0, 0, 0, 1]])
T2 = np.dot(C,P1)
print("Nueva posicion respecto el  sistema de coordenadas de la base")
print(T2)
```

```
            p1 = ((float(T2[0])))
            p2 = ((float(T2[1])))
            p3 = ((float(T2[2])))

            POS=(p1,p2,p3)
            POS1=str(POS)
            print("POSICION A ENVIAR")
            print(POS1)

            POS2=(r1, r2, r3)
            POS3 = str(POS2)
            print(POS3)


            c.send(POS1.encode())
            print("Data sent")

            c.send(POS3.encode())
            print("Data sent 2")

            msg2 = c.recv(1024)
            print("Posicion enviada al UR3")
            print(msg2)
            #time.sleep(1)


            msg3 = c.recv(1024)
            print("Posicion 2 enviada al UR3")
            print(msg3)
            #time.sleep(1)
            msg4 = c.recv(1024)
            print("Posicion Final UR3")
            print(msg4)
            #time.sleep(1)

            print("")
            #time.sleep(0.5)

        except socket.error as socketerror:
            print("Error")

    c.close()
    s.close()
    print("Program finish")
```

- Código Movimiento Coordinado, experimento 2

```
HOST = "192.168.1.103" # The remote host
PORT = 30000 # The same port as used by the server #Robot UR3
PORT2 = 40000 # The same port as used by the server #Robot URe
print("Starting Program")
i = 0

while i<1:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Robot UR3
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT)) # Bind to the port

    r = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  #Robot URe
    r.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    r.bind((HOST, PORT2))  # Bind to the port

    print("Port binded")
    s.listen(5) # Now wait for client connection.
    r.listen(5)  # Now wait for client connection.
    print("Client connected")
    c, addr = s.accept() # Establish connection with client. #Robot UR3
    d, addr = r.accept()                                     #Robot URe
    print("Connection with client")
```

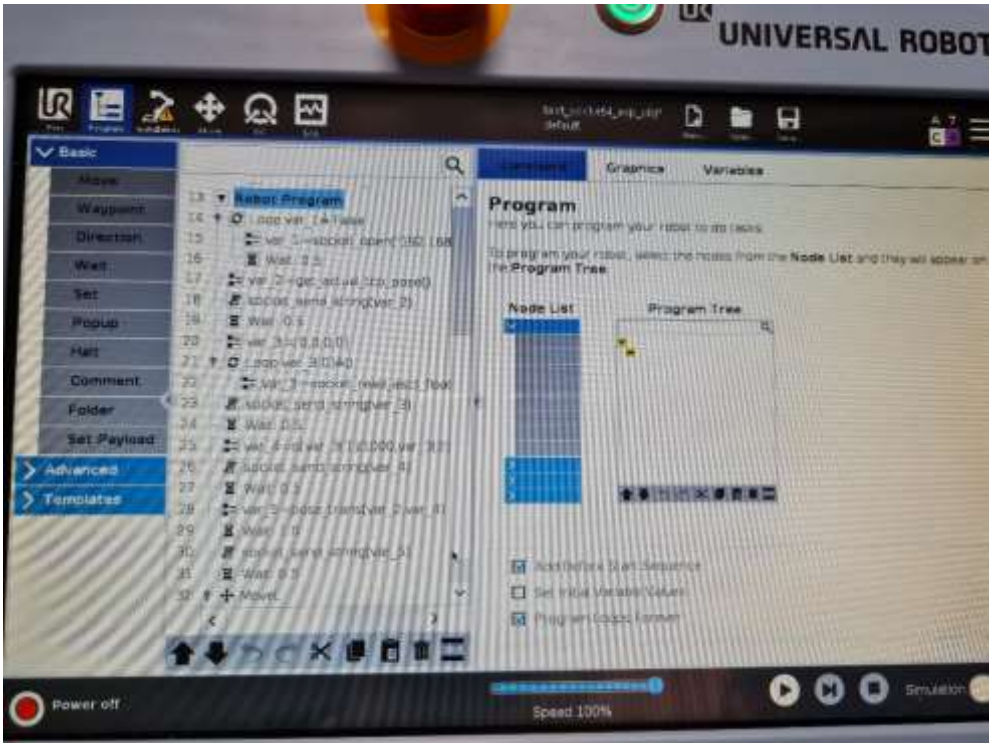
```
    try:
        msg = c.recv(1024)
        msg2 = d.recv(1024)
        print("Posicion Inicial UR3")
        print(msg)
        print("Posicion Inicial URe")
        print(msg2)
        time.sleep(0.5)

        #URe
        pos2 = msg2.decode()
        pos2 = pos2[2:len(pos2)]
        b = pos2.replace("b'p[", "").replace("]", "")
        lst = [x for x in b.split(',')]
        print(lst)

        p1 = (float(lst[0]))
        p2 = (float(lst[1]))
        p3 = (float(lst[2]))
        r1 = (float(lst[3]))
        r2 = (float(lst[4]))
        r3 = (float(lst[5]))
```

```
print("Posiciones recibidas del URe")
print(p1)
print(p2)
print(p3)

A = np.array([[0, 1, 0, -0.0994], [-1, 0, 0, -0.878], [0, 0, 1, -0.206], [0, 0, 0, 1]])
B = np.array([[p1], [p2], [p3], [1]])
T = np.dot(A, B)
print("Posicion del URe respecto el nuevo sistema de coordenadas")
print(T)

time.sleep(0.5)
i = i + 1
pos = np.array([[0.260], [0], [0], [0]])  # SE INDICA EL DESPLAZAMIENTO QUE SE QUIERE R
# X,X,Z,-;
P1 = T + pos
print("Nueva posicion del URe  respecto el nuevo sistema de coordenadas")
print(P1)

C = np.array([[0, -1, 0, -0.878], [1, 0, 0, 0.0994], [0, 0, 1, 0.206], [0, 0, 0, 1]])
T2 = np.dot(C, P1)
print("Nueva posicion del URe respecto el  sistema de coordenadas de la base")
print(T2)
```

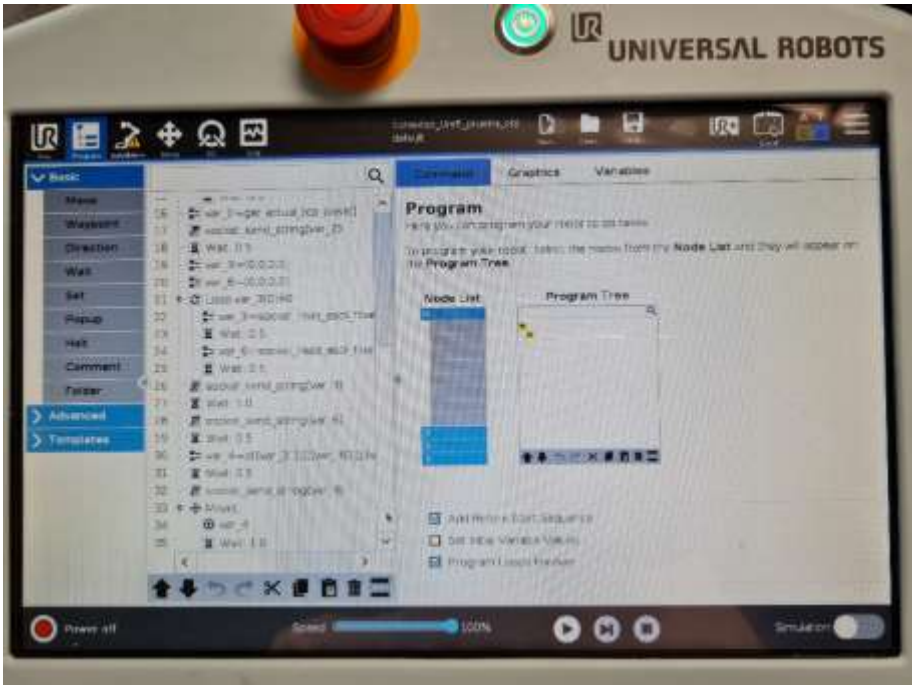
```
p1 = ((float(T2[0])))
p2 = ((float(T2[1])))
p3 = ((float(T2[2])))

POS = (p1, p2, p3)
POS1 = str(POS)
print("POSICION A ENVIAR AL URe")
print(POS1)

POS2 = (r1, r2, r3)
POS3 = str(POS2)
print(POS3)
time.sleep(0.5)

d.send(POS1.encode())
print("Data sent 1 URe")
time.sleep(0.5)

d.send(POS3.encode())
print("Data sent 2 URe")
time.sleep(0.5)

#UR3
pos = "(260,0,0)"  # (X,Y,Z,0,0,0)
c.send(pos.encode())
```

```
    print("Data sent UR3")
    time.sleep(0.5)

    #URe
    msg3 = d.recv(1024)
    print("Posicion enviada al URe")
    print(msg3)
    time.sleep(0.5)

    msg4 = d.recv(1024)
    print("Posicion 2 enviada al URe")
    print(msg4)
    time.sleep(0.5)

    #UR3
    msg5 = c.recv(1024)
    print("Posicion enviada al UR3")
    print(msg5)
    time.sleep(0.5)

    # URe
    msg6 = d.recv(1024)
    print("Posicion Final URe")
    print(msg6)
        # UR3
        msg7 = c.recv(1024)
        print("Posicion recibida en el UR3, var_4")
        print(msg7)
        msg8 = c.recv(1024)
        print("Posicion Final del UR3 (var_5)")
        print(msg8)


    except socket.error as socketerror:
        print("Error")

c.close()
d.close()
s.close()
r.close()
print("Program finish")
```

- Experimento 3. Código de Polyscope
  - Robot maestro:



  - Robot esclavo:

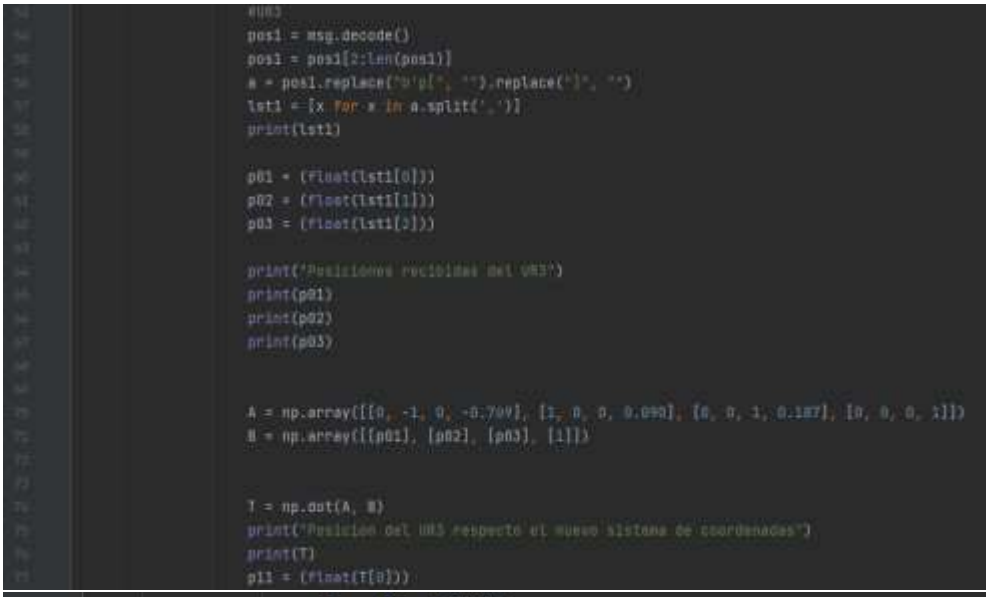


- Experimento 6. Control de fuerza

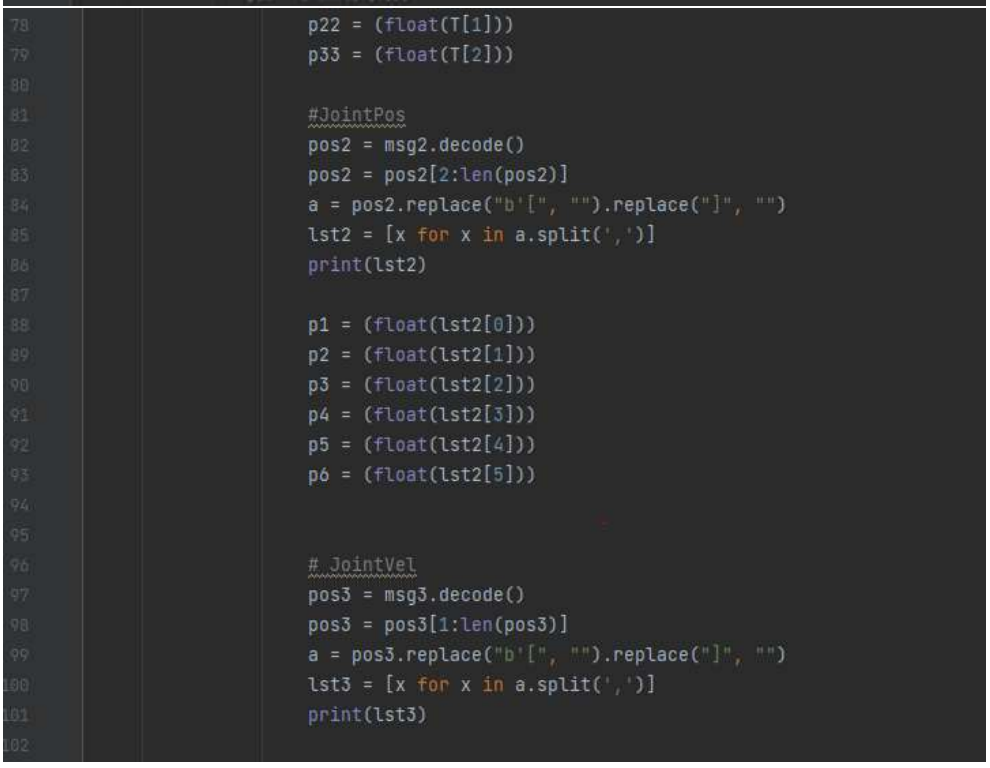
```
print("Fuerzas")
msgó = d.recv(1024)
print(msgó)
msg7 = msgó.decode()
msg7 = msg7[2:len(msg7)]
b = msg7.replace("b'p[", "").replace("]", "")
lst = [x for x in b.split(',')]
print("Fx: {0} , Fy: {1} , Fz: {2} ".format(lst[0], lst[1], lst[2]))

Fx = float(lst[0])
print(Fx)

Fxref=-0.5


print("Errores")
e_x = Fxref - Fx
e_y = p22 - p2
e_z = p33 - p3
C = np.array([[e_x], [e_y], [e_z]])
print(C)
```

- Experimento 6. Guardar datos en fichero

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Feb  7 19:55:09 2022

@author: Raul
"""

# Echo client program
import ...

HOST = "192.168.1.103" # The remote host
PORT = 30000 # The same port as used by the server #Robot UR3

print("Starting Program")
Tiempo = 0

#while i<1:
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Robot UR3
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT)) # Bind to the port

print("Port binded")
s.listen(5) # Now wait for client connection.
print("Client connected")
c, addr = s.accept() # Establish connection with client. #Robot UR3
print("Connection with client")
i = 0
datos2 = []

with open("Trayectoria3.csv", "w") as archivo_csv:
    cabecera = ("Pos(p1,p2,p3),JointP,JointV,Tiempo\n")
    archivo_csv.write(cabecera)

    while 1:
        try:
            start = time.time()
            msg = c.recv(1024)
            print("Posicion Inicial UR3")
            print(msg)
            #time.sleep(0.5)

            msg2 = c.recv(1024)
            print("Joint Position UR3")
            print(msg2)

            msg3 = c.recv(1024)
            print("Joint Speed UR3")
            print(msg3)
```



```
#UR3
pos1 = msg.decode()
pos1 = pos1[2:len(pos1)]
a = pos1.replace("b'p[", "").replace("]", "")
lst1 = [x for x in a.split(',')]
print(lst1)

p01 = (float(lst1[0]))
p02 = (float(lst1[1]))
p03 = (float(lst1[2]))

print('Posiciones recibidas del UR3')
print(p01)
print(p02)
print(p03)


A = np.array([[0, -1, 0, -0.709], [1, 0, 0, 0.098], [0, 0, 1, 0.187], [0, 0, 0, 1]])
B = np.array([[p01], [p02], [p03], [1]])


T = np.dot(A, B)
print("Posicion del UR3 respecto al nuevo sistema de coordenadas")
print(T)
p11 = (float(T[0]))
```



```
p22 = (float(T[1]))
p33 = (float(T[2]))

#JointPos
pos2 = msg2.decode()
pos2 = pos2[2:len(pos2)]
a = pos2.replace("b'[", "").replace("]", "")
lst2 = [x for x in a.split(',')]
print(lst2)

p1 = (float(lst2[0]))
p2 = (float(lst2[1]))
p3 = (float(lst2[2]))
p4 = (float(lst2[3]))
p5 = (float(lst2[4]))
p6 = (float(lst2[5]))


# JointVel
pos3 = msg3.decode()
pos3 = pos3[1:len(pos3)]
a = pos3.replace("b'[", "").replace("]", "")
lst3 = [x for x in a.split(',')]
print(lst3)
```

```
                v1 = (float(lst3[0]))
                v2 = (float(lst3[1]))
                v3 = (float(lst3[2]))
                v4 = (float(lst3[3]))
                v5 = (float(lst3[4]))
                v6 = (float(lst3[5]))

                end = time.time()
                Tiempo = Tiempo+(end - start)
                print("Tiempo de ejecucion")

                print(Tiempo)

                datos = ("{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{}\n".format(p11,p22,
                p33,p1,p2,p3,p4,p5,p6,v1,v2,v3,v4,v5,v6,Tiempo))
                archivo_csv.write(datos)

            except socket.error as socketerror:
                    print("Error")

archivo_csv.close()
c.close()
s.close()

print("Programa terminado")
```

- Experimento 6. Prueba SpeedL con base de datos

```
print("Connection with client")
i = 0
p1 = []
p2 = []
p3 = []
t_ = []
while 1:

    try:
            start = time.time()
            msg = c.recv(1024)
            msg2 = d.recv(1024)
            print("Posicion Inicial UR3")
            print(msg)
            #time.sleep(0.5)
            print("Posicion Inicial URe")
            print(msg2)
            #time.sleep(1)


            if msg != " ":

                #Lesctura de datos guardados
                with open("Trayectoria8.txt", "r") as archivo:
```

```
            for linea in archivo:

                lst1 = [x for x in linea.split(',')]
                p1.append(float(lst1[0]))
                p2.append(float(lst1[1]))
                p3.append(float(lst1[2]))
                t.append(float(lst1[15]))

        print(p1[i])
        print(p2[i])
        print(p3[i])
        print(t[i])

        #UR3
        pos1 = msg.decode()
        pos1 = pos1[2:len(pos1)]
        a = pos1.replace("b'p[", "").replace("]", "")
        lst1 = [x for x in a.split(',')]
        print(lst1)

        p01 = (float(lst1[0]))
        p02 = (float(lst1[1]))
        p03 = (float(lst1[2]))

        print("Posiciones recibidas del UR3")
```

```
print(p01)
print(p02)
print(p03)
A = np.array([[0, -1, 0, -0.769], [1, 0, 0, 0.094], [0, 0, 1, 0.186], [0, 0, 0, 1]])
#A = np.array([[0, -1, 0, -0.029], [1, 0, 0, 0.094], [0, 0, 1, 0.186], [0, 0, 0, 1]])
B = np.array([[p01], [p02], [p03], [1]])

T = np.dot(A, B)
print("Posicion del UR3 respecto el nuevo sistema de coordenadas")
print(T)
p11 = (float(T[0]))
p22 = (float(T[1]))
p33 = (float(T[2]))

# UR3
pos2 = msg2.decode()
pos2 = pos2[2:len(pos2)]
b = pos2.replace("b'p[", "").replace("]", "")
lst = [x for x in b.split(',')]
print(lst)

p1R = (float(lst[0]))
p2R = (float(lst[1]))
p3R = (float(lst[2]))
```

```
            print("Posiciones recibidas del URe")
            print(p1R)
            print(p2R)
            print(p3R)


            print("Errores")
            e_x1 = p11 - p1R
            e_y1 = p22 - p2R
            e_z1 = p33 - p3R
            C = np.array([[e_x1], [e_y1], [e_z1]])
            print(C)

            #Respecto la base de datos
            print(i)
            e_x2 = p1[i] - p1R
            e_y2 = p2[i] - p2R
            e_z2 = p3[i] - p3R
            i=i+1
            D = np.array([[e_x2], [e_y2], [e_z2]])
            print(D)

            kpx1 = 1
            kpy1 = 1
            kpz1 = 1
```

```
        kpx2 = 1
        kpy2 = 1
        kpz2 = 1

        Kp1 = np.array([[kpx1, 0, 0], [0, kpy1, 0], [0, 0, kpz1]])
        Kp2 = np.array([[kpx2, 0, 0], [0, kpy2, 0], [0, 0, kpz2]])
        Vref = (np.dot(Kp1, C) + np.dot(Kp2, D))
        print("Vref")
        print(Vref)

        v1 = ((float(Vref[0])))
        v2 = ((float(Vref[1])))
        v3 = ((float(Vref[2])))

        if abs(e_x1) < 0.001:
            v1 = 0
        if abs(e_y1) < 0.001:
            v2 = 0
        if abs(e_z1) < 0.001:
            v3 = 0

        VEL = (v1, v2, v3)
        VEL1 = str(VEL)
        print("POSICION A ENVIAR AL URe")
        print(VEL1)
```

```
                    d.send(VEL1.encode())
                    print("Data sent URe")
                    #time.sleep(0.5)

                    msg5 = d.recv(1024)
                    print("Posicion enviada al URe")
                    print(msg5)

                    end = time.time()
                    Tiempo = Tiempo+(end - start)
                    print("Tiempo de ejecución")
                    print(Tiempo)

                else:
                    print("No se ha recibido posición")

        except socket.error as socketerror:
                print("Error")

    archivo.close()
    c.close()
    s.close()
    d.close()
    r.close()
    print("Programa terminado")
```

- Código Matlab, Prueba inversa generador de trayectoria.

```
Datos=readcell('Trayectoria_V2_A1.csv')
n=length(Datos)
P11=cell2mat(Datos(2:n,1))'
P22=cell2mat(Datos(2:n,2))'
P33=cell2mat(Datos(2:n,3))'

%JoinP
P1=cell2mat(Datos(2:n,4))'
P2=cell2mat(Datos(2:n,5))'
P3=cell2mat(Datos(2:n,6))'
P4=cell2mat(Datos(2:n,7))'
P5=cell2mat(Datos(2:n,8))'
P6=cell2mat(Datos(2:n,9))'

%Tiempo
t=cell2mat(Datos(2:n,16))'

%JoinV
V1=cell2mat(Datos(2:n,10))'
V2=cell2mat(Datos(2:n,11))'
V3=cell2mat(Datos(2:n,12))'
V4=cell2mat(Datos(2:n,13))'
V5=cell2mat(Datos(2:n,14))'
V6=cell2mat(Datos(2:n,15))'

%JointV en grados
V11=V1*(180/pi)
V22=V2*(180/pi)
```

```
V33=V3*(180/pi)
V44=V4*(180/pi)
V55=V5*(180/pi)
V66=V6*(180/pi)

%representacion JointV en grados
subplot(2,3,1)
plot(t,V11)
xlabel('t')
title('V1')
subplot(2,3,2)
plot(t,V22)
xlabel('t')
title('V2')
subplot(2,3,3)
plot(t,V33)
xlabel('t')
title('V3')
subplot(2,3,4)
plot(t,V44)
xlabel('t')
title('V4')
subplot(2,3,5)
plot(t,V55)
xlabel('t')
title('V5')
subplot(2,3,6)
plot(t,V66)
```

```
57  xlabel('t')
58  title('V6')
59
60  A1=[]
61  A2=[]
62  A3=[]
63  A4=[]
64  A5=[]
65  A6=[]
66
67  %aceleracion
68  for r= 1:n
69      v1=V11(1,r)
70      v2=V11(1,r+1)
71      t1=t(1,r)
72      t2=t(1,r+1)
73      a=(v2-v1)/(t2-t1)
74      A1(r)=a
75
76      v12=V22(1,r)
77      v22=V22(1,r+1)
78      t12=t(1,r)
79      t22=t(1,r+1)
80      a2=(v22-v12)/(t22-t12)
81      A2(r)=a2
82
83      v13=V33(1,r)
84      v23=V33(1,r+1)
```

```
85          t13=t(1,r)
86          t23=t(1,r+1)
87          a3=(v23-v13)/(t23-t13)
88          A3(r)=a3
89
90          v14=V44(1,r)
91          v24=V44(1,r+1)
92          t14=t(1,r)
93          t24=t(1,r+1)
94          a4=(v24-v14)/(t24-t14)
95          A4(r)=a4
96
97          v15=V55(1,r)
98          v25=V55(1,r+1)
99          t15=t(1,r)
100         t25=t(1,r+1)
101         a5=(v25-v15)/(t25-t15)
102         A5(r)=a5
103
104         v16=V66(1,r)
105         v26=V66(1,r+1)
106         t16=t(1,r)
107         t26=t(1,r+1)
108         a6=(v26-v16)/(t26-t16)
109         A6(r)=a6
110
111     end
112     t1=t(1:r-1)
```

```
113
114     %Aceleracion
115     subplot(2,3,1)
116     plot(t1,A1)
117     xlabel('t')
118     title('A1')
119     subplot(2,3,2)
120     plot(t1,A2)
121     xlabel('t')
122     title('A2')
123     subplot(2,3,3)
124     plot(t1,A3)
125     xlabel('t')
126     title('A3')
127     subplot(2,3,4)
128     plot(t1,A4)
129     xlabel('t')
130     title('A4')
131     subplot(2,3,5)
132     plot(t1,A5)
133     xlabel('t')
134     title('A5')
135     subplot(2,3,6)
136     plot(t1,A6)
137     xlabel('t')
138     title('A6')
139
```

```
%JoinP en grados
p11=P1*(180/pi)
p22=P2*(180/pi)
p33=P3*(180/pi)
p44=P4*(180/pi)
p55=P5*(180/pi)
p66=P6*(180/pi)

%Representacion JoinP en grados
subplot(2,3,1)
plot(t,p11)
xlabel('t')
title('P1')
subplot(2,3,2)
plot(t,p22)
xlabel('t')
title('P2')
subplot(2,3,3)
plot(t,p33)
xlabel('t')
title('P3')
subplot(2,3,4)
plot(t,p44)
xlabel('t')
title('P4')
subplot(2,3,5)
plot(t,p55)
xlabel('t')
```

```
168     title('P5')
169     subplot(2,3,6)
170     plot(t,p66)
171     xlabel('t')
172     title('P6')
173
174     %Representacion JoinP
175     subplot(2,3,1)
176     plot(P1,t)
177     subplot(2,3,2)
178     plot(P2,t)
179     subplot(2,3,3)
180     plot(P3,t)
181     subplot(2,3,4)
182     plot(P4,t)
183     subplot(2,3,5)
184     plot(P5,t)
185     subplot(2,3,6)
186     plot(P6,t)
187
188     %Representacion JoinV
189     subplot(2,3,1)
190     plot(t,V1)
191     subplot(2,3,2)
192     plot(t,V2)
193     subplot(2,3,3)
194     plot(t,V3)
195     subplot(2,3,4)
196     plot(t,V4)
197     subplot(2,3,5)
198     plot(t,V5)
199     subplot(2,3,6)
200     plot(t,V6)
```

UNIVERSIDAD POLITECNICA DE VALENCIA

Dpto. de Ingeniería de Sistemas y Automática

Control distribuido del robot colaborativo UR3

2. PLANOS

TRABAJO DE FIN DE MASTER

Master Universitario en Automática e Informática Industrial

AUTOR/A: Alfonso Safont, Raúl

Tutor/a: Zotovic Stanisic, Ranko

Curso Académico: 2021-2022

Índice

Espacio de trabajo UR3e **1**

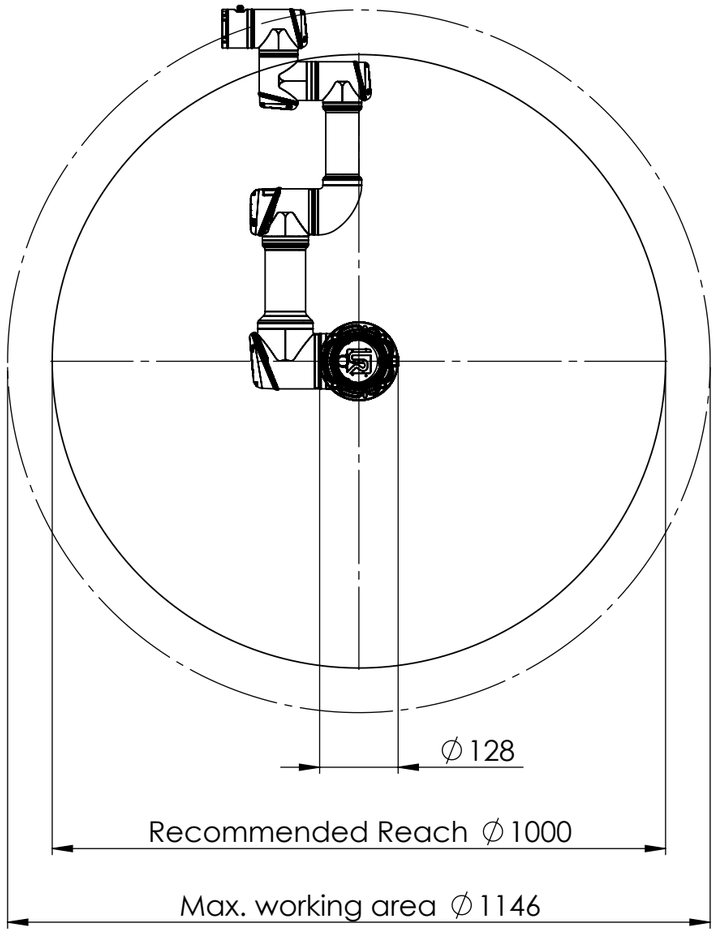
Espacio de trabajo UR3e con dimensión de los eslabones **2**

Esquema eléctrico general UR3e **3**

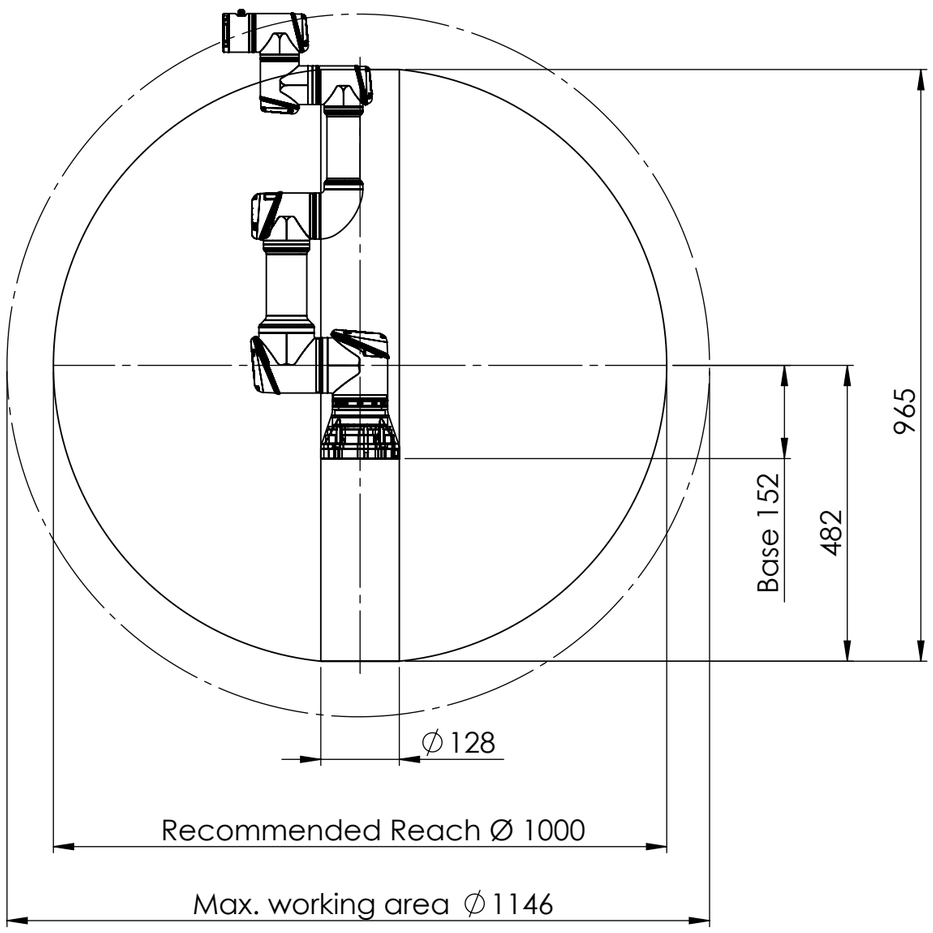
Esquema eléctrico del Controlador **4**

Esquema eléctrico del brazo robótico **5**

UR3 working area, top view



UR3 working area, side view

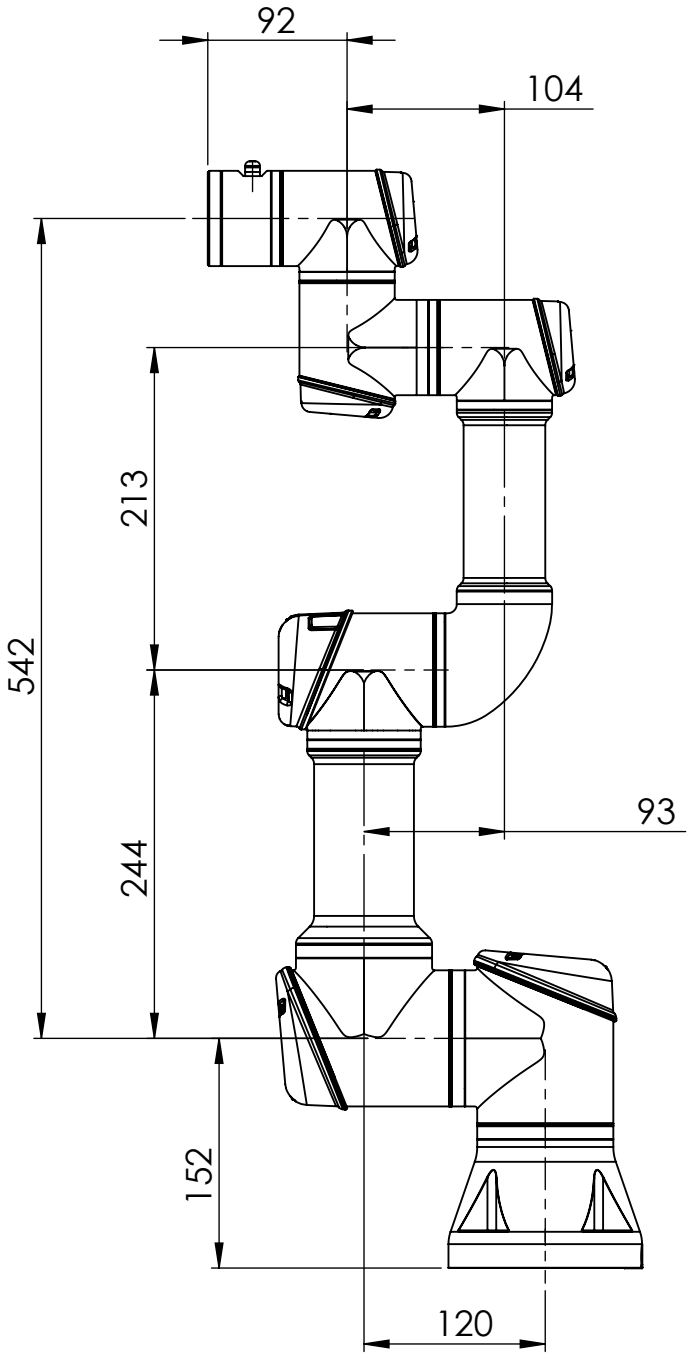


All dimension is in mm
For public use

UNIVERSAL ROBOTS

TEL: +45 89 93 89 89 FAX: +45 38 79 89 89 WEB: universal-robots.com

TITLE:	UR3e Working Area
DATE	28-05-2018
Status change date:	
DWG NO.	1000697
REV.	0

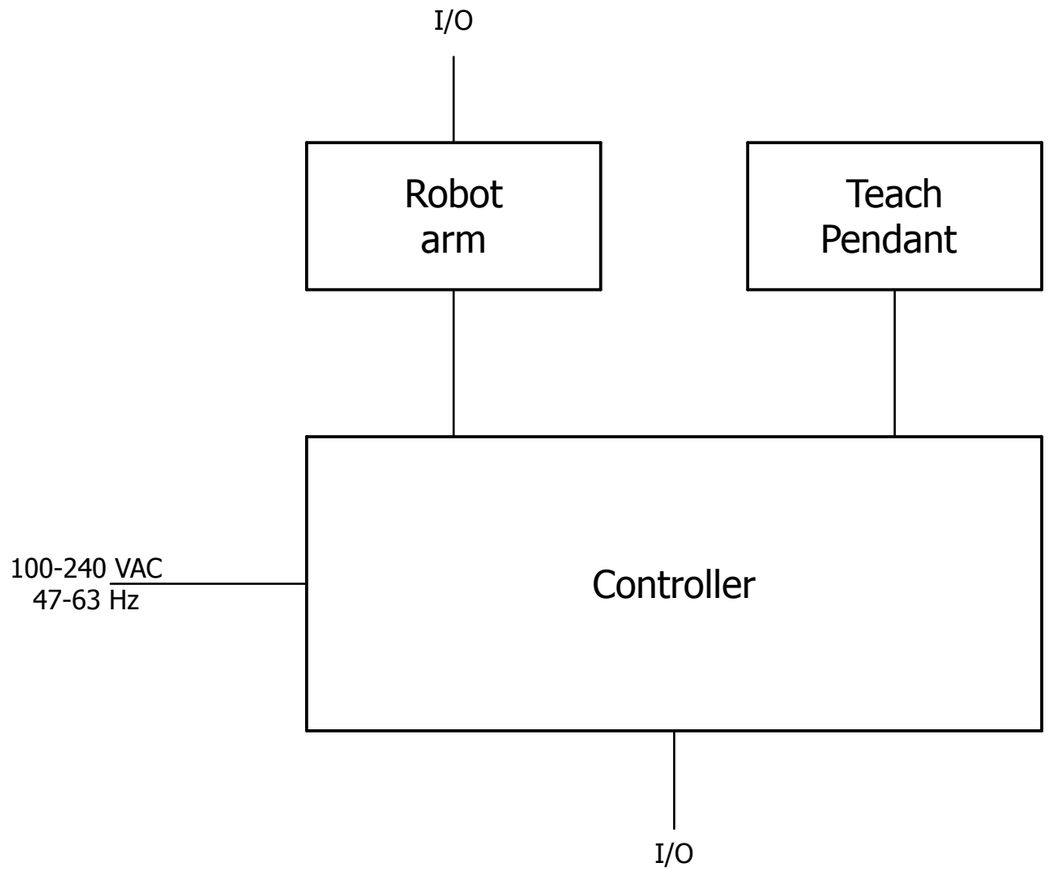


All dimension is in mm
For public use

UNIVERSAL ROBOTS
TEL: +45 89 93 89 89 FAX: +45 38 79 89 89 WEB: universal-robots.com

TITLE:	UR3e Working Area
DATE	28-05-2018
Status change date:	
DWG NO.	1000697
REV.	0

CB3.1 Overview UR3



			Date	10-03-2015	Universal Robots		Overview UR3		= UR3
			Ed.						+
			Appr					Dwg. No.: 3.0.3.1	Page 10
Modification	Date	Name	Original		Replacement of	Replaced by			Pages in total 18

CB3.1 Controller UR3

Wire colour codes	
Black	BK
Blue	BU
Brown	BN
Green	GN
Grey	GY
Orange	OG
Pink	PK
Red	RD
White	WH
Yellow	YE
Green/Yellow	GNYE

Supply:
100-240 VAC
47-63 Hz

Current Distributor
CON
L
N
GND
12V PSU
48V PSU1
48V PSU2

12V PSU
48V PSU

BN
BU
GNYE
BK
RD

Energy Eater
+48V
GND

GND
+IN 48V
+OUT 48V
RS485
12V
48V Relay
TCP/IP 12V

Motherboard CB3.1
Touch
RS232
DVI
TCP/IP
USB
USB BLUE
SYSTEM-SW S/N

Safety Control Board
SAFETY
ON/OFF
24V
Euromap E67
Config. DI
Config. DI
Config. DO
Config. DO
DI
DI
DO
DO
AI/AO

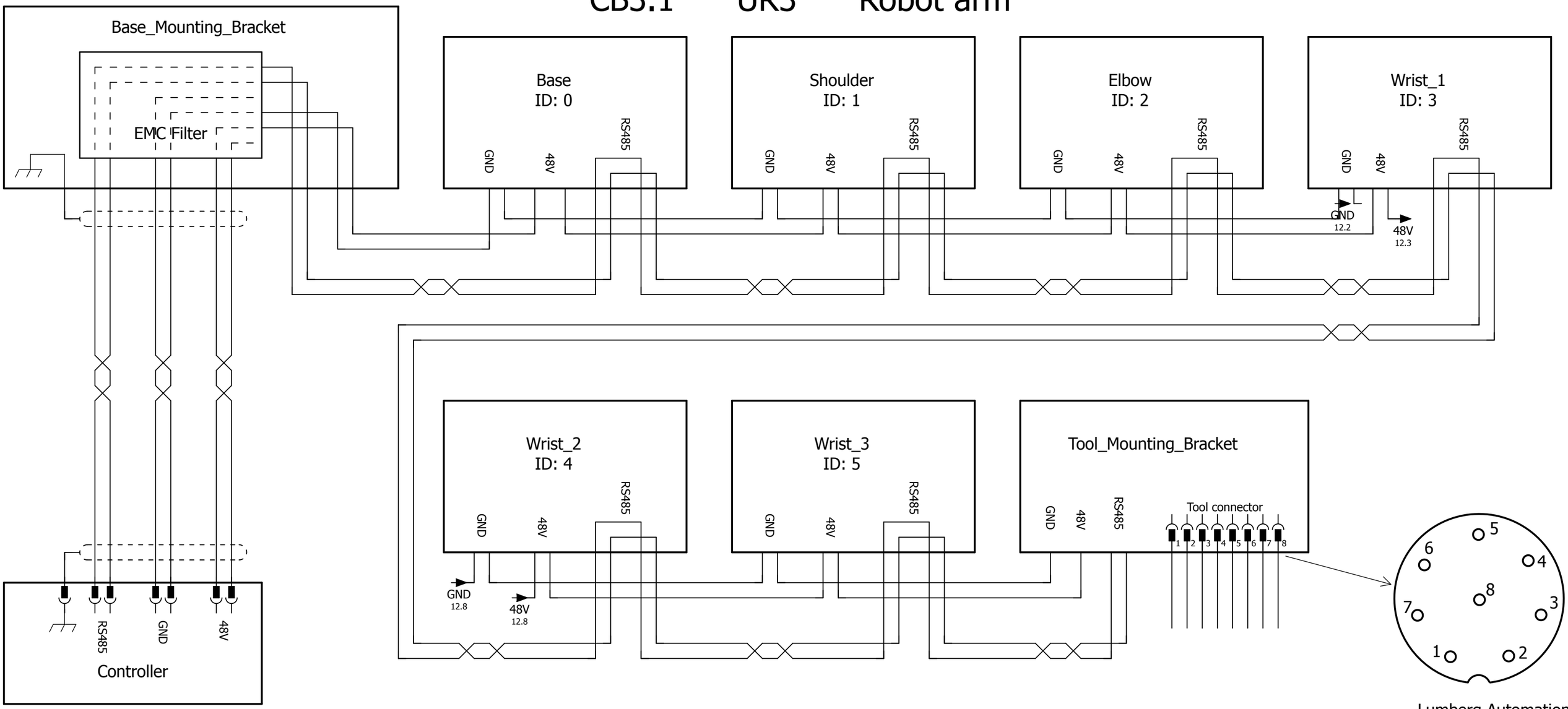
GND
8 4 7 6 5 3 1
EMC Filter

Robot
RS485
48V
ID=0 Base
ID=1 Shoulder
ID=2 Elbow
ID=3 Wrist 1
ID=4 Wrist 2
ID=5 Wrist 3
Tool
24v DI/DO AI

Teach Pendant
USB

			Date	10-03-2015	Universal Robots		Controller UR3	= UR3	
			Ed.					+	
			Appr					Dwg. No.: 3.0.3.1	Page 11
Modification	Date	Name	Original		Replacement of	Replaced by			Pages in total 18

CB3.1 UR3 Robot arm



Lumberg Automation
Type: RSMEDG 8

1 = white	AI[2]
2 = brown	AI[3]
3 = green	DI[9] (pnp)
4 = yellow	DI[8] (pnp)
5 = grey	12/24V (power)
6 = pink	DO[9] (npn)
7 = blue	DO[8] (npn)
8 = red	GND

11 15

			Date	10-03-2015	Universal Robots		Robot arm UR3	= UR3
			Ed.					+
			Appr				Dwg. No.: 3.0.3.1	Page 12
Modification	Date	Name	Original		Replacement of	Replaced by		Pages in total 18

UNIVERSIDAD POLITECNICA DE VALENCIA

Dpto. de Ingeniería de Sistemas y Automática

Control distribuido del robot colaborativo UR3

3. PRESUPUESTO

TRABAJO DE FIN DE MASTER

Master Universitario en Automática e Informática Industrial

AUTOR/A: Alfonso Safont, Raúl

Tutor/a: Zotovic Stanisic, Ranko

Curso Académico: 2021-2022

Índice

Cuadro mano de obra **1**

Cuadro maquinaria **3**

Cuadro materiales **5**

Justificación de precios **7**

Cuadro de precios descompuestos **10**

Medición **11**

Presupuesto **13**

Resumen **17**

Cuadro de mano de obra

Cuadro de mano de obra

Núm.	Código	Denominación de la mano de obra	Precio	Horas	Total
1	TRPP	Horas tutorizadas por el profesorado	15,000	65,000 h	975,00
2	TRPA	Horas realizadas por el alumnado	10,000	450,000 h	4.500,00
				Total mano de obra:	5.475,00

Cuadro de maquinaria

Total maquinaria:	0,00

Cuadro de materiales

Cuadro de materiales

Núm.	Código	Denominación del material	Precio	Cantidad	Total
1	SMTO	Torre ordenador de mesa	339,000	1,000 1	339,00
2	SMPO	Monitor para el ordenador de 20''	55,000	1,000 1	55,00
3	SMMW	Router wifi	13,390	1,000 1	13,39
4	SMTC	Teclado ordenador	5,990	1,000 1	5,99
5	SMRT	Ratón ordenador	5,990	1,000 1	5,99
				Total materiales:	419,37

Anejo de justificación de precios

Núm.	Código	Ud	Descripción			Total
1	SMBR	ud	Placa metalica para sujetar el robot de 500x250 mm			
			Sin descomposición			185,000
			0,000 % Costes indirectos		185,000	0,000
			Total por ud .....:			185,00
			Son CIENTO OCHENTA Y CINCO EUROS por ud.			
2	SMCB	ud	Compute Box para el robot UR3 le permite utilizar cualquier pinza de OnRobot con todos los Universal Robots. Incluye la compute box, un cable de 5 m, una memoria USB, fuente de alimentación, cables de E/S, bloques de terminales, cables de E/S, correas para cables, etc. La compute box puede comunicarse a través de una interfaz de E/S digital o Ethernet. Está conectado al controlador del robot y al cambiador rápido de herramientas.			
			Sin descomposición			812,570
			0,000 % Costes indirectos		812,570	0,000
			Total por ud .....:			812,57
			Son OCHOCIENTOS DOCE EUROS CON CINCUENTA Y SIETE CÉNTIMOS por ud.			
3	SMMT	ud	Mesa de trabajo			
			Sin descomposición			332,000
			0,000 % Costes indirectos		332,000	0,000
			Total por ud .....:			332,00
			Son TRESCIENTOS TREINTA Y DOS EUROS por ud.			
4	SMOB	ud	Caja de cartón para realizar los experimentos 200x150x100 mm			
			Sin descomposición			0,910
			0,000 % Costes indirectos		0,910	0,000
			Total por ud .....:			0,91
			Son NOVENTA Y UN CÉNTIMOS por ud.			
5	SMPC	ud	Ordenador y accesorios			
	SMTO		1,000 1 Torre ordenador de mesa		339,000	339,00
	SMPO		1,000 1 Monitor para el ordenador		55,000	55,00
	SMRT		1,000 1 Ratón ordenador		5,990	5,99
	SMTC		1,000 1 Teclado ordenador		5,990	5,99
	SMMW		1,000 1 Router wifi		13,390	13,39
	%		2,000 % Coster directos complementarios		419,370	8,39
			0,000 % Costes indirectos		427,760	0,000
			Total por ud .....:			427,76
			Son CUATROCIENTOS VEINTISIETE EUROS CON SETENTA Y SEIS CÉNTIMOS por ud.			
6	SMSP	ud	Soporte para la base del robot de 200mm de altura			
			Sin descomposición			120,000
			0,000 % Costes indirectos		120,000	0,000
			Total por ud .....:			120,00
			Son CIENTO VEINTE EUROS por ud.			

Núm.	Código	Ud	Descripción			Total
7	SMSR	ud	Sensor de la marca Onrobot para el robot UR3, dispone de seis grados de medición de fuerza y par.			
				Sin descomposición		3.575,000
			0,000 %	Costes indirectos	3.575,000	0,000
				Total por ud .....:		3.575,00
			Son TRES MIL QUINIENTOS SETENTA Y CINCO EUROS por ud.			
8	SMUR	ud	Robot UR3e incluye instalación			
				Sin descomposición		22.700,000
			0,000 %	Costes indirectos	22.700,000	0,000
				Total por ud .....:		22.700,00
			Son VEINTIDOS MIL SETECIENTOS EUROS por ud.			
9	TRPR	ud	Realización del trabajo			
	TRPP		65,000 h	Ingeniero en Automática Industrial	15,000	975,00
	TRPA		450,000 h	Ingeniero en Automática e Informática Industrial	10,000	4.500,00
			0,000 %	Costes indirectos	5.475,000	0,000
				Total por ud .....:		5.475,00
			Son CINCO MIL CUATROCIENTOS SETENTA Y CINCO EUROS por ud.			

Cuadro de precios nº 1

Advertencia

Los precios designados en letra en este cuadro, con la rebaja que resulte en la subasta en su caso, son los que sirven de base al contrato, y se utilizarán para valorar la obra ejecutada, siguiendo lo prevenido en la Cláusula 46 del Pliego de Cláusulas Administrativas Generales para la Contratación de Obras del Estado, considerando incluidos en ellos los trabajos, medios auxiliares y materiales necesarios para la ejecución de la unidad de obra que definan, conforme a lo prescrito en la Cláusula 51 del Pliego antes citado, por lo que el Contratista no podrá reclamar que se introduzca modificación alguna en ello, bajo ningún pretexto de error u omisión.

Nº	Designación	Importe	
		En cifra (Euros)	En letra (Euros)
1	ud Placa metalica para sujetar el robot de 500x250 mm	185,00	CIENTO OCHENTA Y CINCO EUROS
2	ud Compute Box para el robot UR3 le permite utilizar cualquier pinza de OnRobot con todos los Universal Robots. Incluye la compute box, un cable de 5 m, una memoria USB, fuente de alimentación, cables de E/S, bloques de terminales, cables de E/S, correas para cables, etc. La compute box puede comunicarse a través de una interfaz de E/S digital o Ethernet. Está conectado al controlador del robot y al cambiador rápido de herramientas.	812,57	OCHOCIENTOS DOCE EUROS CON CINCUENTA Y SIETE CÉNTIMOS
3	ud Mesa de trabajo	332,00	TRESCIENTOS TREINTA Y DOS EUROS
4	ud Caja de cartón para realizar los experimentos 200x150x100 mm	0,91	NOVENTA Y UN CÉNTIMOS
5	ud Ordenador y accesorios	427,76	CUATROCIENTOS VEINTISIETE EUROS CON SETENTA Y SEIS CÉNTIMOS
6	ud Soporte para la base del robot de 200mm de altura	120,00	CIENTO VEINTE EUROS
7	ud Sensor de la marca Onrobot para el robot UR3, dispone de seis grados de medición de fuerza y par.	3.575,00	TRES MIL QUINIENTOS SETENTA Y CINCO EUROS
8	ud Robot UR3e incluye instalación	22.700,00	VEINTIDOS MIL SETECIENTOS EUROS
9	ud Realización del trabajo	5.475,00	CINCO MIL CUATROCIENTOS SETENTA Y CINCO EUROS

Valencia, 14/09/2022

Master en Automática e Informática Industrial

Raul Alfonso Safont

1 Materiales

Nº	Ud	Descripción		Medición
1.1	Ud	Robot UR3e		
			Total ud :	2,000
1.2	Ud	Mesa de trabajo		
			Total ud :	1,000
1.3	Ud	Soporte para la base del robot		
			Total ud :	1,000
1.4	Ud	Soporte metalico para sujetar el robot		
			Total ud :	1,000
1.5	Ud	Ordenador y accesorios		
			Total ud :	1,000
1.6	Ud	Caja de cartón		
			Total ud :	1,000
1.7	Ud	Sensor Onrobot		
			Total ud :	1,000
1.8	Ud	Compute Box		
			Total ud :	1,000

2 Trabajo

Nº	Ud	Descripción	Medición
2.1	Ud	Realización del trabajo	
		Total ud :	1,000

Presupuesto

Presupuesto parcial nº 1 Materiales

Núm.	Código	Ud	Denominación	Cantidad	Precio (€)	Total (€)
1.1	SMUR	ud	Robot UR3e incluye instalación	2,000	22.700,00	45.400,00
1.2	SMMT	ud	Mesa de trabajo	1,000	332,00	332,00
1.3	SMSP	ud	Soporte para la base del robot de 200mm de altura	1,000	120,00	120,00
1.4	SMBR	ud	Placa metalica para sujetar el robot de 500x250 mm	1,000	185,00	185,00
1.5	SMPC	ud	Ordenador y accesorios	1,000	427,76	427,76
1.6	SMOB	ud	Caja de cartón para realizar los experimentos 200x150x100 mm	1,000	0,91	0,91
1.7	SMSR	ud	Sensor de la marca Onrobot para el robot UR3, dispone de seis grados de medición de fuerza y par.	1,000	3.575,00	3.575,00
1.8	SMCB	ud	Compute Box para el robot UR3 le permite utilizar cualquier pinza de OnRobot con todos los Universal Robots. Incluye la compute box, un cable de 5 m, una memoria USB, fuente de alimentación, cables de E/S, bloques de terminales, cables de E/S, correas para cables, etc. La compute box puede comunicarse a través de una interfaz de E/S digital o Ethernet. Está conectado al controlador del robot y al cambiador rápido de herramientas.	1,000	812,57	812,57
			Total presupuesto parcial nº 1 Materiales :			**50.853,24**

Presupuesto parcial nº 2 Trabajo

Núm.	Código	Ud	Denominación	Cantidad	Precio (€)	Total (€)
2.1	TRPR	ud	Realización del trabajo	1,000	5.475,00	5.475,00
			Total presupuesto parcial nº 2 Trabajo :			**5.475,00**

Presupuesto de ejecución material

	Importe (€)
1 Materiales	50.853,24
2 Trabajo	5.475,00
Total	56.328,24

Asciende el presupuesto de ejecución material a la expresada cantidad de CINCUENTA Y SEIS MIL TRESCIENTOS VEINTIOCHO EUROS CON VEINTICUATRO CÉNTIMOS.

Valencia, 14/09/2022
Master en Automática e Informática Industrial

Raul Alfonso Safont

Proyecto: ROBOT UR3

Capítulo	Importe
Capítulo 1 Materiales	50.853,24
Capítulo 2 Trabajo	5.475,00
Presupuesto de ejecución material	56.328,24
13% de gastos generales	7.322,67
6% de beneficio industrial	3.379,69
Suma	67.030,60
21% IVA	14.076,43
Presupuesto de ejecución por contrata	81.107,03

Asciende el presupuesto de ejecución por contrata a la expresada cantidad de OCHENTA Y UN MIL CIENTO SIETE EUROS CON TRES CÉNTIMOS.

Valencia, 14/09/2022
Master en Automática e Informática Industrial

Raul Alfonso Safont