UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DESARROLO DE CONTROLADORES PARA ROBOTS PARALELOS MEDIANTE INGENIERÍA DE SOFTWARE BASADA EN COMPONENTES. APLICACIÓN A ROBOTS DE REHABILITACIÓN DE MIEMBRO INFERIOR.

AUTOR: OLIVER CHIVA, ERNESTO

TUTOR: VALERA FERNÁNDEZ, ÁNGEL

COTUTORA: VALLÉS MIQUEL, MARINA

Curso Académico: 2017-18

Índice general

Índice de figuras

Índice de tablas

Parte I

Memoria

Capítulo 1

Introducción, motivación y objetivos

Introducción

El diseño de sistemas electromecánicos basados en robots industriales se está expandiendo continuamente hacia otros sectores. Esto se debe a que la sociedad va creando nuevas demandas y necesidades; por ejemplo, dispositivos mecánicos que cubren necesidades sociales y humanas [17].

En los últimos 30 años los robots han ido ocupando espacios más allá de la industria como las casas privadas, hospitales o áreas de servicio. El uso de estos robots supone una estrecha interacción entre los robots manipuladores y los seres humanos. Esta interacción puede ir desde compartir un mismo espacio de trabajo sin establecer ningún tipo de contacto físico hasta el contacto mecánico hombre-máquina [18].

En este aspecto, el campo de la rehabilitación presenta ciertos aspectos que lo hacen interesante para la investigación con robots [19]:

- Los sistemas sanitarios se enfrentan a grandes pérdidas de forma sistemática.
- La demanda de servicios de rehabilitación se verá incrementada en las próximas décadas debido al envejecimiento de la población.

Por lo tanto el aumento de pacientes que necesiten rehabilitación pueden suponer un incremento del gasto en salud en un presupuesto ya muy ajustado.

La rehabilitación busca devolver a los pacientes las facultades físicas, sensoriales o mentales que han perdido debido a lesiones o enfermedad, y para dar apoyo a pacientes para compensar las deficiencias que no pueden ser tratadas médicamente [20].

En general, una persona con problemas de movilidad en brazos o piernas necesita realizar ejercicios terapéuticos durante largos periodos de tiempo. Las sesiones suponen una serie de movimientos terapéuticos que acaban siendo repetitivos y rutinarios, en el que se necesita la asistencia y supervisión de un fisioterapeuta. Además, el transporte del paciente al centro médico o el desplazamiento del profesional sanitario son factores que hacen aumentar el coste del tratamiento [20]. Esto hace que las sesiones de rehabilitación se vean fuertemente limitadas por la escasez de personal y la fatiga del fisioterapeuta, no del paciente, haciendo que la efectividad del tratamiento se pueda ver afectado. Por otro lado, el uso de ejercicios automatizados con robots permite aumentar el número y la duración a la vez que reduce el número de profesionales necesarios por paciente [18]. En [20] se listan cuatro importantes razones por las que los robots pueden ayudar a mejorar los tratamientos de rehabilitación:

- Los robots pueden completar fácilmente los movimientos cíclicos usados en rehabilitación.
- Los robots pueden leer con precisión los movimientos del paciente.
- Los robots son capaces de reproducir de manera precisa las fuerzas necesarias en los ejercicios.
- Los robots pueden ser mucho más precisos en lo que se refiere a las condiciones de la terapia.

Las primeras generaciones de robots para rehabilitación se caracterizan por forzar el movimiento del paciente a una trayectoria dada, sin que este tenga apenas la posibilidad de influir en el movimiento. [21]. Los ejercicios pasivos ayudan a restablecer la flexibilidad y el rango de movimientos, pero deben complementarse con ejercicios resistivos que fortalezcan las musculatura y aumenten la resistencia [20]. Es por eso que [22] propone clasificar los robots para rehabilitación en tres grupos dependiendo del si se comportan de una forma más pasiva o más resistiva:

1. Robots asistentes para ayudar en el día a día a personas con necesidades especiales.
2. Robots que permiten la movilidad de algún miembro impedido.
3. Robots para ejercicios terapéuticos dónde sirven de apoyo a los fisioterapeutas y pacientes en la realización de ejercicios repetitivos.

Este trabajo se ha centrado en el último grupo, donde se investigan las posibilidades de un robot capaz de realizar movimientos complejos de forma precisa a la vez que recibe información acerca de la fuerza que está realizando el paciente.

Además, para la interacción con el paciente, además de contar con la información de fuerza, es posible contar con la información proporcionada por señales de electromiografía (EMG). Por

ejemplo, la señal EMG acompañada por contracciones musculares contiene información acerca de la intención de movimiento por parte del usuario, ya que todo movimiento depende de alguna contracción muscular, las cuales vienen controladas por el sistema nervioso centras [23].

En la primera generación de robots para rehabilitación se han utilizado las mismas estrategias de control que en los robots industriales, basados en un control de trayectoria para los robots pasivos, e incluyendo control de fuerza y par en los robots con ejercicios resistivos [24] [25]. En comparación con las estrategias de control que encontramos en los diferentes robots industriales, los robots centrados en la interacción humana presentan unos requisitos totalmente diferentes [26]. Estos requisitos incluyen la seguridad, precisión mecánica, adaptabilidad al usuario, facilidad de uso, gentileza en los movimientos, etc [18]. En [25] se presenta un control basado en modelos para un robot con cadenas cinemáticas paralelas (o robot paralelo). Este control tienen la ventaja de tener un control híbrido de posición/fuerza donde la velocidad de las diferentes articulaciones se obtiene a partir de un observador de estados. Además, el modelo del sistema se resuelve con una identificación de parámetros aproximada donde se extraen únicamente los parámetros más relevantes del sistema. Esto hace que el cálculo *online* del control sea rápido y suficientemente preciso.

Este texto se estructura del siguiente modo: el primer capítulo empieza con una introducción que expone el estado del arte de los robots para rehabilitación, así como la motivación y objetivos que han llevado a la realización de este trabajo. El segundo capítulo introduce el concepto de robot paralelo y explica de forma detallada tanto el hardware como el software que se ha empleado en la última versión del robot paralelo. En el capítulo tercero se detallan los cálculos, tanto dinámicos como cinemáticos, de esta última versión del robot. El texto sigue en el capítulo cuatro con una explicación más detallada acerca del sistema de software basado en componentes utilizado, Orocos, el cual ya introducido en el capítulo segundo en el apartado de software. El capítulo cinco recoge la información acerca de los diferentes sensores externos probados. Los resultados se presentan en el capítulo seis, donde se comparan las diferentes estrategias de control implementadas. Por último, en el capítulo siete se desglosa un presupuesto aproximado de lo que ha supuesto el desarrollo del proyecto. Además, en los anexos A y B se incluyen dos manuales de los robots paralelo3DOF y paralelo4DOF respectivamente, redactados por el autor de este texto.

Motivación

El siguiente trabajo tiene como motivación dar respuesta a distintos retos aparecidos por la creciente expansión de la robótica fuera del campo de la industria, donde tradicionalmente se han dado los mayores avances. El sector de la medicina es complejo y generalmente requiere de profesionales experimentados que difícilmente se pueden sustituir por un autómata, pero también existen aplicaciones donde se requiere de instrumental preciso capaz de realizar tareas repetitivas. Son en estos últimos casos en los que los robots pueden suponer una diferencia tanto para el profesional sanitario como para el paciente. Este es el caso de los ejercicios de rehabilitación, dónde un fisioterapeuta debe realizar ejercicios repetitivos con la precaución de no ejercer demasiada fuerza sobre las articulaciones afectadas. El desarrollo de la parte del robot paralelo correspondiente a este trabajo se ha completado utilizando las instalaciones de la Universidad Politécnica de Valencia bajo la beca formativa de colaboración *Utilización de sensores biomédicos para el control de sistemas robotizados (15/0103).*

Objetivos

El objetivo de este trabajo ha sido el desarrollo de controladores para un robot paralelo de 4 grados de libertad, de forma que se consiga desarrollar ejercicios de rehabilitación. Este objetivo se enmarca dentro de los objetivos del proyecto de investigación *Metodología de diseño de sistemas biomecatrónicos. Aplicación al desarrollo de un robot paralelo híbrido para diagnóstico y rehabilitación (MEBIOMEC, 2016)*, dentro del programa estatal de investigación, desarrollo e innovación orientada a los retos de la sociedad, del Ministerio de economía y competitividad del Gobierno de España.

Otro de los objetivos que se ha buscado ha sido que el control del robot se consiga con una arquitectura basada en componentes, lo que ha permitido que se puedan probar diferentes estrategias de control realizando pequeñas modificaciones y reutilizando componentes. Esto ha sido posible gracias a los proyectos de software libre como ROS y Orocos.

Capítulo 2

El robot paralelo

Los robots paralelos

Si buscamos en la literatura nos encontramos que un robot paralelo se define como aquel se compone de una base fija conectada a una base móvil mediante cadenas cinemáticas (a las que se denomina como piernas o patas). De tal forma que la carga se distribuye entre las piernas del robot [27]. Los robots paralelos se suelen clasificar por el número de grados de libertad que tienen. La mayoría de los robots paralelos tienen entre 2 y 6 grados de libertad (o DoF). Un sondeo de 82 arquitecturas paralelas diferentes refleja que el 40 % tienen 6DoF, el 3.5 % tienen 5DoF, el 6 % son de 4DoF, el 40 % de 3DoF y el restante 10.5 % tienen solo 2DoF [28].

Un conocido ejemplo de robot paralelo es el robot *Detla*, desarrollado por Reymond Clavel en su tesis doctoral *Conception d'un robot parallele rapide a 4 degres de liberte* [1] (ver Figura 2.1).

La idea del robot Delta surgió a partir de una visita a una fábrica de chocolate en 1985, donde un grupo de empleadas se encargaban de paletizar a mano las chocolatinas. Esta tarea exige un alto grado de higiene y puede considerarse como un trabajo extremadamente aburrido y estresante para las personas.
En aquel momento no existía en el mercado ningún robot que pudiera ejecutar esa tarea debido a la gran cantidad de restricciones del problema. Por lo tanto se vio que existía la necesidad de buscar un nuevo diseño [1] [29].

Aunque este tipo de robots predominan en el sector industrial, donde son ideales para tareas de *pick-and-place* donde se necesita mover objetos ligeros a altas velocidades y con mucha precisión [16]. En otros sectores como en el aeroespacial tenemos el ejemplo del *Vertical Motion*

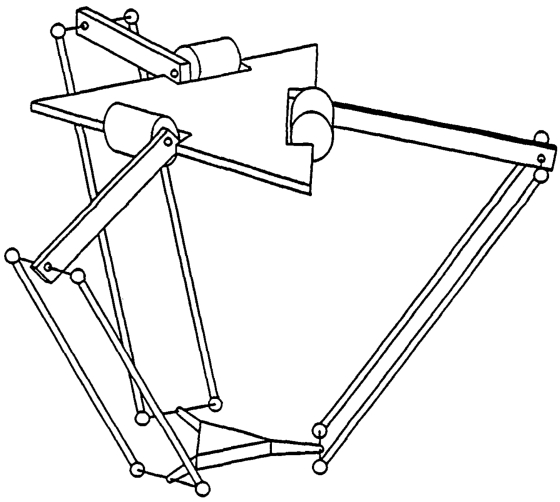


Figura 2.1: Estructura móvil de un robot Delta. [1]

Simulator, considerado el simulador de vuelo más realista del mundo [2], la Figura 2.2 muestra imágenes del simulador.

También en el sector médico, donde tradicionalmente no se piensa en robots, encontramos una tendencia al alza en el uso de esta tecnología. También ha sido este el campo elegido para el desarrollo del presente trabajo. En este campo, donde la precisión es vital, los robots paralelos toman ventaja sobre los robots seriales. En [30] se nos presenta un robot paralelo de 6 grados de libertad que realiza operaciones de neurocirugía con una precisión de 20 micrómetros (ver Figura 2.3).

Los robots paralelos siguen generando lineas de investigación que intentan sacar el máximo provecho a esta tecnología. Este es el caso del robot MilliDelta(Figura 2.4). Un pequeño robot de $15mmx15mmx20mm$ con una masa de $430mg$ que puede cargar hasta $1.31g$ y que cuenta

(a) Base fija **(b)** Cabina interior **(c)** Base móvil

Figura 2.2: Imágenes del VMS [2]

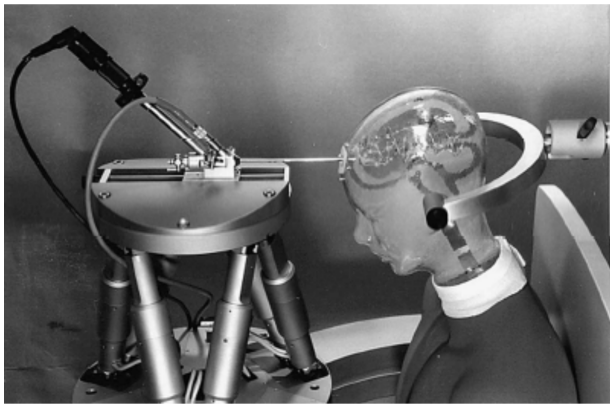


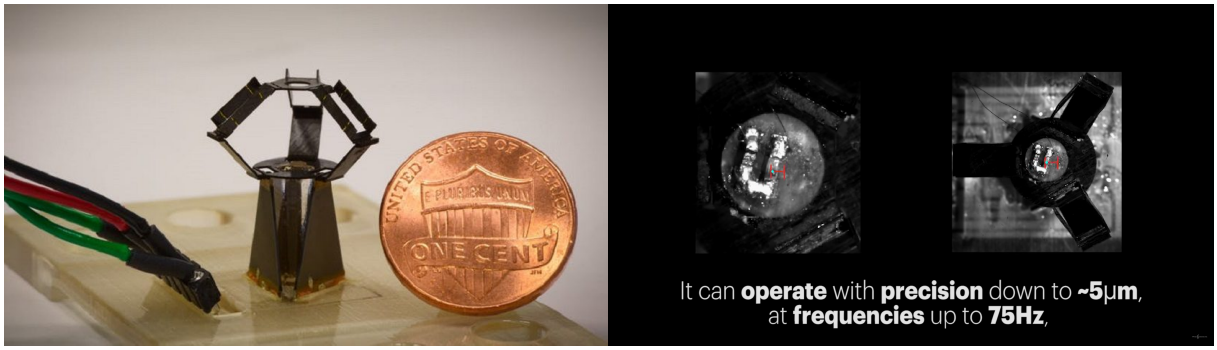
Figura 2.3: Plataforma de Stewart para neurocirugía

con una precisión de 5μ. Este pequeño robot es capaz de seguir trayectorias con una frecuencia de hasta $75Hz$, a una velocidad de $0.45m/s$ y una aceleración de $215m/s^2$ [3].

Los robots paralelos tienen ciertas ventajas cuando se comparan con los robots seriales. Una de las más importantes es la posibilidad de poder mantener los motores fijos en la base, lo que ayuda a reducir considerablemente la masa móvil [31].

En [32] los autores presentan un ejemplo donde comparan un robot paralelo con otro robot serial equivalente y demuestras que, sin generalizar, el robot paralelo presenta mejores prestaciones como ya defendían otros autores:

- "La tecnología del robot paralelo promete ofrecer ventajas relativas a las maquina-herramientas, como por ejemplo alta precisión" [33]
- "Se prefieren los robots paralelos frente a los robots seriales por su capacidad de posicionamiento con alta precisión" [34]



(a) MilliDelta compasado con una moneda [3] **(b)** MilliDelta realizando una trayectoria [3]

Figura 2.4: Imágenes del robot MilliDelta [3]

- "Los errores de los robots paralelos son el resultado de la media de los error en sus cadenas cinemáticas, mientras que los errores de los robots seriales son la acumulación de dichos errores" [35]

Por otro lado, los robots paralelos tienen la desventaja de que su modelo dinámico es considerablemente más complicado de calcular que en el caso de un robot serial. El desarrollo del modelo dinámico de los robots paralelos ha despertado gran interés en las últimas décadas. La principal dificultad recae en encontrar una solución que sea suficientemente representativa del sistema real y a la vez sea fácil de calcular en tiempo real para poder implementarla en el algoritmo de control [31].

Para el proyecto en el que se recoge el presente texto se ha trabajado con dos robots paralelos desarrollados en la Universidad Politécnica de Valencia y el Instituto Universitario de Automática e Informática Industrial (ai2) bajo el proyecto de investigación: *Metodología de Diseño de Sistemas Biomecatrónicos. Aplicación al desarrollo de un Robot Paralelo híbrido para diagnóstico y rehabilitación* (MEBIOMEC, 2016).

Por un lado se trata de un robot funcional paralelo de 3DoF, el cual se le denominará como *paralelo3DoF*, con el que se han podido desarrollar diferentes estrategias de control, el cual e muestra en la Figura 2.5.

Por otro lado, y partiendo del robot paralelo3DoF, se ha desarrollado una nueva versión denominada paralelo4DoF, ver Figura 2.6, en el cual se ha empezado a trabajar desde la instalación del software.

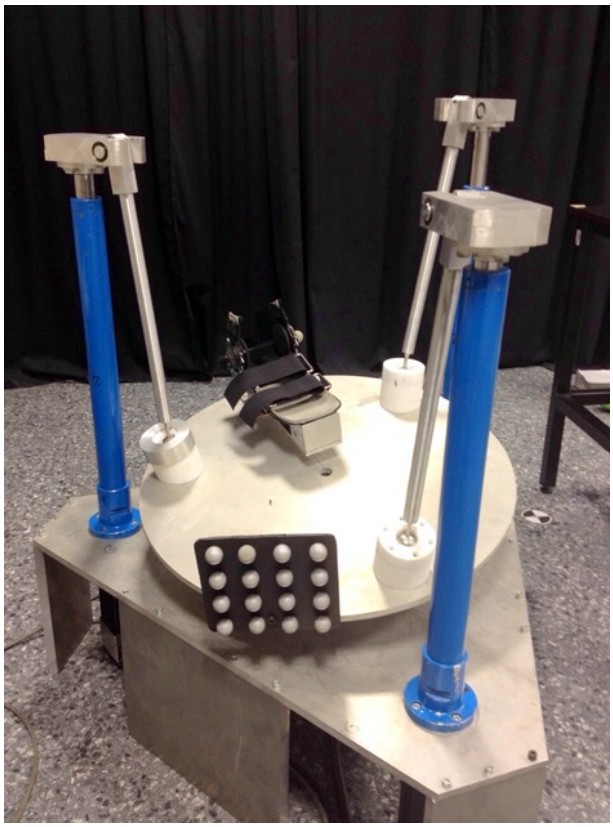


Figura 2.5: Robot paralelo3DoF utilizado en para el desarrollo de controladores

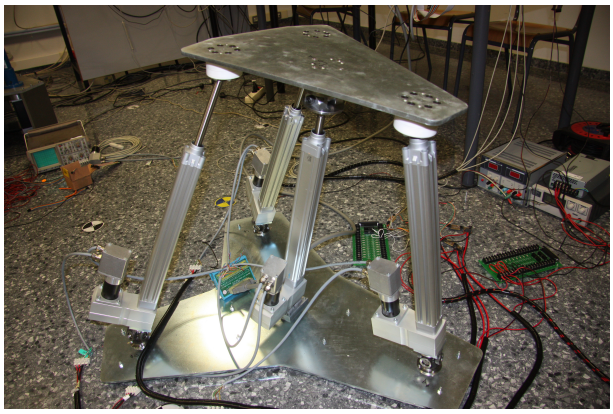


Figura 2.6: Robot paralelo4DoF en su fase inicial

Hardware

La estructura mecánica del robot paralelo con 4 grados de libertad

En la figura 2.7 se muestra un esquema de la estructura del robot paralelo con 4 grados de libertad. Como se ve, la estructura se compone de dos plataformas, una fija y una móvil, unidas por tres cadenas cinemáticas con articulaciones U-P-E en los tres extremos y una cadena con articulaciones R-P-U en el centro.

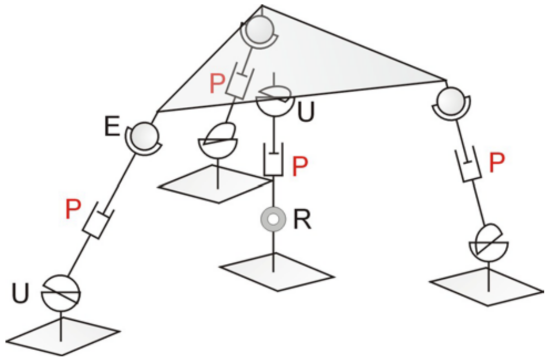


Figura 2.7: Esquema del robot paralelo4DoF. Fuente [4]

Las articulaciones rotacionales (R) permiten el giro alrededor de un eje, mientras que las articulaciones universales (U) se componen de dos articulaciones rotacionales con sus ejes de rotación perpendiculares. Por otro lado las articulaciones esféricas (E) giran alrededor de una esfera, lo que da como resultado infinitos ejes de rotación. Por último están las articulaciones prismáticas (P), que permiten el movimiento lineal en una dirección.

Las articulaciones P son las encargadas de transmitir el movimiento al robot paralelo. Para ello se ha dispuesto de actuadores lineales, compuestos por un motor DC conectado a un husillo de bolas que transforma el movimiento rotatorio del robot en un desplazamiento lineal. Para facilitar el control del motor también se le ha acoplado un freno así como un encoder que devuelve la posición del rotor en cada instante.

El actuador lineal

Quizás el elemento de hardware más importante del robot paralelo, tanto de la versión de 3DOF como la de 4DOF, sea el actuador lineal (ver figura 2.8. Estas piezas son las encargadas directas de generar el movimiento por lo que el criterio con el que se diseñan será crucial para sacar el máximo rendimiento al robot. El mecanismo de estos actuadores está compuesto por un husillo de bolas que transforma el movimiento giratorio en un desplazamiento lineal, un motor DC con escobillas de grafito que es el encargado último de hacer funcionar el mecanismo, un encoder

incremental que nos permitirá conocer la posición del actuador y por último se ha añadido un freno para poder fijar el husillo en una posición cuando sea necesario.

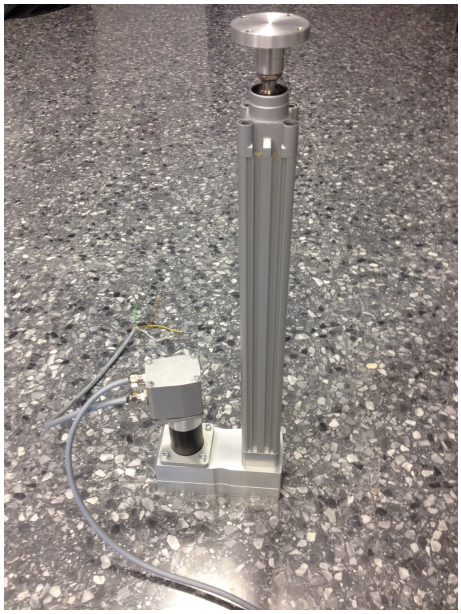
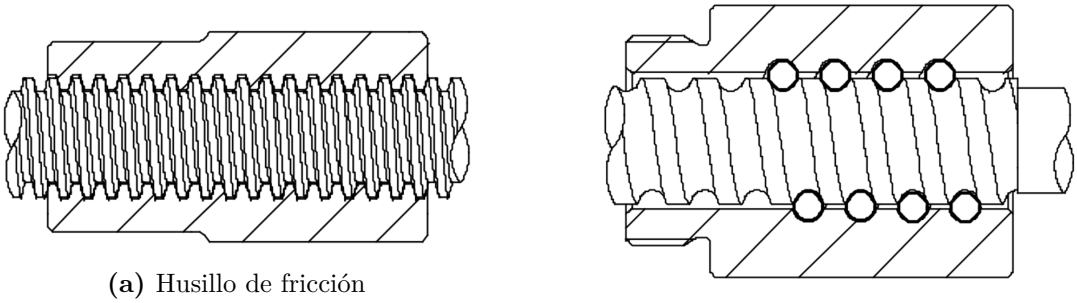


Figura 2.8: Imagen de un actuador lineal del robot robot paralelo4DOF.

El husillo de bolas

Se trata de un mecanismo giratorio con elementos fileteados (puede ser un husillo de fricción o un husillo de bolas. Ver Figura 2.9) que convierte el movimiento giratorio de un motor en un movimiento lineal. De esta manera, el vástago se desplaza hacia delante y hacia atrás. El vástago puede estar asegurado contra giros y protegido contra sobrecarga mediante un acoplamiento de deslizamiento. Además, puede detectarse la posición del vástago mediante detectores de proximidad [5].



(a) Husillo de fricción

(b) Husillo de bolas

Figura 2.9: Esquema de los dos tipos de husillos [5]

Las aplicaciones son diferentes dependiendo del tipo de husillo que se utilice.

- **Husillo de fricción:** Está diseñado para la colocación lenta de masas con fuerzas elevadas.
- **Husillo de bolas:** Está diseñado para el posicionamiento de precisión a velocidades elevadas.

Una gran cantidad de actuadores lineales utilizan los husillos de bolas por su alto grado de eficiencia, no reaccionan tan bruscamente en las paradas comparado con los husillos de fricción y además son más robustos frente a impactos [36].

Dada la información presentada se ha optado por un husillo de bolas, concretamente se trata del modelo DNCE-32-300-BS "10" P-Q del fabricante Festo. El cual presenta las siguientes características:

Tamaño/Tipo de rosca	**32-BS**
Paso del husillo	**3**
Forma constructiva	Cilindro eléctrico con husillo giratorio
Velocidad máxima $[m/s]$	0.15
Número de revoluciones máx. $[1/min]$	3000
Aceleración máxima $[m/s^2]$	6
Constante del avance (Paso del husillo) $[mm/U]$	3
Precisión en la repetición $[mm]$	± 0.02
Perfil del cilindro, culata trasera, émbolo	Aluminio
Husillo, rodamiento de bolas, vástago	Acero
Tuerca del husillo	Acero
Junta rascadora	Caucho nitrílico

Tabla 2.1: Especificaciones técnicas del husillo [5]

El motor DC

Los motores eléctricos, como máquina que transforma energía eléctrica en energía mecánica, son los elementos del sistema encargados de hacer el trabajo convirtiéndose en la "bestia de carga" del sistema. Esto supone que independientemente de lo avanzado que sea el control, los motores van a influir en gran medida en el comportamiento del sistema [37].

Actualmente en el mercado existe una gran variedad de motores eléctricos así como una infinidad de aplicaciones para estos. Dependiendo de la señal de alimentación estos pueden ser AC o DC, otra diferencia está en si el campo magnético del rotor está generado por inducción o por el contrario utiliza imanes permanentes, si hace uso de escobillas o por el contrario carece de ellas, y diferentes combinaciones más. Para el proyecto de los robots paralelos se ha optado por utilizar un motor alimentado con corriente continua con escobillas y que tiene imanes permanentes en el rotor. Esta decisión se justificará a continuación.

Los motores AC, por lo general, se utilizan en aplicaciones donde se requiere gran potencia, mientra que el segmento de mercado en el que predominan los motores DC trabaja con bajas potencia [38]. Aunque los motores de corriente alterna son más simples y robustos, en [37] se nos presentan una serie de ventajas que hacen que los motores DC sean preferibles para ciertas aplicaciones:

1. Los motores de corriente continua son capaces de entregar un mayor par de eje en el encendido que el par en plena carga y tiene un mayor ratio de par/inercia.
2. Un amplio rango de velocidades de operación y la simplicidad del control de velocidad los hacen preferibles para aplicaciones con variación de velocidades. Por otro lado, para aplicaciones con que necesitan una velocidad constante y precisa como relojes, etc., se puede utilizar motores AC síncronos ya que funcionan intrínsecamente a velocidad constante.
3. Presentan una alta eficiencia comparados con los motores de corriente alterna y requieren de drivers y circuitos de control más simples.
4. Se pueden implementar como freno dinámico y regenerativo (la energía generada por el motor sirve para alimentar una parrilla de resistencias o la alimentación respectivamente).
5. Los motores DC responden rápidamente a cambios en la señal de control gracias a la baja constante de tiempo del sistema eléctrico y el alto ratio par/inercia.
6. Además, para un mismo rango de potencia, los motores de corriente continua son más pequeños y ligeros comparados con los motores de corriente alterna.

Por otro lado, [37] también destaca la limitación que suponen las escobillas en los motores de corriente continua. Estás se deslizan sobre una anillo conectado al rotor que permite la conmutación eléctrica en la armadura. Aunque las escobilla de grafito, comercialmente muy populares, presentan baja resistencia eléctrica, bajo coeficiente de fricción y muy bajo desgaste debido al vapor de agua en el aire, las escobillas añaden fricción al sistema y acaban desgastándose por lo que requieren mantenimiento.

En general los motores con escobillas tienen a ser más económicos. A pesar que actualmente el aumento de aplicaciones que requieren motores *brushless* ha hecho que el precio de estos baje, los motores con escobillas siguen teniendo una cuota de mercado que les hace ser los más económicos.
La corriente que alimenta el bobinado de la armadura llega a través de las escobillas que se deslizan alrededor del un conmutador. El conmutador permite que el flujo magnético de la armadura esté orientado siempre hacia al campo que generan los imanes permanentes. La disposición consiste en un conjunto de guías montadas cerca del conmutador. Las guías son unas cajas metálicas abiertas por arriba y por abajo más un resorte, es posible encontrar infinidad

de variaciones de este mecanismo. La escobilla se introduce en la guía y es empujada contra el conmutador por el resorte. El material de la escobilla es normalmente grafito natural, carbón duro, electro-grafito, o grafito metálico. La calidad de la conmutación dependerá del voltaje y los materiales utilizados [38].

Por otro lado, el uso de imanes permanentes para los motores eléctricos se ha popularizado debido a las propiedades únicas que otorgan las tierras raras a los imanes, aportándoles una alta densidad energética, alta densidad de flujo magnético y una alta coercitividad. Todo esto añadido a un mercado maduro ha facilitado la fabricación de mecanismos con una bajo momento de inercia, un alto ratio fuerza/peso, alta eficiencia, mejor respuesta dinámica, volumen reducido y una alta capacidad de sobrecarga de par [38].

Para el robot *paralelo4DOF* se ha utilizado un motor DC de 150W con escobillas de grafito de la compañía *Maxon* (ver Figura 2.10).

Este motor presenta las siguientes características técnicas [6] recogidas en las tabla 2.2

Figura 2.10: Imagen del motor Maxon utilizado. [6]

Maxon Motor RE 40 ⌀40 mm Graphite Brushes 150Watt	
Valores a tensión nominal	
Voltaje nominal	$24V$
Velocidad nominal	$7580rpm$
Corriente sin carga	$137mA$
Par nominal (máx. par constante)	$177mNm$
Corriente nominal (máx. corriente constante)	$6A$
Par inicial	$2420mNm$
Corriente de arranque	$80.2A$
Máx. eficiencia	$91\,\%$
características	
Resistencia entre terminales	0.299Ω
Inductancia entre terminales	$0.082mH$
Constante de par	$30.2mNm/A$
Constante de velocidad	$317rpm/A$
Gradiente de velocidad/par	$3.14rpm/mNm$
Inercia del rotor	$142gcm^2$
Datos mecánicos	
Tipo de rodamiento	rodamiento de bolas
Velocidad máxima	$12000rpm$
Juego axial	$0.05 - 0.15mm$
Juego radial	$0.025mm$
Máx. carga axial	$5.6N$
Máx. carga radial	$28N$
Otras especificaciones	
Número de pares de polos	1
Numero de segmentos del conmutador	13
Peso	$480g$

Tabla 2.2: Especificaciones técnicas del motor Maxon [6]

El freno

Además, con la intención de asegurar que el robot *paralelo4DOF* se mantiene en la posición deseada cuando el motor no recibe alimentación y por lo tanto los actuadores lineales no se retraen debido a la fuerza de la gravedad, se ha dispuesto de un freno conectado al motor (ver Figura 2.11). Las características de este freno, también del fabricante Maxon, se recogen en la tabla 2.3.

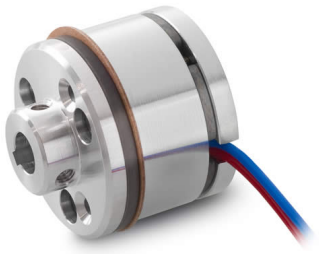


Figura 2.11: Imagen del freno Maxon utilizado. [7]

Maxon Brake AB 28, 24 VDC, 0.4 Nm	
Peso	$50g$
Inercia	$10gcm^2$
Velicidad máxima	1600rpm
Voltaje nominal	$24V$
Par sostenido	$400mNm$

Tabla 2.3: Especificaciones técnicas del freno Maxon [7]

Los encoders

Las articulaciones del robot están controladas por actuadores lineales que consisten en un tornillo sin fin y una reductora conectada a un motor brushed. Por lo tanto, el desplazamiento lineal del actuador se puede calcular conocidas las revoluciones del tornillo sin fin. Los encoders miden estas revoluciones. La posición analógica del motor puede ser determinada por una señal sinusoidal del encoder, o más común es determinar la posición digital del motor contando los pulsos que se obtienen al hacer pasar la señal sinusoidal por un comparador [39].

A fin de obtener una buena respuesta dinámica es esencial tener en cuenta la precisión del encoder. Muchos sistemas de control de alta precisión para motores eléctricos utilizan los encoders rotatorios [40].

Un caso concreto de encoder rotatorio es el encoder óptico, dónde un foco de luz que apunta a un fotodetector es periódicamente interrumpido por un patrón codificado en opaco/translúcido impreso en un disco dispuesto en el eje en el que se pretende medir la velocidad. Estos discos pueden estar hechos de cromo sobre vidrio, metal grabado o tinta impresa en plástico, por ejemplo [41].

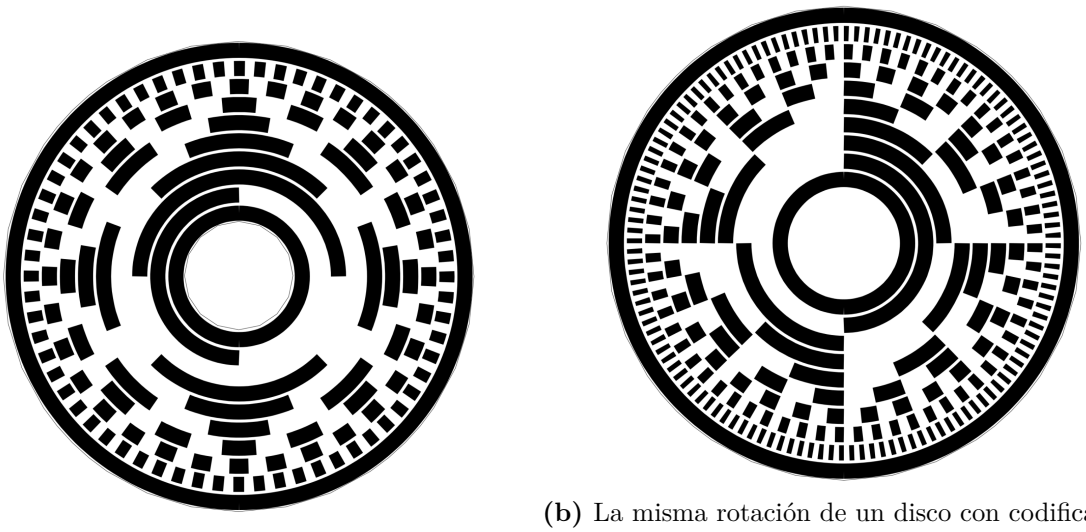
Los encoders rotatorios pueden ser absolutos o incrementales. Mientras que los encoders absolutos devuelven el valor de la posición absoluta del encoder por lo que para aumentar la resolución es necesario aumentar el número de conexiones, los encoders incrementales devuelven el número de pulsos a medida que rotan. Por lo que sabiendo el número de pulsos por radián podemos saber el desplazamiento entre lecturas. Esto hace que su construcción sea más simple que la de

los encoders absolutos [42], no necesita aumentar el número de cables para mejorar su resolución [40]. Si no se requiere conocer la posición de manera no volátil, los encoders incrementales son más sencillos de implementar y de coste mucho que los encoders absolutos para la misma resolución [43].

Los encoders ópticos absolutos

Los encoders absolutos se usan normalmente en aplicaciones donde se da una baja velocidad de rotación y donde es necesario conocer la posición del eje sin ser admisible perder esta información cuando se interrumpe la alimentación. Para ello se dispone de varios anillos concéntricos, cada uno con un patrón distinto, que interrumpen la fuente de luz devolviendo una única "palabra" para cada posición (ver Figura 2.12). La asignación de una pista dedicada para cada bit de resolución hace necesario aumentar el tamaño del disco y con ello reducir la tolerancia a vibraciones. Se toma como norma general que cada pista adicional en el disco duplica la resolución pero cuadruplica el costei [44].

La potencial desventaja de los encoders absolutos es su trama de datos en paralelo, los cuales requieren una interfaz más compleja debido al gran número de conexiones que necesitan. Por ejemplo, un encoder absoluto de 13-bits usando señales de salida complementarias por inmunidad al ruido necesitaría 28 conexiones (13 pares de conexiones para las señales más la alimentación y masa), en cambio, un encoder incremental únicamente necesitaría seis conexiones [45].



(a) El incremento de una posición en sentido antihorario hará que únicamente cambie un bit.

(b) La misma rotación de un disco con codificación binaria hará que todos los bits cambien en el caso particular (255 a 0) ilustrado por la linea de referencia que marca las 12h

Figura 2.12: Rotación de un disco absoluto con codificación de 8-bits [8]

Los encoders ópticos incrementales

Estos, relativamente baratos, dispositivos son ideales para ser utilizados para medir la velocidad en sistemas de control de velocidades medias o altas, pero presentan problemas de ruido y estabilidad a bajas velocidades debido a que el error está cuantificado [46].

La señal de salida de un encoder incremental son dos señales cuadradas desfasadas 90°, este desfase nos permite saber la dirección de rotación del rotor. Esto también nos permite cuadruplicar el numero de pulsos por revolución (PPR) detectando los flancos de la señal cuadrada. [40]. Para aplicaciones en las que existen continuamente rotaciones de más de 360° muchos encoders incluyen un tercer canal que envía la señal cada vez que el eje pasa por 0 y por lo tanto completa una vuelta (ver Figura 2.13. Por lo tanto la posición del eje se puede calcular a partir de los pulsos contados a partir de este tercer canal, o canal *Índice*. El mayor inconveniente es que se debe esperar a volver a detectar el 0 cada vez que se interrumpe la alimentación [43].

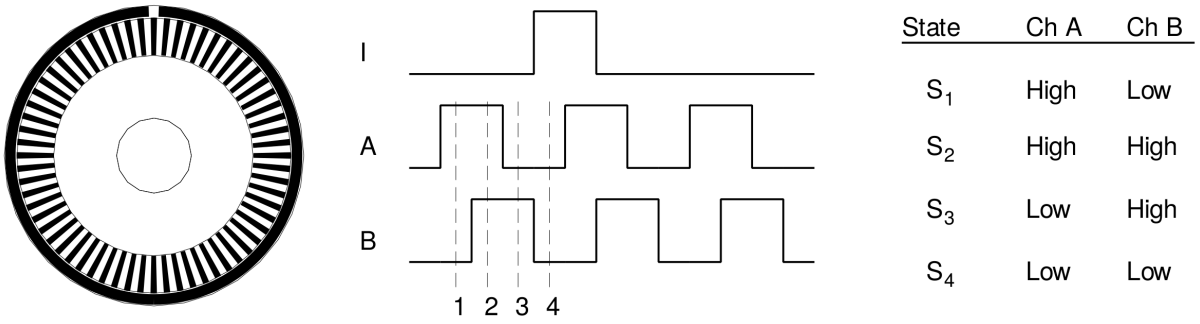


Figura 2.13: Ejemplo de como la relación de fases entre el canal A y el canal B puede ser utilizada para determinar la dirección de rotación con un encoder en cuadratura. La única ranura del disco exterior permite detectar cuando el eje pasa por cero [8].

La velocidad de rotación se puede obtener de dos formas: contando el número de pulsos en un tiempo dado, i.e. un frecuenciómetro, o midiendo el tiempo de los pulsos, i.e. un periodímetro. En este caso se ha utilizado el primer método por se más simple [42]. Si definimos n, como el número de pulsos en un tiempo fijo T, y siendo α el ángulo entre pulsos tenemos que el ángulo recorrido es:

$$\alpha_t = n \cdot \alpha \tag{2.1}$$

y por tanto la velocidad angular se define como:

$$\omega_r = \alpha_t / T \tag{2.2}$$

Dado que α_t es un múltiplo entero de α, se puede cuantificar el error máximo como α/T [40].

Además, cuando el tiempo entre pulsos es mayor que el periodo de muestreo en bajas velocidades, el error aumenta pudiendo llegar a dar problemas de estabilidad en el controlador.Esta velocidad mínima viene dada por la siguiente expresión:

$$\omega_{min} = \frac{90 \cdot f_s \cdot \alpha}{1 - 4 \cdot f_s \cdot T} \tag{2.3}$$

Siendo $f_s(Hz)$ la respuesta en frecuencia del controlador utilizado [42].

PCL-833

El *robot3DOF* se compone de tres actuadores lineales que disponen de un encoder cada uno, para ellos se dispuso de la tarjeta *PCL-833* de *Advantech* con tres canales de entrada para encoders en cuadratura [47].

En la siguiente página se recogen los datos técnicos correspondiente a la tarjeta *PCL-833* [48].

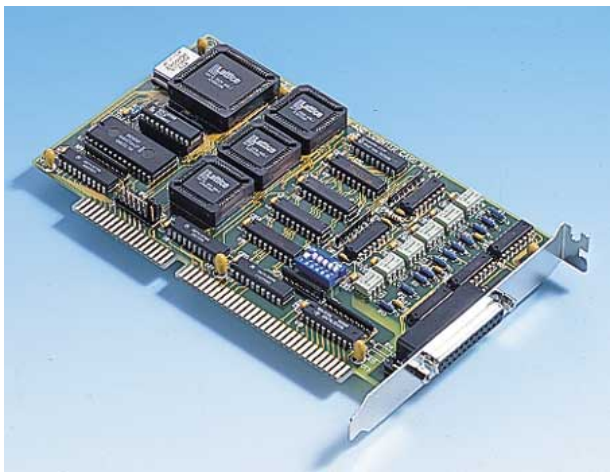


Figura 2.14: Imagen de la tarjeta PCL-833

Interfaz

- **Entrada**
  Cuadratura (fase A/B)
- **Cuentas por encoder**
  x1, x2, x4 (Configurable por S/W)
- **Rango de entrada**
  12 V máx.
- **Driver**
  Diferencial o single-ended
- **Protección de aislamiento**
  $2500V_{RMS}$ (óptico)
- **Frecuencia máxima de entrada**
  2.4 MHz

Contador/Timer

- **Canales**
  3
- **Resolución**
  24bits
- **Compatibilidad**
  5V/TTL
- **Modos del contador**
  3 (cuadratura, subida/bajada, pulso/dirección)
- **Canales con interrupción**
  Contador $0 \sim 2$
- **Filtro digital de ruido**
  4 etapas

Entradas digitales aisladas

- **Canales**
  5(Sin x 3 + DI0 + DI1)
- **Voltaje de entrada**
  0 Lógico: 1V máx.
  1 Lógico: 5V mín. (12V máx.)
- **Canales con interrupción**
  DI0, DI1
- **Protección de aislamiento**
  $2500V_{RMS}$ (ópticos)

General

- **Tipo de BUS**
  ISA
- **Certificaciones**
  Certificaciones
- **Conector**
  1 x DB25 conector hembra
- **Dimensión (L+H)**
  185 x 100 mm (7.3"x 3.9")
- **Consumo**
  Normal: 5V @ 700mA
  Máx.: 12V @ 15mA
- **Absorción de humedad**
  $5 \sim 95\,\%$ HR, no condensado (IEC 68-2-3)
- **Temperatura de funcionamiento**
  $0 \sim 60°C$
- **Temperatura de almacenamiento**
  $-20 \sim 70°C$

PCI - 1784

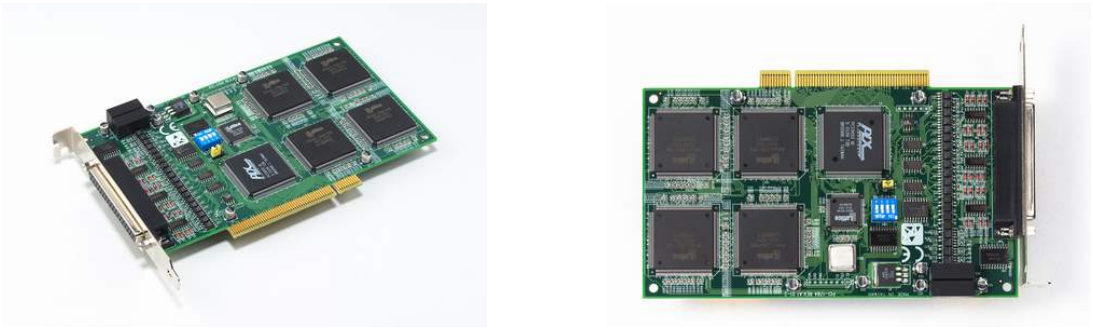


Figura 2.15: Imágenes de la tarjeta PCI-1784 [9]

Contador del encoder

- **Canales**
  4
- **Resolución**
  32bits
- **Modos del contador**
  3 (cuadratura, subida/bajada, pulso/dirección)
- **Frecuencia máxima de entrada**
  8 MHz para subida/bajada y pulso/dirección
  2 MHz para cuadratura sin filtro digital
  1 MHz para cuadratura con filtro digital
- **Filtro digital de ruido**
  4 etapas
- **Protección de aislamiento**
  $2500V_{DC}$
- **Compatibilidad**
  5V/TTL
- **Canales con interrupción**
  Contador $0 \sim 2$
- **Frecuencia del reloj**
  8, 4, 2 o 1 MHz
- **Voltajes de entrada**
  Single-ended:
  0 Lógico: 0.8V máx.
  1 Lógico: 2.8V máx. (12V máx.)
  Diferencial:
  0 Lógico: -0.2V máx. (-12V máx.)
  1 Lógico: 0.2V máx. (12 V máx.)

Entradas digitales aisladas

- **Canales**
  4
- **Voltaje de entrada**
  0 Lógico: 3V máx.
  1 Lógico: 10V máx. (30V máx.)
- **Canales con interrupción**
  Los 4 canales
- **Protección de aislamiento**
  $2500V_{DC}$
- **Respuesta del opto-aislador**
  $100\mu s$
- **Protección de sobrevoltaje**
  $70V_{DC}$

Salidas digitales aisladas

- **Canales**
  4
- **Voltaje de salida**
  0 Lógico: 0.8V máx.
  1 Lógico: 2V máx.
- **Capacidad de salida**
  50mA @ 0.8V
  -50mA @ 2V
- **Aislamiento**
  2500V_{DC}
- **Respuesta del opto-aislador**
  2μs

General

- **Tipo de BUS**
  Universal PCI V2.2
- **Conector**
  37-pin D-sub hembra
- **Dimensión (L+H)**
  175 x 100 mm (6.9"x 3.9")
- **Consumo**
  Normal: 5V @ 200mA
  MÁX.: 5V @ 450mA
- **Absorción de humedad**
  5 $\sim$ 95 % HR, no condensado (IEC 68-2-3)
- **Temperatura de funcionamiento**
  0 $\sim$ 60°C
- **Certificaciones**
  Certificaciones
- **Temperatura de almacenamiento**
  $-20 \sim 70$°C

Tarjeta con salidas analógicas

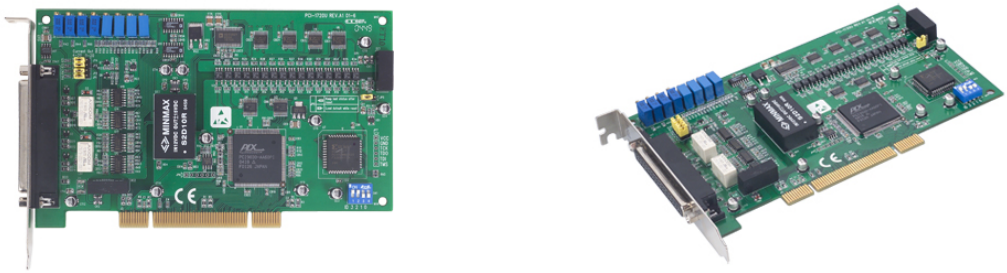


Figura 2.16: Imágenes de la tarjeta PCI-1720U [10]

Las acciones de control se envían a los motores como diferentes niveles de tensión a una tarjeta de control que lo convierte en PWM. Para poder enviar diferentes niveles de voltaje se ha dispuesto de la tarjeta de salidas analógicas *PCI-1720U* de *Advantech* [47]. Las especificaciones técnicas de esta tarjeta vienen recogidas a continuación [10]:

Salidas digitales

- **Canales**
  4
- **Resolución**
  12 bits
- **Rango de salida**

Bipolar (V)	$\pm 5, \pm 10$
Unipolar (V)	$0 \sim 5, 0 \sim 10$
Corriente (mA)	$0 \sim 20, 4 \sim 20$

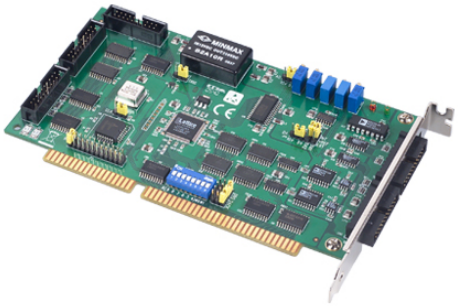
- **Slew rate**
  $2V/\mu s$
- **Protección de aislamiento**
  $2500V_{DC}$
- **Precisión**
  $\pm 1LSB$

General

- **Tipo de Bus**
  Universal PCI V2.2
- **Conector de I/O**
  1 x DB37 connector hembra
- **Consumo**
  5V @ 350mA
  12V @ 200mA

Tarjeta multifunción

Por último se ha implementado la tarjeta multifunción *PCL-812PG* de Advantech [47]. Esta tarjeta tiene varias entradas digitales y analógicas pensadas para un uso general, por ejemplo, se ha utilizado para activar y desactivar el freno o indicar el sentido del actuador lineal. A continuación se muestran las especificaciones técnicas [11]:



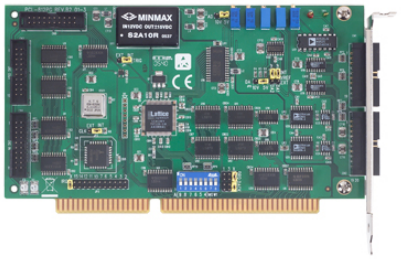


Figura 2.17: Imágenes de la tarjeta PCL-812PG [11]

Entradas analógicas

- **Canales**
  16
- **Resolución**
  12bits
- **Ratio máx. muestreo**
  $30kS/s$
- **Tamaño FIFO**
  0
- **Protección por sobrevoltaje**
  $30V_{p-p}$
- **Impedancia de entrada**
  $> 10M\Omega$
- **Modos**
  Por software, continuo o disparador externo
- **Rangos de entrada**
  $\pm 10, \pm 5, \pm 2.5, \pm 1.25, \pm 0.625, \pm 0.3125$
- **Precisión**
  0.4 % de la lectura $\pm 1LSB$

Salidas analógicas

- **Canales**
  2
- **Resolución**
  12 bits
- **Rango de salid**

Referencia interna	$0 \sim 5$, $0 \sim 10$
Referencia externa	± 10 máx.

- **Corriente máx.**
  $5mA$

Entradas digitales

- **Canales**
  16
- **Compatibilidad**
  5V/TTL
- **Voltaje de entrada**
  0 lógico: 0.8V
  1 lógico: 2.0V

Salidas digitales

- **Canales**
  16
- **Compatibilidad**
  5V/TTL
- **Voltaje de salida**
  0 lógico: 0.5V máx.
  1 lógico: 2.4V mín.
- **Corriente máx.**
  Entrada: 8.0mA
  Salida: 0.4 mA

Contador/Timer

- **Canales**
  1
- **Resolución**
  16 bits
- **Compatibilidad**
  5V/TTL
- **Frecuencia máx. de entrada**
  500kHz
- **Reloj de referencia**
  Interno: 2MHz
  Frecuencia externa: 10MHz
  Rango de voltaje externo: 5V/TTL

Software

Desarrollo de software basado en componente

Fue en la conferencia de Ingeniería del Software en 1968 cuando se introdujo por primera vez el concepto de componente de software [49]. A partir de entonces se ha desarrollado la ingeniería de software basado en componentes, *CBSE (Component-Based Software Engineering) o CBSD (Component-Based Software Development)*. Aunque esta tecnología no ha conseguido popularizarse a lo largo de los años, si que se han desarrollado un gran número de trabajos que aprovechan las ventajas del software basado en componentes, como son [16]:

- Reutilización
- Coste (económico y desarrollo)
- Calidad
- Integración

Para entender en qué consiste el *CBSD* tenemos que buscar la definición, dentro de la extensa bibliografía una de las definiciones más utilizadas es la presentada por [50], que define un componente de software como *una unidad funcional y básica de composición con interfaces y un contexto bien definidos*. Otras definiciones que encontramos lo definen como *una unidad que encapsula una funcionalidad, restringiendo el acceso a esa funcionalidad a interfaces explícitamente definidas* [51], o también como *una caja negra que proporciona y requiere servicios a través de interfaces bien definidas* [52]. Aunque cada autor presenta un concepto de lo que es un componente, en general comparten la idea de que un componente de software debe tener por lo menos dos partes estructurales [52].

- **Puertos.** Los puertos son canales de comunicación que utilizan los componentes para intercambiar información. En [52] se equipara un puerto de software con un conector en hardware.
- **Interfaces.** Las interfaces componen la capa más externa de un componente. Los componentes utilizan las interfaces para interactuar entre ellos, esto supone que hay que encontrar el equilibrio entre diseñar componentes demasiado específicos que resulten poco reutilizables y otros demasiado genéricos que utilicen demasiados recursos o muy complejos de implementar [16] [53].

Al igual que con la definición de componente, no hay un consenso entre los autores para la definición de *Desarrollo o Ingeniería del Software basado en Componentes (CBSD o CBSE)*, aunque se parecen bastante parecido. Por ejemplo, en [53] se dice que el *CBSD consiste en la creación de sistemas a partir de componentes reusables, manteniendo separado el desarrollo del componente en sí, con el desarrollo del sistema.* O una definición más simple sería que el CBSD consiste en *el desarrollo de sistemas distribuidos a partir de componentes de software reusables, independientes, y debidamente testeados* [54].

Aunque el CBSD presenta ventajas en otras muchas áreas de ingeniería de software, este trabajo se centra en el área de la robótica. Los sistemas robóticos generalmente son sistemas complejos, los cuales comparten unas propiedades que los convierten en candidatos ideales para implementar el CBSD [55]:

Complejidad: Un robot necesita de sensores, actuadores y algoritmos de control que hagan uso de estos. La complejidad aumenta con el número de elementos y la necesidad de intercambiar información entre ellos.

Flexibilidad: Generalmente los investigadores se centran en un único aspecto del sistema (i.e. visión artificial, mapeado, etc.). El investigador, por lo tanto, tiene que ser capaz de modificar una parte del sistema sin que esto afecte al resto del conjunto.

Distribución: Cuando un robot esta compuesto por más de un procesador o se pretende sustituir un procesador por otro, es importante que el código sea compatible entre procesadores.

Heterogenidad: Un robot puede estar integrado por una gran cantidad de componentes de diferentes fabricantes, por lo que es importante que estos se puedan integrar sin demasiados problemas.

Por lo visto hasta ahora los diferentes componentes de software deben seguir unos criterios para poder sacar el máximo provecho a los recursos disponibles. En [56] se proponen cuatro conceptos, *las cuatro C's*, a los que prestar atención cuando se pretenden diseñar los componentes. Además en [57] , se propone un quinto concepto pasando así a ser *las cinco C's.*

Computación: La computación consiste en el procesado de datos por parte del algoritmo que se necesita en la aplicación [58]. Esto se entiende como el intercambio de información, la gestión de recursos y el acceso a memoria.

Configuración: La gestión óptima de las interfaces supone encontrar el equilibrio entre un componente más configurable, con muchas interfaces y mayor complejidad, o un componente menos configurable, menos interfaces y por lo tanto más simple.

Comunicación: Se deberá decidir que protocolo de comunicación es el más adecuado para los componentes dependiendo del ancho de banda que necesitemos, la latencias que podamos tolerar o la fiabilidad de la información transmitida, entre otros aspectos.

Coordinación: Esto es la escala de prioridad de los componentes y la interacción entre ellos.

Y el quinto elemento introducido por [57]

Composición: Describe como se estructura cada uno de los componentes.

Como se ha comentado en las primeras líneas de esta sección, el desarrollo de software basado en componentes tiene algunos puntos débiles que han hecho que no se popularice. El autor en [55] destaca que es necesario que los componentes sigan un mismo estándar para poder funcionar. Por otro lado [59] argumenta que si se utiliza un *framework* (o *middleware*) para desarrollar el software, este siempre va a necesitar del *framework* para funcionar. Además, normalmente los *frameworks* no cumplen con los requisitos que se necesitan, por lo que se tiende a *implementar otro nuevo framework desde cero, teniendo que reinventar la rueda* [52], aunque el mismo autor también sugiere que estos problemas se resolverían si existiera un protocolo de *CBSE* que compartieran los *frameworks*.

Pese a los problemas del CBSD o los puntos en contra que puedan surgir, se ha considerado que el desarrollo de software por componentes es la mejor opción para este proyecto. Sobre todo porque reduce el coste y tiempo de un proyecto dinámico, esto se debe a que no es un producto con una única funcionalidad final. Al tratarse de un robot para un proyecto de investigación se requiere de un sistema que sea fácil de reprogramar y que una persona o equipo se pueda centrar en una única parte del sistema, i.e. se necesita que cada módulo sea lo más independiente posible. Además existen varias versiones del robot, por lo que la reusabilidad de los componentes entre distintos sistemas supone un importante ahorro de trabajo para los investigadores.

Para conseguir dicha modularidad en el sistema se ha hecho uso de un *middleware* o *framework*. Como se describe en [16], un *middleware* no es más que una capa de abstracción que se sitúa entre el sistema operativo y la capa de aplicación, siendo el principal objetivo de éste, abstraer al desarrollador de tareas complejas tales como la gestión de información que se transmite por la red, gestión de recursos compartidos, notificación de eventos, gestión de prioridades, etc. Para este proyecto se ha elegido *Orocos*, creado en el año 2000 a partir de la idea de Herman Bruynickx [60] e incluido entre los paquetes de ROS, otro *framework* más generalista. A continuación se describen con más detalle estos *frameworks*.

Robot Operating System [ROS]

> *Robot Operating System* (ROS) is un framework flexible para programar robots. Es una colección de herramientas, bibliotecas y normas de uso para simplificar la tarea de crear complejos y robustos comportamientos en robots dentro de una gran variedad de plataformas.
>
> ¿Por qué? Porque crear software robusto de uso general es complicado. Desde la perspectiva del robot, problemas que parecen triviales para las personas normalmente varían ampliamente entre tareas a realizar y el entorno. Tratar de afrontar estas variaciones is tan costoso que ningún individuo, laboratorio, o institución puede esperar hacerlo por si solo.
>
> Como resultado, ROS fue creado desde la base para fomentar el desarrollo colaborativo de software para robots. Por ejemplo, un laboratorio podría tener expertos en mapeado de entornos interiores, y podría contribuir a la creación de un sistema para producir mapas. Otro grupo podría tener expertos en el uso de mapas para navegación, e incluso otro grupo podría haber descubierto un sistema de visión por computadora que funciona bien reconociendo pequeños objetos dentro de un embrollo. ROS fu diseñado especialmente para que grupos como estos puedan colaborar y contribuir en el trabajo de los otros.
>
> (www.ros.org)

ROS ha demostrado ser una herramienta muy potente para el desarrollo de software para el control de robots. Podemos encontrar ejemplos de robots que utilizan este software en distintos ámbitos, desde su uso en robots comerciales como son los de la empresa española Robotnik (utilizados en otros proyectos del instituto Ai2), hasta los robots industriales de ABB, e incluso existen proyectos puramente de investigación como es el caso del proyecto *Robonaut 2* (ver figura 2.18), un robot humanoide diseñado por la NASA para operar en la Estación Espacial Internacional [61].

La unidad básica de ROS son los *packages* o paquetes. Un paquete puede contener programas completos, bibliotecas, archivos de configuración, datos almacenados o cualquier otra cosa que pueda resultar útil tenerla junta. Cuando una colección de paquetes tienen un control de versiones común y pueden ser liberados a la vez se le denomina repositorio.

Una de las ventajas que tiene ROS sobre otras plataformas es la opción de compartir estos paquetes. ROS anima a convertirse en desarrollador y subir paquetes que sean útiles para la

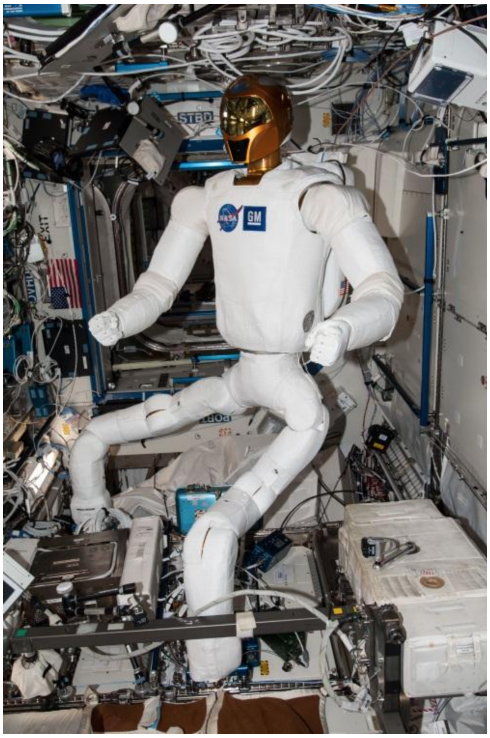


Figura 2.18: Robonaut 2 en la ISS. Fuente: NASA

comunidad. Estos paquetes podrán ser instalados por los usuarios haciendo uso del la función `rosinstall`, con algún gestor de paquetes del sistema operativo o directamente descargando el código fuente. Este es el caso de *Orocos*, un software para la gestión de procesos en tiempo real utilizado en los robots paralelos y que se desarrollará más adelante. *Orocos* es un proyecto independiente que se incorporó a ROS en forma de diferentes repositorios. Concretamente para el control de los robots paralelos se usa *Orocos Toolchain*, que se puede instalar con el repositorio *rtt_ros_integration*.

Aunque no es objeto de este trabajo el uso extenso de ROS, es conveniente desarrollar brevemente cómo ROS gestiona la comunicación entre diferentes procesos y los conceptos que se trabaja a lo largo del proyecto. Estos son los conceptos básicos que intervienen en el proceso de comunicación [61]:

- ***Nodes:*** Los nodos (o *nodes*) son procesos que realizan una tarea. ROS está diseñado para ser modular en una escala muy precisa, por lo que un robot puede llegar a utilizar muchos nodos. Por ejemplo, un nodo se encarga de almacenar el valor de un encoder, otro podría calcular la acción de control, otro mostrar por pantalla una representación gráfica del robot, etc.

- ***Master:*** El Master es un nodo que registra, gestiona y supervisa la comunicación entre nodos. Sin el Master, los nodos no son capaces de comunicarse.

- ***Messages:*** Los nodos se comunican entre sí enviando mensajes (o *messages*). Los mensajes son simples estructuras de datos que pueden contener tipos estándar de variables (int, float, bool, etc.) así como arrays. Similar a las estructuras en C.

- ***Topics:*** Los mensajes son dirigidos a través de un sistema de transporte con una semántica publica/subscribe (*publish/subscribe*). Un nodo envía un mensaje publicándolo en un *topic*. El *topic* es un nombre usado para identificar el contenido del mensaje, entonces el nodo interesado en cierto mensaje se subscribirá al *topic* correspondiente. Pueden haber varios *publisher* y *subscriber* concurrentes a un mismo *topic*, y un mismo nodo puede publicar o subscribirse a múltiples *topics*. En general, *publishers* y *subscribers* no se percatan de la existencia del otro. La idea is desacoplar la producción de información del consumidor. Lógicamente, se puede pensar en un *topic* como un bus de datos con un tipo de mensaje definido. Cada bus tiene un nombre, y todos pueden conectarse al bus para enviar o recibir mensajes siempre y cuando los mensajes son del tipo correcto.

- ***Services:*** El modelo publica/subscribe is bastante flexible, pero la comunicación en una única dirección no es óptima para interacciones del tipo petición/respuesta (*request/reply*), normalmente necesarias en un sistema distribuido. Las comunicaciones petición/respuesta se hacen vía servicios (o *services*), definidos por un par de estructuras de mensaje: uno para la petición y otro para la respuesta. Un nodo ofrece un servicio con un nombre y un cliente usa este servicio enviando un mensaje de petición y esperando el mensaje de respuesta.

- ***Bags:*** Se trata de un formato para guardas y reproducir datos de mensajes en ROS. Los *bags* son un importante mecanismo para almacenar datos, como los proporcionados por un sensor, que pueden ser complicados de recoger pero que son importantes para el desarrollo y el test de algoritmos.

El nodo ROS Master almacena la información de registros de los *topics* y *services* de los diferentes nodos. Los nodos se comunican con el Master para indicarle su información de registro, al comunicarse con el Master estos pueden recibir información acerca de los otros nodos registrados y realizar las conexiones apropiadas. El Master también realizará las diferentes llamadas (*callbacks*) a estos nodos cuando la información de registro cambie, lo que permite que los nodos

puedan ir creando nuevas conexiones a medida que los nuevos nodos se ejecutan.

Los nodos se conectan entre ellos directamente, el Master solo proporciona información de búsquedas, similar a un servidor DNS. Los nodos que se subscriben a un *topic* pedirán información sobre los nodos que publican ese *topic*, y se establecerá una conexión mediante un protocolo acordado. El protocolo más común usado en ROS es el TCPROS, el cual usa estándar TCP/IP sockets [61].

Orocos

> **¿Qué es Orocos?**
>
> *Orocos* es el acrónimo de ***O**pen **Ro**bot **Co**ntrol **S**oftware project*. El objetivo es desarrollar un *framework* de uso general, software libre y modular para el control de robots y maquinaria.
>
> Un *framework* es una colección de código fuente desde el que se pueden crear aplicaciones en un dominio particular [Johnson97]. Por lo tanto, el *framework* no es una aplicación en si, pero proporciona la infraestructura y las funcionalidades para crear aplicaciones en C++. Normalmente, los programadores de aplicaciones tienen que conectarse a algún "punto de acceso" específico para su aplicación, y por lo tanto, no puede ser proporcionado por el *framework*.
>
> [. . .]
>
> *Orocos* es un proyecto de software libre, por lo tanto su código y documentación están liberadas bajo *Free Software Licenses.*
>
> [. . .]
>
> (www.orocos.org)

Orocos es el software en el que se basaba, y se basa, el control del robot Paralelo3DOF en el momento de empezar este trabajo. La versatilidad que supone programar de forma modular y la gestión de recursos pensado para la ejecución en tiempo real han sido las razones para seguir confiando en Orocos para el control del robot Paralelo4DOF.

Dentro del proyecto Orocos coexisten tres herramientas: *Orocos Toolchain*, *Kinematics and Dynamics library* y *Bayesian Filtering library*[60].

- **Orocos Toolchain** Es un *framework* para la creación de componentes en tiempo real, manejo interactivo mediante *scripts*, maquinas de estado, procesos distribuidos y generación de código.
- **Kinematics and Dynamics library:** Es una biblioteca para el cálculo de cadenas cinemáticas, el cálculo tanto de cinemática directa como inversa en tiempo real y adaptación de bibliotecas para Python.
- **Bayesian Filtering library:** Es una biblioteca para el cálculo de *Dynamic Bayesian Networks*, filtros de Kalmann extendidos, y filtros de partículas (o métodos de Monte Carlo secuenciales).

Estas herramientas están actualmente integradas en ROS en forma de "paquetes".

Tanto el robot Paralelo3DOF como el robot Paralelo4DOF únicamente hacen uso de *Orocos Toolchain*. Este *framework* permite la creación y manipulación de módulos llamados componentes. Estos componentes son piezas de software que realizan una función básica y pueden transmitir información entre sí, de esta forma se crean aplicaciones conectando los diferentes módulos. El programa encargado de la gestión de los componentes y sus conexiones se denomina *deployer*, este a la vez ejecuta un *Orocos Program Script*(ops) con la información sobre los componentes que utiliza la aplicación, las conexiones y cómo se deben lanzar los distintos componentes.

La interfaz *componente* está definida por la clase *TaskContext*. Existen cinco formas diferentes por las cuales un componente de Orocos se puede interconectar: A través de *properties*, *events*, *methods*, *commands* y *data flow ports*. Todos estos tipos de interconexiones son opcionales y tanto su propósito como su uso están descritos en *The Orocos Component Builder's Manual*[60].

- *Properties:* Parámetros que pueden ser modificados durante el tiempo de ejecución, almacenados en archivos XML.
- *Events:* Se trata de un mecanismo de *callback* para comunicar (*publish*) un cambio a otro componentes(*subscriber*).
- *Methods:* Piezas de código que pueden ser llamadas por otros componentes para obtener resultados inmediatos, similar a una función en C.
- *Commands:* Estos son enviado por otro componente para indicarle al receptor que haga una función.
- *Data-Flow Ports:* Canales de comunicación para enviar datos entre componentes.

En el capítulo 4 se describen con más detalles los diferentes componentes creados para ambos robots paralelos. Toda la información sobre *Orocos Toolchain* se puede encontrar en la página oficial en *The Orocos Component Builder's Manual* [62].

Capítulo 3

Modelo cinemático y dinámico del robot paralelode 4 grados de libertad

Modelo cinemático.

Para realizar el control del robot es necesario conocer su cinemática y así poder calcular la posición y velocidad en cada instante. El cálculo detallado de la cinemática del robot está recogido en el documento interno del proyecto [4]. En esta sección se presentan los resultados necesarios para implementar el software de control del robot.

Modelación cinemática

Un robot manipulador puede ser considerado como una serie de enlaces rígidos unidos por articulaciones. Considerando un sistema de coordenadas para cada enlace rígido usando la notación de Denavit-Hartenberg podemos crear matrices de transformaciones homogéneas, A_i, que describen la relación de translación y rotación entre dos sistemas de coordenadas consecutivos. La relación entre cualquiera de lo sistemas de coordenadas vendrá dado por el producto de las matrices de transformación homogénea [63].

El robot paralelo4DOF tiene una configuración 3UPE-RPU. Esto indica que el robot tiene, como se ve representado en la figura 3.1, tres cadenas cinemáticas de tipo UPE (Universal-Prismática-Esférica) y una de tipo RPU (Rotacional-Prismática-Universal). Estas cadenas cinemáticas unen dos plataformas. Una plataforma fija en la base y otra móvil que será la que queremos controlar.

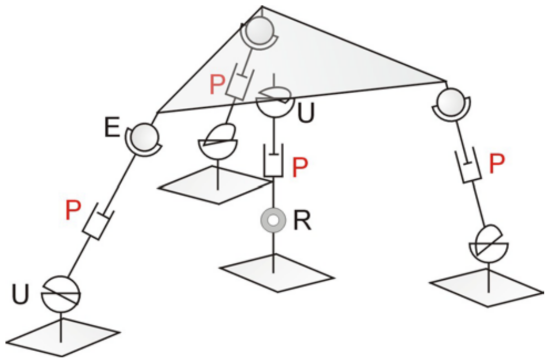


Figura 3.1: Robot 3UPE-RPU. Fuente [4]

De acuerdo a esto se establecen los sistemas de referencia fijo $\{O_F - X_F Y_F Z_F\}$ y móvil $\{O_M - X_M Y_M Z_M\}$ cómo se indica en la figura 3.2.

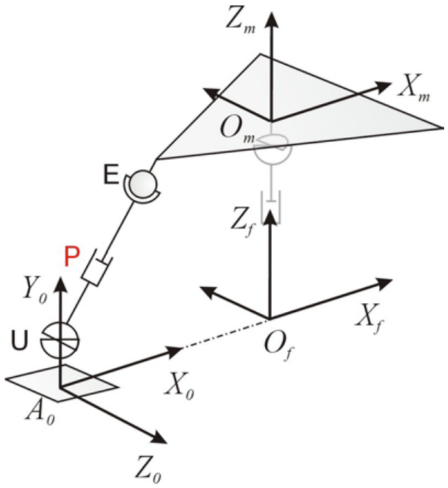


Figura 3.2: Sistemas de referencia fijo y móvil. Fuente [4]

Así mismo, se establece una relación entre el sistema de referencia fijo y un sistema de referencia fijo en la base de cada una de las cadenas cinemáticas o *patas*. Este nuevo sistema de referencia servirá para el cálculo de los parámetros de Denavit-Hartenberg y la matriz de transformaciones homogéneas A_i.

$$
\begin{aligned}
A_i &= Rot(z_i,\theta_i)Trans(z_i,d_i)Trans(x_i,a_i)Rot(x_i,\alpha_i) \\
&= \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & acos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & asin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3.1}
$$

Donde $Rot(z_i,\theta_i)$ es una rotación de θ_i sobre el eje z_i, $Trans(z_i,d_i)$ es una translación de d_i en el eje z_i, $Trans(x_i,a_i)$ es una translación de a_i en el eje x_i y $Rot(x_i,\alpha_i)$ es una rotación de α_i sobre el eje x_i. El subíndice i indica que se trata de una translación de la cadena cinemática $i-1$ a la cadena cinemática i [63].

Cadenas cinemáticas UPE.

Para modelizar las patas se asigna un sistema de referencia a cada uno de los pares. El par universal(U) se modeliza mediante dos pares rotacionales(R) perpendiculares entre sí. Los denotaremos con los subíndices 1 y 2. El par prismático(P) los denotaremos con el subíndice 3 y por último el par esférico (E) se modeliza mediante 3 pares rotacionales a los que se asignan los subíndices 4,5 y 6 [4]. La asignación de los sistemas de referencia correspondientes a los nudos U, P y E se ve representada en la figura 3.3.

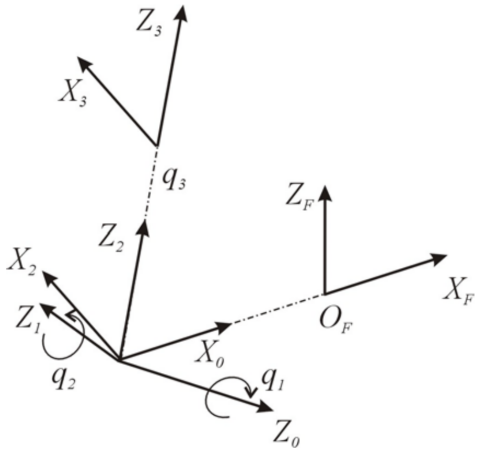


Figura 3.3: Asignación de los sistemas de referencia correspondientes a los nudos U, P y E. Fuente [4]

En la tabla 3.1 se recogen los parámetros para crear las transformaciones homogéneas A_i para las barras UPE.

Tabla 3.1: Tabla de Denavit-Hartenberg para cada barra UPE

Barra i-ésima	α_i	a_i	d_i	θ_i
1	$-\frac{\pi}{2}$	0	0	q_1
2	$\frac{\pi}{2}$	0	0	q_2
3	0	0	q_3	0
4	$\frac{\pi}{2}$	0	0	q_4
5	$\frac{\pi}{2}$	0	0	q_5
6	$\frac{\pi}{2}$	0	0	q_6

Cadena cinemática RPU

Al igual que se ha hecho con las barras UPE, se asigna un sistema de referencia a todos los pares del sistema. Al par rotacional lo denotamos con el subíndice 1, el par prismático con el subíndice 2 y y el par universal se modeliza mediante dos pares rotacionales, 3 y 4.

La asignación de los sistemas de referencia se ve representada en la figura 3.4.

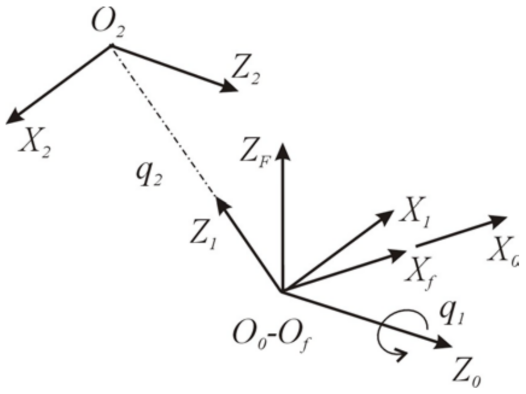


Figura 3.4: Asignación de los sistemas de referencia correspondientes a los nudos R, P y U. Fuente [4]

En la tabla 3.2 se recogen los parámetros para crear las transformaciones homogéneas A_i para la barra central.

Tabla 3.2: Tabla de Denavit-Hartenberg para la barra RPU

Barra i-ésima	α_i	a_i	d_i	θ_i
1	$-\frac{\pi}{2}$	0	0	q_1
2	$-\frac{\pi}{2}$	0	q_2	π
3	$+\frac{\pi}{2}$	0	0	q_3
4	0	0	0	q_4

Posición por cinemática inversa

El objetivo es determinar las coordenadas correspondientes a los actuadores prismáticos, q_3 para las patas exteriores y q_2 para la pata central, a partir de las coordenadas de la plataforma móvil $\{x_m, y_m, z_m, \phi, \theta, \psi\}$.

Como se aprecia en la figura 3.5, se han definido los puntos A_0, B_0 y C_0 como los puntos de unión entre la plataforma fija y las patas exteriores; y los puntos A, B y C como los puntos de unión entre la plataforma móvil y las patas exteriores.

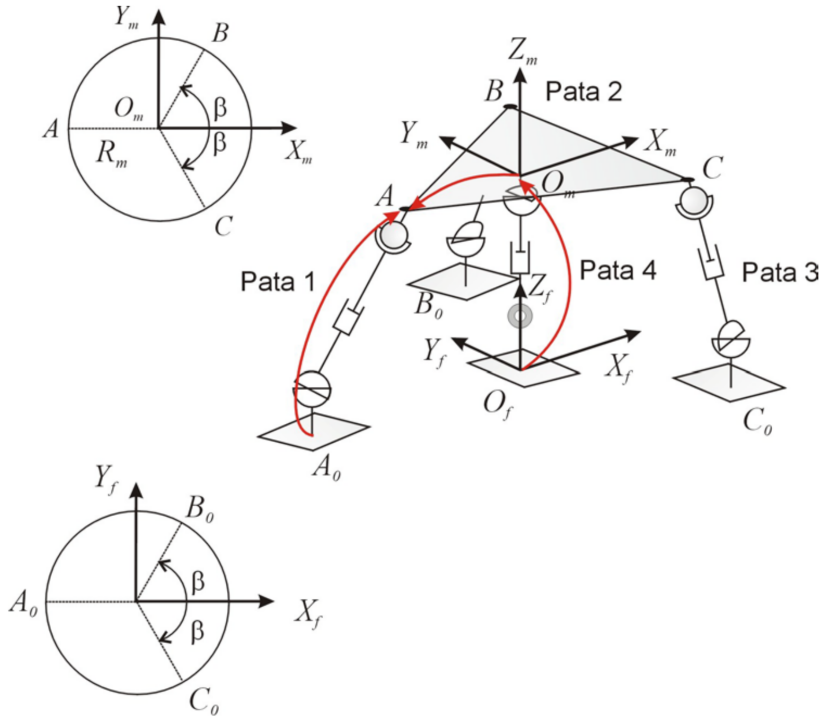


Figura 3.5: Obtención de las coordenadas de los puntos A, B y C por dos caminos diferentes.

Primero, las coordenadas generalizadas q_1, q_2 y q_3 de las patas exteriores y q_1 y q_2 de la pata central se calculan mediante la posición de los puntos A, B y C a los que se llega por dos caminos diferentes como se muestra en la figura 3.5. Y segundo, las coordenadas q_4, q_5 y q_6 de la patas

exteriores y q_3 y q_4 de la central se calculan haciendo uso de la orientación de la plataforma móvil.
La relación entre el sistema de referencia fijo $\{O_f - X_f Y_f Z_f\}$ y el fijo de cada pata viene dado por las expresiones siguientes:

Pata 1
$$^{f}R_{A0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \vec{r}_{O_f A} = \begin{bmatrix} -R \\ 0 \\ 0 \end{bmatrix}$$

Pata 2
$$^{f}R_{B0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \vec{r}_{O_f B} = \begin{bmatrix} R \cdot cos(\beta) \\ R \cdot sen(\beta) \\ 0 \end{bmatrix}$$

Pata 3
$$^{f}R_{C0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \vec{r}_{O_f C} = \begin{bmatrix} R \cdot cos(\beta) \\ -R \cdot sin(\beta) \\ 0 \end{bmatrix}$$

Pata 4
$$^{f}R_{0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \vec{r}_{O_f O_f} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Plataforma móvil
$$^{f}R_{m} = R_Z(\psi) \cdot R_Y(\theta) \cdot R_X(\phi) \qquad \vec{r}_{O_f O_m} = \begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix}$$

Las coordenadas de los puntos A, B y C por los dos caminos diferentes vienen dadas por:

Para el punto A

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} + {}^{f}R_m \cdot \begin{bmatrix} -R_m \\ 0 \\ 0 \end{bmatrix} = \Big[H_{O_f A_0} \cdot H_{01}(q_1) \cdot H_{12}(q_2) \cdot H_{23}(q_3)\Big] \tag{3.2a}$$

Para el punto B

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} + {}^{f}R_m \cdot \begin{bmatrix} R_m \cdot cos(\beta) \\ R_m \cdot sen(\beta) \\ 0 \end{bmatrix} = \Big[H_{O_f B_0} \cdot H_{01}(q_1) \cdot H_{12}(q_2) \cdot H_{23}(q_3)\Big] \tag{3.2b}$$

Para el punto C

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} + {}^{f}R_m \cdot \begin{bmatrix} R_m \cdot cos(\beta) \\ -R_m \cdot sen(\beta) \\ 0 \end{bmatrix} = \Big[H_{O_f C_0} \cdot H_{01}(q_1) \cdot H_{12}(q_2) \cdot H_{23}(q_3)\Big] \tag{3.2c}$$

Para el punto O_m

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} = \Big[H_{O_f 0} \cdot H_{01}(q_1) \cdot H_{12}(q_2)\Big] \tag{3.2d}$$

Desarrollando las expresiones anteriores y teniendo en cuenta que el par R de la base impone las restricciones $y_m = 0$ y $\theta = 0$, obtenemos el siguiente conjunto de ecuaciones para cada una de las patas:

Pata 1

$$cos(q_1) \cdot sin(q_2) \cdot q_3 - R = x_m - R_m \cdot cos(\theta) \cdot cos(\psi) \tag{3.3a}$$

$$cos(q_2) \cdot q_3 = R_m \cdot cos(\theta) \cdot sin(\psi) \tag{3.3b}$$

$$sin(q_1) \cdot sin(q2) \cdot q_3 = z_m + R_m \cdot sin(\theta) \tag{3.3c}$$

Pata 2

$$\begin{aligned} cos(q_1) \cdot sin(q_2) \cdot q_3 + R \cdot cos(\beta) = x_m + R_m \cdot cos(\beta) \cdot cos(\theta) \cdot cos(\psi) + \\ + R_m \cdot sin(\beta) \cdot (-sin(\psi) \cdot cos(\phi) + cos(\psi) \cdot sin(\theta) \cdot sen(\phi)) \end{aligned} \tag{3.4a}$$

$$cos(q_2) \cdot +R \cdot sin(\beta) = R_m \cdot cos(\beta) \cdot cos(\theta) \cdot sin(\psi) + R_m \cdot sin(\beta) \cdot (cos(\psi) \cdot cos(\phi) + sin(\psi) \cdot sin(\theta) \cdot sin(\phi)) \tag{3.4b}$$

$$sin(q_1) \cdot (q_2) \cdot (q_3) = z_m + R_m \cdot (-cos(\beta) \cdot sin(\theta) + cos(\theta) \cdot sin(\phi) \cdot sin(\beta)) \tag{3.4c}$$

Pata 3

$$cos(q_1) \cdot sin(q_2) \cdot q_3 + R \cdot cos(\beta) = x_m + R_m \cdot cos(\beta) \cdot cos(\theta) \cdot cos(\psi) + +R_m \cdot sin(\beta) \cdot (-sin(\psi) \cdot cos(\phi) + cos(\psi) \cdot sin(\theta) \cdot sen(\phi)) \tag{3.5a}$$

$$cos(q_2) \cdot +R \cdot sin(\beta) = R_m \cdot cos(\beta) \cdot cos(\theta) \cdot sin(\psi) + R_m \cdot sin(\beta) \cdot (cos(\psi) \cdot cos(\phi) + sin(\psi) \cdot sin(\theta) \cdot sin(\phi)) \tag{3.5b}$$

$$sin(q_1) \cdot (q_2) \cdot (q_3) = z_m + R_m \cdot (-cos(\beta) \cdot sin(\theta) + cos(\theta) \cdot sin(\phi) \cdot sin(\beta)) \tag{3.5c}$$

Pata 4

$$q_2 \cdot sin(q_1) = -x_m \tag{3.6a}$$

$$q_2 \cdot cos(q_1) = z_m \tag{3.6b}$$

$$sin(q_1) \cdot (q_2) \cdot (q_3) = z_m + R_m \cdot (-cos(\beta) \cdot sin(\theta) + cos(\theta) \cdot sin(\phi) \cdot sin(\beta)) \tag{3.6c}$$

El problema de cinemática inversa tiene solución analítica para las coordenadas generalizadas q_1, q_2 y q_3. Se deberá comprobar también que los valores de q_3 están dentro del rango que permite el actuador físico.

Como se ha indicado anteriormente, el resto de coordenadas generalizadas se obtienen calculando la orientación de la plataforma móvil por dos caminos diferentes.

Para las patas exteriores tendremos que:

$$^{f}R_3(q_1, q_2, q_3) \cdot {}^{3}R_6(q_4, q_5, q_6) = {}^{f}R_m(\phi, \theta, \psi)$$

Donde ${}^{3}R_6(q_4, q_5, q_6)$ es la relación entre el último sistema de referencia antes del par prismático y el último sistema de referencia correspondiente al par prismático. O expresado de otra manera:

$$^3R_6(q_4, q_5, q_6) = (^fR_3(q_1, q_2, q_3))^T \cdot {}^0R_m(\phi, \theta, \psi) = R^*$$

$$R^* = \begin{bmatrix} cos(q_4) \cdot cos(q_5) \cdot cos(q_6) + sin(q_4) \cdot sin(q_6) & cos(q_4) \cdot sin(q_5) & cos(q_4) \cdot cos(q_5) \cdot sin(q_6) - sin(q_4) \cdot cos(q_6) \\ sin(q_4) \cdot (q_5) \cdot cos(q_6) - cos(q_4) \cdot sin(q_6) & sin(q_4) \cdot sin(q_5) & sin(q_4) \cdot cos(q_5) \cdot sin(q_6) + cos(q_4) \cdot cos(q_6) \\ sin(q_5) \cdot cos(q_6) & -cos(q_5) & sin(q_5) \cdot sin(q_6) \end{bmatrix}$$

y resolviendo se obtiene que

$$q_4 = arctan(\frac{R^*_{2,2}}{sin(q_5)}, \frac{R^*_{1,2}}{sin(q_5)}) \tag{3.7a}$$

$$q_5 = arctan(+\sqrt{(R^*_{1,2})^2 + (R^*_{3,3})^2}, -R^*_{3,2}) \tag{3.7b}$$

$$q_6 = arctan(\frac{R^*_{3,2}}{sin(q_5)}, \frac{R^*_{3,1}}{sin(q_5)}) \tag{3.7c}$$

La matriz R^* será singular cuando q_5 sea 0 o múltiplo de 2π dando lugar a infinitas soluciones de q_4 y q_6.

Se procede de manera análoga para la pata 4.

$$^fR_0 \cdot {}^0R_1(q_1) \cdot {}^1R_2(q_2) \cdot {}^2R_3(q_3) \cdot {}^3R_4(q_4) = {}^fR_m$$

de donde,

$$^2R_4(q_3, q_4) = (^fR_0 \cdot {}^0R_1(q_1)^1R_2(q_2))^T \cdot {}^fR_m = R^*$$

$$R^* = \begin{bmatrix} cos(q_3) \cdot cos(q_4) & -cos(q_3) \cdot sin(q_4) & sin(q_3) \\ sin(q_3) \cdot cos(q_4) & -sin(q_3) \cdot sin(q_4) & -cos(q_3) \\ sin(q_4) & cos(q_4) & 0 \end{bmatrix}$$

dando como resultados

$$q_3 = arctan(\sqrt{(R^*_{2,1})^2 + (R^*_{2,2})^2}, \sqrt{(R^*_{1,1})^2 + (R^*_{1,2})^2}) \tag{3.8a}$$

$$q_4 = arctan(\sqrt{(R^*_{1,2})^2 + (R^*_{2,2})^2}, \sqrt{(R^*_{1,1})^2 + (R^*_{2,1})^2}) \tag{3.8b}$$

El uso de las coordenadas relativas al par esférico de las patas exteriores y el par universal de la pata central se limita a comprobar que los valores se encuentran dentro del rango que permiten los pares.

Posición por cinemática directa

Dados los valores de las coordenadas generalizadas correspondientes a los actuadores se pretende obtener la posición y orientación de la plataforma móvil así como las restantes coordenadas generalizadas. El problema se ha abordado numéricamente, primero se ha obtenido la posición del sistema de referencia móvil y las coordenadas generalizadas de las patas, representado gráficamente en 3.6. Desarrollando las ecuaciones de restricción (3.2) para la cinemática directa se obtienen las siguientes expresiones [4],

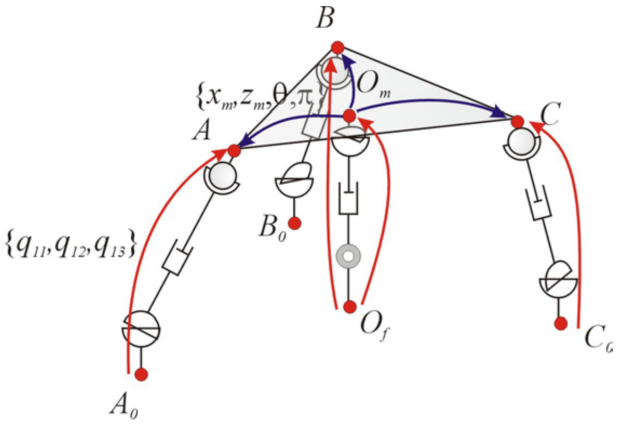


Figura 3.6: Planteamiento gráfico de las ecuaciones de restricción. Fuente [4]

$$\Phi_1 \equiv \cos(q_{11}) \cdot \sin(q_{12}) \cdot q_{13} - R - x_m + R_m \cdot (\phi) \cdot \cos(\psi) = 0$$
$$\Phi_2 \equiv -\cos q_{12} \cdot q_{13} + R_m \cdot \cos(\theta) \cdot \sin(\psi) = 0$$
$$\Phi_3 \equiv \sin(q_{11}) \cdot \sin(q_{12}) \cdot -z_m - R_m \cdot \sin(\theta) = 0$$
$$\Phi_4 \equiv \cos(q_{21}) \sin(q_{22}) \cdot q_{23} + R \cdot \cos(\beta) - x_m - R_m \cdot \cos(\beta) \cdot \cos(\phi) \cdot \cos(\psi) + R_m \cdot \sin(\beta) \cdot \sin(\psi) = 0$$
$$\Phi_5 \equiv -\cos(q_{22}) \cdot q_{23} + R \cdot \sin(\beta) - R_m \cdot \cos(\beta) \cdot \cos(\theta) \cdot \sin(\psi) - R_m \cdot \sin(\beta) \cdot \cos(\psi) = 0$$
$$\Phi_6 \equiv \sin(q_{21}) \cdot \sin(q_{22}) \cdot q_{23} - z_m + R_m \cdot \cos(\beta) \cdot \sin(\theta) = 0$$
$$\Phi_7 \equiv \cos(q_{31}) \cdot \sin(q_{32}) \cdot q_{33} + R \cdot \cos(\beta) - x_m - R_m \cdot \cos(\beta) \cdot \cos(\theta) \cdot \cos(\psi) - R_m \cdot \sin(\beta) \cdot \sin(\psi) = 0$$
$$\Phi_8 \equiv -\cos(q_{32}) \cdot q_{33} - R \cdot \sin(\beta) - R_m \cdot \cos(\beta) \cdot \cos(\theta) \cdot \sin(\psi) + R_m \cdot \sin(\beta) \cdot \cos(\psi) = 0$$
$$\Phi_9 \equiv \sin(q_{31}) \cdot \sin(q_{32}) \cdot q_{33} - z_m + R_m \cdot \cos(\beta) \cdot \sin(\theta) = 0$$
$$\Phi_{10} \equiv -\sin(q_{41}) \cdot q_{42} - x_m = 0$$
$$\Phi_{11} \equiv \cos(q_{41}) \cdot q_{42} - z_m = 0$$

Por lo que se tiene un sistema de once ecuaciones no lineales con once incógnitas.

$$\Phi_i(q_{11}, q_{12}, q_{21}, q_{22}, q_{31}, q_{32}, q_{41}, x_m, z_m, \phi, \psi) = 0, \qquad i = 1, 2, \cdots, 11 \tag{3.9}$$

Este sistema se podrá resolver empleando Newton-Raphson.

Velocidad por cinemática inversa

A partir de la velocidad de la plataforma móvil se calculan las velocidades de las coordenadas generalizadas de los pares U y P de las patas exteriores y los pares R y P de la pata central. Como se ha comentado en el punto anterior, la velocidad y aceleración de las coordenadas de los pares esféricos para las patas exteriores y universal para la pata central no se calculan al carecen de utilidad.

La velocidad de la plataforma viene dada por la velocidad del centro del sistema de referencia móvil $\{\dot{x}_m, \dot{y}_m, \dot{z}_m\}$ y la velocidad angular de la plataforma $\vec{\omega}_m = [\omega_x\ \omega_y\ \omega_z]^T$. Primero se obtienen las velocidades angulares a partir de las derivadas temporales de los ángulos de Euler de la plataforma móvil.

$$\vec{\omega}_m = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R_Z(\psi) \cdot \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_Z(\psi) \cdot R_Y(\theta) \cdot \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix}$$

Operando y teniendo en cuenta la restricción $\dot{\phi} = 0$ se obtienen las siguientes expresiones:

$$\begin{aligned} \omega_x &= -sin(\psi) \cdot \dot{\theta} \\ \omega_y &= cos(\psi) \cdot \dot{\theta} \\ \omega_z &= \dot{\psi} \end{aligned}$$

La velocidad de los puntos A, B y C de la plataforma móvil vendrá dada por,

$$\vec{v}_A = \begin{bmatrix} \dot{x}_m \\ \dot{y}_m \\ \dot{z}_m \end{bmatrix} + \vec{\omega} \times \left({}^{f}R_m \cdot {}^{m}\vec{r}_{O_m,A} \right)$$

Por otro lado, la velocidad se puede obtener derivando respecto del tiempo el vector de posición de cada punto. Por ejemplo, para el punto A tenemos la siguiente expresión,

$$\begin{aligned} \vec{r}_{A_0,A} &= \left[{}^{f}H_0 \cdot {}^{0}H_1(q_1) \cdot {}^{1}H_2(q_2) \cdot {}^{2}H_3(q_3) \right] \\ \vec{v}_A &= \frac{\partial \vec{r}_{A_0,A}}{\partial q_1} \cdot \dot{q}_1 + \frac{\partial \vec{r}_{A_0,A}}{\partial q_2} \cdot \dot{q}_2 + \frac{\partial \vec{r}_{A_0,A}}{\partial q_3} \cdot \dot{q}_3 \end{aligned}$$

Igualando ambas expresiones para la velocidad y procediendo de manera análoga para los puntos B y C se obtienen soluciones explícitas para las velocidades. También se podría resolver por métodos numéricos.

Para el punto O_m se usan las ecuaciones de la pata central,

$$\vec{r}_{O_f,O_m}(q_1,q_2) = \left[{}^{f}H_0 \cdot {}^{0}H_1(q_1) \cdot {}^{1}H_2(q_2)\right]$$

$$\vec{v}_{O_m} = \frac{\partial \vec{r}_{O_f,O_m}(q_1,q_2)}{\partial q_1} \cdot \dot{q}_1 + \frac{\partial \vec{r}_{O_f,O_m}(q_1,q_2)}{\partial q_2} \cdot \dot{q}_2 = \begin{bmatrix} \dot{x}_m \\ 0 \\ \dot{z}_m \end{bmatrix}$$

A continuación se presentan las expresiones para la velocidad de las coordenadas generalizadas $\dot{q}_1$, $\dot{q}_2$ y $\dot{q}_3$ de las patas exteriores y $\dot{q}_1$ y $\dot{q}_2$ de la pata central [64].

Pata 1

$$\dot{q_{11}} = \frac{(\dot{z}_m + \dot{\theta} R_m \cos\theta)\cos q_{11} - (\dot{x}_m + R_m\dot{\theta}\cos\psi\sin\theta + \dot{\psi} R_m \cos\theta \sin\psi)\sin q_{11}}{q_{13}\sin q_{12}} \tag{3.10a}$$

$$\begin{aligned} \dot{q_{12}} = {} & \frac{(\dot{x}_m + \dot{\theta} R_m \cos\psi\sin\theta + \dot{psi} R_m \cos\theta\sin\psi)\cos q_{11}\cos q_{12}}{q_{13}\cos 2q_{12}} + \\ & + \frac{(\dot{z}_m + \dot{\theta} R_m \cos\theta)\sin q_{11}\cos q_{12}}{q_{13}\cos 2q_{12}} + \frac{\dot{\theta}\sin\psi R_m \sin\theta + \dot{\psi} R_m \cos\theta\cos\psi) sin q_2}{q_{13}\cos 2q_{12}} \end{aligned} \tag{3.10b}$$

$$\begin{aligned} \dot{q}_{13} = {} & -\frac{(\dot{\theta}\sin\psi R_m \sin\theta + \dot{\psi} R_m \cos\theta\cos\psi)\cos q_{12}}{\cos 2q_{12}} - \\ & - \frac{(\dot{x}_m + R_m\dot{\theta}\cos\psi\sin\theta + \dot{\psi} R_m \cos\theta\sin\psi)\cos q_{11}\sin q_{12}}{\cos 2q_{12}} - \\ & - \frac{(\dot{z}_m + \dot{\theta} R_m \cos\theta)\sin q_{11}\sin q_{12}}{\cos 2q_{12}} \end{aligned} \tag{3.10c}$$

Pata 2

$$\begin{aligned}\dot{q}_{21} = &\frac{(\dot{z}_m + \dot{\theta}R_m\cos\psi(\sin\beta\sin\psi - \cos\beta\cos\theta\cos\psi))\cos q_{21}}{q_{23}\sin q_{22}} - \\ &- \frac{\dot{\theta}R_m\sin\psi(\sin\beta\cos\psi - \cos\beta\cos\theta\sin\psi)\cos q_{21}}{q_{23}\sin q_{22}} + \\ &+ \frac{(\dot{x}_m - \dot{\psi}R_m(\sin\beta\cos\psi + \cos\beta\cos\theta\sin\psi) - \dot{\theta}R_m\cos\beta\cos\psi\sin\theta)\sin q_{21}}{q_{23}\sin q_{22}}\end{aligned} \tag{3.11a}$$

$$\begin{aligned}\dot{q_{22}} = &\frac{(\dot{x}_m - \dot{\psi}R_m(\sin\beta\cos\psi + \cos\beta\cos\theta\sin\psi) - \dot{\theta}R_m\cos\beta\cos\psi\sin\theta)\cos q_{21}\cos q_{22}}{q_{23}\cos 2q_{22}} + \\ &+ \frac{(\dot{z}_m + \dot{\theta}R_m\cos\psi(\sin\beta\sin\psi - \cos\beta\cos\theta\cos\psi))\sin q_{21}\cos q_{22}}{q_{23}\cos 2q_{22}} - \\ &- \frac{\dot{\theta}R_m\sin\psi(\sin\beta\cos\psi - \cos\beta\cos\theta\sin\psi)\sin q_{21}\cos q_{22}}{q_{23}\cos 2q_{22}} + \\ &+ \frac{\sin q_{22}(\dot{\psi}R_m(\sin\beta\sin\psi - \cos\beta\cos\theta\cos\psi) + \dot{\theta}R_m\cos\beta\sin\psi\sin\theta)}{q_{23}\cos 2q_{22}}\end{aligned} \tag{3.11b}$$

$$\begin{aligned}\dot{q}_{23} = &-\frac{(\dot{x}_m - \dot{\psi}R_m(\sin\beta\cos\psi + \cos\beta\cos\theta\sin\psi) - \dot{\theta}R_m\cos\beta\cos\psi\sin\theta)\cos q_{21}\sin q_{22}}{\cos 2q_{22}} - \\ &- \frac{(\dot{z}_m + \dot{\theta}R_m\cos\psi(\sin\beta\sin\psi - \cos\beta\cos\theta\cos\psi))\sin q_{21}\sin q_{22}}{\cos 2q_{22}} + \\ &+ \frac{\dot{\theta}R_m\sin\psi(\sin\beta\cos\psi - \cos\beta\cos\theta\sin\psi)\sin q_{21}\sin q_{22}}{cos2q_{22}} - \\ &- \frac{\cos q_{22}(\dot{\psi}R_m(\sin\beta\sin\psi - \cos\beta\cos\theta\cos\psi) + \dot{\theta}R_m\cos\beta\sin\psi\sin\theta}{\cos 2q_{22}}\end{aligned} \tag{3.11c}$$

Pata 3

$$\begin{aligned}\dot{q}_{31} = &-\frac{(\dot{z}_m - \dot{\theta}R_m\cos\psi(\sin\beta\sin\psi + \cos\beta\cos\theta\cos\psi))\cos q_{31}}{q_{33}\sin q_{32}} + \\ &+ \frac{\dot{\theta}R_m\sin\psi(\sin\beta\cos\psi - \cos\beta\cos\theta\sin\psi)\cos q_{31}}{q_{33}\sin q_{32}} + \\ &+ \frac{(\dot{x}_m - \dot{\psi}R_m(-\sin\beta\cos\psi + \cos\beta\cos\theta\sin\psi) - \dot{\theta}R_m\cos\beta\cos\psi\sin\theta)\sin q_{31}}{q_{33}\sin q_{32}}\end{aligned} \tag{3.12a}$$

$$\begin{aligned}\dot{q_{32}} = &\frac{(\dot{x}_m - \dot{\psi}R_m(-\sin\beta\cos\psi + \cos\beta\cos\theta\sin\psi) - \dot{\theta}R_m\cos\beta\cos\psi\sin\theta)\cos q_{31}\cos q_{32}}{q_{33}\cos 2q_{32}} + \\ &+ \frac{(\dot{z}_m - \dot{\theta}R_m\cos\psi(\sin\beta\sin\psi + \cos\beta\cos\theta\cos\psi))\sin q_{31}\cos q_{32}}{q_{33}\cos 2q_{32}} - \\ &- \frac{\dot{\theta}R_m\sin\psi(\sin\beta\cos\psi - \cos\beta\cos\theta\sin\psi)\sin q_{31}\cos q_{32}}{q_{33}\cos 2q_{32}} + \\ &+ \frac{\sin q_{32}(\dot{\psi}R_m(\sin\beta\sin\psi - \cos\beta\cos\theta\cos\psi) + \dot{\theta}R_m\cos\beta\sin\psi\sin\theta)}{q_{33}\cos 2q_{32}}\end{aligned} \tag{3.12b}$$

$$\begin{aligned}\dot{q}_{33} = &-\frac{(\dot{x}_m - \dot{\psi}R_m(-\sin\beta\cos\psi + \cos\beta\cos\theta\sin\psi) - \dot{\theta}R_m\cos\beta\cos\psi\sin\theta)\cos q_{31}\sin q_{32}}{\cos 2q_{22}} + \\ &+ \frac{(\dot{z}_m - \dot{\theta}R_m\cos\psi(\sin\beta\sin\psi + \cos\beta\cos\theta\cos\psi))\sin q_{31}\sin q_{32}}{\cos 2q_{32}} - \\ &- \frac{\dot{\theta}R_m\sin\psi(\sin\beta\cos\psi - \cos\beta\cos\theta\sin\psi)\sin q_{31}\sin q_{32}}{cos2q_{32}} - \\ &- \frac{\cos q_{32}(\dot{\psi}R_m(\sin\beta\sin\psi - \cos\beta\cos\theta\cos\psi) + \dot{\theta}R_m\cos\beta\sin\psi\sin\theta}{\cos 2q_{32}}\end{aligned} \tag{3.12c}$$

Pata 4 (central)

$$\dot{q}_{41} = -\frac{\dot{x}_m\cos q_{41} + \dot{z}_m\sin q_{41}}{q_{42}} \tag{3.13a}$$

$$\dot{q_{42}} = -\dot{x}_m\sin q_{41} + \dot{z}_m\cos q_{41} \tag{3.13b}$$

Velocidad por cinemática directa

Conocida la posición y las velocidades en los actuadores, $\dot{q}_{13}, \dot{q}_{23}, \dot{q}_{43}, \dot{q}_{42}$, se quiere determinar la velocidad de la plataforma móvil, $\dot{x}_m, \dot{z}_m, \dot{\phi}, \dot{\psi}$. Derivando las ecuaciones de restricción (3.9) y separando las coordenadas generalizadas independientes, $q^i = [q_{13}, q_{23}, q_{33}, q_{42}]^T$ y las secundarias $q^s = [q_{11}, q_{12}, q_{21}, q_{22}, q_{31}, q_{32}, q_{41}, x_m, z_m, \phi, \psi]^T$, con sus correspondientes velocidades generalizadas se tiene que,

$$\left[\frac{\partial \Phi_i}{\partial q^s}\right]_{11\times 11} \begin{bmatrix} \dot{q}_{11} \\ \dot{q}_{12} \\ \dot{q}_{21} \\ \dot{q}_{22} \\ \dot{q}_{31} \\ \dot{q}_{32} \\ \dot{q}_{41} \\ \dot{x}_m \\ \dot{z}_m \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix} = - \left[\frac{\partial \Phi_i}{\partial q^i}\right]_{11\times 4} \begin{bmatrix} \dot{q}_{13} \\ \dot{q}_{23} \\ \dot{q}_{33} \\ \dot{q}_{42} \end{bmatrix} \tag{3.14}$$

Por otro lado, las componentes de la velocidad angular absoluta de la plataforma, expresada en el sistema de referencia fijo, se pueden obtener mediante la siguiente expresión,

$$\begin{aligned} \omega_x &= -\sin(\psi) \cdot \dot{\psi} \\ \omega_y &= \cos(\psi) \cdot \dot{\psi} \\ \omega_z &= \dot{\psi} \end{aligned}$$

Aceleración por cinemática inversa

Se conoce la aceleración del origen del sistema de referencia móvil de la plataforma $\ddot{x}_m, 0, \ddot{z}_m$ y las derivadas segundas respecto al tiempo de los ángulos de Euler, $0, \ddot{\phi}, \ddot{\psi}$ [64][4]. La aceleración de angular de la barra vendrá dada por,

$$\begin{aligned}\dot{\omega}_x &= -\sin(\psi)\cdot\ddot{\psi} - \cos(\psi)\cdot\dot{\psi}\dot{\psi}\\ \dot{\omega}_y &= \cos(\psi)\cdot\ddot{\psi} - \sin(\psi)\cdot\dot{\psi}\dot{\psi}\\ \dot{\omega}_z &= \ddot{\psi}\end{aligned}$$

La aceleración de los puntos significativos de la plataforma móvil tiene la siguiente expresión,

$$\vec{a}_A = \begin{bmatrix}\ddot{x}_m\\ 0\\ \ddot{z}_m\end{bmatrix} f\vec{\omega}\times\left(\vec{\omega}\times\left({}^fR_m\cdot{}^m\vec{r}_{O_mA}\right)\right) + \vec{\dot{\omega}}\times\left({}^fR_m\cdot{}^m\vec{r}_{O_mA}\right)$$

Por otro lado, derivando respecto al tiempo las velocidades, $\vec{v}_A$, se tendrá que,

$$\vec{a}_A = \frac{\partial\vec{v}_A}{\partial q_1}\cdot\dot{q}_1 + \frac{\partial\vec{v}_A}{\partial\dot{q}_1}\ddot{q}_1 + \frac{\partial\vec{v}_A}{\partial q_3}\cdot\dot{q}_3 + \frac{\partial\vec{v}_A}{\partial\dot{q}_3}\ddot{q}_3\frac{\partial\vec{v}_A}{\partial q_3}\cdot\dot{q}_3 + \frac{\partial\vec{v}_A}{\partial\dot{q}_3}\ddot{q}_3$$

igualando ambas expresiones se tiene un sistema de tres ecuaciones con tres incógnitas, $\ddot{q}_1, \ddot{q}_2, \ddot{q}_3$, para las patas 1, 2, 3. De modo análogo se obtiene un sistema de dos ecuaciones no triviales con dos incógnitas, $\ddot{q}_1, \ddot{q}_2$, para la pata central 4.

Aceleración por cinemática directa

La aceleración por cinemática directa se calcula derivando respecto al tiempo las ecuaciones de velocidad (3.14), dando como resultado la siguiente expresión[4],

$$\left[\frac{\partial\Phi_i}{\partial q^s}\right]\cdot[\ddot{q}^s] + \frac{d\left[\frac{\partial\Phi_i}{\partial q^s}\right]}{dt}[\dot{q}^s] = -\left[\frac{\partial\Phi_i}{\partial q^i}\right]\cdot\left[\ddot{q}^i\right] - \frac{d\left[\frac{\partial\Phi_i}{\partial q^i}\right]}{dt}\cdot\left[\dot{q}^i\right] \tag{3.15}$$

Modelo dinámico.

Pesos y fuerzas de inercia generalizadas

Se ha aplicado el Principio de D'Alembert para el cálculo de los pesos y las fuerzas generalizadas. Para esto se ha utilizado la cinemática de cada una de las patas así como de la plataforma móvil.

En primer lugar se considerarán las patas exteriores, utilizando el primer subíndice para referirnos a la pata y el segundo subíndice para las barras de esta. Se resolverá el problema con la *Pata 1*, el procedimiento es análogo para las patas 2 y 3. Salvo que se indique lo contrario todas las magnitudes se expresarán en el sistema de referencia fijo a la base del robot.

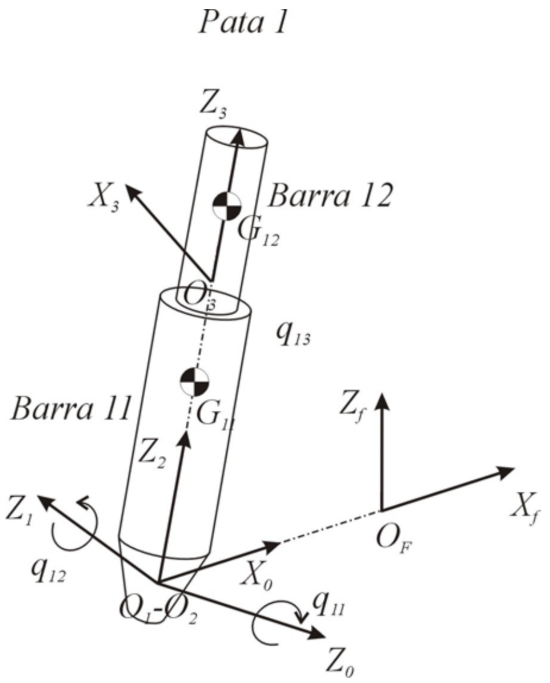


Figura 3.7: Representación gráfica del modelo dinámico de la Pata 1. Fuente [4]

Las velocidades angulares de las barras serán,

$$\vec{\omega}_{11} = \dot{q}_{11} \cdot \vec{u}_{Z_0} + \dot{q}_{12} \cdot \vec{u}_{Z_1}$$
$$\vec{\omega}_{12} = \vec{\omega}_{11}$$

donde

$$\vec{u}_{Z_0} = \begin{bmatrix} 0 & -1 & 0 \end{bmatrix}^T$$
$$\vec{u}_{Z_1} = \begin{bmatrix} -\sin(q_1) & 0 & 0 \end{bmatrix}^T$$

Al tratarse de un par prismático las aceleraciones angulares de ambas barras serán iguales, $\vec{\alpha}_{12} = \vec{\alpha}_{11}$. Para la *Pata 1* tendremos que,

$$\vec{\alpha}_{11} = \dot{\vec{\omega}}_{11} = \ddot{q}_{11} \cdot \vec{u}_{Z_0} + \ddot{q}_{12} \cdot \vec{u}_{Z_1} + \dot{q}_{12} \cdot (\dot{\omega}_1 \cdot^f R_1 \cdot^1 \vec{u}_{Z_1})$$

donde $\vec{\omega}_1 = \dot{q}_{11} \cdot \vec{u}_{Z_0}$. Operando se obtiene que,

$$\vec{\alpha}_{11} = \begin{bmatrix} -\ddot{q}_{12} \cdot \sin(q_{11}) - \dot{q}_{11} \cdot \dot{q}_{12} \cdot \cos(q_{11}) \\ -\ddot{q}_{11} \\ \ddot{q}_{12} \cdot \cos(q_{11}) - \dot{q}_{11} \cdot \dot{q}_{12} \cdot \sin(q_{11}) \end{bmatrix}$$

La velocidad y aceleración del CdG de la barra 11 viene dadas por,

$$\vec{v}_{G_{11}} = \vec{\omega}_{11} \times \vec{r}_{O_2 G_{11}}$$
$$\vec{a}_{G_1 1} = \vec{\omega}_{11} \times (\vec{\omega}_{11} \times \vec{r}_{O_2 G_{11}}) + \vec{\alpha}_{11} \times \vec{r}_{O_2 G_{11}}$$

siendo

$$\vec{r}_{O_2 G_{11}} = \begin{bmatrix} x_{G_{11}} & y_{G_{11}} & z_{G_{11}} \end{bmatrix}^T$$

la localización del sistema de referencia de la barra 11 respecto al sistema de referencia que se mueve con ella.

Para la barra 12 la velocidad y la aceleración serán, considerando el sistema de referencia móvil $\{O_2 - X_2Y_2Z_2\}$ a la hora de aplicar las ecuaciones del movimiento relativo,

$$\vec{v}_{G_{12}} = \vec{v}_{O_2} + \vec{\omega}_{11} \times \vec{r}_{O_2G_{12}} + \dot{q}_{13} \cdot \vec{u}_{Z_2}$$

donde

$$\vec{r}_{O_2G_{12}} = \vec{r}_{O_2O_3} + {}^{f}R_3 \cdot {}^{3}\vec{r}_{O_2G_{12}}$$

y

$$\vec{r}_{O_2O_3} = {}^{f}R_2 \cdot \begin{bmatrix} 0 \\ 0 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(q_{11}) \cdot \sin(q_{11}) \cdot q_{13} \\ -\cos(q_{12}) \cdot q_{13} \\ \sin(q_{11}) \cdot \sin(q_{11}) \cdot q_{13} \end{bmatrix}$$
$${}^{3}\vec{r}_{O_3G_{12}} = \begin{bmatrix} x_{G_{12}} & y_{G_{12}} & z_{G_{12}} \end{bmatrix}^T$$

La aceleración será,

$$\vec{a}_{G_{12}} = \vec{v}_{O_2} + \vec{\omega}_{11} \times (\vec{\omega}_{11} \times \vec{r}_{O_2G_{12}}) + \vec{\alpha}_{11} \times \vec{r}_{O_2G_{12}} + \ddot{q}_{13} \cdot \vec{u}_{Z_2} + 2 \cdot (\vec{\omega}_{12} \times (q_{i3} \cdot \vec{u}_{Z_2}))$$

Y por lo tanto las acciones inerciales que hay que considerar en ambas barras son,

$$\vec{F}_{in_{11}} = -m_{11} \cdot \vec{a}_{G_{11}}$$
$$\vec{T}_{in_{11}} = -\left(I_{G_{11}} \cdot \vec{\alpha}_{11} + \vec{\omega}_{11} \cdot (I_{G_{11}} \cdot \vec{\omega}_{11})\right)$$

y

$$\vec{F}_{in_{12}} = -m_{12} \cdot \vec{a}_{G_{12}}$$
$$\vec{T}_{in_{12}} = -\left(I_{G_{12}} \cdot \vec{\alpha}_{12} + \vec{\omega}_{12} \cdot (I_{G_{12}} \cdot \vec{\omega}_{12})\right)$$

siendo los tensores de inercia locales

$$I_{G_{11}} = \begin{bmatrix} I_{XX_{11}} & -I_{XY_{11}} & -I_{XZ_{11}} \\ & I_{YY_{11}} & -I_{YZ_{11}} \\ \ddots & & I_{ZZ_{11}} \end{bmatrix}$$
$$I_{G_{12}} = \begin{bmatrix} I_{XX_{12}} & -I_{XY_{12}} & -I_{XZ_{12}} \\ & I_{YY_{12}} & -I_{YZ_{12}} \\ \ddots & & I_{ZZ_{12}} \end{bmatrix}$$

y por tanto,

$$I_{G_{11}} = {}^{f}R_2 \cdot {}^{2}I_{G_{11}} \cdot \left({}^{f}R_2\right)^T$$
$$I_{G_{12}} = {}^{f}R_3 \cdot {}^{3}I_{G_{12}} \cdot \left({}^{f}R_3\right)^T$$

y los pesos de las patas serán,

$$\vec{P}_{11} = \begin{bmatrix} 0 & 0 & -m_{11} \cdot g \end{bmatrix}^T$$
$$\vec{P}_{12} = \begin{bmatrix} 0 & 0 & -m_{12} \cdot g \end{bmatrix}^T$$

Las patas 2 y 3 se resuelven con un procedimiento análogo.

Fuerzas externas activas generalizadas

Como acciones externas activas se tienen las fuerzas comunicadas por los cuatro actuadores, tal y como se indican en las figura 3.8.

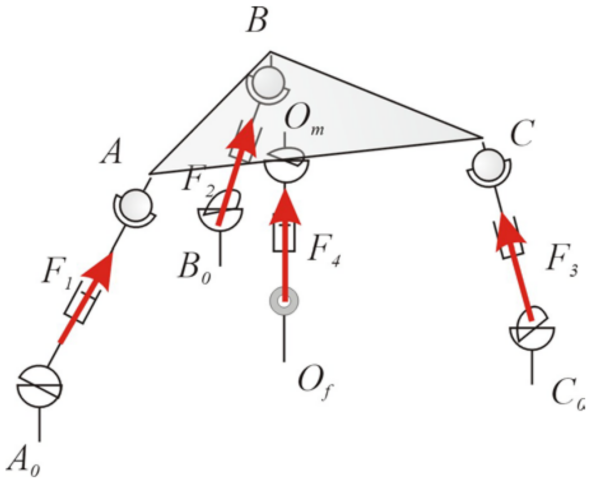


Figura 3.8: Acciones externas activas. Fuente [4]

Las fuerzas activas serán,

$$\vec{F}_1 = {}^{f}R_2 \cdot \begin{bmatrix} 0 \\ 0 \\ F_1 \end{bmatrix} \qquad \vec{F}_2 = {}^{f}R_2 \cdot \begin{bmatrix} 0 \\ 0 \\ F_2 \end{bmatrix} \qquad \vec{F}_3 = {}^{f}R_2 \cdot \begin{bmatrix} 0 \\ 0 \\ F_3 \end{bmatrix} \qquad \vec{F}_4 = {}^{f}R_1 \cdot \begin{bmatrix} 0 \\ 0 \\ F_4 \end{bmatrix}$$

y las fuerzas externas generalizadas vendrán dadas por,

$$\vec{Q}_{ex} = \vec{F}_1 \cdot \frac{\partial \vec{v}_{G_{12}}}{\partial \dot{\vec{q}}} + \vec{F}_2 \cdot \frac{\partial \vec{v}_{G_{22}}}{\partial \dot{\vec{q}}} + \vec{F}_3 \cdot \frac{\partial \vec{v}_{G_{32}}}{\partial \dot{\vec{q}}} + \vec{F}_4 \cdot \frac{\partial \vec{v}_{G_{42}}}{\partial \dot{\vec{q}}} \tag{3.16}$$

Pesos generalizados

Considerando los pesos de las barras móviles del mecanismo, se tendrá que

$$\vec{Q}_{grav} = \vec{P}_{11} \cdot \frac{\partial \vec{v}_{G_{11}}}{\partial \dot{\vec{q}}} + \vec{P}_{12} \cdot \frac{\partial \vec{v}_{G_{12}}}{\partial \dot{\vec{q}}} + \ldots + \vec{P}_m \cdot \frac{\partial \vec{v}_{G_m}}{\partial \dot{\vec{q}}} \tag{3.17}$$

Fuerzas externas aplicadas a la barra móvil

En este punto se tienen en cuenta las acciones exteriores aplicadas sobre la plataforma móvil. Para este caso se considerarán las acciones , teniendo en cuenta el sistema actual (bota + célula de carga), como una fuerza y un par conocidos y referenciados al sistema de referencia asociado a la plataforma móvil como se ve en la figura 3.9.

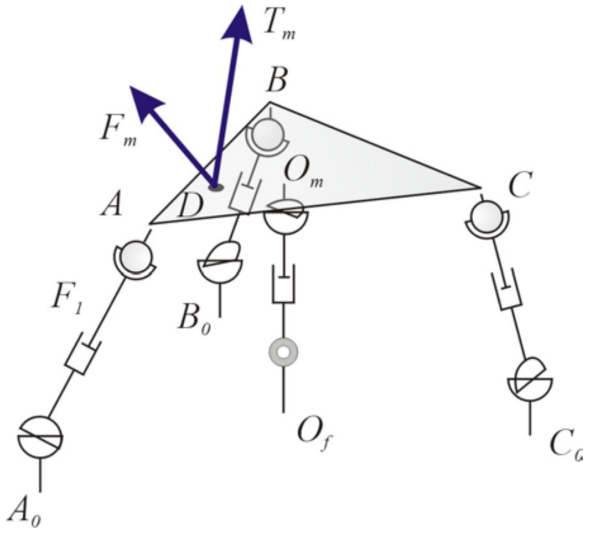


Figura 3.9: Acciones externas activas sobre la barra móvil. Fuente [4]

Estas acciones exteriores vendrán dadas, respecto al sistema de de referencia asociado a la plataforma como,

$$\vec{F}_m = \begin{bmatrix} F_{m_x} & F_{m_y} & F_{m_z} \end{bmatrix}^T$$
$$\vec{T}_m = \begin{bmatrix} T_{m_x} & T_{m_y} & T_{m_z} \end{bmatrix}^T$$

Las correspondientes acciones generalizadas serán,

$$\vec{Q}_{ent} = \vec{F}_m \cdot \frac{\partial \vec{v}_D}{\partial \dot{\vec{q}}} + \vec{T}_m \cdot \frac{\partial \vec{\omega}_m}{\partial \dot{\vec{q}}} \tag{3.18}$$

donde la velocidad del punto de aplicación se puede obtener como

$$\vec{v}_D = \vec{v}_{O_m} + \vec{\omega}_m \times \vec{r}_{O_m D}$$

donde,

$$\vec{r}_{O_m D} = {}^{f}R_m \cdot \begin{bmatrix} x_D \\ y_D \\ z_D \end{bmatrix}$$

Ecuación del movimiento en forma aumentada

Dado el uso de coordenadas dependientes, la ecuación del movimiento será,

$$\vec{Q}_{in} + \vec{Q}_{grav} + \vec{Q}_{ex} + \vec{Q}_{ent} - [\Phi_q]^T \cdot \vec{\lambda} = 0 \tag{3.19}$$

donde $[\Phi_q]$ es el jacobiano de las restricciones y $\vec{\lambda}$ el vector de multiplicadores indeterminados de Lagrange.

En el sistema anterior de 15 ecuaciones escalares, las incógnitas habituales (problema dinámico directo), son las aceleraciones (15) más los multiplicadores (10), por tanto faltan 10 ecuaciones adicionales, que se obtendrán derivando respecto al tiempo dos veces las ecuaciones de restricción. De este modo se tendrá la denominada formulación aumentada, que se puede poner como,

$$\begin{aligned} &- \vec{Q}_{in} + [\Phi_q]^T \cdot \vec{\lambda} = \vec{Q}_{grav} + \vec{Q}_{ex} + \vec{Q}_{ent} \\ &[\Phi_q] \cdot \ddot{\vec{q}} = \vec{b} \end{aligned}$$

y en forma matricial, agrupando los términos que incluyan incógnitas a la izquierda,

$$\begin{bmatrix} M_{15\times15} & [\Phi_q]^T_{15\times11} \\ [\Phi_q]_{11\times15} & 0_{11\times11} \end{bmatrix}_{26\times26} \cdot \begin{bmatrix} \ddot{\vec{q}}_{15\times1} \\ \vec{\lambda}_{10\times1} \end{bmatrix} = \begin{bmatrix} \vec{Q}_{cyc} + \vec{Q}_{ex} + \vec{Q}_{grav} + \vec{Q}_{ent} \\ \vec{b} \end{bmatrix}$$

donde $\vec{Q}_{cyc}$ son las fuerzas generalizadas que dependen del cuadrado de las velocidades generalizas (términos centrífugo y Coriolis).

Ecuaciones del movimiento en forma compacta

En el caso de que no se quieran determinar los multiplicadores de Lagrange, se pueden eliminar de la ecuación del movimiento empleando un complemento ortogonal. De las muchas posibilidades que hay, vamos a emplear Partición de Coordenadas, por dos motivos:

- Está clara cuál es la partición entre coordenadas independientes y dependientes y no parece que vayamos a tener problemas con singularidades.
- Requiere de menos herramientas de cálculo (SVD, por ejemplo) que otros métodos todos.

La idea es obtener una matriz R^* tal que verifique que

$$(R^*)^T \cdot [\Phi_q]^T = 0$$

por lo tanto, multiplicando ambos lados de la ecuación del movimiento por el complemento ortogonal, desaparecerían las fuerzas internas generalizadas que incluyen a los multiplicadores.

$$(R^*)^T \cdot \left(-\vec{Q}_{in} + [\Phi_q]^T \cdot \vec{\lambda}\right) = (R^*)^T \cdot \left(\vec{Q}_{grav} + \vec{Q}_{ex} + \vec{Q}_{ent}\right)$$

y serán

$$-(R^*)^T \cdot \vec{Q}_{in} = (R^*)^T \cdot \left(\vec{Q}_{grav} + \vec{Q}_{ex} + \vec{Q}_{ent}\right)$$

La matriz R^* relaciona velocidades generalizadas con independientes y se puede obtener como,

$$\begin{bmatrix} \vec{\dot{q}}^s \\ \vec{\dot{q}}^i \end{bmatrix}_{N\times 1} = \begin{bmatrix} -\left[\Phi_q^s\right]^{-1} \cdot \left[\Phi_q^i\right]_{M\times F} \\ 1_{F\times F} \end{bmatrix} \cdot \left[\vec{\dot{q}}^i\right]_{F\times 1}$$

por lo que

$$R^*_{N\times F} = \begin{bmatrix} -\left[\Phi_q^s\right]^{-1} \cdot \left[\Phi_q^i\right]_{M\times F} \\ 1_{F\times F} \end{bmatrix}_{N\times F}$$

Por lo tanto, la ecuación del movimiento se puede poner en forma compacta del siguiente modo,

$$(R^*)^T_{F\times N} \cdot \left(M_{n\times N} \cdot \vec{\ddot{q}}_{N\times 1} - Q_{cyc_{N\times N}} \cdot \vec{\dot{q}}_{N\times 1} - \vec{Q}_{grav_{N\times 1}} - Q_{ent_{N\times 1}}\right) = (R^*)^T_{F\times N} \cdot \left(\vec{Q}_{ext_{N\times F}} \cdot \vec{\dot{F}}_{act_{F\times 1}}\right)$$

donde Q_{cyc} es una matriz que corresponde a los términos centrífugo y de Coriolis y Q_{ext} una matriz que corresponde a las fuerzas externas activas.

`Nota importante:` A partir de este momento, y para facilitar la comprensión, le cambiaremos el signos a los términos centrífugos y de Coriolis, gravitatorios y correspondiente a las fuerzas que transmite el entorno, por lo que la expresión anterior quedará,

$$(R^*)^T_{F\times N}\cdot\left(M_{n\times N}\cdot\vec{\ddot{q}}_{N\times 1}+Q_{cyc_{N\times N}}\cdot\vec{\dot{q}}_{N\times 1}+\vec{Q}_{grav_{N\times 1}}+Q_{ent_{N\times 1}}\right)=(R^*)^T_{F\times N}\cdot\left(\vec{Q}_{ext_{N\times F}}\cdot\vec{F}_{act_{F\times 1}}\right) \tag{3.20}$$

Aceleraciones generalizadas en función de las independientes

En la ecuación del movimiento 3.20 se aprecia que aparecen todas las aceleraciones generalizadas (dependientes + independientes), en algunos casos será interesante expresarlas solo en función de las independientes. Para ello se parte de la ecuación de velocidades,

$$\left[\frac{\partial\Phi_i}{\partial q^s}\right]\cdot[\dot{q}^s]=-\left[\frac{\partial\Phi_i}{\partial q^i}\right]\cdot\left[\dot{q}^i\right]$$

y derivando respecto al tiempo, se tendrá que,

$$\left[\frac{\partial\Phi_i}{\partial q^s}\right]\cdot[\ddot{q}^s]=-\left[\frac{\partial\Phi_i}{\partial q^i}\right]\cdot\left[\dot{q}^i\right]-\frac{d}{dt}\left(\left[\frac{\partial\Phi_i}{\partial q^s}\right]\right)\cdot[\dot{q}^s]-\frac{d}{dt}\left(\left[\frac{\partial\Phi_i}{\partial q^i}\right]\right)\cdot\left[\dot{q}^i\right]$$

el término

$$B=\frac{d}{dt}\left(\left[\frac{\partial\Phi_i}{\partial q^s}\right]\right)\cdot[\dot{q}^s]+\frac{d}{dt}\left(\left[\frac{\partial\Phi_i}{\partial q^i}\right]\right)\cdot\left[\dot{q}^i\right]$$

se obtendrá considerando que los jacobianos no dependen directamente del tiempo, sino de las coordenadas generalizadas, por lo que se aplicará la regla de derivación en cadena,

$$\frac{d}{dt}\left(\left[\frac{\partial\Phi_i}{\partial q^s}\right]\right)\cdot[\dot{q}^s]=\left(\sum_{j=1}^{N}\frac{\partial\left(\left[\frac{\partial\Phi_i}{\partial q^s}\right]\right)}{\partial q_j}\cdot\dot{q}_j\right)\cdot[\dot{q}^s]$$

$$\frac{d}{dt}\left(\left[\frac{\partial\Phi_i}{\partial q_i}\right]\right)\cdot\left[\dot{q}^i\right]=\left(\sum_{j=1}^{N}\frac{\partial\left(\left[\frac{\partial\Phi_i}{\partial q^i}\right]\right)}{\partial q_j}\cdot\dot{q}_j\right)\cdot\left[\dot{q}^i\right]$$

nótese que aparecen términos centrífugos y de Coriolis. Por tanto, se tendrá que

$$[\ddot{q}^s] = -\left[\frac{\partial \Phi_i}{\partial q^s}\right]^{-1} \cdot \left[\frac{\partial \Phi_i}{\partial q^i}\right] \cdot \left[\dot{q}^i\right] - \left[\frac{\partial \Phi_i}{\partial q^s}\right]^{-1} \cdot B$$

Finalmente todas las aceleraciones generalizadas se podrán poner en función de las independientes del siguiente modo,

$$\begin{bmatrix} \ddot{q}^s_{11\times 1} \\ \ddot{q}^i_{4\times 1} = \end{bmatrix} \begin{bmatrix} -\left[\frac{\partial \Phi_i}{\partial q^s}\right]^{-1}_{11\times 11} \cdot \left[\frac{\partial \Phi_i}{\partial q^i}\right]_{11\times 4} \\ I_{4\times 4} \end{bmatrix}_{15\times 4} \cdot \left[\dot{q}^i\right]_{4\times 1} + \begin{bmatrix} -\left[\frac{\partial \Phi_i}{\partial q^s}\right]^{-1}_{11\times 11} \cdot B \\ 0_{4\times 1} \end{bmatrix}_{15\times 1} \tag{3.21}$$

Capítulo 4

Componentes

Como ya se ha comentado en las sección 2.3.1, donde se discute la importancia de un desarrollo de software basado en componentes, el concepto de *middleware* juega un papel clave a la hora de desarrollar una aplicación que cubra las necesidades que se especifican consumiendo los mínimos recursos posibles. Haciendo uso de *Open RObot COntrol Software (OROCOS)* se puede desarrollar aplicaciones interactivas basadas en componentes y totalmente configurables [65].

Existen en la literatura diferentes definiciones, como se ha visto en la sección 2.3.1, de lo que es un componente, cada una enfocada hacia un tipo de aplicación o pensada para un framework en concreto pero todas coinciden en lo fundamental. En esta parte del texto se prefiere utilizar la definición propuesta por Bruyninckx [66].

Bruyninckx define un componente como un pedazo de software que ejecuta su funcionalidad (o entrega un servicio) independientemente del contexto en el que se usa, i.e., durante el diseño e implementación del componente no debe ser necesario conocer de antemano quien lo va a utilizar y para qué. Por otro lado cabe destacar que esta independencia nunca podrá ser absoluta, por ejemplo, va a ser necesario determinar la memoria interna de la máquina donde se ejecutará el componente o la capacidad de procesamiento de la misma. El componente deberá de proporcionar algún mecanismo de intercambio de información suficientemente bien definido de manera que otros componentes puedan interactuar con él.

Como ya se ha comentado varias veces a lo largo de este trabajo, Orocos es un *framework* para el desarrollo de software basado en componentes para el control de máquinas y robots, se trata de un proyecto *open-source* utilizado en muchos proyectos tanto de investigación como de uso comercial y que ha tenido una gran aceptación en el campo de la robótica. Sin embargo, trabajar con OROCOS supone una curva de aprendizaje muy lenta. Puede llegar a ser muy confuso debido

a la ingente cantidad de características que tiene y, por lo tanto, es posible implementar la misma funcionalidad de diferentes maneras. Desgraciadamente, OROCOS no define una política de como utilizar los diferentes mecanismos disponibles debido a la complejidad del *framework* y por lo tanto puede resultar muy complicado de entender y decidir que elemento resulta mejor en cada caso, sobre todo para usuarios noveles [13]. En relación a esto, todas las definiciones que se dan a continuación están enfocadas al proyecto de los robots paralelos y, por lo tanto, las soluciones que se han propuesto han sido siempre enfocadas a estos proyectos.

Creación de un componente

Los componentes de Orocos se crean dentro de *paquetes*, estos *paquetes* son directorios que contienen el código fuente del componente y diferentes archivos con la configuración necesaria para compilarlo y hacer que el nuevo componente esté disponible en el entorno de trabajo.

Orocos proporciona herramientas de generación de código que nos permite crear un componente con un simple comando. Dependiendo de cómo se tenga instalado Orocos la forma de crear los paquetes variará. En caso de tener Orocos instalado de forma independiente se puede crear el paquete introduciendo la siguiente línea en la terminal:

```
$ orocreate-pkg component/package my_component
```

Este método se ha quedado un poco obsoleto desde que Orocos pasó a ser parte del repositorio de paquetes de ROS. En el caso de utilizar Orocos como parte de un nodo de ROS el comando para generar el código que acompaña al componente es el siguiente:

```
$ rosrun rtt_ros orocreate-pkg my_component component
```

Estos comandos crean un directorio con el nombre del componente y los archivos auxiliares necesarios:

```
my_component
├── CMakeList.txt
├── manifest.xml
└── src
    ├── my_component-component.cpp
    └── my_component-component.hpp
```

- **CMakeList.txt:** Se trata del archivo utilizado por el software *CMake* con los detalles de la compilación. Por ejemplo, en este archivo se pueden indicar los *flags* necesarios para compilar correctamente el código fuente, indicar los directorios dónde se pueden localizar los diferentes archivos de cabecera o incluir las librerías externas.

- **manifest.xml:** Este archivo es útil para añadir información relativa al componente como número de la versión, información del desarrollador u otros componentes que se van a integrar dentro del mismo. Cabe destacar que en la versión de Orocos integrada en ROS este archivo se sustituye por *package.xml*. Esto permite usar la herramienta *Catkin*, que es la encargada de compilar todo el espacio de trabajo en los proyectos de ROS.
- **my_component-component.cpp:** También se crea una carpeta `/src` en la que deberá incluirse todo el código fuente. En esta carpeta se crea un archivo con el nombre del componente acabado en `-component.cpp`. En este archivo, escrito en C++, se genera el código de las funciones básicas de cada componente que se comentarán más adelante en esta misma sección.
- **my_component-component.hpp:** Similar al archivo anterior, se trata de un archivo de cabecera con el código mínimo necesario generado automáticamente por `orocreate-pkg`.

Cuando se compila con *Catkin*, o directamente con *CMake* en las versiones aisladas, se crea una biblioteca `.so`. Estas bibliotecas están disponibles en el entorno haciéndolas accesibles al *Deployer Component* [65], el cual se trata de un componente maestro que gestiona todos los demás y se explicará a continuación.

Estructura interna de un componente

Al crear un componente se genera el código de unas funciones denominadas *hooks*, como se puede ver en el ejemplo del componente vacío `Master`.

(Archivo Master-componen.hpp)

```
#ifndef OROCOS_MASTER_COMPONENT_HPP
#define OROCOS_MASTER_COMPONENT_HPP
 #include <rtt/RTT.hpp>
using namespace RTT; //Usamos este namspace para no tener que poner RTT en todas
    las funciones de orocos
class Master : public RTT\dotsTaskContext{
public:
    Master(std\dotsstring const& name);
    bool configureHook(;
    bool startHook(;
    void updateHook(;
    void stopHook(;
    void cleanupHook(;

 };
#endif
```

(Archivo Master-componen.cpp)

```
#include ‘‘master-component.hpp’’
#include <rtt/Component.hpp>
#include <iostream>

Master::Master(std::string const& name) : TaskContext(name){
    std::cout << ‘‘Master constructed !’’ <<std::endl;
}

bool Master::configureHook(){
  std::cout << ‘‘Master configured !’’ <<std::endl;
  return true;
}

bool Master::startHook(){
  std::cout << ‘‘Master started !’’ <<std::endl;
  return true;
}

void Master::updateHook(){
  std::cout << ‘‘Master executes updateHook !’’ <<std::endl;
}

void Master::stopHook() {
  std::cout << ‘‘Master executes stopping !’’ <<std::endl;
}

void Master::cleanupHook() {
  std::cout << ‘‘Master cleaning up !’’ <<std::endl;
}

/*
* Using this macro, only one component may live
* in one library *and* you may *not* link this library
* with another component library. Use
* ORO_CREATE_COMPONENT_TYPE()
* ORO_LIST_COMPONENT_TYPE(Master)
* In case you want to link with another library that
* already contains components.
*
* If you have put your component class
* in a namespace, don’t forget to add it here too:
*/
ORO_CREATE_COMPONENT(Master)
```

Cada componente se comporta como una *Máquina de Estados Finitos* en el que cada *hook* representa una transición, ver figura 4.1.

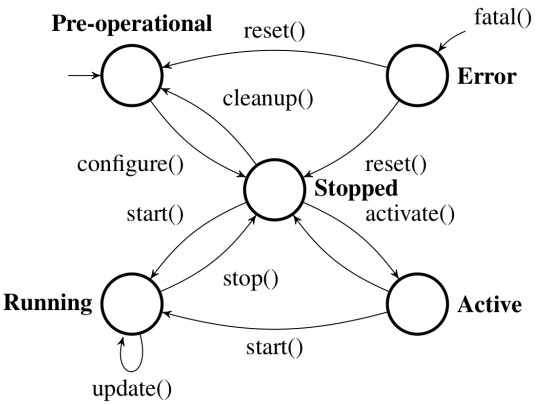


Figura 4.1: Máquina de estados finitos de un componente de Orocos [12]

- **configureHook:** Es el primero en ejecutarse y lee el archivo XML con todos los parámetros de configuración. Se pueden añadir también llamar a otras funciones o definir variables iniciales dentro de este *hook*. Solo se ejecuta una vez al iniciar.
- **startHook:** Se ejecuta cada vez que se reinicia la máquina de estados. Sirve para inicializar variables, abrir ficheros, hacer comprobaciones de seguridad, etc.
- **updateHook:** Se trata de la función principal, este *hook* se ejecuta de forma cíclica, ya sea por que se le define un periodo con `setPeriod`, es llamado por un `Event` o recibe información a través de un `eventport`.
- **stopHook:** La máquina de estados se puede detener de forma manual o de forma automática llamado a este *hook*.
- **cleanupHook:** Es el último en ejecutarse. Una vez ha terminado el proceso se ejecuta `cleanup` pudiendo escribir la información de estado en el archivo XML, liberar recursos, etc.

Nótese que existen otros *hooks* menos utilizados como `activateHook`, `resetHook` o `fatalHook`. En este texto se ha preferido limitarse a nombrarlos ya que están reservados para sistemas más complejos.

La interacción entre componentes

Cada componente se basa en la clase *TaskContext*, esta clase crea una interfaz pública donde se definen las interacciones entre componente: *Events*, *Properties*, *Commands*, *Methods* y *Data Ports* como se puede ver en la figura 4.2 [13].

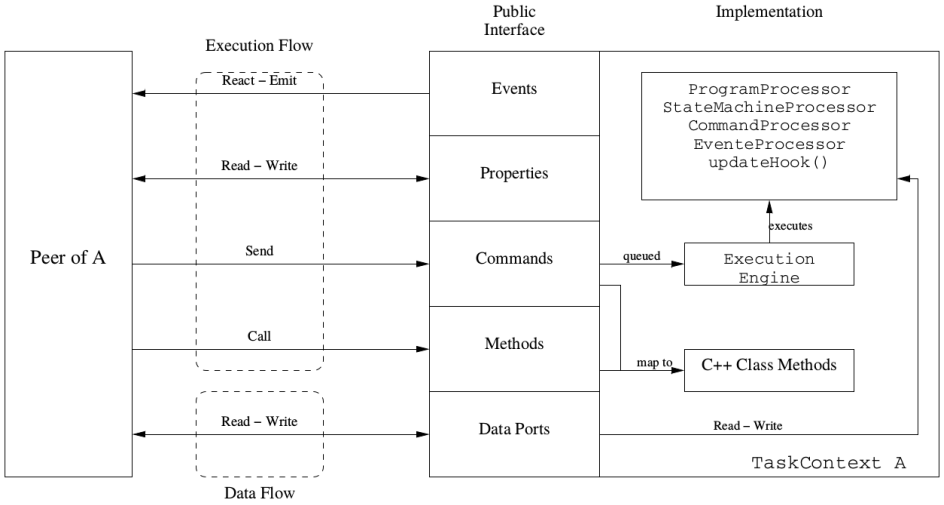


Figura 4.2: Interfaz de un componente de Orocos [13]

Cuando un *Event* o Evento ocurre se ejecuta la función asociada a dicho evento. Esto se puede dar de dos formas, ejecutando la función de forma síncrona o de forma asíncrona. La diferencia reside en que los eventos síncronos se ejecutan como parte del proceso que los ha activado, es decir, en el mismo hilo de ejecución. Mientras que los eventos asíncronos esperan a ser ejecutados cuando se activa el componente en el que están definidos.

Los *Methods* o Métodos se comportan como funciones que pueden ser llamadas desde otros componentes o desde *scripts*. Al igual que ocurre con las funciones en C++, los métodos se ejecutan de forma síncrona con respecto al componente que los llama [13].

Los *Commands* o Comandos son similares a los métodos, con la diferencia de que los primeros se comportan de forma asíncrona. Los comandos no se ejecutan hasta que lo hace el componente en el que han sido definidos.

Las *Properties* o Propiedades son variables que pueden ser utilizadas para configurar los componentes. Estas son accesibles para lectura y escritura a través de un archivo con formato XML, por lo que se pueden utilizar para almacenar valores [67].

Los *Data Ports* constituyen los canales de comunicación entre componentes. Cada puerto se define con un nombre único dentro de cada componente, el tipo de dato y el tipo de puerto. Estos pueden ser de lectura y escritura, solo de lectura que se define como `InputPort` o solo de escritura `OutputPort` [67]. La información se transmite en tiempo real, esto hace que la comunicación

se deba hacer de manera *thread-safe*. Esto es, mientras se está leyendo un puerto se ejecuta un proceso de exclusión mutua que evita que la información cambie durante el proceso [13]. Además, un puerto de entrada se puede configurar para que lance un *Event* cada vez que reciba información. Se definen como `eventports` [67].

El siguiente ejemplo de un componente con el nombre `reset` ilustra como se utiliza la clase `TaskContext` para definir un puerto de salida, un puerto de entrada con evento y un atributo:

(En el archivo reset-component.cpp)

```
Reset::Reset(std::string const& name) : TaskContext(name){

  this->ports(->addPort(``output_port'',output_port);
  this->ports(->addEventPort(``input_port'',input_port);
  this->addAttribute(``max'',max);

  std::cout << ``Reset constructed !'' <<std::endl;
}
```

(En el archivo reset-component.hpp)

```
class Reset : public RTT::TaskContext{
  public:
      Reset(std::string const& name);

      OutputPort <bool> output_port; //Declaramos los puertos y el tipo
      InputPort <int> input_port;

  private:
      int max;
};
```

El Deployer Component

El componente *Deployer*, de la biblioteca *OCL Library*, es el encargado de ejecutar la aplicación de Orocos. La cual, a través del *TaskBrowser*, intercambia datos y comandos entre los componentes y el *Deployer* [65].

El *Deployer* carga y configura el resto de componentes bien introduciéndolos a mano a través de la terminal o, lo más habitual, cargando la descripción de los componentes almacenada en un archivo XML [13], como se muestra en la figura 4.3.

Primero se cargan todos los componentes como bibliotecas usando el comando `import`. Acto seguido se inicializan con el comando `loadComponent`, se pueden inicializar varios componentes diferentes de la misma biblioteca. Se especifican las conexiones entre componentes y los valores de las propiedades, luego el comando `configureComponent` realiza las conexiones especificadas

y almacena los valores de las propiedades. Finalmente se ejecuta el comando `start` que llama al *hook* homónimo del componente.

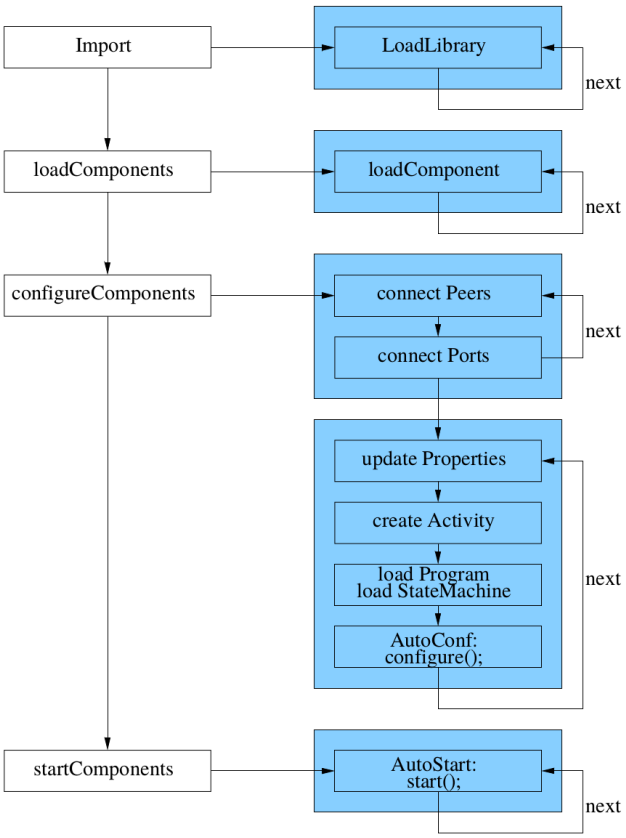


Figura 4.3: Secuencia de inicialización del systema del *Deployer* [13]

A continuación se ve un ejemplo simple de un archivo XML.

```
import(‘‘rtt_ros’’);
ros.import(‘‘master’’); //Importamos los paquetes
ros.import(‘‘reset’’);

//Cargamos los componentes.
loadComponent(‘‘contador’’,‘‘Master’’); //El nombre dle paquete tiene que estar en mayúscula
loadComponent(‘‘reset’’,‘‘Reset’’); //Podemos elegir el nombre y cargar tantos como nos hagan falta

connect(‘‘contador.output_port’’,‘‘reset.input_port’’,ConnPolicy);
connect(‘‘reset.output_port’’,‘‘contador.input_port’’,ConnPolicy);

contador.start_num = 3; //Definimos el valor del atributo
contador.max_reset = 3;
reset.max = 9 ;

reset.configure ;
contador.configure ;

contador.setPeriod(0.5); //El componente contador ( del paquete master) actuará como principal

reset.start; //Iniciamos el componente reset
//contador.start; //El componente contador lo iniciamos desde el deployer, aquí se ha dejado comentado como indicación
```

Capítulo 5

Sensorización externa

Posicionamiento usando realidad aumentada

Normalmente cuando un robot realiza algún tipo de movimiento es necesario que el robot conozca su posición con respecto al entorno. Por lo tanto, se deberá implementar algún método de localización. Estos métodos dependen del tipo de robot, en el tipo de sensores, el sistema de coordenadas utilizado o en el escenario en el que el robot esté operando [68].

Aunque existe una gran variedad de sistemas que se puede utilizar para localización de robots; como pueden ser sistemas de odometría, sonars, GPS's, IMU's, etc.; la visión por computación supone una alternativa simple y suficientemente robusta para tal fin. Utilizando marcadores 2D y procesamiento de imagen por software se puede obviar la necesidad de sistemas más complejos necesarios si utilizamos otros métodos [69].

Con solamente una cámara se puede calcular la posición relativa del robot a partir de dos o más capturas y comparando la posición de algún marcador en las imágenes, como texturas características o formas conocidas. Si se requiere conocer las coordenadas sin movimiento, será necesario conocer las posición relativa de al menos dos marcadores [14]. Pero existe otra posibilidad para determinar la posición relativa del robot con un único marcador. Existen diferentes tipos de marcadores en 2D diseñados especialmente para este fin, vemos algunos ejemplos en la Figura 5.1. Por lo tanto, si conocemos los parámetros de calibración de la cámara, es posible calcular la translación y rotación del marcador a partir de una única imagen [68].

Estos marcadores característicos son conocidos como "fiducial markers". Un sistema de marcadores debe estar compuesto por un set válido de marcadores y un algoritmo con el cuál poder detectarlos. Aunque se pueden usar infinidad de formas e imágenes para crear un sistema de

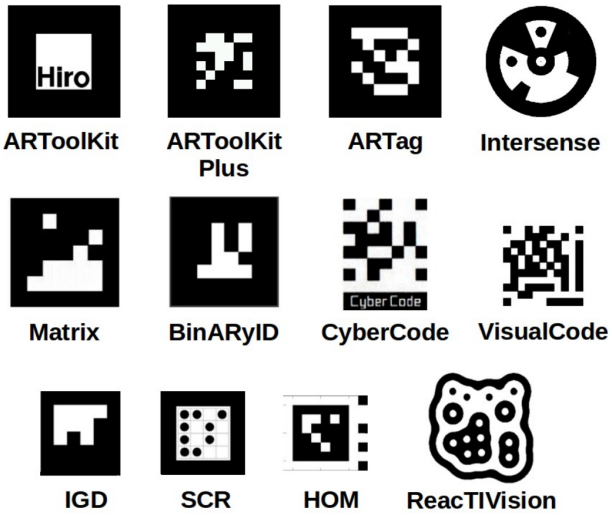


Figura 5.1: Ejemplos de marcadores usados por diferentes softwares [14]

marcadores, los más populares son los marcadores cuadrados y la razón es que se puede calcular la posición de la cámara a partir de las cuatro esquinas [14] [70].

El proceso para detectar este tipo de marcadores se puede dividir en dos pasos. El primer paso consiste en buscar el candidato, esto es, buscar formas cuadradas que parezcan marcadores dentro de la imagen. El segundo paso es la etapa de identificación, donde la codificación del candidato es analizada para determinar si efectivamente se trata de un marcador, y si pertenece al set de marcadores válidos considerados, también llamado diccionario [70].

Para este proyecto se ha optado por utilizar *Aruco*.

Los marcadores de Aruco están definidos como un array de cuadrados de 7x7. Los cuadrados exteriores son negros, pero los interiores pueden ser tanto blancos como negros. El sistema viene con algoritmos implementados en funciones para reconocer estos patrones y gracias al uso de teoría de sistema digitales codificados se consigue un riesgo bastante bajo en cuanto al nivel de confusión y detecciones en falso. Para una correcta detección de los marcadores se asume que el tamaño de estos y los parámetros de la cámara son conocidos. Después de detectar el marcador, las funciones de la biblioteca de ArUco devuelven la posición relativa al centro óptico de la cámara y la matriz de transformación estándar o cuaternión para la rotación [68].

Electromiografía

La electromiografía, o EMG, es una técnica experimental dedicada al desarrollo, almacenamiento y análisis de señales mioeléctricas. Las señales mioeléctricas están formadas por variaciones fisiológicas en el estado de las membranas de las fibras musculares [71].

Esta definición de electromiografía abarca tanto la EMG neurológica como la EMG kinesiológica. A diferencia de la clásica EMG neurológica, donde se analiza en condiciones estáticas la respuesta artificial del músculo debido a una estimulación externa, la EMG kinesiológica se centra en el estudio de la activación neuromuscular de los diferentes músculos en diferentes posturas realizando una tarea, movimientos funcionales, condiciones de trabajo y regímenes de tratamientos o entrenamientos [72].

Dado que el objetivo de este proyecto es el desarrollo de un robot para la rehabilitación, se ha buscado incorporar un sensor de electromiografía al sistema con el que obtener datos adicionales del usuario.

La rehabilitación es una terapia que tiene el propósito de recuperar parcial o totalmente las habilidades motoras de un paciente [73].

Otros investigadores han medido las señales producidas por los músculos para desarrollar sistemas de rehabilitación o prótesis, aunque muchos de estos sistemas solo responden a respuestas binarias, hay o no movimiento. Esto es debido a la gran dificultad que resulta de utilizar sistemas no-lineales y no-estacionarios como son los músculos y la información que se puede obtener de un EMG [74].

El desarrollo de un control basado en EMG queda fuera del alcance de este proyecto debido a la complejidad y la utilización de recursos que supondría. Este proyecto se ha limitado a desarrollar las herramientas necesarias para obtener las señales EMG y poder utilizarlas dentro del framework utilizado en el robot.

Para ello se ha hecho uso de un equipo de la empresa *CED (Cambridge Electronic Design Limited)* que consta del sistema de adquisición de datos *Power 1401* y el preamplificador *1902 amplifier*.

Power 1401

El sistema de adquisición de datos es el encargado de leer la señal analógica enviada por el pre-amplificador y digitalizarla para poder ser usada y almacenada por el ordenador. De esta forma se puede visualizar y/o enviar los datos a un sistema de control.

En la figura 5.2 se muestra una fotografía del sistema de adquisición de datos.

La empresa *CED*, proveedor de estos aparatos, proporciona un software privativo con todas las funcionalidades necesarias para sacarle el máximo partido a sus sistemas. El inconveniente de

Figura 5.2: Imagen del Power 1401 utilizado [15]

este software es que se distribuye a parte del hardware, por lo que también se cobra a parte. El precio de la licencia ronda los 2000€ [15], por lo que enseguida se buscó una alternativa. Por suerte *CED* también proporciona una herramientas de software para programadores de manera gratuita. Dado que el uso que se le quiere dar al sistema de adquisición de datos es muy específico y no se requieren muchas líneas de código, se ha optado por esta solución.

Otro inconveniente surgido debido a los recursos de los que se disponen es que la herramientas de software disponibles para el *power 1401* son para el sistema operativo *Windows*, mientras que todo el sistema que acompaña al robot está desarrollado en *Ubuntu*.

Esta incompatibilidad entre software se ha abordado añadiendo una máquina con *Windows* a la que se conecta el EMC, y mediante *sockets*, se envía la información a un componente de *Orocos* que permite que sea utilizada por cualquier otro componente del sistema.

Siguiendo las indicaciones de [75] se ha configurado el *Power 1401* de manera que esté continuamente leyendo las entradas. Para ellos se deben instalar las librerías, que se pueden descargar de la web [15], y enviarle un *string* con la instrucción adecuada:

```
ADCMEM, I, 2, 0, 512, 0 1 2 3, 0, C, 25;.
```

`ADCMEM, kind, byte, st, sz, chan, rpt, clock, pre, cnt;`

`kind` Modo de operación. 'I' para leer por interrupciones y 'F' para hacerlo de modo secuencial.

`byte` Indica el tamaño de los datos. Se guardan en formato 8-bits si es igual a 1 o 16-bits si es igual a 2. Nota: El formato 8-bits está obsoleto.

`st` Posición de memoria dónde se empiezan a almacenar los datos.

`sz` Tamaño del buffer a almacenar. Si `byte` es 2, `sz` tiene que ser el doble del número de datos que queramos almacenar.

`chan` Canales, y orden, que queremos leer. Estos se pueden repetir.

`rpt` Número de repeticiones. Si igual a 0 está leyendo continuamente.

`clock` Tipo de reloj. C es para 1MHz.

`pre*cnt` El reloj se divide por la multiplicación de estos dos parámetros para dar las lecturas por segundo.

Una vez enviada la orden de leer al sistema de adquisición de datos, se recoge la información con la función `U14toHost()`.

Pre-amplificador aislado 1902

El otro aparato utilizado para las lecturas de EMG es el *pre-amplificador aislado 1902*, el cual se muestra en la figura 5.3. La funcionalidad de este aparato es la de acondicionar la señal recibida de los electrodos para que, por un lado, esté dentro de los niveles de entrada al sistema de adquisición de señales y por otro sea una señal útil para ser utilizada para diagnósticos médicos u otro funcionalidad que se le quiera dar.

Figura 5.3: Imagen del preamplificador 1902 [15]

Si entramos más en detalle, el *1902* tiene dos funcionalidades claramente separadas: El acondicionador de señal y el generador de trigger [76].

El acondicionador de señal: Este proporciona una señal de salida, que dependiendo de las características de la señal de entrada se configura el rango de entrada por medio de software. Esto se consigue modificando las siguiente opciones: la ganancia del amplificador, filtros y offset, y detección de fuera de rango.

El generador de trigger: El generador de trigger proporciona un canal de salida con una señal de pulsos, derivada de dos entradas a elegir seleccionadas por software. Se utiliza para generar señales clara de pulsos con niveles TTL desde una variedad de entradas.

A pesar de que el *pre-amp 1902*, como el *power 1401*, está pensado para ser utilizado con el software distribuido por *CED Spike2* y/o *Signal*, el pre-amplificador se puede configurar y utilizar con el software gratuito *Try1902*, disponible en la web oficial de *CED* [15].

A continuación se describen las diferentes opciones que se pueden encontrar en el panel de control del *Try1902* [76]:

Port: Este es la entrada del conversor ADC del *pre-amp 1401*, cuidado, NO se trata de la entrada RS232 del *1902*.

Input: Esta opción permite seleccionar las señales de entrada desde un transductor o desde el amplificador aislado. Tiene varios modos de operación, como e single-ended o modo diferencial para el transductor, o varios retardos para el clamp del amplificador aislado.

Gain: Esto es la ganancia del voltaje de la entrada seleccionada. Puede no ser igual a uno: algunas entradas tienen una ganancia mínima de 100x o 1000x.

Offset: Esto es, sumar una tensión DC a la entrada y así centrar la señal en cero; valores positivos o negativos (en milivoltios) se pueden introducir por teclado o con las flechas en saltos del 5 %

Low-pass & high-pass filters: Los filtros digitales se pueden configurar con cuatro desplegables, dos para seleccionar el tipo (Bessel o Butterworth) y otros dos para seleccionar el número de polos (2 o 3 polos).

EMG processing: La señal puede ser rectificada (i.e. todos los valores negativos de la señal se invierten). Esto es una práctica habitual en las señales de EMG, además se incluyen las opciones de un filtro paso bajo y una ganancia x10.

Notch filter: Este filtro se usa para atenuar la frecuencia de la red, 50Hz o 60Hz.

AC couple: Este opción bloquea la componente DC de la señal. Puede ser muy útil, por ejemplo, si la señal tiene un pequeño rizado con un nivel de DC muy alto. Se atenúan frecuencias muy bajas, la frecuencia de corte está aproximadamente en 0.16Hz.

Trigger: Elije entre dos entradas de trigger, estas entradas tienen una funcionalidad idéntica. Se puede elegir que el trigger detecte el flanco de subida o el de bajada y siempre habrá uno de los dos triggers seleccionados.

El sensor de fuerza

El control de fuerza, también conocido como retroalimentación táctil, le da a los robots instrucciones precisas basadas en la fuerza que está siendo aplicada en un punto antes de reaccionar. Los seres humanos son infinitamente más adaptables que los robots, pero en el momento que estos últimos son programados para una tarea son capaces de repetirla sin apenas variabilidad [77].

La aplicaciones donde se requiere un control de fuerza son innumerables pero es la industria del automóvil la que más rápido está creciendo y demandando esta tecnología. En la línea de producción, por ejemplo, encontramos la inserción de los ejes, transporte de la carrocería o el montaje de la caja de cambios. Además, el sensado de pares y fuerzas será cada vez más necesario a medida que las tareas colaborativas humano-robot vayan aumentando creando la necesidad de sensores de fuerza y estrategias de control cada vez más sofisticadas [78].

Pero no solo se requieren sensores de fuerza en la industria, diversos campos de investigación hacen usos de estos sensores. Investigadores odontológicos investigan cual es la fuerza y dirección que hacemos al morder, ingenieros en robótica quieren medir el par y fuerza en los dedos de un guante robótico, desarrolladores de prótesis que necesitan un mapa de fuerzas preciso de las muñecas, dedos, pies, y demás articulaciones, otros investigadores necesitan saber el progreso de pacientes en tratamiento de rehabilitación o empresas que necesitan monitorizar el estado de los trabajadores con riesgo de sufrir Síndrome del Tunel Carpiano son solo algunos de los ejemplos en los que se necesitan sensores de alta precisión capaces de medir simultáneamente las fuerzas y pares en los ejes x, y y z [79].

Se ha dispuesto del *F/T Transducter Delta* de *ATI*(Figura 5.4) al que se la ha acoplado un arnés para posicionar el tobillo. Este sensor es capaz de medir altos valores de fuerza(ver tabla 5.1) y par con gran precisión(ver tabla 5.2) [80].

Eje sobrecargado	Fxy	Fz	Txy	Tz
	$\pm 3700N$	$\pm 10000N$	$\pm 280Nm$	$400Nm$

Tabla 5.1: Máximos valores de par y fuerza soportados

El acceso a las lecturas del sensor se pueden hacer por diferentes canales, dependiendo de cual sean las especificaciones y necesidades del sistema se optará por uno u otro. El sensor viene con una *NetBox*(Figura 5.5) que permite la conexión vía Ethernet o BUS CAN. Si se conecta el cable de Ethernet se tienen varias opciones de comunicación [81]:

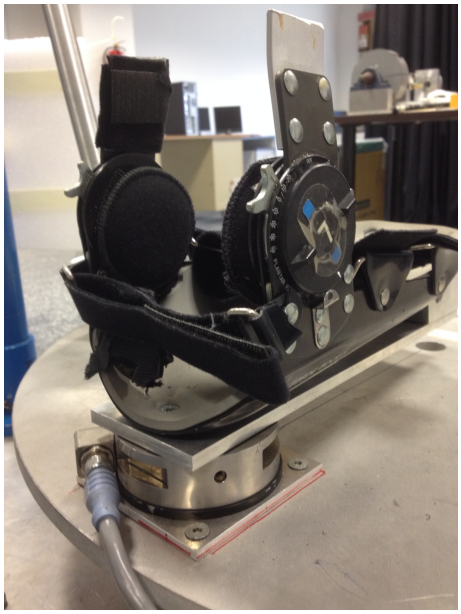


Figura 5.4: Imagen del sensor de fuerza con arnés instalado.

Calibración (SI)	**Fx,Fy (N)**	**Fz (N)**	**Tx, Ty (Nm)**	**Tz (Nm)**	**Fx,Fy (N)**	**Fz (N)**	**Tx, Ty (Nm)**	**Tz (Nm)**
SI-165-15	165	495	15	15	1/32	1/16	1/528	1/528
SI-330-30	330	990	30	30	1/16	1/8	5/1333	5/1333
SI-660-60	660	1980	60	60	1/8	1/4	10/1333	10/1333
	Rangos				Resolución (DAQ, Net F/T)			

Tabla 5.2: Resolución del sensor Delta con diferentes modos de calibración.

- EtherNet/IP
- DeviceNet
- UDP
- TCP

Incluir las lecturas del sensor en el sistema control implica una programación a bajo nivel, ya que se trabaja con el Middleware Orocos. Por esto se ha preferido crear un componente en el que se ejecuta un socket UDP que se comunica con el sensor. Además, se tiene la ventaja que este sistema es más rápido que utilizar DeviceNet o EtherNet/IP ya que estos trabajan en unas capas superiores.

Figura 5.5: Imagen de la NetBox

Capítulo 6

Controladores

A lo largo de la vida del proyecto se han ido probando diferentes estrategias de control con el robot *paralelo3DOF*. El más básico es un control pasivo con compensación de las acciones gravitatorias (Sección 6.1), más adelante se ha añadido la compensación de otros efectos dinámicos del robot como son la inercia y el efecto Coriolis (Sección 6.2).

Además, se ha buscado poder medir la acción que el usuario está haciendo sobre la plataforma del robot. Para esto se ha añadido un sensor de fuerza con una estrategia de control híbrida que combina la posición y una fuerza externa ejercida sobre el robot (Sección 6.4).

Control pasivo con compensación de la gravedad

Se ha empezado probando estrategias de control basadas en la pasividad. Como se define en [82], los controladores basados en la pasividad resuelven el problema de controlar un robot mediante la explotación de la estructura física del sistema robotizado, y en especial de su propiedad de pasividad. La filosofía de diseño de estos controladores es reconfigurar la energía natural del sistema de tal forma que se consiga el objetivo de seguimiento del control.

Este primera estrategia de control se basa en un PID al que se le añade un término compensatorio para corregir los efectos de la gravedad (6.1).

$$u(t) = K_{\mathrm{p}} e(t) + K_{\mathrm{i}} \int_0^t e(\tau)\, d\tau + K_{\mathrm{d}} \frac{de(t)}{dt} i - G(q,t) \tag{6.1}$$

Donde $e(t)$ es el error de la posición del robot, $u(t)$ es la acción de control y K_{p}, K_{i} y K_{d} son las constantes que regulan el PID. Ademas, al PID se le añade el termino gravitacional

$G(q,t)$ que depende de las coordenadas generalizadas. O dicho de otra manera, se calcula en cada instante igualando las velocidades y aceleraciones a cero [83]. Se trata de la fuerzas que tiene que compensar cada actuador debido al propio peso del robot.

A pesar de su simplicidad, este sistema de control presenta dos problemas importantes. Por un lado, el término gravitacional puede llegar a ser muy complejo de calcular, ya que dependiendo del robot y su modelo dinámico el método de cálculo puede no converger o resultar demasiado lento para hacer un control en tiempo real. El otro problema es que puede dar lugar al fenómeno de zona muerta o que cualquier error en la estimación del término de gravedad puede causar una variación en el punto de equilibrio y por tanto un error de posición estacionario [84].

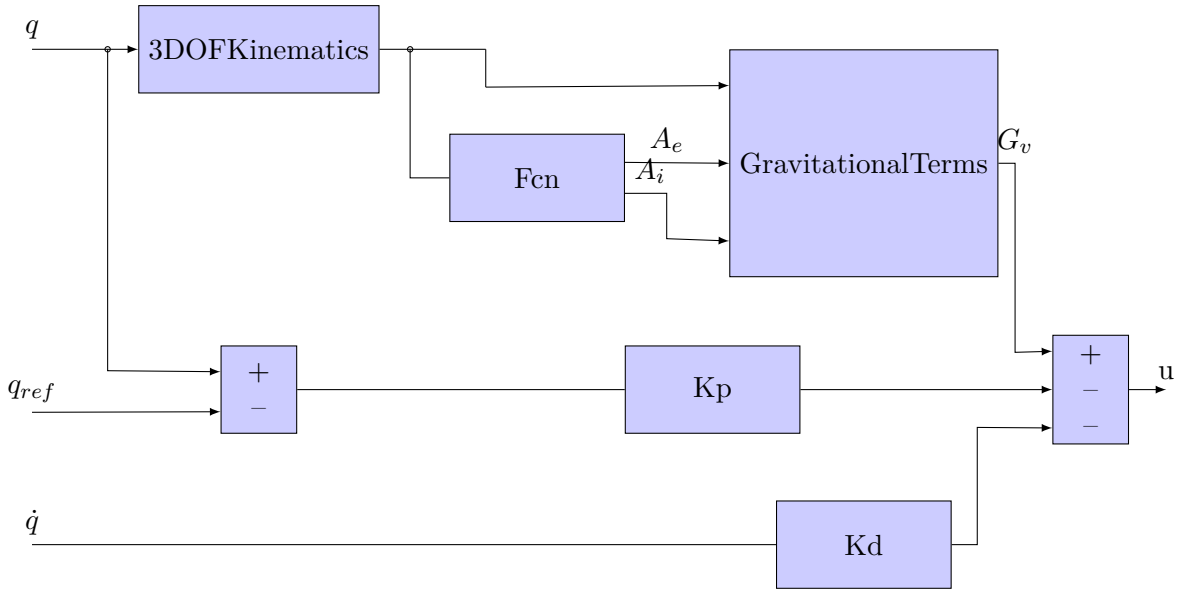


Figura 6.1: Resultado experimentales del control pasivo con compensación de la gravedad

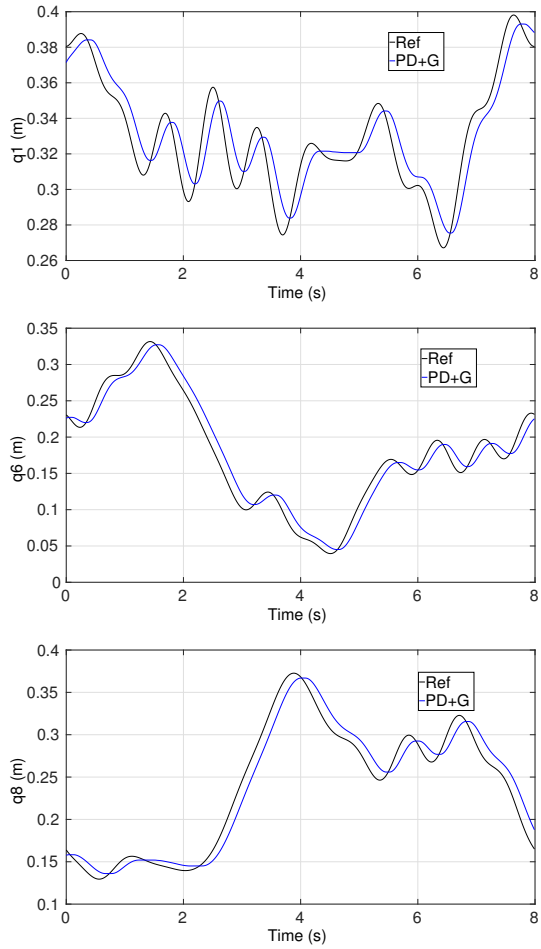


Figura 6.2: Resultado experimentales del control pasivo con compensación de la gravedad

Control pasivo con compensación de la dinámica del robot

Esta estrategia de control es una extensión de la anterior en la que, siguiendo el sistema propuesto por [85], se utiliza la ecuación del movimiento completa (6.2) añadiendo los término de Coriolis y de inercia además del término gravitacional.

$$\vec{\tau} = \mathbf{M}(\vec{q}, \vec{\Phi})\ddot{\vec{q}} + \vec{C}(\vec{q}, \dot{\vec{q}}, \vec{\Phi}) + \vec{G}(\vec{q}, \vec{\Phi}) \tag{6.2}$$

Donde $\mathbf{M}$ es la matriz de inercia, $\vec{C}$ es el vector de Coriolis y $\vec{G}$ es el vector de fuerzas gravitacionales [83].

Cabe destacar que se trata de un robot paralelo, los cuales tienen un acoplamiento dinámico más fuerte que los robots seriales. Una de las diferencias más importantes es que mientras que en los robots seriales todas las articulaciones son activas, no ocurre lo mismo en el caso de los

RP donde pueden haber articulaciones pasivas. Como ya se ha comentado anteriormente, estas coordenadas pasivas tienen que ser calculadas en cada loop de control con a partir de la dinámica directa con métodos iterativos como *Newton - Raphson* los cuales no siempre convergen o no lo hacen suficientemente rápido [83]. Esto se resuelve modificando la energía cinética y potencial de tal forma que el controlador sea global y asimptóticamente estable [84].

La ecuación (6.2) se completa con un PD para tener un control en bucle cerrado (6.3).

$$\vec{\tau}_c = \mathbf{M}(\vec{q}, \vec{\Phi})\ddot{\vec{q}} + \vec{C}(\vec{q}, \dot{\vec{q}}, \vec{\Phi}) + \vec{G}(\vec{q}, \vec{\Phi}) + -K_{\mathrm{p}}e - K_{\mathrm{d}}\dot{e} \tag{6.3}$$

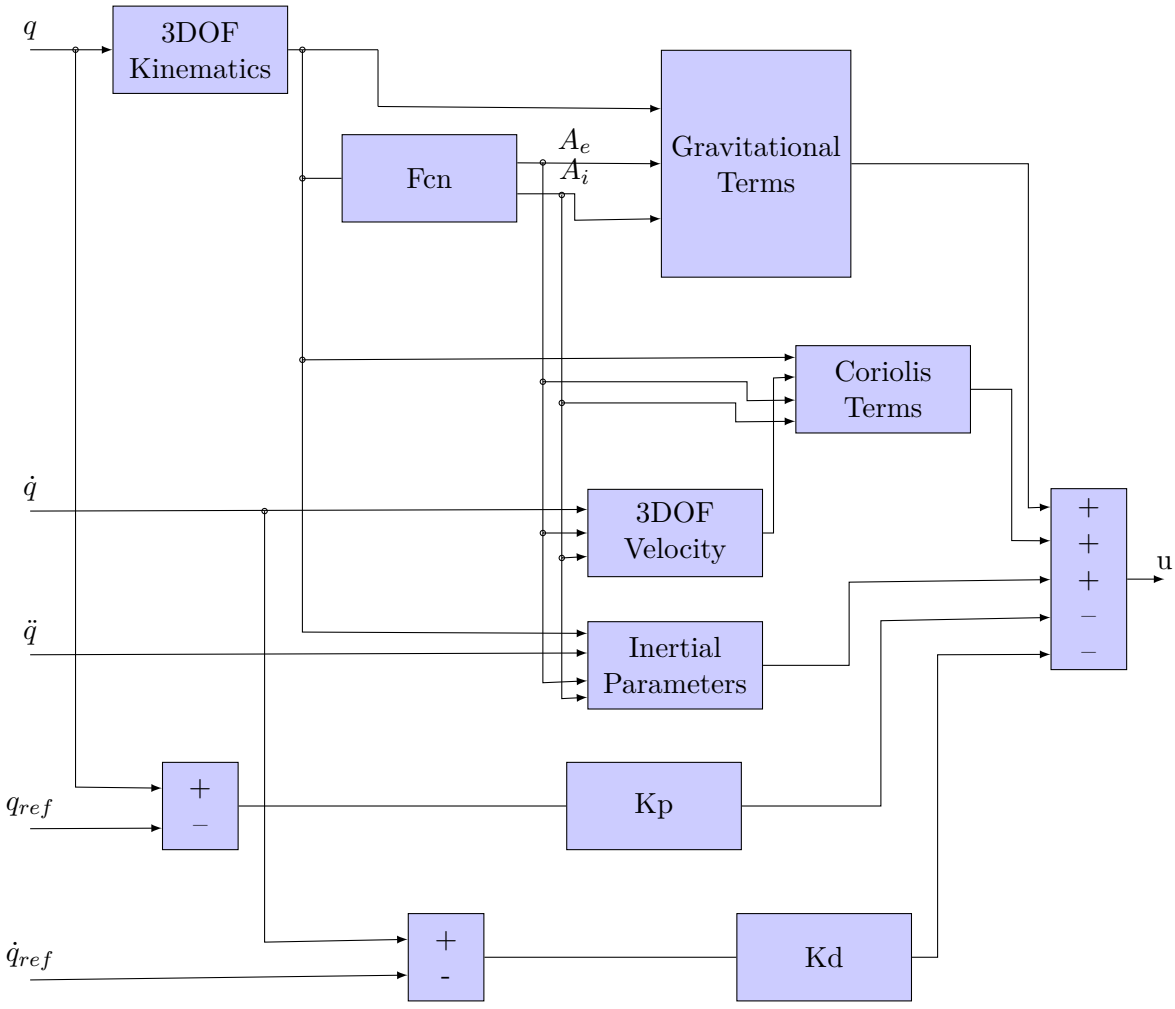


Figura 6.3: Diagrama de bloques del control pasivo con compensación de la dinámica del robot

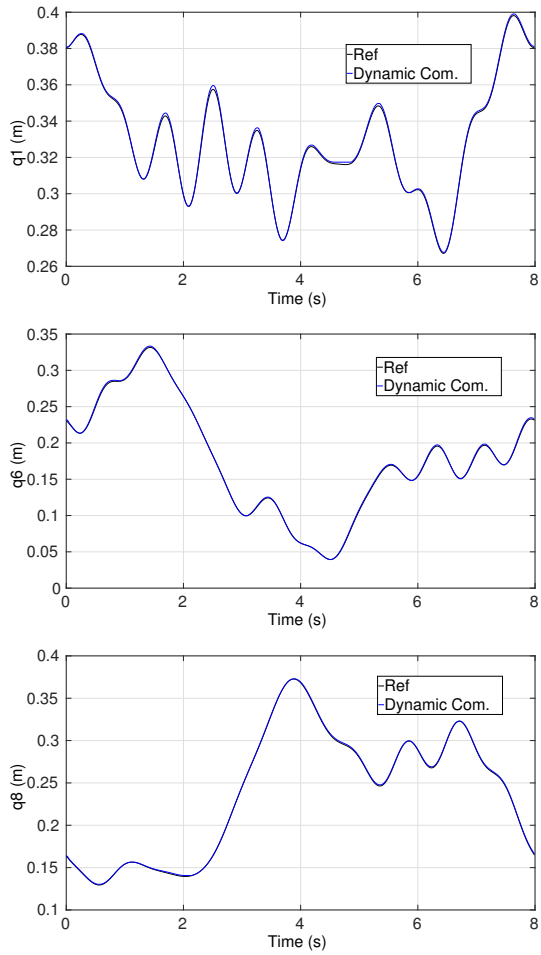


Figura 6.4: Resultado experimentales del control pasivo con compensación de la dinámica del robot

Control por dinámica inversa

Dada la naturaleza no lineal de la ecuación 6.2 debido a las matrices **C** y **G** se ha propuesto un control por dinámica inversa [16].

La idea detrás de este control es encontrar una acción de control

$$\tau_{dinv} = f(q, \dot{q}, t) \tag{6.4}$$

tal que al sustituirla en la ecuación 6.2 el resultado sea un sistema lineal en bucle cerrado [86]. Por lo general, para sistemas no lineales esta acción de control puede ser realmente complicada o incluso imposible de encontrar pero para robots con una dinámica como la representada en 6.2 el problema tiene una solución bastante sencilla [87].

Ley de control	a
Control Punto a Punto	$-K_d\dot{q} - K_p e$
Control por Trayectoria	$\ddot{q}_d - K_d\dot{e} - K_p e$

Tabla 6.1: Controladores por dinámica inversa [16]

Dado un sistema con la forma de (6.5)

$$x^{(n)} = f(x) + b(x)u \tag{6.5}$$

donde $f(x)$ es una función de estados no lineales y u la acción de control. Se cancelan las no linealidades cuando

$$u = \frac{1}{b}\left[a - f\right] \tag{6.6}$$

Para el caso que nos ocupa, despejando la aceleración de la ecuación 6.2 queda como

$$\ddot{q} = M^{-1}(q)(\tau - C(q,\dot{q})\dot{q} - G(q)) \tag{6.7}$$

y a partir de las ecuaciones (6.5) y (6.7) obtenemos que

$$f(x) = M^{-1}(q)(-C(q,\dot{q})\dot{q}) - G(q)) \tag{6.8a}$$

$$b(x) = M^{-1}(q) \tag{6.8b}$$

Si sustituimos en la ecuación (6.6) y simplificamos llegamos a la expresión

$$\tau = M(q)a + C(q,\dot{q})\dot{q} - G(q) \tag{6.9}$$

Donde se deduce que $a = \ddot{q}$ [16][86][87].

Esto implica que a puede usarse para controlar u sistema escalar lineal [87]. En [16] se han propuesto dos estrategias de control, un control punto a punto y otro por trayectoria (ver tabla 6.1).

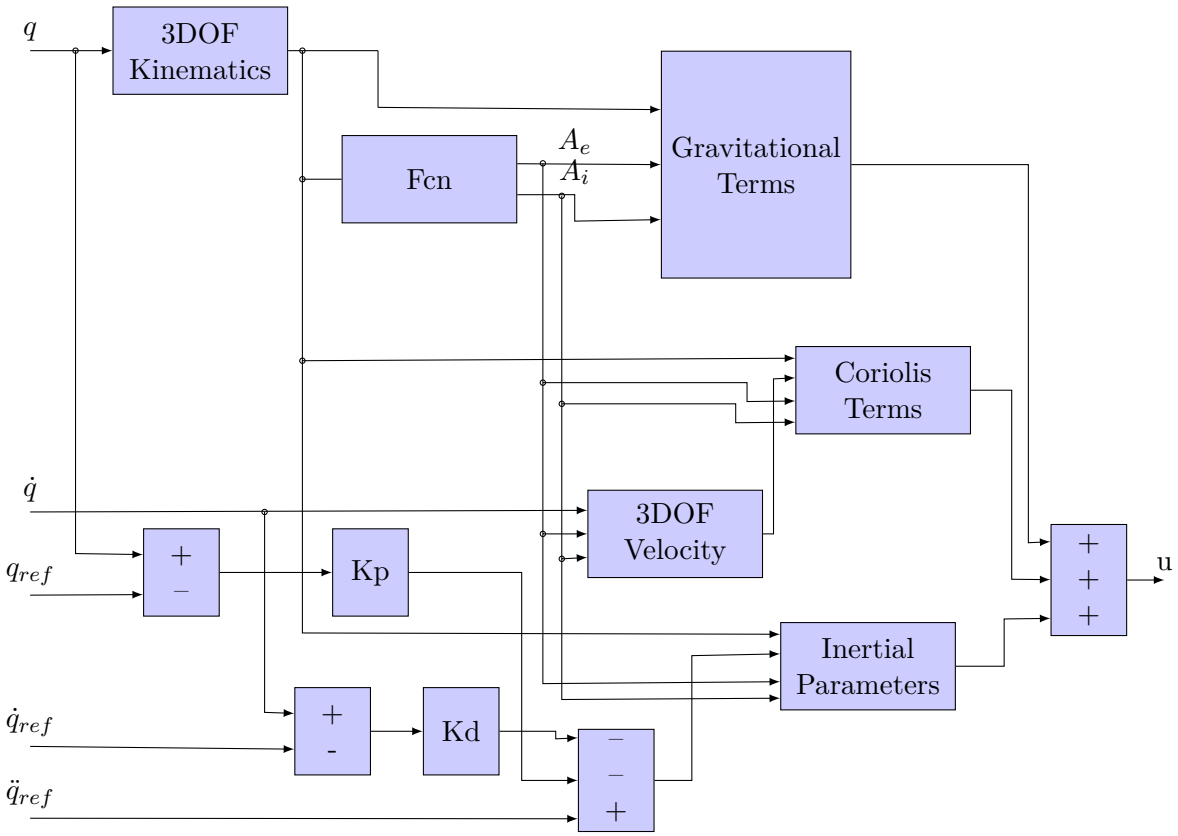


Figura 6.5: Diagrama de bloques del control por dinámica inversa

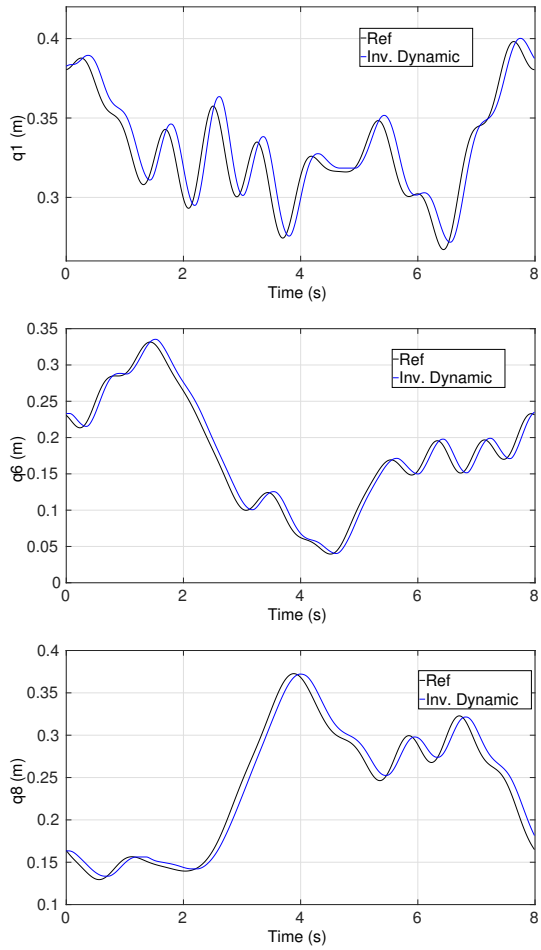


Figura 6.6: Resultado experimentales del control por dinámica inversa

Control híbrido de posición y fuerza

Por lo general los robots únicamente tienen un control de posición pero añadiendo un control de fuerza es posible realizar una adaptación flexible que corrija las imperfecciones del sistema. El movimiento controlado por fuerza en diferentes sistemas mecánicos ha sido un caso de estudio experimental durante dos décadas pero no ha sido hasta recientemente que los controladores de robots industriales han empleado esta tecnología en aplicaciones donde se conocen las posiciones y fuerzas están en las especificaciones [88].

Aunque existen varias estrategias de control de fuerza en este proyecto se ha optado por un control explícito, el cual consiste en indicar una fuerza de referencia y medir los valores de fuerzas con el objetivo de que los valores medidos sigan los de referencia con el menor error posible [89].

Normalmente se trata de un control lineal, como es un control PID (6.10a). Con un término integral que proporciona un error igual a cero frente a una entrada constante y un término derivativo que amortigua el sistema.

Se ha optado por un control simple ya que es importante empezar por un control conocido y poder destacar sus puntos fuertes y débiles. Si con un PID el control es suficientemente bueno no harán falta desarrollar controladores más avanzados como no lineales o adaptativos, en caso contrario habrá que recurrir a otras técnicas más sofisticadas [89].

Se puede probar matemáticamente que el termino integral puede hacer que el sistema resulte inestable. Se propone utilizar una prealimentación (o *feedforward*) de fuerza que sustituya al integrador (6.10b). Si, además, el sensor de fuerza tiene cierto ruido el cálculo de la derivada acumula tanto error que resulta inútil o incluso contraproducente para controlar el robot. Por esto, se suele utilizar la velocidad como equivalente de la derivada de la fuerza [16][89][88]. El resultado viene dado en (6.10c).

$$F = K_p \left(F_{ref} - f\right) + K_I \int \left(F_{ref} - f\right) dt + K_d \frac{d}{dt} \left(F_{ref} - f\right) \tag{6.10a}$$

$$F = F_{ref} + K_p \left(F_{ref} - f\right) + K_d \frac{d}{dt} \left(F_{ref} - f\right) \tag{6.10b}$$

$$F = F_{ref} + K_p \left(F_{ref} - f\right) - K_v \dot{x} \tag{6.10c}$$

Donde F es la acción de control, F_{ref} es la referencia de fuerza, f la fuerza aplicada por el robot, $\dot{x}$ la velocidad y K_p, K_i, K_d y K_v las constantes proporcional, integral, derivativa y de amortiguación respectivamente.

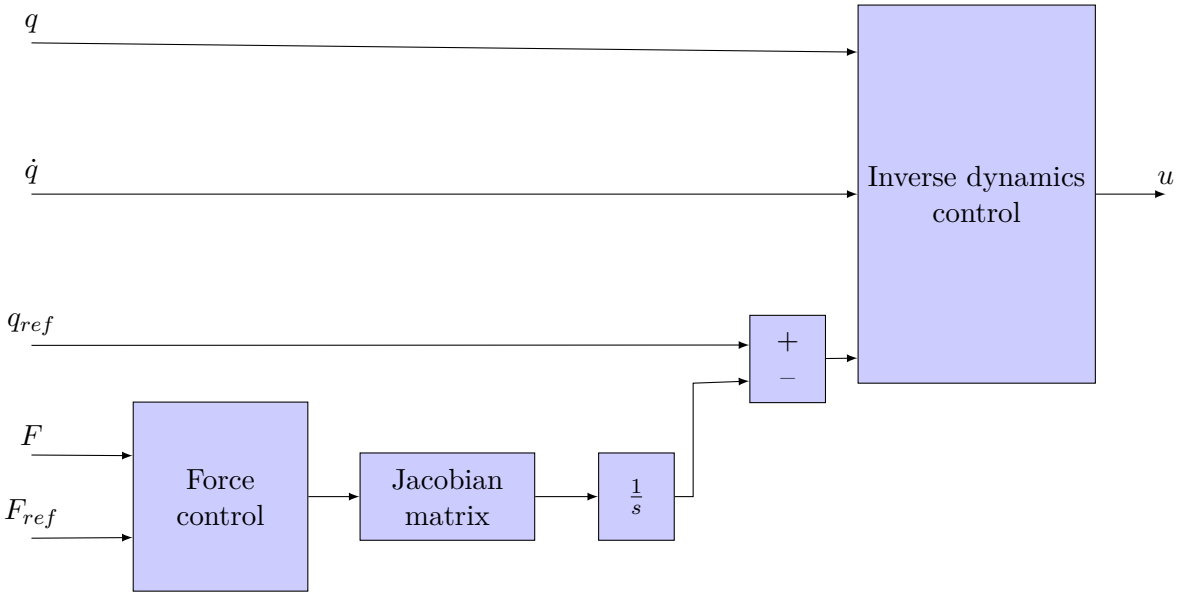
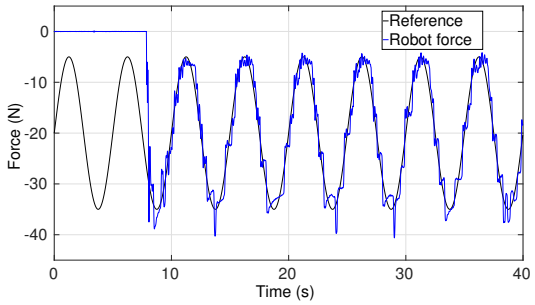
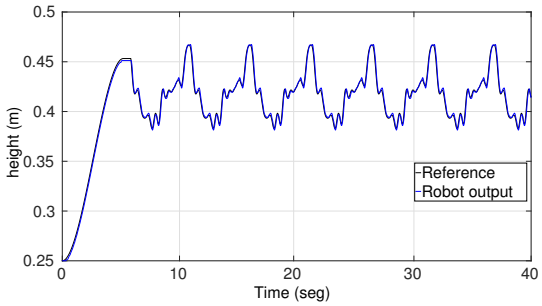


Figura 6.7: Diagrama de bloques del control híbrido de posición y fuerza.



(a) Resultado experimentales del control de fuerza. Nótese que el control no empieza a actuar hasta que el robot ha alcanzado la posición inicial.



(b) Resultado experimentales del control de posición (solo altura) siguiendo la referencia compensada con el jacobiano.

Control por dinámica inversa con observador de estados

Cuando se diseña una estrategia de control generalmente se asume que se dispone de toda la información necesaria para la retroalimentación del bucle cerrado, en realidad, en un sistema real no es práctico disponer de un sensor para cada medida que recoja toda esa información, ya que encarecería excesivamente el conjunto. Por lo tanto se opta por otros métodos para obtener esta información.

Este es el caso de la velocidad. Normalmente la velocidad se calcula diferenciando la señal de la posición posición conocida la frecuencia de muestreo, lo que puede dar problemas de precisión sobre todo a bajas velocidades [90]. Otra manera de abordar el problema es utilizando un *observador*. Este método calcula las variables de estado no medidas de un sistema observable a partir las variables de estado disponibles.

El diseño de un observador de estados consta de dos pasos:

- Se construye un sistema dinámico auxiliar que, a partir de las entradas y salidas, es capaz de reconstruir las variables de estados necesarias. Esto es el observador en sí.
- Se diseña un controlador que se encarga de reducir el error entre las variables de estado conocidas y las variables de estado calculadas a partir de las observadas.

Esta técnica se puede realizar en dos pasos diferenciados en los sistemas lineales gracias al principio de separación [91]. Dado que este no es el caso de nuestro sistema, el ajuste del observador se tiene que hacer teniendo en cuenta el controlador utilizado [25].

Para diseñar el observador de la velocidad se construye un sistema dinámico auxiliar que reconstruye asimptóticamente la señal de la velocidad a partir de las medidas de entrada y salida, i.e. el par (τ) y la posición (q). El observador propuesto por Berghuis, Löhnberg y Nijmeijer (1991) [90] se explica a continuación:

$$\begin{cases} \dot{\hat{q}} = z + L_d \cdot \tilde{q} \\ \dot{z} = M^{-1}(q,\theta) \cdot (\tau - C(q,\dot{q}_0,\theta) \cdot \dot{q}_0 - G(q,\theta) + L_{p1} \cdot \tilde{q}) + L_{p2} \cdot \tilde{q} \end{cases} \tag{6.11}$$

donde $\left[\hat{q}^T z^T\right]$ es el estado del observador, $\dot{\hat{q}}$ representa la velocidad estimada, $\tilde{q} \equiv q - \hat{q}$ es el error en la estimación de la posición por el observador, $L_d = L_d^T > 0$, $L_{p1} = L_{p1}^T \geq 0$,$L_{p2} = L_{p2}^T \geq 0$ y $\dot{q}_0 = \dot{\hat{q}} - \Lambda_2$ con $\Lambda_2 = \Lambda_2^T$.

El par que aplica el controlador será

$$\begin{cases} \tau_c = M(q,\theta) \cdot a + C\left(q, \dot{\hat{q}}, \theta\right) + G(q,\theta) \\ a = \ddot{q}_d - K_d \cdot \left(\dot{\hat{q}} - \dot{q}_d\right) - L_p \cdot e \end{cases} \tag{6.12}$$

donde $\dot{\hat{q}}$ representa la velocidad estimada, obtenida a partir del observador no lineal en bucle cerrado. Ajustando el observador a la estrategia de control utilizada se tiene que

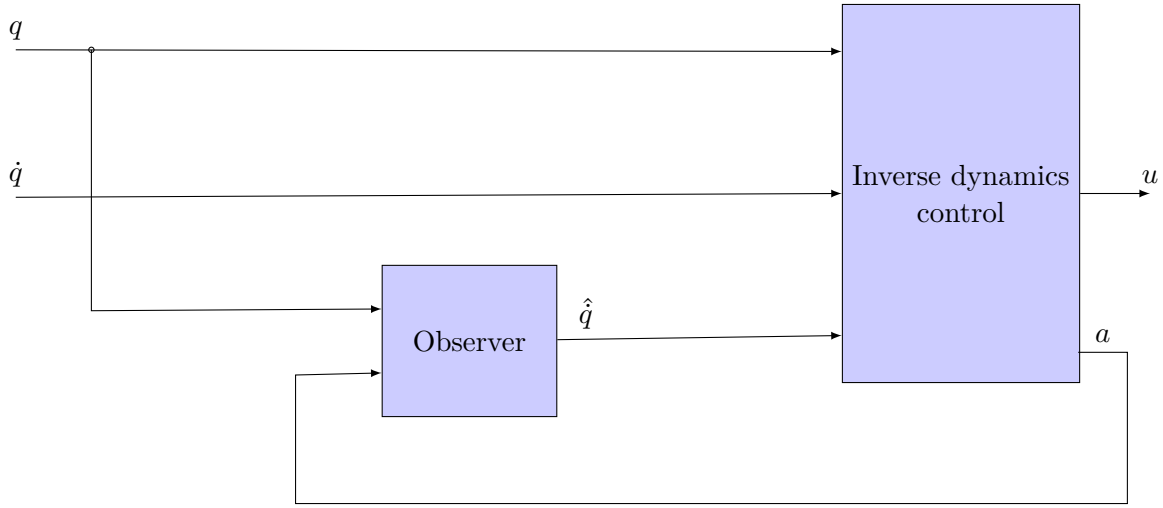
$$\begin{cases} \dot{\hat{q}} = z + L_d \cdot \tilde{q} \\ \dot{z} = M^{-1}(q,\theta) \cdot \left(\tau - C\left(q, \dot{\hat{q}}_0, \theta\right) \cdot \dot{\hat{q}}_0 - G(q,\theta)\right) + L_{p2} \cdot \tilde{q} \end{cases} \tag{6.13}$$

Y combinando las ecuaciones (6.12) y (6.13) se obtiene:

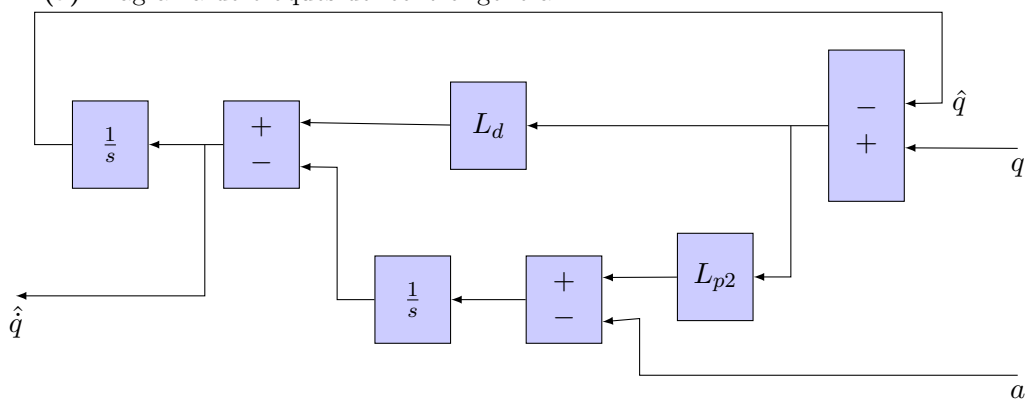
$$Controlador \begin{cases} \tau_c = M(q,\theta) \cdot a + C\left(q, \dot{\hat{q}}, \theta\right) \cdot \dot{\hat{q}} + G(q,\theta) \\ a = \ddot{q}_d - K_d \cdot \left(\dot{\hat{q}} - \dot{q}_d\right) - K_p \cdot e \end{cases} \tag{6.14}$$

$$Observador \begin{cases} \dot{\hat{q}} = z + L_d \cdot \tilde{q} \\ \dot{z} = a + L_{p2} \cdot \tilde{q} \end{cases} \tag{6.15}$$

Es interesante destacar que el observador resulta ser lineal gracias a la acción de control, esto es debido a la propiedad de linealización por retroalimentación del método del par calculado.



(a) Diagrama de bloques del control general



(b) Diagrama de bloques del control del observador de estados

Figura 6.9: Diagrama de bloques de control por dinámica inversa con observador de estados

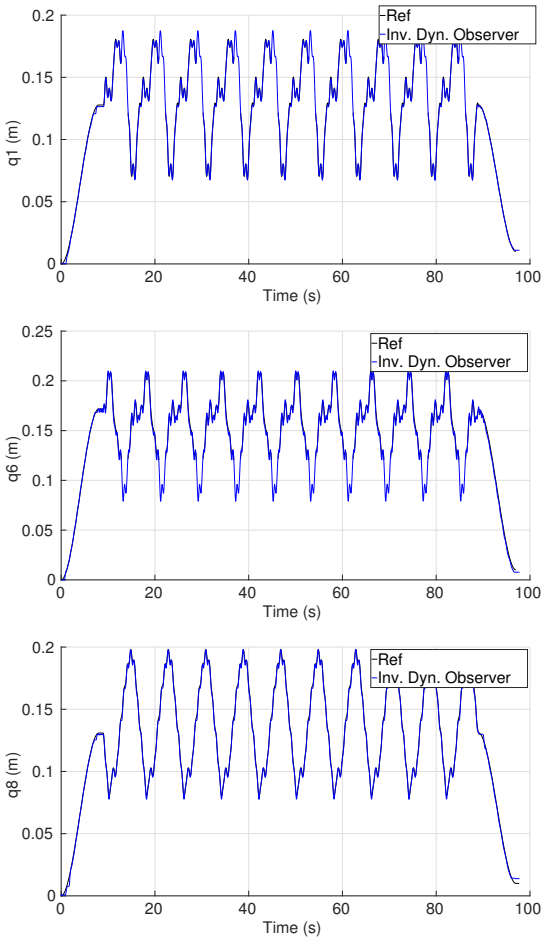


Figura 6.10: Resultado experimentales del control por dinámica inversa con observador de estados

Capítulo 7

Conclusiones

A lo largo de este trabajo centrado en el desarrollo de un robot paralelo para la rehabilitación se ha podido experimentar con dos campos tan diferentes como la robótica y la medicina. La robótica ha estado tradicionalmente ligada a la industria, donde la interacción con personas se limita a su programación o mantenimiento. En cambio cuando se utilizan los robots en la medicina, la interacción con estos es total. Esto hace que cualquier fallo en el sistema pueda poner en peligro la integridad física del usuario.

Se han buscado nuevas técnicas de desarrollo que aumenten la eficiencia del sistema. Esto se ha conseguido con *Ingeniería de Software Basada en Componentes*, en donde se construyen bloques de código independientes que se conectan entre sí para crear un sistema completo. Al tratarse de bloques independientes, estos se pueden modificar, rehusar, sustituir o añadir a una parte del sistema de forma que no se tenga que reprogramar el resto del conjunto. Con este método el desarrollo de software se vuelve muy flexible, ya que un desarrollador se puede centrar en un único componente del sistema y compartirlo con el resto del equipo, el cual solo tendría que conectarlo. Además, para hacerlo más accesible, se ha escogido el *frameworks Orocos* que se encuentran bajo licencias de software libre, el cual está incluido dentro de los paquetes de *ROS*, también software libre.

La modularidad del software basado en componentes ha sido clave a la hora de desarrollar las estrategias de control. Se ha empezado con un control simple con los componentes básicos y a partir de ahí se han ido acoplando módulos hasta llegar a un control por dinámica inversa al que se le ha añadido un observador de estados que calcula la velocidad de los actuadores a partir de los estados del robot. Los resultados de este control con observador sirvieron para la publicación del artículo titulado *Controller-observer design and dynamic parameter identification for model-based control of an electromechanical lower-limb rehabilitation system.*

Otro aspecto en el que se ha investigado ha sido el uso de diferentes sensores externos. Se ha visto que un sensor de fuerza con el correspondiente control hace que el usuario pueda "introducir" información al sistema de manera que se pueda interactuar con el robot. También se ha visto que es posible introducir información al sistema por medio de visión artificial gracias a la librería *ArUco*, o incluso leer las señales electromiográficas (EMG) de los músculos del paciente lo que podría ser un indicador del nivel de esfuerzo que se está realizando.

Como conclusión final se extrae que es posible introducir la robótica en el sector médico y que esto puede suponer un gran avance para ambos campos. Por otro lado esto no va a ser un camino fácil ya que hay que asegurar que las interacciones hombre-máquina son seguras y no ponen en peligro la integridad física de los usuarios. Por suerte se está investigando y desarrollando tecnologías que en un futuro permitirán facilitar el trabajo de los profesionales sanitarios, y por tanto, mejorar la atención recibida por los pacientes.

Bibliografía

[1] R. Clavel, *Conception d' un robot parallèle rapide à 4 degrés de liberté.* PhD thesis, Ecole Polytechnique Federale de Lausanne, 1991.

[2] "The NASA Ames Vertical Motion Simulator A Facility Engineered for Realism,"

[3] H. McClintock, F. Z. Temel, N. Doshi, J.-s. Koh, and R. J. Wood, "The millidelta: A high-bandwidth, high-precision, millimeter-scale delta robot," *Science Robotics*, vol. 3, no. 14, 2018.

[4] V. Mata, "Robot 3UPE-RPU." Documento interno., Marzo 2017.

[5] Festo, *Electric cylinder DNCE. Operating instructions.* Festo, 1109e ed.

[6] Maxon, *Maxon DC Motors RE 40 Graphite Brushes 150Watts. Technical Specifications.* Maxon, 1.0 ed.

[7] Maxon, *Maxon Brake AB 28, 24VDC, 0.4 Nm . Technical Specifications.* Maxon, 1.0 ed.

[8] H. R. Everett, *Sensors for Mobile Robots: Theory and Applications.* 1995.

[9] Advantech, *PCL-1784 Datasheet.*

[10] Advantech, *PCI-1720U Datasheet.*

[11] Advantech, *PCL-812pg Datasheet.*

[12] G. Infantes, C. Lesire, H. de Plinval, and F. Teichteil, "An orocos-based decisional architecture for the ressac missions," *CAR, Toulouse, France*, 2009.

[13] D. C. Santini and W. F. Lages, "An architecture for robot control based on the orocos framework," in *Proceedings of the 4th Workshop on Applied Robotics and Automation*, 2010.

[14] S. Garrido-Jurado, R. M. noz Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280 – 2292, 2014.

[15] CED, "Cambridge Electronic Design Limited." http://ced.co.uk/.

[16] J. I. Cazalilla-Morenas, *Diseño e Implementación de un sistema de control de robots mediante la ingeniería del software basada en componentes. Aplicación a un robot paralelo de 3DOF.* PhD thesis, Universitat Politècnica de València, España, Mayo 2017.

[17] W. Alcocer, L. Vela, A. Blanco, J. Gonzalez, and M. Oliver, "Major trends in the development of ankle rehabilitation devices.," *DYNA*, vol. 79, pp. 45 – 55, 12 2012.

[18] R. Riener, "Control of robots for rehabilitation," in *EUROCON 2005 - The International Conference on "Computer as a Tool"*, vol. 1, pp. 33–36, Nov 2005.

[19] H. Krebs, J. Palazzolo, L. Dipietro, M. Ferraro, J. Krol, K. Rannekleiv, B. Volpe, and N. Hogan, "Rehabilitation robotics: Performance-based progressive robot-assisted therapy," *Autonomous Robots*, vol. 15, pp. 7–20, Jul 2003.

[20] E. Akdoan and M. A. Adli, "The design and control of a therapeutic exercise robot for lower limb rehabilitation: Physiotherabot," *Mechatronics*, vol. 21, no. 3, pp. 509 – 522, 2011.

[21] H. Vallery, J. Veneman, E. Van Asseldonk, R. Ekkelenkamp, M. Buss, and H. Van Der Kooij, "Compliant actuation of rehabilitation robots," *IEEE Robotics & Automation Magazine*, vol. 15, no. 3, 2008.

[22] M. Lee, M. Rittenhouse, and H. A. Abdullah, "Design issues for therapeutic robot systems: results from a survey of physiotherapists," *Journal of Intelligent and robotic systems*, vol. 42, no. 3, pp. 239–252, 2005.

[23] O. Fukuda, T. Tsuji, A. Ohtsuka, and M. Kaneko, "Emg-based human-robot interface for rehabilitation aid," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 4, pp. 3492–3497, IEEE, 1998.

[24] J. Cao, S. Q. Xie, R. Das, and G. L. Zhu, "Control strategies for effective robot assisted gait rehabilitation: the state of art and future prospects," *Medical Engineering and Physics*, vol. 36, no. 12, pp. 1555–1566, 2014.

[25] A. Valera, M. Diaz-Rodríguez, M. Vallés, E. Oliver, V. Mata, and A. Page, "Controller-observer design and dynamic parameter identification for model-based control of an electromechanical lower-limb rehabilitation system," pp. 1–45, 07 2016.

[26] M. Zinn, O. Khatib, B. Roth, and J. K. Salisbury, "A new actuation approach for human friendly robot design," in *Experimental Robotics VIII*, pp. 113–122, Springer, 2003.

[27] Ángel Valera, V. Mata, Álvaro Page, M. Vallés, and E. Oliver, *Modelado y Simulación de Robots Paralelos con V-REP. Aplicación de tele-operación y monitorización de un Robot de 3 Grados de Libertad*, ch. 9. Pontificia Universidad Católica de Ecuador, 2017.

[28] J.-P. Merlet, "Parallel robots: open problems," in *Robotics research*, pp. 27–32, Springer, 2000.

[29] L. Rey and R. Clavel, "The delta parallel robot," in *Parallel Kinematic Machines*, pp. 401–417, Springer, 1999.

[30] M. Wapler, V. Urban, T. Weisener, J. Stallkamp, M. Dürr, and A. Hiller, "A stewart platform for precision surgery," *Transactions of the Institute of Measurement and Control*, vol. 25, no. 4, pp. 329–334, 2003.

[31] A. Codourey, "Dynamic modeling of parallel robots for computed-torque control implementation," *The International Journal of Robotics Research*, vol. 17, no. 12, pp. 1325–1336, 1998.

[32] S. Briot and I. Bonev, "Are parallel robots more accurate than serial robots?," *CSME Transactions*, vol. 31, no. 4, pp. 445–456, 2007.

[33] A. J. Wavering, "Parallel kinematic machine research at nist: Past, present, and future," in *Parallel Kinematic Machines*, pp. 17–31, Springer, 1999.

[34] A. Rauf, "A new measurement device for complete parameter identification of parallel manipulators with partial pose measurements," in *Proceedings of Parallel Kinematics Seminar (PKS 04), Chemnitz, Germany*, 2004.

[35] L. Guan, J. Wang, Y. Yun, and L. Wang, "Kinematics of a tricept-like parallel robot," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 6, pp. 5312–5316, IEEE, 2004.

[36] V. L. Orekhov, C. S. Knabe, M. A. Hopkins, and D. W. Hong, "An unlumped model for linear series elastic actuators with ball screw drives," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2224–2230, Sept 2015.

[37] S. Murugesan, "An overview of electric motors for space applications," *IEEE Transactions on Industrial Electronics and Control Instrumentation*, vol. IECI-28, pp. 260–265, Nov 1981.

[38] K. Hameyer and R. J. M. Belmans, "Permanent magnet excited brushed dc motors," *IEEE Transactions on Industrial Electronics*, vol. 43, pp. 247–255, Apr 1996.

[39] J.-C. Kim, J.-M. Kim, C.-U. Kim, and C. Choi, "Ultra precise position estimation of servomotor using analog quadrature encoders," in *Twenty-First Annual IEEE Applied Power Electronics Conference and Exposition, 2006. APEC '06.*, pp. 5 pp.–, March 2006.

[40] F. Briz, J. A. Cancelas, and A. Diez, "Speed measurement using rotary encoders for high performance ac drives," in *Industrial Electronics, Control and Instrumentation, 1994. IECON '94., 20th International Conference on*, vol. 1, pp. 538–542 vol.1, Sep 1994.

[41] S. L. Henkel, *Optical Encoders: A Review.* pp. 9-12, Sensors, September 1987.

[42] M. S. Wang, Y.-S. Kung, Y.-M. Tu, and T.-T. Lin, "Novel interpolation method for quadrature encoder square signals," in *2009 IEEE International Symposium on Industrial Electronics*, pp. 333–338, July 2009.

[43] J. Borenstein, H. R. Everett, and L. Feng, *Where am I ? Sensors and Methods for Mobile Robot Positioning.* University of Michigan, April 1996.

[44] A. Agent, "The advantages of absolute encoders for motion control," *Sensors*, pp. 19–24, 1991.

[45] G. Avolio, "Principles of rotary optical encoders," *SENSORS-PETERBOROUGH-*, vol. 10, pp. 10–10, 1993.

[46] P. Nickson, *Solid-State Tachometry.* pp. 23-26, Sensors, April 1985.

[47] Advantech, "Advantech.com." http://www.advantech.com.

[48] Advantech, *PCL-833 Datasheet.*

[49] I. Crnkovic, J. Stafford, and C. Szyperski, "Software components beyond programming: From routines to services," *IEEE Software*, vol. 28, pp. 22–26, May 2011.

[50] C. Szyperski, D. Gruntz, and S. Murer, *Component Software: Beyond Object-Oriented Programming.* ACM Press and Addison-Wesley, 2nd edition ed., 2002.

[51] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice.* Wiley Publishing, 2009.

[52] A. Shakhimardanov, N. Hochgeschwender, and G. K. Kraetzschmar, "Component models in robotics software," in *Proceedings of the 10th Performance Metrics for Intelligent Systems Workshop*, PerMIS '10, (New York, NY, USA), pp. 82–87, ACM, 2010.

[53] D. Brugali and P. Scandurra, "Component-based robotic engineering (part i) [tutorial]," *IEEE Robotics Automation Magazine*, vol. 16, pp. 84–96, December 2009.

[54] G. Wang and C. K. Fung, "Architecture paradigms and their influences and impacts on component-based software systems," in *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9 - Volume 9*, HICSS '04, (Washington, DC, USA), pp. 90272.1–, IEEE Computer Society, 2004.

[55] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback, "Towards component-based robotics, principles and practices of software development in robotics (sdir2005)," in *ICRA2005 Workshop, Barcelona, Spain*, 2005.

[56] D. Brugali and A. Shakhimardanov, "Component-based robotic engineering (part ii)," *IEEE Robotics Automation Magazine*, vol. 17, pp. 100–112, March 2010.

[57] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The brics component model: A model-based development paradigm for complex robotics software systems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, (New York, NY, USA), pp. 1758–1764, ACM, 2013.

[58] M. Radestock and S. Eisenbach, *Coordination in evolving systems*, pp. 162–176. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996.

[59] A. Shakhimardanov and E. Prassler, "Comparative evaluation of robotic software integration systems: A case study," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 29 - November 2, 2007, Sheraton Hotel and Marina, San Diego, California, USA*, pp. 3031–3037, 2007.

[60] Orocos, "The Orocos project." http://www.orocos.org.

[61] ROS, "Robot Operating System." http://www.ros.org.

[62] Peter Soetens, *The Orocos Component Builder's Manual*. Open RObot COntrol Software , 2012.

[63] R. P. Paul, *Robot manipulators : mathematics, programming, and control : the computer control of robot manipulators*. MIT Press series in artificial intelligence, MIT Press, 1981.

[64] L. A. Fernandez-Yánez and L. F. Sotomayor-Reinoso, "Ánalisis cinemático inverso y directo del robot paralelo," Master's thesis, Escuela Politécnica Nacional, Quito, Ecuador, Noviembre 2016.

[65] T. Machhi, N. Rastogi, P. Dutta, and P. Paliwal, "Orocos-easy way to develop open source robot control application," 2016.

[66] H. Bruyninckx, "Orocos: design and implementation of a robot control software framework," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2002.

[67] D. Ioris, W. F. Lages, and D. C. Santini, "Integrating the orocos framework and the barrett wam robot," in *Proceedings of the 5th Workshop on Applied Robotics and Automation. Sociedade Brasileira de Automática*, 2012.

[68] A. Babinec, L. Juriica, P. Hubinský, and F. Ducho, "Visual localization of mobile robot using artificial markers," vol. 96, 12 2014.

[69] M. Fiala, "Vision guided control of multiple robots," in *First Canadian Conference on Computer and Robot Vision, 2004. Proceedings.*, pp. 241–246, May 2004.

[70] S. Garrido-Jurado, R. M. noz Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using mixed integer linear programming," *Pattern Recognition*, vol. 51, pp. 481 – 491, 2016.

[71] J. Basmajian and C. De Luca, *Muscles Alive: Their Functions Revealed by Electromyography.* Williams & Wilkins, 1985.

[72] P. Konrad, "The ABC of EMG," 2005.

[73] M. Mulas, M. Folgheraiter, and G. Gini, "An emg-controlled exoskeleton for hand rehabilitation," in *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.*, pp. 371–374, June 2005.

[74] S. K. Au, P. Bonato, and H. Herr, "An emg-position controlled system for an active ankle-foot prosthesis: an initial experimental study," in *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.*, pp. 375–379, June 2005.

[75] Cambridge Electronic Design Limited, *1401 family programming manual*, September 2012.

[76] Cambridge Electronic Design Limited, *The CED 1902 Owners Handbook*, May 2006.

[77] N. Wrigth, "Taking adaptive measures. Adding sense of vision and touch to robots offers huge production protential for job shops and manufacturers alike.," *FFJournal*, March 2014.

[78] T. Brogårdh, "Present and future robot control developmentan industrial perspective," *Annual Reviews in Control*, vol. 31, no. 1, pp. 69 – 79, 2007.

[79] R. Boggs, "Tiny sensors assist. Medical and Robotics research.," *Design News*, March 1995.

[80] ATI Industrial automation, *F/T Transducter. Six-Axis Force/Torque Sensor system.* ATI Industrial automation, Pinnacle Parc, 1031 Goodworth Drive, March 2016.

[81] ATI Industrial automation, *Net F/T.Network Force/Torque Sensor system.* ATI Industrial automation, Pinnacle Parc, 1031 Goodworth Drive, 2007.

[82] R. Ortega and M. W. Spong, "Adaptive motion control of rigid robots: a tutorial," in *Proceedings of the 27th IEEE Conference on Decision and Control*, pp. 1575–1584 vol.2, Dec 1988.

[83] M. Díaz-Rodríguez, A. Valera, V. Mata, and M. Valles, "Model-based control of a 3-dof parallel robot based on identified relevant parameters," *IEEE/ASME Transactions on Mechatronics*, vol. 18, pp. 1737–1744, Dec 2013.

[84] J. I. Cazalilla Morenas, "Implementación basada en el Middleware Orocos de controladores dinámicos para un robot paralelo," , Universitat Politecnica de Valencia, , 2012. .

[85] B. PADEN and R. PANJA, "Globally asymptotically stable pd+ controller for robot manipulators," *International Journal of Control*, vol. 47, no. 6, pp. 1697–1712, 1988.

[86] C. C. de Wit, G. Bastin, and B. Siciliano, eds., *Theory of Robot Control*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1st ed., 1996.

[87] M. W. Spong, *Robot Dynamics and Control*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 1989.

[88] A. Valera, F. Benimeli, J. Solaz, H. D. Rosario, A. Robertsson, K. Nilsson, R. Zotovic, and M. Mellado, "A car-seat example of automated anthropomorphic testing of fabrics using force-controlled robot motions," *IEEE Transactions on Automation Science and Engineering*, vol. 8, pp. 280–291, April 2011.

[89] R. Volpe and P. Khosla, "A theoretical and experimental investigation of explicit force control strategies for manipulators," *IEEE Transactions on Automatic Control*, vol. 38, pp. 1634–1650, Nov 1993.

[90] H. Berghuis, P. Lohnberg, and H. Nijmeijer, "Tracking control of robots using only position measurements," in *[1991] Proceedings of the 30th IEEE Conference on Decision and Control*, pp. 1039–1040 vol.1, Dec 1991.

[91] H. Kwakernaak, *Linear Optimal Control Systems*. New York, NY, USA: John Wiley & Sons, Inc., 1972.

[92] M. de Hacienda, "Real Decreto 1098/2001, de 12 de octubre, por el que se aprueba el Reglamento general de la Ley de Contratos de las Administraciones Públicas.." BOE núm. 257, Octubre 2001. .

Parte II

Presupuesto

Capítulo 8

Presupuesto

En el presente capítulo se presenta el presupuesto de la primera fase de desarrollo del robot *paralelo4DoF*. Este presupuesto se ha elaborado siguiendo el Real Decreto 1098/2001, de 12 de octubre, por el que se aprueba el Reglamento general de la Ley de Contratos de las Administraciones Públicas [92]. Concretamente los artículos:

- Artículo 130. Cálculo de los precios de las distintas unidades de obra
- Artículo 131. Presupuesto de ejecución material y presupuesto base de licitación.

Se ha asumido que tanto los ordenadores como las licencias de software se pueden amortizar en 10 proyectos. Se han considerado unos gastos generales del 13 %, un beneficio industrial del 6 %, un IVA del 21 % y unos honorarios del 3 %.

Índice del presupuesto

1. **Diseño teórico**
   - 1.1. Propuesta de diseño de la versión inicial
   - 1.2. Modelado del robot en software CAD/CAM Solid Works®
   - 1.3. Modelado del robot en software de simulación V-REP®
   - 1.4. Desarrollo del modelo dinámico
   - 1.5. Simulación del modelo dinámico
   - 1.6. Implementación de la estrategia de control en Simulink®

2. **Montaje del prototipo**

    2.1. Montaje de la estructura

    2.2. Instalación del hardware

    2.3. Instalación del software

3. **Desarrollo del software de control**

    3.1. Comunicación con los sensores y actuadores

    3.2. Desarrollo de los componentes de control

    3.3. Implementación del control

4. **Ensayo y extracción de datos**

    4.1. Primer ensayo de prueba

    4.2. Ensayo con estrategia de control simple

    4.3. Ensayo con estrategia de control avanzada

5. **Presentación de los resultados**

    5.1. Compilación de los resultados

    5.2. Análisis de los resultados

    5.3. Redacción guía del robot paralelo3DoF

    5.4. Redacción guía del robot paralelo4DoF

    5.5. Redacción y presentación de un artículo científico

Cuadro de precios unitarios

Código	Unidades	Descripción	Precio
1		**Diseño teórico**	
1.1	ud.	Propuesta de diseño de la versión inicial	2,025.00€
1.2	ud.	Modelado del robot en software CAD/CAM Solid Works®	4,085.00€
1.3	ud.	Modelado del robot en software de simulación V-REP®	850.00€
1.4	ud.	Desarrollo del modelo dinámico	4,025.00€
1.5	ud.	Simulación del modelo dinámico	1,375.00€
1.6	ud.	Implementación de la estrategia de control en Simulink®	900.00€
		Total Diseño teórico	13,260.00€
2		**Montaje del prototipo**	
2.1	ud.	Montaje de la estructura	23,730.00€
2.2	ud.	Instalación del hardware	4,475.00€
2.3	ud.	Instalación del software	1,650.00€
		Total Montaje del prototipo	29,855.00€
3		**Desarrollo del software de control**	
3.1	ud.	Comunicación con los sensores y actuadores	2,475.00€
3.2	ud.	Desarrollo de los componentes de control	5,115.00€
3.3	ud.	Implementación del control	1,650.00€
		Total Desarrollo del software de control	9,240.00€
4		**Ensayo y extracción de datos**	
4.1	ud.	Primer ensayo de prueba	1,700.00€
4.2	ud.	Ensayo con estrategia de control simple	1,700.00€
4.3	ud.	Ensayo con estrategia de control avanzada	3,400.00€
		Total Ensayo y extracción de datos	6,800.00€
5		**Presentación de los resultados**	
5.1	ud.	Compilación de los resultados	2,400.00€
5.2	ud.	Análisis de los resultados	5,400.00€
5.3	ud.	Redacción guía del robot paralelo3DoF	1,600.00€
5.4	ud.	Redacción guía del robot paralelo4DoF	800.00€
5.5	ud.	Redacción y presentación de un artículo científico	4,700.00€
		Total Presentación de los resultados	14,900.00€
		Total	74,055.00€

Cuadro de precios descompuestos

Código	Unidades	Descripción	Rendimiento	Precio unitario	Importe
1		**Diseño teórico**			
1.1		*Propuesta de diseño de la versión inicial*			
	h	Director del proyecto	40	50€/h	2,000.00€
	ud	Ordenador con prestaciones estándar	0.025	1000€	25€
				Total	2,025.00€
1.2		*Modelado del robot en software CAD/CAM Solid Works®*			
	h	Director del proyecto	10	50€/h	500€
	h	Ingeniero de automática y control	80	35€/h	2800€
	ud	Ordenador para diseño gráfico	0.05	1500€	75€
	ud	Licencia de software Solid Works®	0.1	7100€	710€
				Total	4,085.00€
1.3		*Modelado del robot en software de simulación V-REP®*			
	h	Director del proyecto	2.5	50€/h	125€
	h	Ingeniero de automática y control	20	35€/h	700€
	ud	Ordenador con prestaciones estándar	0.025	1000€	25€
	ud	Licencia de software V-REP	1	0€	0€
				Total	850.00€

Código	Unidades	Descripción	Rendimiento	Precio unitario	Importe
1.4		*Desarrollo del modelo dinámico del robot paralelo de 4 grados de libertad*			
	h	Director del proyecto	80	50€/h	4000€
	ud	Ordenador con prestaciones estándar	0.025	1000€	25€
				Total	4,025.00€
1.5		*Simulación del modelo dinámico del robot con el software de simulación de dinámica de cuerpos Adams®*			
	h	Director del proyecto	2.5	50€/h	125€
	h	Ingeniero de automática y control	20	35€/h	700€
	ud	Ordenador para diseño gráfico	0.1	1500€/h	150€
	ud	Licencia para software Adams®	0.1	4000€/h	400€
				Total	1,375.00€
1.6		*Implementación de la estrategia de control en el software de simulación basada en modelos Simulink®*			
	h	Director del proyecto	2.5	50€/h	125€
	h	Ingeniero de automática y control	20	35€/h	700€
	ud	Ordenador con prestacione estándar	0.025	1000€	25€
	ud	Licencia educativa de software Matlab®y Simulink®	0.1	500€	50€
				Total	900.00€
Total Diseño teórico					13,260.00€

Código	Unidades	Descripción	Rendimiento	Precio unitario	Importe
2		**Montaje del prototipo**			
2.1		*Montaje del prototipo*			
	h	Técnico de laboratorio	40	30€/h	1200€
	ud	Fuente de alimentación variable de 0V a 30V	1	230€	230€
	ud	Fuente de alimentación variable de 24V y 15A	1	150€	150€
	ud	Robot paralelo con 4 grados de libertad	1	14230€	14230€
	ud	Motor con encoder y freno incorporado	4		
	ud	Actuador lineal	4		
	ud	Articulación universal	4		
	ud	Articulación rotacional	3		
	ud	Articulación esférica	1		
	ud	Plataforma fija	1		
	ud	Plataforma móvil	1		
	ud	Sensor de fuerza con receiver NETZEIL	1	7040€	7040€
	ud	Etapa de control para motor DC	4	220€	880€
				Total	23,730.00€

Código	Unidades	Descripción	Rendimiento	Precio unitario	Importe
2.2		*Instalación del hardware*			
	h	Técnico de laboratorio	8	30€/h	240€
	ud	Ordenador de desarrollo	1	1360€	1360€
	ud	PC Industrial de control	1	1850€	1850€
	ud	Tarjeta encoder	1	365€	365€
	ud	Tarjeta convertidora A/D	1	375€	375€
	ud	Webcam	1	120€	120€
	ud	Cables y regletas de conexión	1	165€	165€
				Total	4,475.00€
2.3		*Instalación del software*			
	h	Director del proyecto	5	50€/h	250€
	h	Ingeniero de automática y control	40	35€/h	1400€
	ud	Sistema operativo Ubuntu	1	0€	0€
	ud	Driver tarjeta encoder	1	0€	0€
	ud	Driver tarjeta A/D	1	0€	0€
	ud	Software ROS	1	0€	0€
	ud	Software Orocos	1	0€	0€
				Total	1,650.00€
Total Montaje del prototipo					29,855.00€

Código	Unidades	Descripción	Rendimiento	Precio unitario	Importe
3		**Desarrollo del software de control**			
3.1		Comunicación con los sensores y actuadores			
	ud	Lectura del encoder - Componente medio			
	h	Director del proyecto	2.5	50€/h	125€
	h	Ingeniero de automática y control	20	35€/h	700€
	ud	Lectura del sensor de fuerza - Componente medio			
	h	Director del proyecto	2.5	50€/h	125€
	h	Ingeniero de automática y control	20	35€/h	700€
	ud	Estructura de los actuadores lineales - Componente medio			
	h	Director del proyecto	2.5	50€/h	125€
	h	Ingeniero de automática y control	20	35€/h	700€
				Total	2,475.00€
3.2		Desarrollo de los componentes de control			
	ud	PID - Componente simple			
	h	Director del proyecto	1	50€/h	50€
	h	Ingeniero de automática y control	8	35€/h	280€
	ud	Saturación - Componente simple			
	h	Director del proyecto	1	50€/h	50€
	h	Ingeniero de automática y control	8	35€/h	280€
	ud	Suma - Componente simple			
	h	Director del proyecto	1	50€/h	50€
	h	Ingeniero de automática y control	8	35€/h	280€

Código	Unidades	Descripción	Rendimiento	Precio unitario	Importe
	ud	Obervador de estados - Componente medio			
	h	Director del proyecto	2.5	50€/h	125€
	h	Ingeniero de automática y control	20	35€/h	700€
	ud	Dinámica inversa - Componente azanzado			
	h	Director del proyecto	5	50€/h	250€
	h	Ingeniero de automática y control	40	35€/h	1400€
	ud	Términos dinámicos - Componente azanzado			
	h	Director del proyecto	5	50€/h	250€
	h	Ingeniero de automática y control	40	35€/h	1400€
				Total	5,115.00€
3.3		Implementación del control			
	h	Director del proyecto	5	50€/h	250€
	h	Ingeniero de automática y control	40	35€/h	1400€
				Total	1,650.00€
Total Desarrollo del software de control					9,240.00€

Código	Unidades	Descripción	Rendimiento	Precio unitario	Importe
4		**Ensayo y extracción de datos**			
4.1		Primer ensayo de prueba			
	h	Director del proyecto	20	50€/h	1000€
	h	Ingeniero de automática y control	20	35€/h	700€
				Total	1,700.00€
4.2		Ensayo con estrategia de control simple			
	h	Director del proyecto	20	50€/h	1000€
	h	Ingeniero de automática y control	20	35€/h	700€
				Total	1,700.00€
4.3		Ensayo con estrategia de control avanzado			
	h	Director del proyecto	40	50€/h	2000€
	h	Ingeniero de automática y control	40	35€/h	1400€
				Total	3,400.00€
Total Ensayos y extracción de datos					6,800.00€
5		**Presentación de los resultados**			
5.1		Compilación de los resultados			
	h	Director del proyecto	20	50€/h	1000€
	h	Ingeniero de automática y control	40	35€/h	1400€
				Total	2,400.00€
5.2		Análisis de los resultados			
	h	Director del proyecto	80	50€/h	4000€
	h	Ingeniero de automática y control	40	35€/h	1400€
				Total	5,400.00€
5.3		Redacción guía del robot paralelo3DoF			
	h	Director del proyecto	4	50€/h	200€
	h	Ingeniero de automática y control	40	35€/h	1400€
				Total	1,600.00€

Código	Unidades	Descripción	Rendimiento	Precio unitario	Importe
5.4		Redacción guía del robot paralelo4DoF			
	h	Director del proyecto	2	50€/h	100€
	h	Ingeniero de automática y control	20	35€/h	700€
				Total	800.00€
5.5		Redacción y presentación de un artículo científico			
	h	Director del proyecto	80	50€/h	4000€
	h	Ingeniero de automática y control	20	35€/h	700€
				Total	4,700.00€
Total Presentación de los resultados					4,9000.00€

Presupuesto total

Código	descripción	Precio
1	Diseño teórico	13,260.00€
2	Montaje del prototipo	29,855.00€
3	Desarrollo del software de control	9,240.00€
4	Ensayos y extracción de datos	6,800.00€
5	Presentación de los resultados	14,900.00€
	Presupuesto de ejecución material (PEM)	74,055.00€
	Gastos generales (13 %)	9,627.15€
	Beneficio industrial (6 %)	577,63€
	Presupuesto de contrata (PC)	84,259.78€
	IVA (21 %)	17,694.55€
	Honorarios (3 %)	2,527.79€
	Presupuesto total (PT)	104,483.13€

Asciende el presente Presupuesto Total a la cantidad de ciento cuatro mil cuatrocientos ochenta y dos con trece céntimos (104,482.13€).

Parte III

Anexos

Apéndice A

Guía de usuario para el robot paralelo de 3DOF

Guía de usuario para el robot paralelo de 3DOF

Introducción

Este documento pretende ser una guía para aquella persona que pretenda introducirse en el entramado de componentes, script y demás archivos del robot paralelo de tres grados de libertad. Esta guía, por desgracia, no evita la necesidad de leer el código fuente para saber cómo funciona realmente el software que hace que el robot se mueva de una u otra manera. Pero este documento es una buena base con la que empezar a buscar información y conocer el funcionamiento general del sistema como que componentes intervienen en que esquema de control, que información envía y recibe cada componente, si un componente se puede sustituir por otro o en cambio o en cambio necesita otro que haga de puente, etc. Esto hará que en caso de tener que buscar información en el código fuente, sea más fácil saber dónde hay que buscar.

Por una parte se explican los componentes, qué es lo que hacen y que entradas y salidas tienen. Los componentes son los programas base de Orocos, estos se cargan en el deployer y se conectan entre sí para controlar el robot. Hay otra parte dedicada a los scripts, que son la manera rápida de cargar los componentes en el deployer y conectarlos entre sí. Estos archivos son los que, de alguna manera, configuran el esquema de control. Por último se explican los diferentes archivos de texto con los que se introduce información al sistema o los que se generan para guardar información útil.

Contenido

Scripts

DinInvers_derivativo:

Este script presenta un esquema de control de dinámica inversa y la velocidad de las articulaciones se calculan por derivadas discretas. Únicamente implementa un control de posición, no de fuerza. Al usar el componente "ModuloGenRefMod" se carga el ejercicio del archivo "miReferencia.txt" y empieza a ejecutarlo.

Archivos generados:

ActuadoresDinInvers.txt: Valor en voltios que recibe cada actuador. Componente <sumInerGvC>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: A Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

velDinInvDeseada.txt: Almacena la derivada discreta de las posiciones de referencia de q1, q2 y q3. Atributo del componente <derivada>
| dq1_ref | dq2_ref | dq3_ref |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

sumaInerGvC: Suma los parámetros inerciales, gravitacionales y de Coriolis.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

DinInvers_Observador

Este script presenta un esquema de control de dinámica inversa y la velocidad de las articulaciones se calculan con un observador de estado. En este caso no se tiene en cuenta la derivada de la posición de referencia. Únicamente implementa un control de posición, no de fuerza. Al usar el componente "ModuloGenRefMod" se carga el ejercicio del archivo "miReferencia.txt" y empieza a ejecutarlo.

Archivos generados:

ActuadoresDinInvers.txt: Valor en voltios que recibe cada actuador. Componente <sumInerGvC>
| Vmotor1 | Vmotor2 | Vmotor3 |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

velObsX.txt: Almacena la velocidad observada de la articulación X. Atributo del componente <moduloObservadorVel>
| dqX_Obs |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloAuxObservador: Calcula el PD necesario para el obervador. Toma que la velocidad deseada es igual a cero.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa.

moduloObservadorVel: Estima la velocidad a partir del Observador de Luenberger.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

sumaInerGvC: Suma los parámetros inerciales, gravitacionales y de Coriolis.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

DinInvers_Observador2

Este script presenta un esquema de control de dinámica inversa y la velocidad de las articulaciones se calculan con un observador de estado. Tiene en cuenta la derivada de la posición de referencia para hacer el cálculo de la velocidad observada. Únicamente implementa un control de posición, no de fuerza. Al usar el componente "ModuloGenRefMod" se carga el ejercicio del archivo "miReferencia.txt" y empieza a ejecutarlo.

Archivos generados:

ActuadoresDinInvers.txt: Valor en voltios que recibe cada actuador. Componente <sumInerGvC>
| Vmotor1 | Vmotor2 | Vmotor3 |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

velObsX.txt: Almacena la velocidad observada de la articulación X. Atributo del componente <moduloObservadorVel>
| dqX_Obs |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloAuxObservador2: Calcula el PD necesario para el obervador. Toma que la velocidad deseada a partir de la derivada discreta de la posición de referencia.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa.

moduloObservadorVel: Estima la velocidad a partir del Observador de Luenberger.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

sumaInerGvC: Suma los parámetros inerciales, gravitacionales y de Coriolis.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Fuerza_DinInv_refPosFuerza:

Con este script se controla la posición y la fuerza en gamma, beta y zeta. Las referencias de posición las toma el componente "moduloGenRefMod" del archivo "miReferencia.txt" y las referencias de la fuerza en gamma, beta y zeta almacenadas en el archivo "miFuerzaGBZ.txt" las carga el componente "moduloGenerFuerza". El control de la posición se hace por dinámica inversa y para el control de fuerza se calcula el Jacobiano.

Permite activar o desactivar el control de fuerza de gamma, beta y/o zeta mediante atributos de los componentes, así como ajustar los parámetros de los PID.

Archivos generados:

ActuadoresDinInvers.txt: Valor en voltios que recibe cada actuador. Componente <sumInerGvC>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

Fuerzas.txt: Almacena las fuerzas leídas del sensor, las referencias de gamma, beta y zeta y el resultado del control PID para gamma, beta y zeta. Componente <moduloControlFuerzaGBZ>
| Fx | Fy | Fz | Tx | Ty | Tz | g_ref | b_ref | z_ref | PID_g| PID_b | PID_z |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de las derivadas de gamma, beta y zeta usadas en el jacobiano. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

salJacobiano.txt: Almacena los valores de la integral del jacobiano. Componente <moduloJacobiano>
| intJ_q1 | intJ_q2 | intJ_q3 |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

velqrefest: Almacena los valores de la velocidad de referencia estimada a partir del jacobiano de q1, q2 y q3. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloControlFuerzaGBZ: Calcula el control PID a partir de las fuerzas y las referencias de fuerza.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloGenerFuerzaGBZ: Carga y envía las referencias de fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt”.

moduloJacobiano: Calcula del jacobiano y su derivada entre la cinemática y las fuerzas.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

modulSumaRefes: Suma la referencia de posición y la calculada a partir del Jacobiano.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

sumaInerGvC: Suma los parámetros inerciales, gravitacionales y de Coriolis.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Fuerza_DinInv_refPosFuerza_aruco:

Con este script se controla la posición y la fuerza en gamma, beta y zeta. Las referencias de posición las toma el componente “moduloGenRefMod” del archivo “miReferencia.txt” y las referencias de la fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt” las carga el componente “moduloGenerFuerza”. El control de la posición se hace por dinámica inversa y para el control de fuerza se calcula el Jacobiano.

Permite activar o desactivar el control de fuerza de gamma, beta y/o zeta mediante atributos de los componentes, así como ajustar los parámetros de los PID.

La diferencia de este script respecto a “Fuerza_DinInv_refPosFuerza” es que este añade un componente que lee la posición de un marcador de Aruco. Para poder usar este script hace falta lanzar el nodo de ROS que reconoce el marcador y envía la posición y orientación del marcador por un topic. Esto se desde el icono de escritorio “MarcadorAruco” desde el ordenador “parallel4dof”.

Archivos generados:

ActuadoresDinInvers.txt: Valor en voltios que recibe cada actuador. Componente <sumInerGvC>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

aruco_pose.txt: Almacena la posición y rotación de un marcador de Aruco detectado a través de una cámara. Atributo del componente <aruco_read_single>
| X | Y | Z | Roll | Pitch | Yaw |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

Fuerzas.txt: Almacena las fuerzas leídas del sensor, las referencias de gamma, beta y zeta y el resultado del control PID para gamma, beta y zeta. Componente <moduloControlFuerzaGBZ>
| Fx | Fy | Fz | Tx | Ty | Tz | g_ref | b_ref | z_ref | PID_g| PID_b | PID_z |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de las derivadas de gamma, beta y zeta usadas en el jacobiano. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

salJacobiano.txt: Almacena los valores de la integral del jacobiano. Componente <moduloJacobiano>
| intJ_q1 | intJ_q2 | intJ_q3 |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

velqrefest: Almacena los valores de la velocidad de referencia estimada a partir del jacobiano de q1, q2 y q3. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

Componentes:

actuador: Controla la tensión que le llega al motor.

Aruco_read_single: Recibe y almacena la posición y orientación de un marcador de Aruco.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloControlFuerzaGBZ: Calcula el control PID a partir de las fuerzas y las referencias de fuerza.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloGenerFuerzaGBZ: Carga y envía las referencias de fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt”.

moduloJacobiano: Calcula del jacobiano y su derivada entre la cinemática y las fuerzas.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

modulSumaRefes: Suma la referencia de posición y la calculada a partir del Jacobiano.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

sumaInerGvC: Suma los parámetros inerciales, gravitacionales y de Coriolis.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Fuerza_modular_refPosFuerza:

Con este script se controla la posición y la fuerza en gamma, beta y zeta. Las referencias de posición las toma el componente “moduloGenRefMod” del archivo “miReferencia.txt” y las referencias de la fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt” las carga el componente “moduloGenerFuerzaGBZ”. En vez de dinámica inversa usa un control Paden, el control de posición se hace sumando el resultado del PD a los parámetros gravitacionales, inerciales y de Coriolis. Para el control de fuerza se calcula el Jacobiano.

Permite activar o desactivar el control de fuerza de gamma, beta y/o zeta mediante atributos de los componentes, así como ajustar los parámetros de los PID.

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

Fuerzas.txt: Almacena las fuerzas leídas del sensor, las referencias de gamma, beta y zeta y el resultado del control PID para gamma, beta y zeta. Componente <moduloControlFuerzaGBZ>
| Fx | Fy | Fz | Tx | Ty | Tz | g_ref | b_ref | z_ref | PID_g| PID_b | PID_z |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de las derivadas de gamma, beta y zeta usadas en el jacobiano. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

salJacobiano.txt: Almacena los valores de la integral del jacobiano. Componente <moduloJacobiano>
| intJ_q1 | intJ_q2 | intJ_q3 |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

velqrefest: Almacena los valores de la velocidad de referencia estimada a partir del jacobiano de q1, q2 y q3. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloControlFuerzaGBZ: Calcula el control PID a partir de las fuerzas y las referencias de fuerza.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloGenerFuerzaGBZ: Carga y envía las referencias de fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt”.

moduloJacobiano: Calcula del jacobiano y su derivada entre la cinemática y las fuerzas.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

modulSumaRefes: Suma la referencia de posición y la calculada a partir del Jacobiano.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Jacob_modular_vector_fuerza_ErrorObservador:

Este script está basado en "Jacob_modular_vector_fuerza_v2". Un control de posición Paden y el uso del jacobiano para controlar la fuerza. Este script está pensado para comparar la velocidad estimada mediante el uso de derivadas discretas con la velocidad estimada mediante un observador de estados. En esta primera prueba se utiliza el mismo esquema de control que en "Jacob_modular_vector_fuerza_v2", un control Paden y las velocidades calculadas a partir de las derivadas discretas. La entrada "q" del observador recibe la posición real y la entrada "a" recibe el resultado del PD. Se almacenan la derivada discreta de la posición y la velocidad observada.

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

errorObsX.txt: Para la articulación X, almacena la derivada discreta de la posición, la velocidad estimada con el observador y el error entre ambos. Atributo del componente <moduloLeeError>
| dqX_deriv | dqX_obs | (dqX_deriv - dqX_obs) |

Fuerzas.txt: Almacena las fuerzas leídas del sensor, las referencias de gamma, beta y zeta y el resultado del control PID para gamma, beta y zeta. Componente <moduloControlFuerzaGBZ>
| Fx | Fy | Fz | Tx | Ty | Tz | g_ref | b_ref | z_ref | PID_g| PID_b | PID_z |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de las derivadas de gamma, beta y zeta usadas en el jacobiano. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

salJacobiano.txt: Almacena los valores de la integral del jacobiano. Componente <moduloJacobiano>
| intJ_q1 | intJ_q2 | intJ_q3 |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

velObsX.txt: Almacena la velocidad observada de la articulación X. Atributo del componente <moduloObservadorVel>
| dqX_Obs |

velqrefest: Almacena los valores de la velocidad de referencia estimada a partir del jacobiano de q1, q2 y q3. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloAuxErrorObs: Hace de puente entre componentes que no tienen puertos compatibles para "Jacob_modular_vector_fuerza_ErrorObservador"

moduloControlFuerzaGBZ: Calcula el control PID a partir de las fuerzas y las referencias de fuerza.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa a partir del archivo "miReferencia.txt".

moduloGenerFuerzaGBZ: Carga y envía las referencias de fuerza en gamma, beta y zeta almacenadas en el archivo "miFuerzaGBZ.txt".

moduloJacobiano: Calcula del jacobiano y su derivada entre la cinemática y las fuerzas.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

moduloLeeError: Almacena dos variables y la diferencia entre estas.

moduloObservadorVel: Estima la velocidad a partir del Observador de Luenberger.

modulSumaRefes: Suma la referencia de posición y la calculada a partir del Jacobiano.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Jacob_modular_vector_fuerza_ErrorObservador2:

En este script se utiliza un esquema de dinámica inversa y se utiliza por completo el observador. El observador tiene como entradas la posición real y el resultado del PD haciendo uso de la velocidad observada. El cálculo de las velocidades de las variables pasivas se hace usando la velocidad observada y los parámetros inerciales tienen como entrada el PD calculado con la velocidad observada. Se almacenan la derivada discreta de la posición y la velocidad observada.

Archivos generados:

ActuadoresDinInvers.txt: Valor en voltios que recibe cada actuador. Componente <sumInerGvC>
| Vmotor1 | Vmotor2 | Vmotor3 |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

errorObsX.txt: Para la articulación X, almacena la derivada discreta de la posición, la velocidad estimada con el observador y el error entre ambos. Atributo del componente <moduloLeeError>
| dqX_deriv | dqX_obs | (dqX_deriv - dqX_obs) |

Fuerzas.txt: Almacena las fuerzas leídas del sensor, las referencias de gamma, beta y zeta y el resultado del control PID para gamma, beta y zeta. Componente <moduloControlFuerzaGBZ>
| Fx | Fy | Fz | Tx | Ty | Tz | g_ref | b_ref | z_ref | PID_g| PID_b | PID_z |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de las derivadas de gamma, beta y zeta usadas en el jacobiano. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

salJacobiano.txt: Almacena los valores de la integral del jacobiano. Componente <moduloJacobiano>
| intJ_q1 | intJ_q2 | intJ_q3 |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

velObsX.txt: Almacena la velocidad observada de la articulación X. Atributo del componente <moduloObservadorVel>
| dqX_Obs |

velqrefest: Almacena los valores de la velocidad de referencia estimada a partir del jacobiano de q1, q2 y q3. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloAuxErrorObs2: Hace de puente entre componentes que no tienen puertos compatibles para “Jacob_modular_vector_fuerza_ErrorObservador2”

moduloAuxObservador: Calcula el PD necesario para el obervador. Toma que la velocidad deseada es igual a cero.

moduloControlFuerzaGBZ: Calcula el control PID a partir de las fuerzas y las referencias de fuerza.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloGenerFuerzaGBZ: Carga y envía las referencias de fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt”.

moduloJacobiano: Calcula del jacobiano y su derivada entre la cinemática y las fuerzas.

moduloLeeError: Almacena dos variables y la diferencia entre estas.

moduloObservadorVel: Estima la velocidad a partir del Observador de Luenberger.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

moduloSumaRefes: Suma la referencia de posición y la calculada a partir del Jacobiano.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

sumaInerGvC: Suma los parámetros inerciales, gravitacionales y de Coriolis.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Jacob_modular_vector_fuerza_ErrorObservador3:

En este script utiliza el esquema Paden. La entrada a de los parámetros inerciales es la doble derivada de la posición de referencia como ocurre en un esquema sin observador. En cambio, la entrada dq del bloque que calcula las velocidades es la velocidad observada. El observador tiene como entradas la posición real y la doble derivada de la posición deseada. Se almacenan la derivada discreta de la posición y la velocidad observada.

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

errorObsX.txt: Para la articulación X, almacena la derivada discreta de la posición, la velocidad estimada con el observador y el error entre ambos. Atributo del componente <moduloLeeError>
| dqX_deriv | dqX_obs | (dqX_deriv - dqX_obs) |

Fuerzas.txt: Almacena las fuerzas leídas del sensor, las referencias de gamma, beta y zeta y el resultado del control PID para gamma, beta y zeta. Componente <moduloControlFuerzaGBZ>
| Fx | Fy | Fz | Tx | Ty | Tz | g_ref | b_ref | z_ref | PID_g| PID_b | PID_z |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de las derivadas de gamma, beta y zeta usadas en el jacobiano. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

salJacobiano.txt: Almacena los valores de la integral del jacobiano. Componente <moduloJacobiano>
| intJ_q1 | intJ_q2 | intJ_q3 |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

velObsX.txt: Almacena la velocidad observada de la articulación X. Atributo del componente <moduloObservadorVel>
| dqX_Obs |

velqrefest: Almacena los valores de la velocidad de referencia estimada a partir del jacobiano de q1, q2 y q3. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloAuxErrorObs3: Hace de puente entre componentes que no tienen puertos compatibles para “Jacob_modular_vector_fuerza_ErrorObservador3”

moduloControlFuerzaGBZ: Calcula el control PID a partir de las fuerzas y las referencias de fuerza.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenerFuerzaGBZ: Carga y envía las referencias de fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt”.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloJacobiano: Calcula del jacobiano y su derivada entre la cinemática y las fuerzas.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

moduloLeeError: Almacena dos variables y la diferencia entre estas.

moduloObservadorVel: Estima la velocidad a partir del Observador de Luenberger.

modulSumaRefes: Suma la referencia de posición y la calculada a partir del Jacobiano.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Jacob_modular_vector_fuerza_ErrorObservador4:

En este script se utiliza un esquema Paden y se utiliza por completo el observador. El observador tiene como entradas la posición real y el resultado del PD haciendo uso de la velocidad observada. El cálculo de las velocidades de las variables pasivas se hace usando la velocidad observada y los parámetros inerciales tienen como entrada la doble derivada de la posición deseada. La acción de control es la resultante de sumar los términos inerciales, gravitacionales, de Coriolis y el PD haciendo uso de la velocidad observada. Se almacenan la derivada discreta de la posición y la velocidad observada.

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

errorObsX.txt: Para la articulación X, almacena la derivada discreta de la posición, la velocidad estimada con el observador y el error entre ambos. Atributo del componente <moduloLeeError>
| dqX_deriv | dqX_obs | (dqX_deriv - dqX_obs) |

Fuerzas.txt: Almacena las fuerzas leídas del sensor, las referencias de gamma, beta y zeta y el resultado del control PID para gamma, beta y zeta. Componente <moduloControlFuerzaGBZ>
| Fx | Fy | Fz | Tx | Ty | Tz | g_ref | b_ref | z_ref | PID_g| PID_b | PID_z |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de las derivadas de gamma, beta y zeta usadas en el jacobiano. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

salJacobiano.txt: Almacena los valores de la integral del jacobiano. Componente <moduloJacobiano>
| intJ_q1 | intJ_q2 | intJ_q3 |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

velObsX.txt: Almacena la velocidad observada de la articulación X. Atributo del componente <moduloObservadorVel>
| dqX_Obs |

velqrefest: Almacena los valores de la velocidad de referencia estimada a partir del jacobiano de q1, q2 y q3. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloAuxErrorObs: Hace de puente entre componentes que no tienen puertos compatibles para “Jacob_modular_vector_fuerza_ErrorObservador”

moduloAuxObservador: Calcula el PD necesario para el obervador. Toma que la velocidad deseada es igual a cero.

moduloControlFuerzaGBZ: Calcula el control PID a partir de las fuerzas y las referencias de fuerza.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenerFuerzaGBZ: Carga y envía las referencias de fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt”.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloJacobiano: Calcula del jacobiano y su derivada entre la cinemática y las fuerzas.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

moduloLeeError: Almacena dos variables y la diferencia entre estas.

moduloObservadorVel: Estima la velocidad a partir del Observador de Luenberger.

modulSumaRefes: Suma la referencia de posición y la calculada a partir del Jacobiano.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Jacob_modular_vector_fuerza_observador:

Se trata de un control híbrido de fuerza y posición. El control de posición se hace por dinámica inversa y haciendo uso de un observador de estado. El control de fuerza usa el jacobiano.

Archivos generados:

ActuadoresDinInvers.txt: Valor en voltios que recibe cada actuador. Componente <sumInerGvC>
| Vmotor1 | Vmotor2 | Vmotor3 |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

Fuerzas.txt: Almacena las fuerzas leídas del sensor, las referencias de gamma, beta y zeta y el resultado del control PID para gamma, beta y zeta. Componente <moduloControlFuerzaGBZ>
| Fx | Fy | Fz | Tx | Ty | Tz | g_ref | b_ref | z_ref | PID_g| PID_b | PID_z |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de las derivadas de gamma, beta y zeta usadas en el jacobiano. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

RefeActual.txt: Almacena la referencia de posición. Primero en formato gamma, beta y zeta y luego en q1, q2 y q3. Componente <moduloGenRefMod>
| gamma_ref | beta_ref | zeta_ref | q1_ref | q2_ref | q3_ref |

salJacobiano.txt: Almacena los valores de la integral del jacobiano. Componente <moduloJacobiano>
| intJ_q1 | intJ_q2 | intJ_q3 |

velObsX.txt: Almacena la velocidad observada de la articulación X. Atributo del componente <moduloObservadorVel>
| dqX_Obs |

velqrefest: Almacena los valores de la velocidad de referencia estimada a partir del jacobiano de q1, q2 y q3. Componente <moduloJacobiano>
| J_q1 | J_q2 | J_q3 |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloAuxObservador: Calcula el PD necesario para el obervador. Toma que la velocidad deseada es igual a cero.

moduloControlFuerzaGBZ: Calcula el control PID a partir de las fuerzas y las referencias de fuerza.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenerFuerzaGBZ: Carga y envía las referencias de fuerza en gamma, beta y zeta almacenadas en el archivo “miFuerzaGBZ.txt”.

moduloGenRefMod: Calcula la posición de referencia por cinemática inversa.

moduloJacobiano: Calcula del jacobiano y su derivada entre la cinemática y las fuerzas.

moduloObservadorVel: Estima la velocidad a partir del Observador de Luenberger.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

sumaInerGvC: Suma los parámetros inerciales, gravitacionales y de Coriolis.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Paden_eHealthROS:

Este script implementa un control de posición Paden. A diferencia del resto de los scripts, este esquema no lee las posiciones de referencia a partir de un archivo, lo hace a través de un topic de ROS. El robot se eleva hasta una posición segura. En este estado está listo para recibir la nueva posición de referencia a través de un topic de ROS. Cada vez que se recibe una nueva posición, se calcula una trayectoria con forma de spline. Esto puede continuar hasta que se envía la señal de parada o de pausa.

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

GBZ.txt: Almacena los valores de gamma, beta y zeta de referencia. Componente <moduloGenRefRT>
| gamma_ref | beta_ref | zeta_ref |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

prefe.txt: Almacena la posición de referencia de las articulaciones activas. Componente <moduloGenRefRT>
| q1_refe | q2_refe | q3_refe |

puntos.txt: Almacena la gamma, beta y zeta en el instante en el que se le indica. Componente <moduloGenRefRT>
| gamma_ref | beta_ref | zeta_ref | instante |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRefRT: Calcula la posición de referencia por cinemática inversa a partir de los datos recibidos por un topic de ROS.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Paden_modular_vector:

Con este script se controla la posición. Las referencias de posición las toma el componente "moduloGenRef" del archivo "miReferencia.txt". En vez de dinámica inversa usa un control Paden, el control de posición se hace sumando el resultado del PD a los parámetros gravitacionales, inerciales y de Coriolis.

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

Fuerzas.txt: Almacena las fuerzas leídas por el sensor. Componente <moduloGenRef>
| Fx | Fy | Fz | Tx | Ty | Tz | zeta_ref | PID_zeta |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de gamma, beta y zeta de referencia. Componente <moduloGenRef>
| gamma_ref | beta_ref | zeta_ref |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

prefe.txt: Almacena la posición de referencia de las articulaciones activas. Componente <moduloGenRef>
| q1_refe | q2_refe | q3_refe |

puntos.txt: Almacena la gamma, beta y zeta en el instante en el que se le indica. Componente <moduloGenRef>
| gamma_ref | beta_ref | zeta_ref | instante |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloGenRef: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Paden_modular_vector_fuerza:

Con este script se controla la posición y la fuerza en zeta. Las referencias de posición las toma el componente “moduloGenRef” del archivo “miReferencia.txt” y las referencias de la fuerza están fijas en 0. En vez de dinámica inversa usa un control Paden, el control de posición se hace sumando el resultado del PD a los parámetros gravitacionales, inerciales y de Coriolis. Para el control de fuerza se multiplica la lectura del sensor de fuerza y se suma o resta a la posición de referencia. Por lo tanto se trata de un control proporcional. Este control de fuerza se puede desactivar definiendo el atributo “Rfuerza” de “moduloGenRef” como 0.

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

Fuerzas.txt: Almacena las fuerzas leídas por el sensor. Componente <moduloGenRef>
| Fx | Fy | Fz | Tx | Ty | Tz | zeta_ref | PID_zeta |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de gamma, beta y zeta de referencia. Componente <moduloGenRef>
| gamma_ref | beta_ref | zeta_ref |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

prefe.txt: Almacena la posición de referencia de las articulaciones activas. Componente <moduloGenRef>
| q1_refe | q2_refe | q3_refe |

puntos.txt: Almacena la gamma, beta y zeta en el instante en el que se le indica. Componente <moduloGenRef>
| gamma_ref | beta_ref | zeta_ref | instante |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRef: Calcula la posición de referencia por cinemática inversa a partir del archivo "miReferencia.txt".

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Paden_modular_vector_fuerza_alarma:

Con este script se controla la posición y la fuerza en gamma, beta y zeta. Las referencias de posición las toma el componente "moduloGenRef" del archivo "miReferencia.txt" antes de detectarse la alarma y después las genera el componente "moduloGenRefAlarms". Las

referencias de la fuerza están fijas en 0. En vez de dinámica inversa usa un control Paden, el control de posición se hace sumando el resultado del PD a los parámetros gravitacionales, inerciales y de Coriolis. Para el control de fuerza se multiplica la lectura del sensor de fuerza y se suma o resta a la posición de referencia. Por lo tanto se trata de un control proporcional. Este control de fuerza se puede desactivar definiendo el atributo "Rfuerza" de "moduloGenRef" como 0.

La particularidad de este control es que lleva incorporado un sistema de alarma que hace que el ejercicio finalice. Cuando salta la alarma el robot baja hasta la posición inicial y acaba con el spline de bajada. La alarma puede saltar bien porque el usuario ha pulsado un botón físico, se ha enviado la señal de alarma desde un topic de ROS o porque se ha superado la fuerza máxima.

Está preparado para usarse con una referencia de gamma en forma de rampa ascendente y mantener beta igual a cero y zeta a una altura constante. Un ejemplo es el archivo "refeGammaRampa.txt".

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

Fuerzas.txt: Almacena las fuerzas leídas por el sensor. Componente <supervisor>
| Fx | Fy | Fz | Tx | Ty | Tz |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

GBZ.txt: Almacena los valores de gamma, beta y zeta de referencia. Componente <moduloGenRef> + <moduloGenRefAlarm>
| gamma_ref | beta_ref | zeta_ref |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

prefe.txt: Almacena la posición de referencia de las articulaciones activas.
Componente <moduloGenRef> + <moduloGenRefAlarm>
| q1_refe | q2_refe | q3_refe |

puntos.txt: Almacena la gamma, beta y zeta en el instante en el que se le indica.
Componente <moduloGenRef> + <moduloGenRefAlarm>
| gamma_ref | beta_ref | zeta_ref | instante |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRef: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloGenRefAlarm: Calcula la posición de referencia una vez se ha pulsado la alarma y el robot tiene que volver a la posición de reposo.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

supervisor: Hace de switch, se activa con un botón físico, mediante un topic de ROS o cuando detecta que se ha superado la fuerza.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Paden_modular_vector_fuerza_copia_movimiento:

Con este script se controla la posición y la fuerza en gamma, beta y zeta. Las referencias de posición las toma el componente "moduloGenRef" del archivo "miReferencia.txt" antes de detectarse el cambio de estado y después las genera el componente "copia_movimiento". Las referencias de la fuerza están fijas en 0. En vez de dinámica inversa usa un control Paden, el control de posición se hace sumando el resultado del PD a los parámetros gravitacionales, inerciales y de Coriolis. Para el control de fuerza se multiplica la lectura del sensor de fuerza y se suma o resta a la posición de referencia. Por lo tanto se trata de un control proporcional. Este control de fuerza se puede desactivar definiendo el atributo "Rfuerza" de "moduloGenRef" como 0.

La particularidad de este control es que lleva incorporado un sistema que almacena la última posición antes del cambio de estado. Esto le permite moverse hasta una altura indicada en el atributo "Zmed" del componente "copia_movimiento", volver a la posición donde se había cambiado de estado, volver a "Zmed" y finalmente acabar en la posición de reposo.

El cambio de posición y orientación se realiza manualmente, guiando el robot gracias al control de fuerza. El archivo de referencias debe ser un archivo con gamma y beta igual a cero y un zeta a una altura constante. El cambio de estado se puede dar bien porque el usuario ha pulsado un botón físico, se ha enviado la señal de alarma desde un topic de ROS o porque se ha superado la fuerza máxima. El nivel de fuerza máxima se ajusta con el atributo "Rfuerza" del componente "control_estados".

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

Fuerzas.txt: Almacena las fuerzas leídas por el sensor. Componente <supervisor>
| Fx | Fy | Fz | Tx | Ty | Tz |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

fuerzasContrEst1.txt: Almacena las fuerzas leídas por el sensor. Componente <control_estados1>
| Fx | Fy | Fz | Tx | Ty | Tz |

GBZ.txt: Almacena los valores de gamma, beta y zeta de referencia. Componente <moduloGenRef> + <copia_movimiento>
| gamma_ref | beta_ref | zeta_ref |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

prefe.txt: Almacena la posición de referencia de las articulaciones activas. Componente <moduloGenRef>
| q1_refe | q2_refe | q3_refe |

prefe2.txt: Continúa almacenando la posición de referencia de las articulaciones activas. Componente <copia_movimiento>

| q1_refe | q2_refe | q3_refe |

puntos.txt: Almacena la gamma, beta y zeta en el instante en el que se le indica.
Componente <moduloGenRef> + <copia_movimiento>
| gamma_ref | beta_ref | zeta_ref | instante |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

Control_estados1: Hace de switch, se activa con un botón físico, mediante un topic de ROS o cuando detecta que se ha superado la fuerza.

copia_movimiento: Almacena la posición en el instante en el que se ha cambiado de estado y calcula las posiciones de referencia para la segunda parte del ejercicio.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRef: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Paden_modular_vector_fuerza_copia_movimiento2:

Con este script se controla la posición y la fuerza en gamma, beta y zeta. Las referencias de posición las toma el componente "moduloGenRef" del archivo "miReferencia.txt" antes de detectarse el cambio de estado y después las genera el componente "copia_movimiento2". Las referencias de la fuerza están fijas en 0. En vez de dinámica inversa usa un control Paden, el control de posición se hace sumando el resultado del PD a los parámetros gravitacionales, inerciales y de Coriolis. Para el control de fuerza se multiplica la lectura del sensor de fuerza y se suma o resta a la posición de referencia. Por lo tanto se trata de un control proporcional. Este control de fuerza se puede desactivar definiendo el atributo "Rfuerza" de "moduloGenRef" como 0.

La particularidad de este control es que lleva incorporado un sistema que almacena el recorrido que ha estado haciendo el robot para después repetirlo. Esto le permite, una vez se indica que cambie de estado, volver hasta la posición inicial, repetir la trayectoria que se ha ido realizando, volver de nuevo a la primera posición y finalmente acabar en la posición de reposo.

El cambio de posición y orientación se realiza manualmente, guiando el robot gracias al control de fuerza. El archivo de referencias debe ser un archivo con gamma y beta igual a cero y un zeta a una altura constante. El cambio de estado se puede dar bien porque el usuario ha pulsado un botón físico, se ha enviado la señal de alarma desde un topic de ROS o porque se ha superado la fuerza máxima. El nivel de fuerza máxima se ajusta con el atributo "Rfuerza" del componente "control_estados2".

Archivos generados:

ActuadoresPaden.txt: Valor en voltios que recibe cada actuador. Componente <sum2>
| Vmotor1 | Vmotor2 | Vmotor3 |

ArticulacionXPaden.txt: Se trata de tres archivos, uno para cada articulación. Almacena el valor de la referencia de posición y el de la posición real de la articulación respectiva. Componente <pd2>
| qX_ref | qX_real |

cinematica.txt: Almacena la cinemática de las 9 variables (QS1) calculadas a partir de las articulaciones activas. Componente <cine3dof>
| q1 | q2 | q3 | q4 | q5 | q6 | q7 |q8 | q9 |

ejercicio.txt: Almacena las posiciones del ejercicio que se ha ido realizando, así se pueden usar más adelante. Componente <control_estados2>
| q1_refe | q2_refe | q3_refe |

Fuerzas.txt: Almacena las fuerzas leídas por el sensor. Componente <supervisor>
| Fx | Fy | Fz | Tx | Ty | Tz |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

fuerzasContrEst2.txt: Almacena las fuerzas leídas por el sensor. Componente <control_estados2>
| Fx | Fy | Fz | Tx | Ty | Tz |

GBZ.txt: Almacena los valores de gamma, beta y zeta de referencia. Componente <moduloGenRef>
| gamma_ref | beta_ref | zeta_ref |
(Cuidado! El archivo puede ser diferente dependiendo del componente que lo cree.)

preal.txt: Almacena la posición real de las articulaciones activas. Componente <cine3dof>
| q1_real | q2_real | q3_real |

prefe.txt: Almacena la posición de referencia de las articulaciones activas. Componente <moduloGenRef>
| q1_refe | q2_refe | q3_refe |

prefe2.txt: Continúa almacenando la posición de referencia de las articulaciones activas. Componente <copia_movimiento2>

| q1_refe | q2_refe | q3_refe |

puntos.txt: Almacena la gamma, beta y zeta en el instante en el que se le indica. Componente <moduloGenRef>

| gamma_ref | beta_ref | zeta_ref | instante |

velDinInvDerivReal.txt: Almacena la derivada discreta de las posiciones reales de q1, q2 y q3. Atributo del componente <derivada>
| dq1_real | dq2_real | dq3_real |

Componentes:

actuador: Controla la tensión que le llega al motor.

cine3dof: Calcula la cinemática del robot con 3dof.

control_estados2: Hace de switch, se activa con un botón físico, mediante un topic de ROS o cuando detecta que se ha superado la fuerza. También almacena el recorrido del robot antes del cambio de estado.

copia_movimiento2: Gracias al archivo almacenado por "control_estados2", calcula y envía las posiciones de referencia para la segunda parte del ejercicio.

derivada: Calcula la derivada discreta.

fcn: Calcula las matrices Ae i Ai utilizadas para calcular los parámetros gravitacionales, inerciales, de Coriolis y las velocidades.

moduloFuerza: Sensor de fuerza, lee la fuerza del sensor.

moduloGenRef: Calcula la posición de referencia por cinemática inversa a partir del archivo “miReferencia.txt”.

moduloPcl833: Encoder, lee la posición real de las articulaciones.

para_c: Calcula los parámetros de Coriolis.

para_iner: Calcula los parámetros inerciales.

paraGv: Calcula los términos gravitacionales.

pd2: Calcula un control PD.

suma2: Suma los parámetros inerciales, gravitacionales y de Coriolis al resultado del PD.

velo3dof: Calcula las velocidades de las variables pasivas a partir de las variables activas.

Componentes

Actuador

Este componente controla la tarjeta PCI1720 encargada de enviar la señal a uno de los motores. Este componente es el único que usa la función “addOperation()”. Una operación es una función que puede ser ejecutada desde el deployer de orocos.

Entradas:

act: Recibe el valor de la acción de control (en unidades de tensión) que se envía al motor. Tipo <double>

final: Recibe la indicación de que se ha finalizado el ejercicio. Tipo <double>

final2: Otro canal por el que se recibe la indicación de que ha finalizado el ejercicio. Tipo <int>

Salidas:

ticksFin: Instante en ticks de reloj en el que se finaliza el ejercicio. Tipo <long long>

Operaciones:

setChannel: Función que tiene como parámetro el canal del motor que va a controlar. Este guarda el valor en una variable <int>. (Sugerencia: Para guardar un valor en una variable es aconsejable usar atributos)

Aruco_read_single

Lee la posición y orientación de un marcador de Aruco a través de un topic de ROS lanzado desde el ordenador de 4DOF.

La orden de leer la recibe desde el componente “ModuloPcl833”. Cuando este termina un ciclo envía un int “1”. El componente *aruco_read_single* lo lee a través del puerto “*input_ready*”, en ese momento guarda la posición del marcador y la orientación en ángulo de Euler (a partir de cuaterniones).

Entradas:

input_pose: Recibe la posición del marcador de Aruco. Tipo <geometry_msgs::PoseStamped>

input_ready: Recibe la información de que el módulo principal está listo, por lo tanto indica que empieza un ciclo nuevo. Tipo <int>

Atributos:

fileName: Dirección y nombre del archivo donde se guardan la posición y orientación. Tipo <std::string>

Control_estados1

Este componente se comporta como un switch. Tiene como entradas: *entrada_control, entrada_fuerza, entrada_refe y entrada_alarm*. La *entrada_control* se utiliza para recibir el estado del "*ModuloPcl833*" y saber cuándo está listo para otro ciclo. La *entrada_refe* es la encargada de recibir la referencia generada por el componente correspondiente, igual que la *entrada_fuerza* recibe la lectura del sensor de fuerza. Por último, la *entrada_alarm* está pensada para recibir la orden de activar el estado del switch a través de un topic de ROS.

Por defecto la entrada_control y la entrada_fuerza están conectadas con la *salida1_control* y la *salida1_fuerza* respectivamente. Hay tres posibilidades de activar el switch. La primera es la comentada anteriormente a través de un topic de ROS. La segunda es mediante un interruptor físico y la tarjeta PCL812. En el momento en el que el componente hace el start se almacena el estado del interruptor y el switch cambia de estado en el momento que se detecta que el interruptor ha sido pulsado. Por último, se puede fijar la fuerza máxima. Si el sensor detecta que se ha sobrepasado dicho máximo, el switch cambia de estado.

El cambio de estado del switch desconecta las *salida1_control* y *salida1_fuerza* y conecta la *entrada_control* con la *salida2_control*, la *entrada_refe* con la *salida2_refe* y la *entrada_fuerza* con la *salida2_fuerza*. De esta manera se puede cambiar el comportamiento del robot al conectar las salida1 a un componente y las salida2 a otro.

Entradas:

entrada_control: Recibe la información de que el módulo principal está listo, por lo tanto indica que empieza un ciclo nuevo. Tipo <int>

entrada_fuerza: Recibe la información leída por el sensor de fuerza. Tipo <vector <double>>

entrada_refe: Recibe la referencia que debe seguir el robot. Tipo <vector <double>>

entrada_alarm: Canal por el que recibe desde un topic de ROS la orden de cambiar de estado. Tipo <std_msgs::Bool>

Salidas:

salida1_control: Envía la información de que el componente "ModuloPcl833" está listo mientras el switch se encuentre en el estado 1. Tipo <int>

salida1_fuerza: Envía los valores de fuerza recibidos al componente conectado mientras el switch se encuentre en el estado 1. Tipo <vector <double>>

salida2_control: Envía la información de que el componente "ModuloPcl833" está listo cuando el switch se encuentre en el estado 2. Tipo <int>

salida2_refe: Envía la posición de referencia del robot cuando el switch se encuentre en el estado 2. Tipo <vector <double>>

salida2_fuerza: Envía los valores de fuerza recibidos al componente conectado cuando el switch se encuentre en el estado 2. Tipo <vector <double>>

Atributos:

Rfuerza: Límite de fuerza al que el componente cambiará de estado. Tipo <vector <double>>

Control_estados2

Se trata de una actualización del componente control_estados1. Añade un archivo donde se va guardando el recorrido de las posiciones del robot para que otro componente pueda usarlo.

Entradas:

entrada_control: Recibe la información de que el módulo principal está listo, por lo tanto indica que empieza un ciclo nuevo. Tipo <int>

entrada_fuerza: Recibe la información leída por el sensor de fuerza. Tipo <vector <double>>

entrada_refe: Recibe la referencia que debe seguir el robot. Tipo <vector <double>>

entrada_alarm: Canal por el que recibe desde un topic de ROS la orden de cambiar de estado. Tipo <std_msgs::Bool>

Salidas:

salida1_control: Envía la información de que el componente “ModuloPcl833” está listo mientras el switch se encuentre en el estado 1. Tipo <int>

salida1_fuerza: Envía los valores de fuerza recibidos al componente conectado mientras el switch se encuentre en el estado 1. Tipo <vector <double>>

salida2_control: Envía la información de que el componente “ModuloPcl833” está listo cuando el switch se encuentre en el estado 2. Tipo <int>

salida2_refe: Envía la posición de referencia del robot cuando el switch se encuentre en el estado 2. Tipo <vector <double>>

salida2_fuerza: Envía los valores de fuerza recibidos al componente conectado cuando el switch se encuentre en el estado 2. Tipo <vector <double>>

Atributos:

Rfuerza: Límite de fuerza al que el componente cambiará de estado. Tipo <vector <double>>

Copia_movimiento

Se trata de una versión modificada del componente “moduloGenRef” pensado para ser utilizado con “control_estados1”. Cuando se ejecuta espera recibir por “entrada_refe” la última referencia enviada al robot. A partir de ahí se generan referencias para que el robot se sitúe en una posición con Gamma y Beta igual a cero y Zeta igual a “Zmed”, un atributo definido en el script. Una vez alcanzada esta posición volverá a situarse en la posición inicial, de vuelta a la posición donde las articulaciones están en “Zmed” y acaba bajando del todo con un spline.

Entradas:

entrada_control: Sirve para recibir desde “moduloPcl833” la señal para un nuevo ciclo. Tipo <int>

entrada_fuerza: Recibe los valores de fuerza del sensor. Tipo <vector <double>> (SIN FUNCIONALIDAD)

entrada_duracion: Recibe la duración para guardarla en un archivo con los valores de GBZ. Tipo <std_msgs::Float64>

entrada_ROS: Entrada para recibir información desde ROS. Tipo <geometry_msgs::Float64> (SIN FUNCIONALIDAD)

entrada_refe: Sirve para recibir la posición de referencia que se utilizará para repetir el movimiento. Tipo <vector <double>>

Salidas:

salida_Refe: Envía los valores de referencia de las articulaciones (q1,q2,q3). Tipo <vector <double>>

salida_finPcl: Le indica al componente principal (moduloPcl833) que se ha terminado el proceso y que debe cerrarse. Tipo <int>

salida_ticks: Envía el instante en nanosegundos en el que se ha ejecutado el updatehook(). Tipo <long long>

Atributos:

RRos: Activa o desactiva funciones conectadas a ROS. Tipo <bool> (SIN FUNCIONALIDAD)

Zmed: Altura de reposo hasta la que el robot bajará antes de repetir el movimiento. Tipo <double>

Copia_movimiento2

Basado en “moduloGenRef”.Se trata de una versión mejorar de “copia_movimiento” y está pensado para ser usado con “control_estados2”. Este componente usa el archivo “ejercicio.txt” generado por “control_estados2”. Al igual que “copia_movimiento”, este componente se divide en tres etapas: primero devuelve el robot a la primera posición registrada en el archivo “ejercicio.txt”, luego va enviando las referencias guardadas en el

archivo para que el robot repita la misma trayectoria. Cuando ha terminado la repetición vuelve a la primera posición y acaba con el spline de bajada.

Entradas:

entrada_control: Sirve para recibir desde “moduloPcl833” la señal para un nuevo ciclo. Tipo <int>

entrada_refe: Sirve para recibir la posición de referencia que se utilizará para repetir el movimiento. Tipo <vector <double>>

Salidas:

salida_Refe: Envía los valores de referencia de las articulaciones (q1,q2,q3). Tipo <vector <double>>

salida_finPcl: Le indica al componente principal (moduloPcl833) que se ha terminado el proceso y que debe cerrarse. Tipo <int>

salida_ticks: Envía el instante en nanosegundos en el que se ha ejecutado el updatehook(). Tipo <long long>

Derivada

Calcula la derivada del valor de los valores de entrada. Al estar pensado para calcular la derivada de las tres articulaciones al mismo tiempo, el vector de entrada será de dimensión 3.

Entrada:

entrada_deriv: Puerto de entrada de valores. Tipo <vector <double>>

Salida:

salida_deriv: Puerto por el que saca la derivada. Tipo <vector <double>>

Atributos:

Fich: Sirve para indicarle al componente si queremos o no que se genere un archivo con los valores de las derivadas. Fich=1 para generar el archivo. Tipo <int>

fileName: Dirección y nombre del archivo donde se guardarán los valores de las derivadas. Tipo <string>

ModuloAuxErrorObs

Este grupo de componentes se utilizan para conectar los puertos de diferentes componentes que usan el mismo tipo de datos. Este componente tiene como entradas tres puertos por los que recibe variables de tipo <vector <double>> de tres dimensiones. Cada una de estas variables se separan en tres variables de tipo <double> y se envían por un puerto de salida. De esta manera se conectan los componentes "derivada", "moduloPcl833" y "pd2" que trabajan con las componentes de las tres articulaciones a la vez con los componentes "moduloObservadorVel" y "moduloLeeError" que trabajan con solo una articulación.

Entradas:

entrada_q: Recibe la posición real de las articulaciones. Tipo <vector<double>>

entrada_dq: Recibe la derivada de la posición real de las articulaciones. Tipo <vector<double>>

entrada_a: Recibe la salida de "pd2". Tipo <vector<double>>

Salidas:

salida_q1: Envía la posición real de la articulación 1. Tipo <double>

salida_q2: Envía la posición real de la articulación 2. Tipo <double>

salida_q3: Envía la posición real de la articulación 3. Tipo <double>

salida_a1: Envía el valor de "pd2" de la articulación 1. Tipo <double>

salida_a2: Envía el valor de "pd2" de la articulación 2. Tipo <double>

salida_a3: Envía el valor de "pd2" de la articulación 3. Tipo <double>

salida_dq1: Envía el valor de la derivada de la posición real de la articulación 1. Tipo <double>

salida_dq2: Envía el valor de la derivada de la posición real de la articulación 2. Tipo <double>

salida_dq3: Envía el valor de la derivada de la posición real de la articulación 3. Tipo <double>

ModuloAuxErrorObs2

Hace de conexión entre el puerto de salida de la derivada de la velocidad real, que es de tipo <vector<double>> con las entradas de "moduloLeeError" que son de tipo <double>.

Entradas:

entrada_dq: Recibe la derivada de la posición real de las articulaciones. Tipo <vector<double>>

Salidas:

salida_dq1: Envía el valor de la derivada de la posición real de la articulación 1. Tipo <double>

salida_dq2: Envía el valor de la derivada de la posición real de la articulación 2. Tipo <double>

salida_dq3: Envía el valor de la derivada de la posición real de la articulación 3. Tipo <double>

ModuloAuxErrorObs3

Permite conectar la posición real, la derivada de la posición real y la aceleración de la posición real con el "moduloObservadorVel", y también la conexión entre "moduloObservadorVel" con "Velo3dof".

Entradas:

entrada_q: Recibe la posición real de las articulaciones. Tipo <vector<double>>

entrada_dq: Recibe la derivada de la posición real de las articulaciones. Tipo <vector<double>>

entrada_a: Recibe la salida de "pd2". Tipo <vector<double>>

entrada_dqObs1: Recibe la velocidad calculada por el observador para la articulación 1. Tipo <double>

entrada_dqObs2: Recibe la velocidad calculada por el observador para la articulación 2. Tipo <double>

entrada_dqObs3: Recibe la velocidad calculada por el observador para la articulación 3. Tipo <double>

Salidas:

salida_q1: Envía la posición real de la articulación 1. Tipo <double>

salida_q2: Envía la posición real de la articulación 2. Tipo <double>

salida_q3: Envía la posición real de la articulación 3. Tipo <double>

salida_a1: Envía el valor de "pd2" de la articulación 1. Tipo <double>

salida_a2: Envía el valor de "pd2" de la articulación 2. Tipo <double>

salida_a3: Envía el valor de "pd2" de la articulación 3. Tipo <double>

salida_dq1: Envía el valor de la derivada de la posición real de la articulación 1. Tipo <double>

salida_dq2: Envía el valor de la derivada de la posición real de la articulación 2. Tipo <double>

salida_dq3: Envía el valor de la derivada de la posición real de la articulación 3. Tipo <double>

salida_dqObs: Envía los valores de la velocidad observada de las articulaciones. Tipo <vector<double>>

ModuloAuxObservador

Este componente hace algunos cálculos previos necesarios para el "moduloObservadorVel". Para simplificar, en este caso se toma que la derivada de la posición deseada es igual a cero.

Entradas:

entrada_q: Recibe la posición real de las articulaciones. Tipo <vector<double>>

entrada_dq: Recibe la derivada de la posición real de las articulaciones. Tipo <vector<double>>

entrada_dqObs1: Recibe la velocidad calculada por el observador para la articulación 1. Tipo <double>

entrada_dqObs2: Recibe la velocidad calculada por el observador para la articulación 2. Tipo <double>

entrada_dqObs3: Recibe la velocidad calculada por el observador para la articulación 3. Tipo <double>

Salidas:

salida_dqObs: Envía los valores de la velocidad observada de las articulaciones. Tipo <vector<double>>

salida_q1: Envía la posición real de la articulación 1. Tipo <double>

salida_q2: Envía la posición real de la articulación 2. Tipo <double>

salida_q3: Envía la posición real de la articulación 3. Tipo <double>

salida_a1: Envía el valor de a del bucle observador1. Tipo <double>

salida_a2: Envía el valor de a del bucle observador2. Tipo <double>

salida_a3: Envía el valor de a del bucle observador3. Tipo <double>

salida_a: Envía el valor de a en un vector. Tipo <vector<double>>

Atributos:

Kp: Kp del bucle. Tipo <double>

Kd: Kd del bucle. Tipo <double>

ModuloAuxObservador2

Versión actualizada de "moduloAuxObservador". Esta versión tiene en cuenta tanto la derivada como la doble derivada de la posición de referencia.

Entradas:

entrada_q: Recibe la posición real de las articulaciones. Tipo <vector<double>>

entrada_dq: Recibe la derivada de la posición real de las articulaciones. Tipo <vector<double>>

entrada_dqObs1: Recibe la velocidad calculada por el observador para la articulación 1. Tipo <double>

entrada_dqObs2: Recibe la velocidad calculada por el observador para la articulación 2. Tipo <double>

entrada_dqObs3: Recibe la velocidad calculada por el observador para la articulación 3. Tipo <double>

Salidas:

salida_dqObs: Envía los valores de la velocidad observada de las articulaciones. Tipo <vector<double>>

salida_q1: Envía la posición real de la articulación 1. Tipo <double>

salida_q2: Envía la posición real de la articulación 2. Tipo <double>

salida_q3: Envía la posición real de la articulación 3. Tipo <double>

salida_a1: Envía el valor de a del bucle observador1. Tipo <double>

salida_a2: Envía el valor de a del bucle observador2. Tipo <double>

salida_a3: Envía el valor de a del bucle observador3. Tipo <double>

salida_a: Envía el valor de a en un vector. Tipo <vector<double>>

Atributos:

Kp: Kp del bucle. Tipo <double>

Kd: Kd del bucle. Tipo <double>

ModuloControlFuerza

Calcula el control PID de fuerza, aunque solo para la fuerza vertical (eje z). Los valores que multiplican al término derivativo, integral y proporcional están guardados en variables del programa. Tiene como entradas la fuerza leída y la fuerza de referencia, devuelve el valor del control. También tiene una entrada para activar o desactivar el control.

Entradas:

entrada_fuerza: Recibe la información leída por el sensor de fuerza. Tipo <vector <double>>

entrada_refFuerza: Recibe el valor de referencia de la fuerza en el eje z para cada instante. Tipo <double>

entrada_cf: Recibe el estado del control de fuerza. Si recibe 0 se desactiva el control de fuerza, en cambio con 1 se activa. Tipo <int>

Salidas:

salida_controlFuerza: Devuelve el valor del control de fuerza para zeta. Tipo <double>

Atributos:

cf: Indica si se debe hacer o no el control. Tipo <int>

ModuloControlFuerzaGBZ

Se trata de una versión actualizada del componente "moduloControlFuerza". A esta versión se le ha añadido el control para las fuerzas en gamma y beta, además de definir el valor de Kp, Kd y Ki desde el script para no tener que recompilar.

Entradas:

entrada_fuerza: Recibe la información leída por el sensor de fuerza. Tipo <vector <double>>

entrada_refFuerza: Recibe el valor de referencia de la fuerza para cada instante. Tipo vector<<double>>

entrada_cf: Recibe el estado del control de fuerza. Si recibe 0 se desactiva el control de fuerza, en cambio con 1 se activa. Tipo <int>

Salidas:

salida_controlFuerza: Devuelve el valor del control de fuerza para gamma, beta y zeta. Tipo <vector <double>>

Atributos:

cf: Indica si se debe hacer o no el control. Tipo <int>

Kp: Valor que multiplica al error de fuerza respecto a la referencia. Tipo <double>

Ki: Valor que multiplica la integral del error de fuerza. Tipo <double>

Kd: Valor que multiplica la derivada del error de fuerza. Tipo <double>

ModuloFuerza

Este componente es el encargado de leer el valor de la fuerza registrada por la célula de carga mediante sockets.

Entradas:

entrada_control: Sirve para recibir desde "moduloPcl833" la señal para un nuevo ciclo. Tipo <int>

Salidas:

salida_fuerza: Envía el valor de la fuerza leída a los componentes que se contecten. Tipo vector<double>>

ModuloGenerFuerza

Envía la referencia de fuerza en zeta. Este componente abre el archivo "miFuerza.txt", guarda los valores de referencia en un vector y los va enviando uno a uno cuando el componente principal le da la orden de un nuevo ciclo. La longitud de las columnas del archivo deben de ser de 10000 valores.

Entradas:

entrada_control: Sirve para recibir desde “moduloPcl833” la señal para un nuevo ciclo. Tipo <int>

Salidas:

salida_RefeFuerza: Envía el valor de la fuerza deseada en ese instante. Tipo <double>

ModuloGenerFuerzaGBZ

Versión actualizada de “moduloGenerFuerza”. Envía la referencia de fuerza en gamma, beta y zeta. Este componente abre el archivo “miFuerzaGBZ.txt”, guarda los valores de referencia en un vector y los va enviando uno a uno cuando el componente principal le da la orden de un nuevo ciclo.

Entradas:

entrada_control: Sirve para recibir desde “moduloPcl833” la señal para un nuevo ciclo. Tipo <int>

Salidas:

salida_RefeFuerza: Envía el valor de la fuerza deseada en ese instante. Tipo vector<<double>>

ModuloGenRef

La principal función de este componente es enviar la referencia de posición en cada ciclo. Dependiendo de los valores de los atributos “Rfuerza” y “RRos” tomará los valores de referencia del archivo “miReferencia.txt”, teniendo en cuenta o no la fuerza, o a través de un topic de ROS. Es importante destacar que la longitud del vector de datos viene impuesta por la constante “MAXVEC”, por defecto igual a 6800.

Una vez recogidos los valores de referencia para el ejercicio, se calculan los valores de un spline cúbico para que el robot se mueva desde la posición cero hasta la primera posición del ejercicio. De esta manera se asegura que, independientemente de la primera posición del ejercicio, el movimiento hasta esta primera posición va a ser suave.

Cuando se alcanza la primera posición, después del spline, empieza a enviar las referencias. Las referencias que se envían son las correspondientes a la altura de las articulaciones (q1, q2 y q3). Estas se calculan en cada ciclo por dinámica inversa a partir de la posición de referencia en formato GBZ que se ha leído del archivo o se ha recibido mediante ROS.

Si se ha activado “Rfuerza”, la referencia de posición se verá afectada sumando o restando un factor proporcional a la fuerza leída.

Una vez terminado el ejercicio se genera un spline de referencias desde la última posición hasta la posición final donde todas las articulaciones están bajadas. También se le envía a "moduloPcl833" la señal de que debe apagarse.

Entradas:

entrada_control: Sirve para recibir desde "moduloPcl833" la señal para un nuevo ciclo. Tipo <int>

entrada_fuerza: Recibe la información leída por el sensor de fuerza. Tipo <vector <double>>

entrada_duracion: Recibe la duración para guardarla en un archivo con los valores de GBZ. Tipo <std_msgs::Float64>

entrada_ROS: Recibe los valores de las referencias a través de un topic de ROS. Tipo <geometry_msgs::Vector3>

Salidas:

salida_Refe: Envía la referencia de posición de las articulaciones q1, q2 y q3. Tipo <vector<double>>

salida_finPcl: Le indica al componente "moduloPcl833" que se ha terminado la ejecución. Tipo <int>

salida_ticks: Envía el instante en nanosegundos en el que se ha ejecutado el updatehook(). Tipo <long long>

salida_ready: Envía la señal de que ha terminado el spline de subida y se dispone a realizar el ejercicio. Tipo <bool>

Atributos:

Rfuerza: Indica si se debe tener en cuenta o no la lectura de fuerzas. Tipo <bool>

RRos: Indica de dónde se van tomar las referencias. RRos = true indica que se espera que los datos se reciban desde un topic de ROS, en caso contrario se lee del archivo. Tipo <bool>

ctex: Constante por el que se multiplica el valor del momento en gamma antes de sumarlo a la referencia de posición. Tipo <double>

ctex: Constante por el que se multiplica el valor del momento en beta antes de sumarlo a la referencia de posición. Tipo <double>

ModuloGenRefAlarm

Este componente es una modificación de "moduloGenRef". Está pensado para utilizarlo con el componente "supervisor" y una referencia que hace que la articulación 3 se vaya elevando poco a poco. El ejercicio empieza utilizando el componente "moduloGenRef" hasta que el "supervisor" cambia al "moduloGenRefAlarm" y este pone primero la plataforma en horizontal y acaba con el spline de bajada.

Entradas:

entrada_control: Sirve para recibir desde "moduloPcl833" la señal para un nuevo ciclo. Tipo <int>

entrada_duracion: Recibe la duración para guardarla en un archivo con los valores de GBZ. Tipo <std_msgs::Float64>

entrada_ROS: Recibe los valores de las referencias a través de un topic de ROS. Tipo <geometry_msgs::Vector3> (SIN FUNCIONALIDAD)

Salidas:

salida_Refe: Envía la referencia de posición de las articulaciones q1, q2 y q3. Tipo <vector<double>>

salida_finPcl: Le indica al componente "moduloPcl833" que se ha terminado la ejecución. Tipo <int>

salida_ticks: Envía el instante en nanosegundos en el que se ha ejecutado el updatehook(). Tipo <long long>

ModuloGenRefMod

Versión de “moduloGenRef”. Esta versión suprime los atributos sin funcionalidad y añade un puerto que envía los valores de gamma, beta y zeta de cada referencia. Esta versión trabaja con 8000 valores de referencias en vez de las 6800 de “moduloGenRef”.

Entradas:

entrada_control: Sirve para recibir desde “moduloPcl833” la señal para un nuevo ciclo. Tipo <int>

entrada_fuerza: Recibe la información leída por el sensor de fuerza. Tipo <vector <double>>

Salidas:

salida_Refe: Envía la referencia de posición de las articulaciones q1, q2 y q3. Tipo <vector<double>>

salida_finPcl: Le indica al componente “moduloPcl833” que se ha terminado la ejecución. Tipo <int>

salida_cfuerza: Envía la señal de que ha terminado de hacer el spline de subida y se debe activar el control de fuerza. Tipo <int>

salida_GBZ: Envía el valor de la referencia en formato gamma, beta y zeta. Tipo <vector<double>>

Atributos:

Rfuerza: Indica si se debe tener en cuenta o no la lectura de fuerzas. Tipo <bool>

ModuloGenRefRT

Esta versión de “moduloGenRef” está pensada para usarla únicamente con ROS en vez de cargar las referencias desde un archivo. A medida que recibe una nueva referencia por un topic, envía un spline de referencias al robot hasta que llega a la posición recibida. Se para cuando recibe por otro topic la señal de parar.

Entradas:

entrada_control: Sirve para recibir desde “moduloPcl833” la señal para un nuevo ciclo. Tipo <int>

entrada_duracion: Recibe la duración para guardarla en un archivo con los valores de GBZ. Tipo <std_msgs::Float64>

entrada_ROS: Recibe los valores de las referencias a través de un topic de ROS. Tipo <geometry_msgs::Vector3>

entrada_standBy: Recibe la orden de parar. Tipo <std_msgs::Bool>

isPaused: Recibe la orden de pausar el ejercicio. Tipo <std_msgs::Bool>

Salidas:

salida_Refe: Envía la referencia de posición de las articulaciones q1, q2 y q3. Tipo <vector<double>>

salida_finPcl: Le indica al componente “moduloPcl833” que se ha terminado la ejecución. Tipo <int>

salida_ticks: Envía el instante en nanosegundos en el que se ha ejecutado el updatehook(). Tipo <long long>

visualizar_ref: Envía la referencia en cada ciclo a un topic de ROS. Tipo <geometry_msgs::Vector3>

objetivo: Envía la señal de que se ha alcanzado la referencia enviada desde ROS. Tipo <std_msgs::Bool>

status: Envía el estado del programa dependiendo si se está iniciando (wakingUp), si está realizando el ejercicio (movingOn), si está acabando (layingDown) o si ha terminado (standBy). Tipo <std_msgs::String>

fuerza: Envía la lectura de fuerzas a un topic de ROS. Tipo <geometry_msgs::Twist>

ModuloJacobiano

Calcula el Jacobiano entre la cinemática y las fuerzas, lo integra y devuelve el valor resultante. Se puede activar o desactivar cada uno de los controles en las componente gamma, beta y zeta.

Entradas:

entrada_QS1: Lee la cinemática del robot. Tipo <vector<double>>

entrada_ControlFuer: Lee el valor del control de fuerza. Tipo <vector<double>>

entrada_GBZderiv: Lee el valor de las derivadas de gamma, beta y zeta. Tipo <vector<double>>

Salidas:

salida_Jacobiano: Envía el valor de la integral del jacobiano. Tipo <vector<double>>

Atributos:

controlZ: Se utiliza para activar o desactivar el control de la componente zeta. Tipo <bool>

controlGamma: Se utiliza para activar o desactivar el control de la componente gamma. Tipo <bool>

controlBeta: Se utiliza para activar o desactivar el control de la componente beta. Tipo <bool>

ModuloJacobianoNoInt

Versión del "moduloJacobiano". No calcula la integral, por lo que tampoco calcula el control de gamma, beta y/o zeta.

Entradas:

entrada_QS1: Lee la cinemática del robot. Tipo <vector<double>>

entrada_GBZ: Lee el valor de las derivadas de gamma, beta y zeta. Tipo <vector<double>>

Salidas:

salida_Jacobiano: Envía el valor de la integral del jacobiano. Tipo <vector<double>>

ModuloLeeError

Este componente calcula la diferencia entre dos entradas y la almacena en un archivo con tres columnas (variable1 | variable2 | diferencia). Está pensado para calcular el error entre la velocidad calculada usando el componente “derivada” y el “moduloObservadorVel”.

Entradas:

entrada_q1: Recibe el primer valor a comparar. Tipo <double>

entrada_q2: Recibe el segundo valor a comparar. Tipo <double>

Atributos:

fileName: Dirección y nombre del donde se guardarán los valores. Tipo <std::string>

ModuloObservadorVel

Calcula la velocidad observada a partir de la posición real y la entrada “a” calculada por el “moduloAuxObservador” o “moduloAuxObservador2”.

Entradas:

entrada_posReal: Recibe la posición real leída desde el encoder. Tipo <double>

entrada_a: Recibe el valor de “a” desde uno de los módulos auxiliares. Tipo <double>

Salidas:

salida_velObs: Envía la velocidad calculada por el observador. Tipo <double>

Atributos:

fileName: Dirección y nombre del donde se guardarán los valores de la velocidad. Tipo <std::string>

ModuloPcl833

Este componente es el que se ha utilizado como principal. Principalmente se encarga de leer los encoders de los motores que mueven las articulaciones del robot y por tanto conocer la posición real de estas. Los encoders se leen haciendo uso de una tarjeta "PCL833" y la posición en la que se encuentre el robot cuando se ejecute configureHook() es la que se considerará como el cero. Por esto es importante bajar el robot, de forma manual si fuera necesario, antes de empezar cualquier ejercicio.

Este componente también es el encargado de enviar la señal de que está listo y por lo tanto se debe empezar un nuevo ciclo, así como recibir la señal de parar cuando termine el proceso.

Entradas:

entrada_fin: Recibe la orden de parar el programa. Tipo <int>

Salidas:

salida_posReal: Envía la posición de las articulaciones leída por los encoders. Tipo <vector<double>>

salida_control: Envía la señal para empezar el siguiente ciclo. Tipo <int>

posRos: Envía el valor de los encodres a un topic de ROS. Tipo <std_msgs::Int32MultiArray>

joint1: Envía a un topic de ROS el valor de posición de la articulación 1. Tipo <std_msgs::Float64>

joint2: Envía a un topic de ROS el valor de posición de la articulación 2. Tipo <std_msgs::Float64>

joint3: Envía a un topic de ROS el valor de posición de la articulación 3. Tipo <std_msgs::Float64>

ModuloSumaRefes

Suma el valor de la referencia con el valor del Jacobiano y comprueba si satura.

Entradas:

entrada_refe: Recibe el valor de la referencia de posición. Tipo <vector <double>>

entrada_jacob: Recibe el valor del jacobiano. Tipo <vector<double>>

Salidas:

salida_sumaRefe: Envía el valor de la suma o del valor saturado. Tipo <vector<double>>

Pd

Esta familia de componentes calcula un PD a partir del error de posición y la derivada de este. Este primer componente multiplica Kd por la derivada de la posición real y Kp por el error entre la posición real y la deseada.

$$PD = \dot{q} \times K_d + \left(q - q_{ref}\right) * K_p$$

Entradas:

entrada_q: Recibe la posición real de las articulaciones. Tipo <vector<double>>

entrada_qd: Recibe el valor de la posición de referencia (o deseada). Tipo <vector<double>>

entrada_qprima: Recibe el valor de la derivada de la posición real. Tipo <vector<double>>

Salidas:

salida_pd1: Salida del valor del PD de la articulación 1. Tipo <double>

salida_pd2: Salida del valor del PD de la articulación 2. Tipo <double>

salida_pd3: Salida del valor del PD de la articulación 3. Tipo <double>

Pd1

Es igual que “pd” pero envía los tres valores del PD por un mismo puerto.

$$PD = \dot{q} \times K_d + \left(q - q_{ref}\right) * K_p$$

Entradas:

entrada_q: Recibe la posición real de las articulaciones. Tipo <vector<double>>

entrada_qd: Recibe el valor de la posición de referencia (o deseada). Tipo <vector<double>>

entrada_qprima: Recibe el valor de la derivada de la posición real. Tipo <vector<double>>

Salidas:

salida_pd: Salida del valor del PD de las tres articulaciones. Tipo <vector<double>>

Pd2

Este componente multiplica Kd por la diferencia de las derivadas de la posición real y la de referencia y Kp por el error entre la posición real y la deseada. Además se ha declarado Kp y Kd como atributos.

$$PD = (\dot{q} - \dot{q}_{ref}) \times K_d + \left(q - q_{ref}\right) \times K_p$$

Entradas:

entrada_q: Recibe la posición real de las articulaciones. Tipo <vector<double>>

entrada_qd: Recibe el valor de la posición de referencia (o deseada). Tipo <vector<double>>

entrada_qprima: Recibe el valor de la derivada de la posición real. Tipo <vector<double>>

entrada_qdprima: Recibe el valor de la derivada de la posición de referencia. Tipo <vector<double>>

Salidas:

salida_pd: Salida del valor del PD de las tres articulaciones. Tipo <vector<double>>

Atributos:

Kp: Valor de Kp. Tipo <double>

Kd: Valor de Kd. Tipo <double>

Pd3

Este componente se utiliza para calcular el PD del esquema "Adaptativo".

$$PD = \left(q - q_{ref}\right) \times K_p + \left[\left(q - q_{ref}\right) \times K_a + \left(\dot{q} - \dot{q}_{ref}\right)\right] \times K_d +$$

Entradas:

entrada_q: Recibe la posición real de las articulaciones. Tipo <vector<double>>

entrada_qd: Recibe el valor de la posición de referencia (o deseada). Tipo <vector<double>>

entrada_qprima: Recibe el valor de la derivada de la posición real. Tipo <vector<double>>

entrada_qdprima: Recibe el valor de la derivada de la posición de referencia. Tipo <vector<double>>

Salidas:

salida_pd3: Salida del valor del PD de las tres articulaciones. Tipo <vector<double>>

salida_s1: Salida del error entre la posición real y la posición deseada. Tipo <vector<double>>

Sum1

Esta familia de componentes suma todas las entradas y pasa el valor a voltaje para enviarlo a los actuadores. Este primer componente suma los parámetros gravitacionales y los valores del PD.

Entradas:

entrada_Gv: Recibe el valor de los parámetros gravitacionales. Tipo <vector<double>>

entrada_Pd: Recibe los valores del PD. Tipo <vector<double>>

Salidas:

salida_ticks: Envía el valor del instante en nanosegundos en el que se envían los valores a los actuadores. Tipo <long long>

salida_act1: Envía el valor del voltaje al actuador del motor 1. Tipo <double>

salida_act2: Envía el valor del voltaje al actuador del motor 2. Tipo <double>

salida_act3: Envía el valor del voltaje al actuador del motor 3. Tipo <double>

Sum2

Este componente suma los parámetros gravitacionales, los de Coriolis, los inerciales y los valores del PD antes de enviarlo al actuador.

Entradas:

entrada_Gv: Recibe el valor de los parámetros gravitacionales. Tipo <vector<double>>

entrada_Cv. Recibe el valor de los parámetros de Coriolis. Tipo <vector<double>>

entrada_Iner. Recibe el valor de los parámetros inerciales. Tipo <vector<double>>

entrada_Pd: Recibe los valores del PD. Tipo <vector<double>>

Salidas:

salida_ticks: Envía el valor del instante en nanosegundos en el que se envían los valores a los actuadores. Tipo <long long>

salida_act1: Envía el valor del voltaje al actuador del motor 1. Tipo <double>

salida_act2: Envía el valor del voltaje al actuador del motor 2. Tipo <double>

salida_act3: Envía el valor del voltaje al actuador del motor 3. Tipo <double>

SumInerGvC

Este componente suma los parámetros gravitacionales, los de Coriolis y los inerciales antes de enviarlos al actuador.

Entradas:

entrada_Gv: Recibe el valor de los parámetros gravitacionales. Tipo <vector<double>>

entrada_Cv. Recibe el valor de los parámetros de Coriolis. Tipo <vector<double>>

entrada_Iner. Recibe el valor de los parámetros inerciales. Tipo <vector<double>>

Salidas:

salida_ticks: Envía el valor del instante en nanosegundos en el que se envían los valores a los actuadores. Tipo <long long>

salida_act1: Envía el valor del voltaje al actuador del motor 1. Tipo <double>

salida_act2: Envía el valor del voltaje al actuador del motor 2. Tipo <double>

salida_act3: Envía el valor del voltaje al actuador del motor 3. Tipo <double>

Supervisor

Componente similar a “control_estados1” y “control_estados2”. Funciona a modo de switch. Conecta unas entradas con un componente hasta que se le da la orden de cambiar. Entonces conecta las entradas con otro componente. La orden de cambio de estado le puede llegar por un botón físico, por un topic de ROS o porque se ha superado un valor de fuerza máximo.

Entradas:

entrada_control: Recibe la información de que el módulo principal está listo, por lo tanto indica que empieza un ciclo nuevo. Tipo <int>

entrada_fuerza: Recibe la información leída por el sensor de fuerza. Tipo <vector <double>>

entrada_refe: Recibe la referencia que debe seguir el robot. Tipo <vector <double>>

entrada_alarm: Canal por el que recibe desde un topic de ROS la orden de cambiar de estado. Tipo <std_msgs::Bool>

Salidas:

salida1_control: Envía la información de que el componente "ModuloPcl833" está listo mientras el switch se encuentre en el estado 1. Tipo <int>

salida1_fuerza: Envía los valores de fuerza recibidos al componente conectado mientras el switch se encuentre en el estado 1. Tipo <vector <double>>

salida2_control: Envía la información de que el componente "ModuloPcl833" está listo cuando el switch se encuentre en el estado 2. Tipo <int>

salida2_refe: Envía la posición de referencia del robot cuando el switch se encuentre en el estado 2. Tipo <vector <double>>

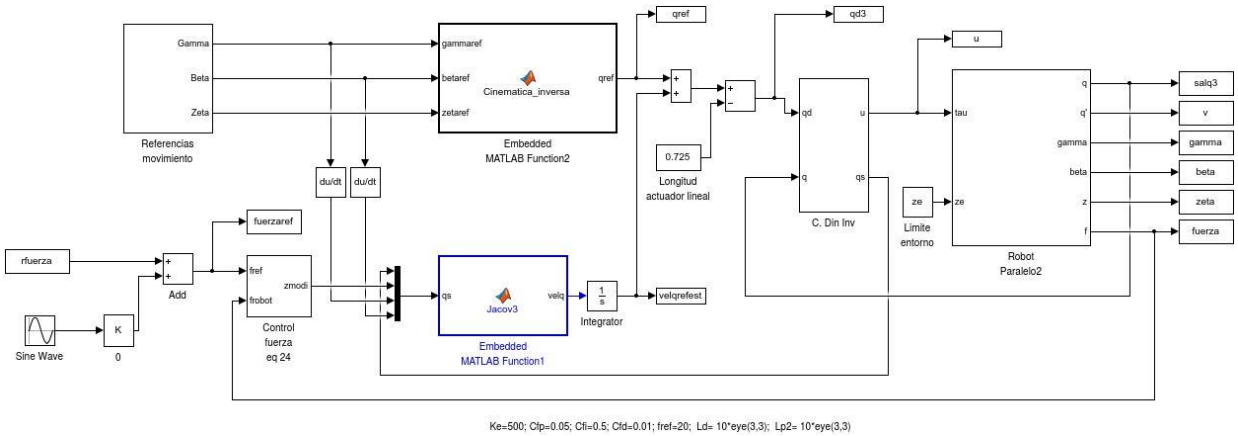
salida2_fuerza: Envía los valores de fuerza recibidos al componente conectado cuando el switch se encuentre en el estado 2. Tipo <vector <double>>

Atributos:

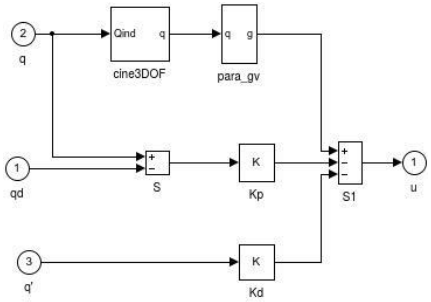
Rfuerza: Límite de fuerza al que el componente cambiará de estado. Tipo <vector <double>>

Esquemas

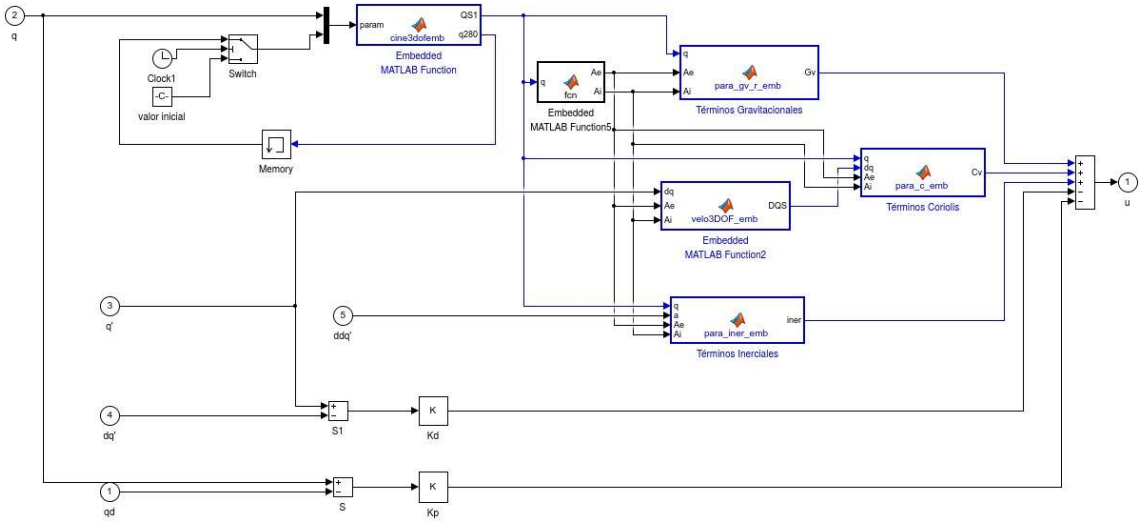
Control híbrido de posición y de fuerza con jacobiano



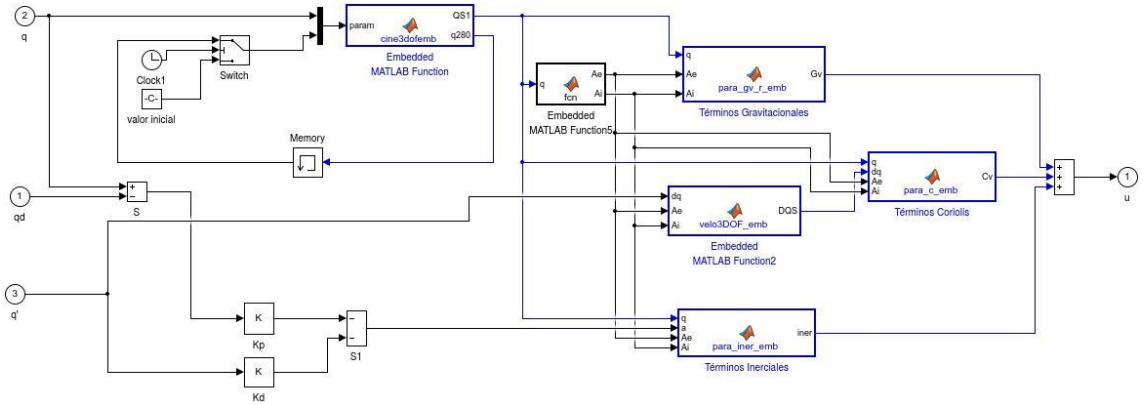
Paden pasivo únicamente con términos gravitacionales



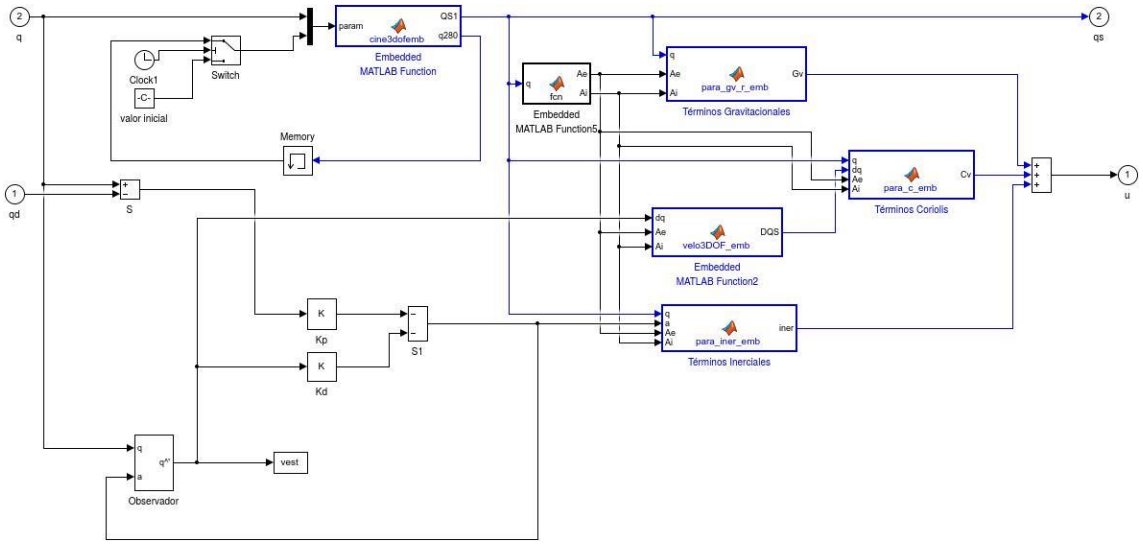
Paden pasivo



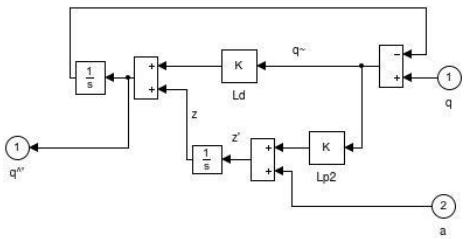
Dinámica inversa



Dinámica inversa con observador de estados.



Observador de estados



Apéndice B

Guía de usuario para el robot paralelo de 4DOF

Guía de usuario para el robot paralelo de 4DOF

1. Introducción

Esta guía de usuario pretende ser una ayuda para todo aquel que vaya a trabajar con el robot paralelo de cuatro grados de libertad. Esta guía se debe ir completando y corrigiendo a medida que se va avanzando con el proyecto, por lo que va a estar alojada en *Bitbucket* en la rama master del repositorio. Así se puede tener un mayor control sobre los cambios que se vayan realizando.

Hay dos secciones diferenciadas, una de hardware donde se intenta explicar cómo se han usado las tarjetas y otra de software dónde se dan los detalles de cómo funcionan los programas escritos para el proyecto.

En la sección de software se destaca la importancia de trabajar de una forma organizada, por eso se ha subido el repositorio a *Bitbucket*. Es recomendable hacer un *commit* para cada cambio que se haga, por pequeño que parezca. De esta manera es más fácil volver a una situación concreta en caso de error, en vez de volver a una situación anterior en la que se han hecho tantos cambios que es difícil identificar el problema. También se recomienda escribir todo en inglés, tanto los *commits* como los comentarios internos del software.

La web oficial de *Orocos* hace tiempo que no se actualiza, aunque mucha de la información sigue siendo útil. También se puede consultar en la página de ROS o utilizar los foros de consultas para más información.
Para obtener información adicional sobre las tarjetas se puede visitar la página web de *Advantech.*

2. Hardware

Analog output card PCI-1720U

Esta tarjeta dispone de 4 salidas analógicas con 12bits de resolución con las que se van a controlar los actuadores del robot. Se van a enviar valores entre 0 y 10V.
Las salidas se conectan a las entradas de las tarjetas de control.

CED Power1401 mk II

El Power1401 se utiliza para leer la señal de hasta 16 canales, aunque aquí solo se van a utilizar 4. Necesita los drivers que se pueden descargar de la página oficial, aunque solo están disponibles para Windows. El software que ofrece el fabricante es privativo y de pago, por lo que se ha optado por escribir uno ad hoc haciendo uso de los documentos que proporciona el fabricante al instalar el driver.

CED 1902

Se trata de un acondicionador de señal programable de la empresa *Cambridge Electronic Design Limited* y se va a utilizar para acondicionar la señal de un electromiograma. Se necesitan los drivers oficiales que se pueden descargar desde la web aunque solo están disponibles para Windows. Una vez instalados los drivers también se habrán instalado dos programas que, con un cable serie, serán suficiente para utilizar el CED1902. El primer programa es Try1902, con este podremos verificar que todo funciona correctamente. Además es necesario para guardar las opciones de puerto series y canales disponibles. Para ello pulsamos en *File/Set port and channels…* Con esto ya podemos utilizar *Ctl1902* para ajustar el acondicionador de señal. Las opciones para leer desde la entrada *Electrodes* son *Isolated ECG* y *Grounded ECG.*

Encoder card PCI-1784U

Esta tarjeta se encarga de leer los cuatro encoders del sistema. Tiene las entradas A+ ,A- , B+, B-, Z+ y Z- para conectar las salidas de los encoders. También cuenta con cuatro salidas digitales TTL que sirven para alimentar a los encoders.

Enhanced Multi-Lab Card PCL-812PG

Esta tarjeta multiusos hará de auxiliar para controlar los motores. En la primera versión de las tarjetas de control de los motores hace falta una entrada de entre 8 y 9V para desactivar el freno de los motores. Esto se hace mediante el canal 1 de las salidas analógicas de la PCL-812PG.
La otra función de esta tarjeta es la de controlar el sentido de marcha del motor. Esto se hace usando las salidas digitales. A cada canal le corresponden dos salidas (dos bits). Cuando los dos bits son iguales, [0 0] o [1 1], el motor no responde. En cambio el actuador sube cuando el primer bit por la derecha es 1 y el segundo bit es 0, es decir [0 1], el actuador baja en caso contrario, [1 0].

3. Software

Bitbucket

El proyecto parallel4dof se aloja en Bitbucket, un servicio de alojamiento basando en web para proyectos que utilizan el control de versiones Mercurial o Git. De esta manera es más fácil coordinar el trabajo de varias personas y de llevar un mayor control sobre los cambios que se van realizando.

Se ha creado un repositorio privado, por lo que es necesario que el administrador de acceso a los nuevos miembros.

Se utiliza el sistema Git de control de versiones y en el ordenador *paralelo4dof* el repositorio se encuentra en el directorio */home/paralelo4dof/parallel_robot/*.
En un primer nivel hay dos carpetas y un archivo Readme.md:

- **src/**: Contiene los códigos fuente, tanto de los componentes como de las bibliotecas.
- **scripts/**: Contiene los scripts con las instrucciones y esquemas de control que se introducen en el deployer.

El archivo **Readme.md** debe contener la información básica del repositorio (recuerda que debe estar escrito en inglés). Se explica cómo se crea un componente nuevo y la lista de los componentes ya existentes con una breve descripción. Dentro del directorio de cada componente se debe crear otro archivo **Readme.md** que contenga toda la información que pueda ser de interés acerca de ese componente.

Por el momento, el criterio para *mergear* un componente a la rama *Master* es que este se pueda compilar. Para nuevos componentes que no estén listos se puede crear una nueva *branch*.

Bibliotecas

Las bibliotecas incluidas en este proyecto se usan principalmente para controlar las diferentes tarjetas y para incluir funciones características del robot paralelo con cuatro grados de libertad como puede ser calcular la posición de los actuadores por cinemática inversa. El código fuente de las librerías se almacena en el directorio *parallel_robot/src/lib/*.

Biodaq

Biodaq es el conjunto de bibliotecas que proporciona el driver de *Advantech* para sus tarjetas PCI. En el directorio */home/paralelo4dof/Documents/advantech/* está toda la información acerca del driver instalado y las tarjetas. Dentro de la carpeta *linux_driver_source_3.2.7.6_64bit* se encuentra el driver propiamente dicho y una carpeta con ejemplos que se pueden consultar para ver cómo se implementan las funciones.

Parallel4dof

Esta biblioteca contiene una clase con todas las funciones intrínsecas al robot paralelo con cuatro grados de libertad.

- `parallel4dof::inverseKinematics( vector<double> &prismaticJoints, const vector<double> &polarCoordinateSys)`

Calcula las 22 componentes cinemáticas de los actuadores a partir de las coordenadas polares [x , z , θ , Ψ].

PCL812

Esta biblioteca contiene una clase con las funciones necesarias para comunicarse con la tarjeta PCL-812PG.

- `Pcl812::ana8digi( unsigned short int canal)`

  Devuelve el valor(double) en voltios que la tarjeta está leyendo por el canal que se le introduce como parámetro.

- `Pcl812::adigi8ana( double volt, unsigned short int canal)`

  Pone la salida del canal indicado como parámetro a la tensión definida por `volt.`

Use1432.dll

Esta biblioteca se usa en el software hecho para el control del Power1401 y proporciona las funciones necesarias para comunicarse con el aparato.

Power1401

No se trata de una biblioteca. Es un programa escrito para leer la señal recogida por el CED Power1401 mk II y enviarla por medio de sockets al componente de Orocos Emg_socket.
El Power1401 se controla enviándole *strings* con las órdenes. Para que lea continuamente de las entradas se le envía: "ADCMEM,I,2,0,512,0 1 2 3,0,C,1,25;"

ADCMEM,kind,byte,st,sz,chan,rpt,clock,pre,cnt;

kind: Modo de operación. I para leer por interrupciones y F para hacerlo de modo secuencial.

byte: Se guardan los datos en formato 8-bits si es 1 o 16-bits si es 2. El formato do 8-bits está obsoleto.

st: Posición de memoria donde empieza a almacenar datos.

sz: Tamaño del buffer a almacenar. Si byte es 2, sz tiene que ser el doble de los datos que queramos almacenar.

chan: Canales que queremos leer (y orden).

rpt: Número de repeticiones. Con 0 está leyendo siempre.

clock: Tipo de reloj. C es 1MHz.

pre*cnt: El reloj se divide por la multiplicación de estos dos parámetros para dar las lecturas por segundo.

Una vez enviada la orden de leer al aparato, se va recogiendo la información con la función U14toHost(), se hace la media y se envía por un socket al componente de Orocos encargado de recogerla.

Componentes de Orocos

Los componentes son la unidad básica de Orocos. Para crearlos hace falta un espacio de trabajo basado en Catkin. Para este proyecto dicho espacio esta creado en */home/paralelo4dof/parallel_robot/.* Dentro de la carpeta *src/* podremos crear un nuevo componente usando la herramienta *orocreate-pkg:*

```
~$ cd ~/parallel_robot/src
~$ rosrun rtt_ros orocreate-pkg my_component component
```

Para que Catkin lo detecte como un componente hay que reemplazar el archivo *Manifest.xml* por *package.xml.*

Actuador

Este componente es el encargado de suministrar la tensión a cada uno de los motores a través de la tarjeta PCI-1720U. Para seleccionar el canal se define el atributo “channel” en el script, este puede ser 0, 1, 2 o 3.

Al configurar el componente se pone el motor correspondiente a tensión cero y se abre el archivo “actuadorX.txt” (X corresponde al número de canal) donde se almacenarán los valores de voltaje que se vayan enviando.

Al ejecutar startHook(), se desactivan los frenos de los motores poniendo la salida analógica 1 de la tarjeta PCL-812PG a 10V.

Durante la ejecución de updateHook() se va enviando el voltaje (0V-10V) a través de la tarjeta PCI-1720U y se controla el sentido del motor con las salidas digitales de la tarjeta PCL-812PG. Si se recibe la señal de fin se pone ponen el motor a tensión 0V.

El stopHook() pone el motor a 0V, pone el sentido de giro del motor a una situación de indeterminación (ni sube, ni baja) y vuelve a activar los frenos.

Entradas:

act: Canal por el que recibe el valor del voltaje. Tipo <double>

entrada_fin: Recibe la señal para que finalice. Tipo <bool>

Salidas:

ticksFin: Envía los ticks del reloj del sistema en el momento en el que se hace el update. Tipo <long long>

Atributos:

channel: Inidica el motor que se va a controlar con ese componente. Tipo <int>

Emg_socket

Recibe los datos de la lectura del electromiograma enviados por el programa Power1401 desde un ordenador con Windows y los envía por cuatro puertos diferentes.

Salidas:

emgX_out: Envía la señal del canal X del electromiograma. Tipo <double>

Encoder

Lee y envía la posición en pulsos leídos por el encoder. 500 pulsos equivalen a una vuelta completa. Este componente es el que hará de principal y al que se le asignará el periodo del ciclo.

Al configurar el componente se prepara la tarjeta para leer desde el canal seleccionado y se abre el archivo “encoderX.txt”(X corresponde al número de canal) donde se almacenarán los valores del encoder conectado al canal del componente.

El startHook() conecta la alimentación del encoder.

En cada update se lee el valor de los pulsos y se envía por el puerto “salida_posReal” mientras no se haya recibido la señal de parar, en dicho caso se ejecuta stopHook().

El stopHook() desconecta la tarjeta y cierra el archivo de texto.

Entradas:

entrada_fin: Recibe la señal para que finalice. Tipo <bool>

Salidas:

salida_cntrl: Envía la señal para empezar un nuevo ciclo. Tipo <bool>

salida_posReal: Envía el valor de los pulsos leídos por el encoder. Tipo <double>

Atributos:

channel: Indica el motor que se va a controlar con este componente. Tipo <int>