Tema 4

Sistemas robotizados.
Opciones de diseño e implementación.

En esta unidad veremos:

4.1 Modelado y control cinemático en robots manipuladores.
4.2 Problemas de los robots. Ideas y soluciones.
4.3 Características diferenciadoras de las técnicas de programación de robots y de sistemas robotizados.
4.4 Opciones en el diseño e implementación de sistemas robotizados.

.

Índice

4.1 Modelado y control cinemático en robots manipuladores.

El 1 de julio de 2021 la empresa nVidia anunció en su página WEB, dedicada a los mejores proyectos basados en la plataforma Jetson, un brazo robótico. El proyecto consiste en el modelado y entrenamiento de un robot, de tal manera que pueda coger un objeto y lanzarlo lejos con intención de hacerlo pasar a través de un hueco practicado a una caja. Este es el principio de funcionamiento de un juego popular estadounidense llamado Cornhole.

David Niewinski empleó un brazo de Kuka, modelo KR20 al que pareó con una unidad del kit de desarrolló nVidia Jetson AGX Xavier. Empleó una cámara con resolución FHD para que sirviera como «ojos» del robot y para facilitar la detección del agujero en la caja, lo coloreó de rojo a fin de que resaltara sobre el fondo. A las pocas horas de entrenamiento, ya ganaba sin dificultad a competidores humanos (Figura 4.1).

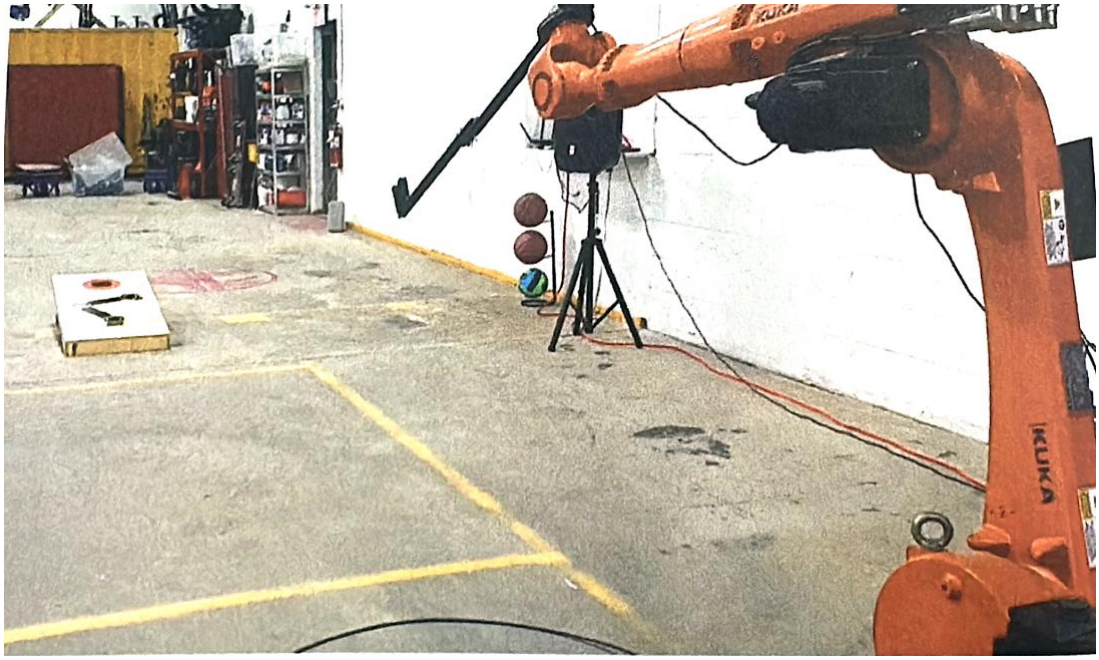

Figura 4.1 Brazo robótico.

Las funciones de captación de imagen y detección de característica (el agujero) fueron implementadas en OpenCV mediante inRange y findContours. La comunicación entre Jetson y Kuka se realizó por medio de KUKA.ethernetkRL.

El entorno de trabajo (el modelado) fue generado por medio de la aplicación de Kuka para asegurar una operación segura en la cual el brazo no colisionaba con ningún obstáculo.

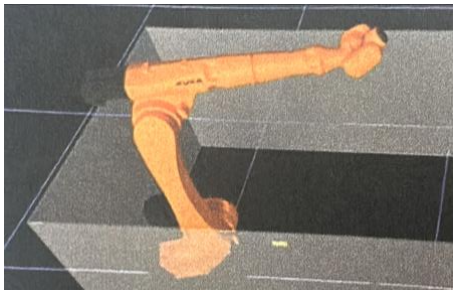

Figura 4.2 Brazo robótico.

Para saber más…

Un brazo KUKA es un tipo de robot industrial fabricado por la empresa alemana KUKA Robotics. Estos brazos robóticos son utilizados en una amplia variedad de aplicaciones industriales, como ensamblaje, soldadura, manipulación de materiales y fabricación automatizada.

Los brazos KUKA están diseñados para ofrecer alta precisión y flexibilidad en la automatización de procesos. Funcionan con servomotores y pueden programarse mediante software avanzado para realizar tareas específicas con gran eficiencia. Además, pueden integrarse con tecnologías de inteligencia artificial y visión artificial para mejorar su rendimiento en entornos industriales.

En el siguiente link se pueden estudiar las características del proyecto Kickstarter: https://www.kickstarter.com/projects/jetmax/jetmax-the-ai-vision-robotic-arm-for-endless-creativity?lang=es

Todo ello es un ejemplo de cómo la inteligencia artificial (IA) puede ser empleada con robots manipuladores para llevar a cabo una tarea concreta. Desde seguidores de junta a robots de antorcha de soldadura, las aplicaciones de la IA en este campo no han hecho más que empezar a florecer.

Por otra parte, un proyecto de Kickstarter ha hecho muy popular a este otro robot manipulador controlado mediante Jetson: Jetmax.

Jetmax ha sido concebido como un diseño que cumple todo lo necesario para poder ser parte del material educativo en un centro:

- Respecto al *hardware*, el montaje es amigable y en sí mismo didáctico, acercando al alumno a la parte de modelado cinemático. Incluye un kit de desarrollo Jetson nano con tarjeta de memoria de 32GB, cámara HD, ventilador, antenas para conexión inalámbrica, un servo HTS-35H y un micro servo, así como bastidor y estructura. Obedece a una cinemática de 4 + 1 ejes, dimensiones de caja de abarque de 450 x 160 x 260 mm, un peso propio de 1,6 kg y una capacidad de izado de unos 0,45 kg. Se alimenta mediante una fuente de 12 V y 5 A, y tres configuraciones de boquillas intercambiables para hacer la cogida y el izado de objetos. Tiene la posibilidad de ser ampliado mediante una determinada cantidad

de sensores (por ejemplo, de proximidad) por medio de los puertos GPIO de la Jetson, que por ser semejantes a los de RPi facilitan que casi cualquier sensor comprado para RPi o Arduino sirvan para Jetson directamente, o con una adaptación mínima.

• En cuanto al *software*, dispone de sistema operativo ROS (Robot Operating System y paquetes y librerías en Python/C/C++ principalmente. Puede conectarse al PC a través de Ethernet, wifi, o telecontrol, igual que el JetBot original.

Ejemplo 1

Tomando como base el proyecto Kickstarter, se estudia cómo se realiza el modelado y el control cinemático de estos robots manipuladores.

En un primer momento, todo parte de un entorno de diseño en las que el diseñador ha de dar forma a la idea que previamente ha plasmado en un boceto. Si bien los bocetos solían ser realizados sobre lienzo o papel a mano alzada, la digitalización del dibujo técnico ha supuesto que actualmente se realice en algunos casos mediante tabletas digitalizadoras, que permiten escribir con un pen digital a un fichero de imagen que queda almacenado en el dispositivo.

Figura 4.3 Modelo de entorno de diseño.

Durante esta etapa, el diseñador solo tiene un concepto general del producto que quiere conseguir, y una idea vaga de las restricciones o de las características necesarias para llegar. En ese momento y en paralelo a los primeros cálculos, dicho diseñador comienza a croquizar en un programa de modelado paramétrico 3D la parte mecánica de su producto. Estos programas permiten definir las piezas, las uniones y los movimientos hasta el punto de poder simular las uniones. Existen muchos paquetes de software capaces de hacer esto. La labor de modelado 3D es relativamente común, y los paquetes más habituales son Solid Works, Solid Edge, Inventor, Mechanical Desktop/AutoCAD Mechanical, Catia, CREO (anteriormente PROEngineer).

Sobre el modelo 3D se puede trabajar ya de distintas maneras según el grado de digitalización de la empresa, aunque en España, hoy por hoy, sigue siendo común el uso de planos de fabricación en los talleres, por lo que a partir del sólido 3D se generan los

planos, y se acotan con instrucciones de diseño (acotación funcional), fabricación (procesos y utillajes asociados) y calidad (planes de inspección).

No obstante, antes de llegar a ese punto, el modelo generado en CAD 3D nativo de alguna de las aplicaciones mencionadas se traslada al entorno de software de simulación del ROBOT, como el antes mencionado de Kuka. Ahí se comprueba que con la programación pretendida del robot no se generen riesgos para las personas físicas, y se puede llevar a cabo un primer entrenamiento por coordenadas, que por otra parte también se podría realizar mediante programación manual en **modo palpador**.

Para saber más…

La **programación manual en modo palpador** es una técnica utilizada en robótica industrial para enseñar manualmente al robot la trayectoria o los puntos clave de un proceso, utilizando un **palpador** o **teach pendant** (dispositivo de control portátil).

¿Cómo funciona?

1. **Modo de enseñanza**: El robot se coloca en un modo especial donde su movimiento se controla manualmente, sin intervención de códigos de programación.
2. **Uso del palpador**: Un operador mueve el brazo robótico utilizando el **palpador** (que puede ser un joystick, botones o una interfaz táctil).
3. **Grabación de posiciones**: El operador lleva el brazo a diferentes posiciones clave y las guarda en la memoria del robot.
4. **Ejecución automática**: Una vez enseñados todos los puntos, el robot puede ejecutar la trayectoria de forma repetitiva con alta precisión.

Ventajas de la programación manual en modo palpador

- **Fácil de aprender**: No requiere conocimientos avanzados de programación.
- **Precisión**: Permite ajustar manualmente los puntos críticos.
- **Rápida implementación**: Útil para tareas repetitivas en la industria, como soldadura, ensamblaje o pintura.

Aplicaciones comunes

- Soldadura robótica (como en brazos KUKA con antorcha de soldadura).
- Corte por láser o chorro de agua.
- Pintura y aplicación de recubrimientos.
- Manipulación de objetos en líneas de producción.

En el siguiente link se muestra el analizador de cadenas cinemáticas de PTC CREO: https://support.ptc.com/help/creo/creo_pma/usascii/index.htm|#page/tutorials_pma/exercise_04/motion analysis.html

4.2 Problemas de los robots. Ideas y soluciones.

En el uso de robots existe una serie de labores complicadas que son conocidas como grandes obstáculos. Algunas de ellas al tenerlas en cuenta se solucionan por sí mismas, mientras que para otras actualmente no hay una conclusión sencilla.

Un primer problema elemental es el **abastecimiento energético**. De la misma manera en que los seres humanos necesitan de los alimentos, los robots precisan energía para funcionar.

Cuando se trata de un **dispositivo robótico estacionario**, basta con haber previsto un punto de conexión a la red eléctrica, y pagar el coste asociado a la misma. Por supuesto nada es trivial y el correcto control de la electrónica de potencia es necesario para que el suministro sea estable, protegido contra sobreintensidades y sobretensiones, y energéticamente eficiente (por ejemplo, las conversiones con ciclo convertidores en contra de la disipación de energía mediante resistencias).

El uso de **PDBs (placas de distribución de energía o Power Distribution Boards) y de BMS (sistemas de monitorizado y carga de baterías o Battery Monitoring Systems**) está ya sobradamente implantado y se considera un reto superado, pero que no debe ser olvidado.

En el caso de **dispositivos móviles o no estacionarios**, puede ser más complicado y **la capacidad de carga y descarga de las baterías eléctricas es finito**. Destacan en este punto **las baterías KOKAM** (https://kokam.com/) de alta eficacia y capacidad de descarga, y **los sistemas de energía híbrido** como los turbogeneradores mediante imanes permanentes en los que apenas se disipa energía por perdidas mecánicas o térmicas. Sistemas como **el frenado regenerativo (similar al KERS de la F1), células de hidrógeno,** etc., se están implantando poco a poco en los sectores más punteros como el aeronáutico o la defensa. **El uso de motores de combustible fósil** (que lleva acarreado el empleo de un depósito para el mismo) **está condicionado en gran medida al tamaño del robot** y el payload (capacidad de carga final) objetivo del producto.

Figura 4.4 Ideas y soluciones para los problemas de los robots.

La inevitable consecuencia del abastecimiento energético es que las cargas y las inercias deben de ser estudiadas en detalle (y en muchas ocasiones minimizadas). Aún así, **uno de los errores más comunes en robots estacionarios**, donde la ausencia

de limitaciones de energía relaja el miedo al sobrepeso **es subestimar la carga asociada al movimiento del robot y sus inercias**, dejando de contabilizar, por ejemplo, el peso de las herramientas de fin de brazo o el del material fungible o de aporte en los brazos robóticos de soldadura con aporte. Obviar las limitaciones de diseño impuestas por el teorema de conservación de la cantidad de movimiento puede, asimismo, dar lugar a que ignorar las fuerzas de inercia generadas por las cargas (en adición al peso propio) sobrecarguen el robot (motores, servos, acoplamientos, embragues, etc.).

Recordando la física elemental, reducir el peso siempre es una solución que permite disminuir la energía del sistema (cinética, potencial o, conjuntamente, mecánica), pero esto no siempre es posible. La segunda alterativa, **reducir la velocidad,** da lugar a un aumento del tiempo de ciclo que suele ser un efecto colateral indeseado. En todo caso, el principal problema a tener en cuenta cuando se calculan las inercias es el ligado al de la seguridad del entorno de trabajo del robot, si la cantidad de movimiento en caso de colisión pudiera hacer peligrar la vida de un ser humano.

Como **segundo problema, se observa la selección del tipo de robot más adecuado para la aplicación deseada**, que es crucial. Metodologías como LOSTPED (Load, Orientation, Speed, Travel, Precission, Enviroment y Duty cycle) de Bosh Rexroth permiten estandarizar el proceso de toma de decisiones y poder determinar con claridad si es preferible un robot SCARA, uno DELTA, uno de seis ejes, uno cartesiano, etc. Este es un primer paso de cara al diseño y modelado mecánico, que condiciona el resto del desarrollo.

LOSTPED es un acrónimo que se refiere a una metodología utilizada en el diseño y selección de robots industriales. Sus siglas representan los siguientes factores clave a considerar en el proceso de toma de decisiones:

- **L**oad (Carga): Peso y características del material a manipular.
- **O**rientation (Orientación): Posición y dirección de la carga en el espacio.
- **S**peed (Velocidad): Ritmo de operación requerido.
- **T**ravel (Desplazamiento): Distancia y recorrido del movimiento del robot.
- **P**recision (Precisión): Nivel de exactitud requerido en la tarea.
- **E**nvironment (Entorno): Condiciones ambientales donde operará el robot.
- **D**uty cycle (Ciclo de trabajo): Duración y frecuencia de las operaciones.

Para saber más….

En el siguiente link se puede encontrar la información relativa a LOSTPED: https://dc-us.resource.bosch.com/media/us/products_13/product_groups_1/assembly_technology_/pdfs/Bosch Rexroth LOSTPED_ Paper.pdf]

La complejidad y la flexibilidad de adaptación del robot es otro asunto importante. Un robot con un conjunto de tareas dispares y muy sobrecargado da lugar a una celda de trabajo difícil de ajustar, prever y, en definitiva, de difícil cálculo del tiempo de ciclo. Además, incluso supuestos todos esos parámetros controlados y correctos, **un robot que deba realizar labores muy distintas y complejas habrá de tener inevitablemente una gran cantidad de sensores, actuadores, posibilidades de movimiento, y un definido tamaño, número de ejes y cadenas cinemáticas, y en último término coste de fabricación y mantenimiento.** Por otra parte, una celda de trabajo excesivamente complicada requiere de personal experto altamente entrenado en esa celda, y la hace más propensa acumular retrasos en casos de paradas de mantenimiento no programado o avería.

Es también fundamental conocer los problemas derivados de no realizar correctamente **el ruteado**. A pesar de lo elemental de este aspecto, existe una fuerte tendencia a no prestar excesiva atención al ruteado de los cables, optimizar el enrutamiento del cable hacia las herramientas de fin de brazo es imprescindible para no limitar los movimientos del robot, y no añadir fuerzas antagonistas al movimiento, ni desgastes en ninguna parte móvil. La rotura de cables puede generar parada de línea y, por tanto, grandes pérdidas. **Rutear el cable en el programa CAD paramétrico (muchos paquetes como CREO lo permiten) consume tiempo en la etapa de diseño, pero agiliza el montaje posterior**, y hay estudios de ingeniería de procesos que indican que el coste asociado al diseño de un producto no supone nunca más del 10-20% de los costes del producto, mientras que solucionar problemas en operación es realmente caro.

La **planificación del movimiento** es un problema en sí mismo, que no está totalmente solucionado. Cómo desplazar un brazo de un punto a otro es una tarea que requiere estudio movimiento a movimiento y que tiene gran cantidad de resoluciones y de opciones (especialmente en sistemas con muchos grados de libertad). Así, por ejemplo, en el desplazamiento de un brazo de un punto a otro, emplear trayectorias que dejen una articulación con excesiva ventaja mecánica puede dar lugar a acodalamientos no deseados.

La ubicación y el SLAM.
La tecnología SLAM permite que se lleve a cabo un mapeo y una localización de manera simultánea,es decir, que el robot pueda **ubicarse en un entorno desconocido mientras crea un mapa de ese entorno al mismo tiempo**. Las siglas son **Simultaneous Localization and Mapping.**

Esto puede ser de gran importancia en algunos dispositivos y aplicaciones, como en la ejecución de un patrón de pintura o en un barrido o aspirado en un robot con desplazamiento, pero no resuelve íntegramente los problemas habituales de localización. Apagar un robot que se halle en una determinada posición, trasladarlo a otro nuevo emplazamiento alejado, y encenderlo, puede dar lugar a que el robot «se pierda» y sea incapaz de ubicarse. Tecnologías de apoyo como la visión artificial o el uso de sensores LIDAR o GPS pueden ayudar, así como los mucho más sofisticados sistemas de navegación inercial (INS) o de guiado inercial, si bien estos no son comunes en robots manipuladores.

Para saber más…

En el siguiente link se puede obtener más información acerca de SLAM: (https://es.mathworks.com/discovery/slam.html)

La **realimentación háptica** es la tecnología que proporciona **sensaciones táctiles o de fuerza** a un usuario a través de un dispositivo, permitiéndole interactuar con entornos virtuales o sistemas robóticos de manera más realista.

¿Cómo funciona?

- Utiliza **actuadores** (vibración, presión, resistencia mecánica) para simular texturas, impactos o fuerzas.
- Se basa en el sentido del **tacto y la propiocepción** para mejorar la percepción en interfaces humano-máquina.

Ejemplo de aplicaciones de la realimentación háptica:

- **Realidad virtual y videojuegos**: Controladores que vibran o generan resistencia para una experiencia inmersiva.
- **Robótica médica**: Cirugías asistidas por robots donde los médicos pueden "sentir" la resistencia de los tejidos.
- **Prótesis avanzadas**: Permiten a los usuarios percibir presión o temperatura en extremidades artificiales.
- **Automoción**: Volantes y pedales con vibración para advertencias en conducción autónoma.

La realimentación háptica marca también uno de los avances robóticos de última generación, pero ni siquiera el uso de galgas extensiométricas y células de carga en combinación con esta tecnología (que no deja de ser una comunicación a través del sentido del tacto, normalmente en forma de un patrón de ondas vibrando) puede permitir que una herramienta de final de brazo sea capaz de manipular objetos como lo hace una mano humana.

Las **galgas extensiométricas** son sensores que miden la **deformación mecánica** de un material cuando se somete a una fuerza o tensión. Funcionan registrando pequeños cambios en su resistencia eléctrica cuando el material sobre el que están adheridas se **estira o comprime**.

¿Cómo funcionan?

1. Se adhieren a la superficie de un material que se va a analizar.
2. Cuando el material se **deforma** por una carga o esfuerzo, la galga también se deforma.
3. Esta deformación cambia su **resistencia eléctrica**, lo que se traduce en un cambio de voltaje medible.
4. Se usa un **puente de Wheatstone** para medir con precisión estos cambios.

Aplicaciones de las galgas extensiométricas

Ingeniería estructural: Monitoreo de puentes, edificios y estructuras para detectar tensiones.
Robótica y biomecánica: Sensores en prótesis y exoesqueletos para medir fuerzas aplicadas.
Industria automotriz y aeroespacial: Pruebas de fatiga en materiales y componentes.
Instrumentación médica: Medición de presión arterial y fuerza en dispositivos médicos.

Son esenciales en **control de calidad, diseño de materiales y monitoreo de seguridad** en diversas industrias.

Algo fácil, como levantar un objeto estacionario con un plano de fácil agarre es un tema de estudio y mejora hoy en día. De manera general, un solo robot no puede tener una herramienta de fin de brazo que le permita amoldarse a los diferentes tipos de objetos que hay en el mundo. Es posible que un brazo manipulador tenga una herramienta de fin de brazo capaz de agarrar una botella de refresco, o por similitud de agua, de tomate o de mayonesa, pero por ejemplo esa herramienta no podrá agarran una zanahoria. El diseño de un set finito de herramientas de fin de brazo es fundamental, en virtud de los usos que se haya pensado dar al robot (Figura 4.5).

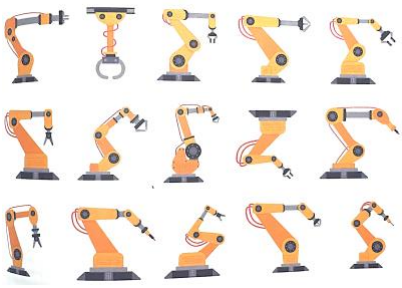

Figura 4.5 Modelos de herramienta de fin de brazo en un robot.

La profundidad es algo difícil de determinar para un robot. Es necesario más de un sensor para ello, de la misma manera que una persona necesita la visión de sus dos ojos para hacer una composición de la profundidad. El problema para un sistema

informático se agrava si hay desplazamientos, es decir, si se desea determinar la profundidad de un objeto en desplazamiento o si el propio brazo está sujeto a una unidad móvil y se está desplazando. Existen algoritmos de aprendizaje profundo que comienzan a dar muy buenos resultados en este campo.

Exactitud y precisión son dos parámetros de gran importancia. Que un robot o un subsistema o mecanismo sea repetible, implica regresar con precisión a la posición de un punto determinado, lo que implica que sensorialmente el robot sea capaz de ubicar ese punto y que mecánicamente el robot sea capaz de un movimiento con suficientes particiones para alcanzarlo. Esto lleva parejo los conceptos de resolución y de tolerancia (baste como ejemplo poner la tolerancia de fabricación de las propias piezas del motor para obtener esa precisión, de manera análoga a lo que le ocurre a los tornos, fresas y máquinas herramientas de gran precisión en relación con sus usillos). Todo ello significa que si no es necesaria una excesiva precisión, no deben exigirse tolerancias de fabricación que encarezcan innecesariamente el coste; sin embargo, si es indispensable esa precisión, eso debe reflejarse en el diseño del robot y en sus componentes (Figura 4.6).

Figura 4.6 Ejemplo de precisión en un robot de brazo.

La importancia de la mecatrónica en relación con el control también ha de ser valorada. Con frecuencia, se elige un robot en virtud de su control, sin atender a su mecánica y a su estructura. Un robot es un sistema mecatrónico, y una vez puesto en servicio es poco probable que en ausencia de cambios en sí mismo o en el entorno el control falle por sí mismo. Lo habitual es que a medida que la parte mecánica se degrada por uso, comiencen a aparecer variables ocultas que acaben sacando al proceso de control, y haciendo necesario actuar a nivel de mantenimiento sobre el robot. Haber diseñado el Robot teniendo en consideración este punto puede reducir drásticamente el tiempo no productivo. Estas labores de integración del diseño en el apoyo logístico reciben internacionalmente el nombre de ILS.

No valorar correctamente el estado de la técnica, o lo que es lo mismo sobreestimar las funciones del dispositivo de control del robot, puede dar lugar a tareas imposibles, que tras mucho esfuerzo en tiempo y en coste cause que se desista en la implementación. Por otra parte, infravalorar las capacidades del robot da lugar a una

pérdida de oportunidad que ha de ser evitada. Un error del primer ejemplo es intentar controlar demasiadas entradas o salidas, servomecanismos adicionales o rutinas simultáneas. Un error de este tipo aún más catastrófico es pretender implementar una solución sin consultar el estado de la técnica. Por ejemplo, los robots de soldadura de seguimiento de juntas funcionan muy bien con chapas de dimensiones determinadas (y espesores habitualmente pequeños, como el empleado en la automoción) para soldaduras en una sola pasada. Aplicar un seguidor de juntas con una soldadura multipasada genera que una vez finalizada la primera pasada el contorno desaparezca, y se haya generado uno nuevo, que si el control no está diseñado para detectar dará lugar a un error o a una mala soldadura. Por otra parte, no comprender las habilidades completas del robot antes de la puesta en marcha, o no tener en cuenta las posibilidades de crecimiento del robot son ejemplos de costes de oportunidad a evitar.

El factor humano es tanto o más importante que el técnico. En mayor o menor medida el robot estará en un centro productivo rodeado de otros robots y de seres humanos. La formación media de los trabajadores y el nivel de aceptación de la robótica pueden afectar significativamente a su implementación, y el propio diseño (aspecto, forma) del robot es posible que genere un rechazo inicial entre los trabajadores. De la misma manera, detalles menores de acabado como que las chapas no tengan aristas vivas pueden mejorar la interacción hombre-máquina.

Por finalizar con algo más novedoso, las interfaces cerebro-computadora (BCI) aun presentan muchas limitaciones a la hora de ser empleadas con las neuroprótesis, los dispositivos de estimulación eléctrica funcional y los exoesqueletos.

4.3 Características diferenciadoras de las técnicas de programación de robots y de sistemas robotizados.

A la hora de programar un robot hay dos métodos o familias de métodos que se pueden utilizar. El primero de ello es el de programación por palpación de coordenadas o Teach Pendant, y el segundo es por programación y simulación estructurada.

4.3.1 Teach Pendant

Según la Asociación Británica de Automatización y Robots, más del 80% de los robots se programan mediante Teach Pendant también conocido como CLI (interfaz de línea de comandos de consola) o programación por puntos de palpador (Figura 4.7).

Figura 4.7 Teach Pendant.

Se le llama CLI porque se emplea un dispositivo similar a una tablet rugerizada: para programar el robot el operario lo mueve a un punto y con la tablet empleando la interfaz de botones de línea de comandos graba el punto, repitiendo este proceso para todos los puntos de la trayectoria que desee. Cuando se termina, el robot puede reproducir los puntos a toda velocidad, con una cierta cantidad de salvedades:

• En primer lugar, el robot necesita interpolar las posiciones del brazo entre un punto y el siguiente. El método de interpolación más habitual es el lineal, es decir, el robot viaja en una línea imaginaria trazada entre las coordenadas de un punto y del siguiente. Si de un lugar a otro la cabeza gira, hace una doble interpolación: de traslación, y de rotación. Sin embargo, robots con controles avanzados pueden encontrar patrones y minimizar (mediante mínimos cuadrados o empleando cualquier otra función) el error trazando elipsoides u otras trayectorias no lineales.

• En segundo lugar, el número de ejes y de grados de libertad condicionan esta interpolación. Insuficientes grados de libertad dan lugar a una no solución de la trayectoria que será detectada inmediatamente por el operario, pero un exceso de grados de libertad produce una indeterminación de un sistema de ecuaciones que, aplicando la regla de Cramer, dará lugar a más de una solución.

4.3.2 Simulación/programación estructurada

La programación estructurada o simulada se utiliza a menudo en la investigación de la robótica para el desarrollo de los algoritmos de control avanzado o para células de trabajo complicadas (Figura 4.8).


Figura 4.8 Programación estructurada.

Está concebida para reducir el tiempo de inactividad y mejorar la eficiencia. En talleres de pequeño tamaño o en lugares de fabricación de muchos productos en pequeños lotes, está también altamente indicada, ya que es más probable que los robots se configuren varias veces que en grandes centros productivos con altas tiradas, como las fábricas de automóviles.

La programación estructurada y simulada permite programar el robot usando una maqueta virtual del mismo y del entorno. Si el software de simulación es de uso in-tuitivo, es una excelente manera de trabajo de «croquizar un programa» probando una programación antes de trasladarla al robot. Estos paquetes informáticos contienen en ocasiones bibliotecas ya generadas por el fabricante con los modelos de sus robots y de otros elementos comunes como cintas y pasarelas.

Los lenguajes de programación son interfaces de comunicación intermedias, escritas en el denominado «alto nivel» con una serie de palabras clave muy similares al razonamiento de un humano (inglés). Por ejemplo, para expresar una condición, en casi todos los lenguajes se emplea una sentencia que contiene la palabra IF, cuyo significado traducido del inglés es «SI...». La mayor parte de los programas están escritos en este pseudo lenguaje de alto nivel, conocido como código fuente, y posteriormente es compilado, para que el procesador lo pueda ejecutar, lo cual quiere decir que se compila para una arquitectura de computadores determinada.

Al código generado por el compilador se llama código máquina, o en ocasiones «ovejota» por analogía a la extensión OBJ que hace referencia en realidad a la palabra Object. Los ejemplos más característicos de este tipo de lenguajes son C/C++ y VISUAL BASIC y sus variantes NET, todas ellas orientadas a objetos.

Python también es muy popular tanto por su uso en el periodo de pruebas de las máquinas y porque puede ser usado para desarrollar paquetes ROS, como por su alta difusión dentro de la comunidad científica. Sin embargo, Python no es un lenguaje compilable, sino que es un lenguaje interpretado, lo que comporta ventajas y desventajas:

• Ventajas: por un lado, no es dependiente de la arquitectura de la máquina, y por otro, existe una gran cantidad de librerías de manejo de robots desarrolladas en código libre para Python.

• Desventajas: respecto de C son muchas y sin llevar a cabo una comparativa, destacan la gestión y la optimización de la memoria, y la velocidad, lo cual en sistemas de ejecución en tiempo real es muy importante.

4.4 Opciones en el diseño e implementación de sistemas robotizados

Desde un punto de vista general, hay algunas consideraciones que tener en cuenta respecto de al diseño e implementación de los sistemas robotizados:

- Cantidad y tamaño: aunque las opciones en los diseños e implementaciones de sistemas robotizados son muy amplias, la corriente actual es el desarrollo de enjambres de robots, que permiten crear unidades modulares más simples y menos costosas que los robots más grandes, y que llegan a ser tan eficaces como estos últimos.

- Calidad y ambiente: si bien casi todos los robots se diseñan para soportar ambientes normales en relación con la normativa de operaciones de cada país (DEF STAN 00-35, MIL-STD-810, IEC-60068, IEC-TC-75, GAM-EG-13, Y NATO STANAG 4370 principalmente), algunos de ellos han de llevar a cabo su operación en ambientes agresivos. No se refiere esto a actividades tales como navegar y explorar en entornos extremos apenas conocidos, como las profundidades marinas, sino a espacios industriales en los que la atmósfera es excesivamente ácida o corrosiva, o en los que la temperatura es extremadamente alta o baja (o tienen ciclos que trabajan en ambos extremos).
- Ética y de seguridad: la ciberseguridad es una disciplina con un mayor nivel de implantación, que, si bien no es objeto de este texto, si hay que tener en cuenta por el gran riesgo de que un entorno operativo sea víctima de un ciberataque y que un brazo robótico manipulador de gran inercia sea hackeado y comience a moverse intempestivamente....

4.4.1 Tipologías físicas de los robots

Desde un punto de vista hardware, el alumno debe recordar todo lo estudiado en la rama de Robótica Industrial, y tener presente a modo de recordatorio que los robots industriales pueden clasificarse según los siguientes aspectos:

• El tipo de movimiento (grados de libertad).
• La aplicación (proceso de fabricación).
• La arquitectura (en serie o en paralelo).
• El fabricante.

Según la forma del hardware los robots se clasifican en las siguientes topologías: cartesianos, multi-DOF, SCARA y DELTA:

• Robot cartesiano: se mueve a lo largo de tres ejes (X, Y, Z) con una trayectoria lineal y la herramienta de efecto final siempre mantiene la misma orientación. Debido a la falta de gestión de la orientación, los casos de uso son limitados.
• Robot Multi-DOF (de varios grados de libertad, caros y capaces): tiene más articulaciones y más ejes. Una configuración bien conocida es un robot con tres ejes para moverse y tres ejes para orientar el efector final (seis grados de libertad). El robot puede alcanzar cualquier punto con cualquier orientación. Este tipo de robot es ideal para operaciones multitarea y complejas, aunque el precio es mayor.
• Robot SCARA: es un tipo intermedio entre los dos anteriores. Puede moverse a lo largo de 3 ejes (X, Y, Z) pero tiene un eje más para orientar el efector final en una dirección, por lo que muchas veces recibe el nombre de 3+1. Funciona muy bien para tareas de «coger y colocar».
• Robot DELTA o Araña: es un modelo paralelo que puede llegar a disponer de hasta cinco grados de libertad y que en función del modelo tiene hastaseis ejes. El cuerpo y el efector final se encuentran unidos por cadenas cinemáticas basadas en la aplicación de paralelogramos, que le confieren una forma similar a las patas de una araña.

4.4.2 Tipologías lógicas de los robots basadas en Jetson

Desde el punto de vista del software, es necesario comenzar hablando del sistema operativo.

El Sistema Operativo de Robots (ROS) es un sistema flexible para programar el software de los robots. Es una colección de herramientas, bibliotecas y protocolos que tienen como objetivo simplificar la tarea de crear un comportamiento robótico complejo y robusto en una amplia variedad de plataformas robóticas. Dicho así esto, no parece aclarar mucho en que difiere ROS de un sistema operativo como Windows, pero las diferencias son muchas. ROS es un middleware, un framework de menor nivel basado en un sistema operativo existente (normalmente Linux Ubuntu).

La clave de ROS es la existencia de nodos. Los subprogramas que se escriben se agrupan por nodos y cada nodo puede ser escrito en cualquier idioma. Así, una aplicación puede tener un nodo escrito en Python comunicándose con un nodo escrito en C++. La clave de estos nodos comporta la capacidad de separar tareas y procesos de tal manera que un fallo en un nodo no «cuelgue» todos los subsistemas. Así un robot como Jetmax con ROS instalado podría tener un nodo para la operación de los sensores de visión (cámaras), otro para el control del brazo, un tercero para su propio desplazamiento (en la versión motorizada), alguno para los sensores adicionales que se le pueden añadir, otro para la parte de control de BMS, etc.

Además, como gran ventaja de ROS, se puede obtener un sistema de tiempo real débil, que es bastante más adecuado en términos de electrónica, que un sistema operativo convencional, de forma similar a las diferencias existentes entre una RPi y un Arduino, en relación con la lectura, procesamiento y actualización de sus entradas y salidas. Cualquier sistema basado en Jetson puede implementar ROS, desde una modesta Jetson nano, hasta una Xavier.

Para saber más…

Para acceder al documento completo de Jetson.GPIO puede utilizar el siguiente link: https://github.com/NVIDIA/jetson-gpio

Ejemplo 1

Ejemplo de aplicación, de cara al control de sensores y de actuadores el kit de desarrollo de Jetson nano o Xavier que incluye la misma disposición de entradas y salidas de propósito general GPIO que una RPi.

Complementan estos puertos los de comunicación I2C, I25, serial UART (Tx, Rx), SPI, CSI y USB. Se adjuntan varios códigos de ejemplo de uso.

GPIO

```
!sudo pip install Jetson.GPIO
!sudo groupadd -f -r gpio
!sudo usermod -a -G gpio your_user_name
```

La librería Jetson.GPIO puede ser instalada con Pip para Python con normalidad. Aunque el documento de uso completo es accesible desde la web su funcionamiento es muy sencillo y se resumen en lo siguiente:

• La placa de desarrollo tiene numeradas los pines del puerto GPIO de cuatro maneras distintas, para mantener la compatibilidad con RPi.
Desde el intérprete de Python, el primer paso será, por tanto, decir cuál de las numeraciones se desea emplear, usando una de estas 4 instrucciones, tras el import correspondiente:

```
import Jetson.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setmode(GPIO.BCM)
GPIO.setmode(GPIO.CVM)
GPIO.setmode(GPIO.TEGRA_SOC)
```

Las dos primeras son exactamente iguales a las empleadas con RPi y por consiguiente serán las elegidas en este texto, especialmente la primera que se refiere a los números de pin visibles en la propia placa. Las dos últimas, CVM y TEGRA, usan las cadenas de texto en lugar de los números de cada pin, correspondientes a las señales en CVM/CVB o en el SOC de TEGRA respectivamente.

• A continuación, a cada pin se le asignará una variable, por comodidad si se llevan a cabo cambios en el cableado físico del robot, y para cada pin se le dice si es de entrada (en ese pin se va a recibir un 0 o un 1) o de salida (en ese pin se va a sacar un 0 o un 1 lógico). Además, en caso de que sea de salida, se le indicará si inicialmente comienza siendo un 0 lógico [LOW, señal baja] o un 1 lógico [HIGH, señal alta]. Por ejemplo, se asigna al pin7 la lectura de un sensor (por tanto, será una entrada) y al pin 11 el control de un motor que inicialmente estará en movimiento (por tanto, será una salida que en nivel alto hará que el motor gire, y en nivel bajo que pare).

```
Sensor = 7
Motor = 11
GPIO.setup(Sensor, GPIO.IN)
GPIO.setup(Motor, GPIO.OUT, initial=GPIO.HIGH)
```

Si se quisiera leer el valor del pin 7 (el sensor) en algún momento, se ejecutaría la siguiente instrucción cuyo resultado podría ser o bien GPIO. LOW (0 lógico) o bien GPIO. HIGH (1 lógico).

```
print(GPIO.input(Sensor))
```

Si se quisiera cambiar el valor del motor para pararlo, el código sería el siguiente:

```
GPIO.output(Motor, GPIO.LOW)
```

Cuando se finalice el programa conviene revertir a la configuración por defecto los pines, para ello se emplea:

```
GPIO.cleanup()
```

Es importante destacar que todas las funciones usadas son para la lectura y escritura de valores digitales (binarios). Jetson incorpora la posibilidad de emular valores analógicos a través de sus pines PWM.

Nota: los pines elegidos (7) y (11) puede ser cualquier pin marcado como GPIO en el esquema de pines del fabricante (nVidia) pero sólo el 32 y el 33 pueden usarse como PWM. No obstante, se aconseja no usar como pines de entrada y salida aquellos que tengan asociada alguna función especial, como por ejemplo el pin 11, que se emplea también como puerto de comunicación de serie UART.

SoC GPIO	Linux GPIO #	Alternate Function	Default Function			Default Function	Alternate Function	Linux GPIO #	SoC GPIO
			3.3 VDC	1	2	5 VDC			
PJ.03	75	GPIO	I2C1_SDA	3	4	5 VDC			
PJ.02	74	GPIO	I2C1_SCL	5	6	GND			
PBB.00	216	AUD_CLK	GPIO	7	8	UART1_TXD	GPIO	48	PG.00
			GND	9	10	UART1_RXD	GPIO	49	PG.01
PG.02	50	UART1_RTS	GPIO	11	12	GPIO	I2S0_SCLK	79	PJ.07
PB.06	14	SPI1_SCK	GPIO	13	14	GND			
PY.02	194		GPIO	15	16	GPIO	SPI1_CS1	232	PDD.00
			3.3 VDC	17	18	GPIO	SPI1_CS0	15	PB.07
PC.00	16	SPI0_MOSI	GPIO	19	20	GND			
PC.01	17	SPI0_MISO	GPIO	21	22	GPIO	SPI1_MISO	13	PB.05
PC.02	18	SPI0_SCK	GPIO	23	24	GPIO	SPI0_CS0	19	PC.03
			GND	25	26	GPIO	SPI0_CS1	20	PC.04
PB.05	13	GPIO	I2C0_SDA	27	28	I2C0_CLK	GPIO	18	PC.02
PS.05	149	CAM_MCLK	GPIO	29	30	GND			
PZ.00	200	CAM_MCLK	GPIO	31	32	GPIO	PWM	168	PV.00
PE.06	38	PWM	GPIO	33	34	GND			
PJ.04	76	I2S0_FS	GPIO	35	36	GPIO	UART1_CTS	51	PG.03
PB.04	12	SPI1_MOSI	GPIO	37	38	GPIO	I2S0_DIN	77	PJ.05
			GND	39	40	GPIO	I2S0_DOUT	78	PJ.06

STREAM BÁSICO CON LA CAMARA EN PUERTO CSI

Puede emplearse la información del siguiente enlace: https://github.com/JetsonHacksNano/CSI-Camera

Tenga en cuenta que openCV ya viene instalado con el flasheo básico de las unidades Jetson y de Jetbot. La función gstreamer_pipeline define los parámetros de la

cámara como los cuadros por segundo y la resolu-ción. La función show_camera() llama a cv2.VideoCapture que recibe como argumento el gstreamer antes definido. Es importante no olvidar emplear cap.release() y cv2.destroyAllWindows() para limpiar la memoria al finalizar el programa.

```
https://github.com/JetsonHacksNano/CSI-Camera/blob/master/simple_camera.py
# MIT License
# Copyright (c) 2019 JetsonHacks
# See license


# Using a CSI camera (such as the Raspberry Pi Version 2) connected to a
# NVIDIA Jetson Nano Developer Kit using OpenCV
# Drivers for the camera and OpenCV are included in the base image

import cv2 #Esto importa OpenCV como cv2

# gstreamer_pipeline returns a GStreamer pipeline for capturing from the CSI camera
# Defaults to 1280x720 @ 60fps
# Flip the image by setting the flip_method (most common values: 0 and 2)
# display_width and display_height determine the size of the window on the screen

def gstreamer_pipeline(
    capture_width=1280,
    capture_height=720,
    display_width=1280,
    display_height=720,
    framerate=60,
    flip_method=0, #cambiar este valor a "1" gira la imagen
):
    return (
        "nvarguscamerasrc ! "
        "video/x-raw(memory:NVMM), "
        "width=(int)%d, height=(int)%d, "
        "format=(string)NV12, framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
        "video/x-raw, format=(string)BGR ! appsink"
        % (
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
            display_height,
        )
    )
def show_camera():
    # To flip the image, modify the flip_method parameter (0 and 2 are the most common)
    print(gstreamer_pipeline(flip_method=0))
    cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)
    if cap.isOpened():
        window_handle = cv2.namedWindow("CSI Camera", cv2.WINDOW_AUTOSIZE)
        # Window
        while cv2.getWindowProperty("CSI Camera", 0) >= 0:
            ret_val, img = cap.read()
```

```
            cv2.imshow("CSI Camera", img)
            # This also acts as
            keyCode = cv2.waitKey(30) & 0xFF
            # Stop the program on the ESC key
            if keyCode == 27:
                break
        cap.release()
        cv2.destroyAllWindows()
    else:
        print("Unable to open camera")

if __name__ == "__main__":
    show_camera()
```

STREAM CON USB-CAM

https://github.com/NVIDIA-AI-IOT/jetcam

```
!git clone https://github.com/NVIDIA-AI-IOT/jetcam
!cd jetcam
!sudo python3 setup.py install

from jetcam.usb_camera import USBCamera
camera = USBCamera(capture_device=1)
image = camera.read()
```

En este caso, es necesario tener en cuenta que el programa lo único que hace es llamar al método read contenido en la clase USBCamera, que ya tiene implementados unos valores de resolución por defecto. También tenga en cuenta el capture_device aunque por defecto sea =1, dependiendo del número de dispositivos conectados a su jetbot podría ser otro desde 0 hasta 3. Se adjunta la clase USBCamera de la libreria jetcam para una mejor compresión.

```
https://github.com/NVIDIA-AI-IOT/jetcam/tree/master/jetcam
from .camera import Camera
import atexit
import cv2
import numpy as np
import threading
import traitlets

class USBCamera(Camera):

    capture_fps = traitlets.Integer(default_value=30)
    capture_width = traitlets.Integer(default_value=640)
```

```
capture_height = traitlets.Integer(default_value=480)
capture_device = traitlets.Integer(default_value=0)

def __init__(self, *args, **kwargs):
    super(USBCamera, self).__init__(*args, **kwargs)
    try:
        self.cap = cv2.VideoCapture(self._gst_str(), cv2.CAP_GSTREAMER)

        re , image = self.cap.read()

        if not re:
            raise RuntimeError('Could not read image from camera.')

    except:
        raise RuntimeError(
            'Could not initialize camera.  Please see error trace.')

    atexit.register(self.cap.release)

def _gst_str(self):
        return 'v4l2src device=/dev/video{} ! video/x-raw, width=(int){},
height=(int){}, framerate=(fraction){}/1 ! videoconvert ! video/x-raw,
format=(string)BGR ! appsink'.format(self.capture_device, self.capture_width,
self.capture_height, self.capture_fps)

def _read(self):
    re, image = self.cap.read()
    if re:
        image_resized = cv2.resize(image,(int(self.width),int(self.height)))
        return image_resized
    else:
        raise RuntimeError('Could not read image from camera')
```

Resumen:

• OpenCV es una librería que incluye funciones para el tratamiento de imágenes, y por tanto puede ser empleada en robótica perceptual.
• ROS (Robotic Operation Sistem) es un sistema operativo nodal apropiado para el desarrollo de soluciones robóticas.
• La programación mediante CLI o Teach Pendant es la más habitual, pero también la menos personalizable.
• Las topologías más habituales de robot son cartesianos, multi-DOF, SCARA y DELTA.

Test de evaluación

1. Un robot cartesiano:
    a) Se mueve a lo largo de tres ejes (XYZ) con una trayectoria lineal.
    b) Es de los más baratos según el punto de vista tipológico.
    c) No permite un control de la orientación.
    d) Todas las anteriores son correctas.
2. Un robot Multi-DOF:
    a) Tiene muchos grados de libertad, por ejemplo 5 o 6.
    b) Son muy caros, desde el punto de vista de la tipología.
    c) Puede alcanzar cualquier punto con cualquier orientación.
    d) Todas las anteriores son correctas.

3. Un robot SCARA:
    a) Tiene 3 + 1 ejes.
    b) Hace bien labores de coger y colocar.
    c) Es moderadamente caro o barato.
    d) Todas las anteriores son correctas.
4. Un robot DELTA:
    a) Puede llegar a tener hasta cinco grados de libertad.
    b) Recibe también el nombre de Araña.
    c) Es globalmente caro, desde un punto de vista de la tipología.
    d) Todas las anteriores son correctas.
5. En una unidad JetBot:
    a) Pueden conectarse cámaras y gestionarse con OpenCV.
    b) Pueden conectarse sensores, servos o motores y ser controlados mediante GPIO, I2C u otros protocolos.
    c) Está basado en Ubuntu pero puede emplearse ROS.
    d) Todas las anteriores son correctas.

Actividades

1. Llevar a cabo el montaje de una unidad JetBot completa. En relación al hardware, podrá fabricar el esqueleto (bastidor) a partir de piezas de diseño propio con ayuda de impresión 3D, o acudir a la lista de materiales de JetBot proporcionada por nVidia.

2. Instalar una imagen de Jetbot, o realizar una instalación de un sistema operativo compatible.

3. Conectar un mando tipo consola, como los de la xBOX o similar, y realizar un sencillo programa para hacer que el robot se mueva en la dirección marcada por la cruceta.

Ejercicios opcionales

Ejercicio 1
Clonar este repositorio de GitHub, que corresponde al repositorio oficial de JetBot, en el que se haya el código para visualizar a través de la cámara, o encender o apagar un motor.
https://github.com/NVIDIA-AI-IOT/jetbot
https://github.com/NVIDIA-AI-IOT/jetbot/tree/master/docs/_examples