Universidad de Alicante

Escuela Politécnica Superior

Proyecto Teórico:

Estudio del funcionamiento de los robots Hexápodos

Ingeniería robótica

Robots Móviles

Autores:
Javier Aparicio Ruiz
Pedro Díaz García
Adrián Díaz Vázquez
Israel Hernández Ramírez

Enero 2022

Índice

1. Introducción

Este proyecto consiste en la construcción de un robot hexápodo de 3 grados de libertad en cada pata y una simulación del robot en ROS. Para ello se ha elegido el robot hexápodo de Freenove comercializado por la tienda online Amazon (1), que incluye instrucciones de montaje y códigos de prueba.

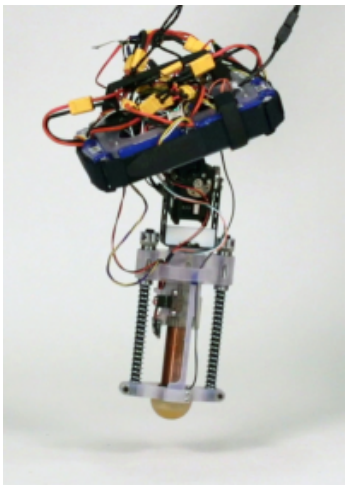
2. Estado del arte

Los robots hexápodos pertenecen al grupo de robots móviles caminantes o con patas. Son más versátiles que los robots con ruedas y pueden atravesar muchos terrenos diferentes, aunque estas ventajas requieren mayor complejidad y consumo de energía. Los robots caminantes se pueden clasificar según su número total de patas, de grados de libertad de las patas y los distintos tipos de marcha o caminar.

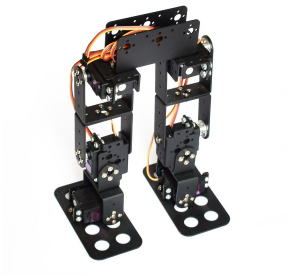
2.1. Clasificaciones de los robots con patas

Los robots con patas se pueden clasificar de diferentes maneras atendiendo a diversos criterios:

- **Según el número de patas:**
  - **Robots de 1 pata o saltador**
    A pesar de la aparente simplicidad de estos robots, resultan muy complejos de implementar. Debido a que no puede utilizar una pata auxiliar de pie de apoyo al caminar, este autómata debe botar para desplazarse como si de un saltador de niños se tratara. Para ello debe calcularse continuamente la dinámica y el punto de aterrizaje, de modo que el robot no se tropiece.
  - **Robots de 2 patas o bípedos**
    Los robots de dos patas se asemejan a la estructura de cualquier ser vivo bípedo, como pueden ser las personas o los canguros. La forma para caminar que se le implementa, suele ser similar a la de las personas, de modo que para dar un paso, el robot mantendrá una pierna fija al suelo. Esto consigue un movimiento más estable, aunque la dinámica resulte más compleja a causa de la necesidad de calcular dos extremos.
  - **Robots de 4 patas o cuadrúpedos**
    Son robots que simulan el caminar de los animales cuadrúpedos. Gracias a este número, el robot se vuelve más estable, pues hay más puntos de contacto sobre el suelo. Esta estabilidad se incrementa bastante a causa del uso del chasis cuasiestáticos. El cálculo de la dinámica es bastante diferente respecto los 2 tipos de robots vistos con anterioridad por que el número de grados de libertad en robots con 4 o más patas incrementa.
  - **Robots de 6 patas o hexápodos**
    Los robots hexápodos están inspirados biológicamente en la locomoción hexápoda. Esto consiste en obtener una alta estabilidad y mantenerla durante el tiempo colocado sus patas de manera que suspenden el cuerpo y conserva el centro de masas bajo. Cabe destacar que la potencia ejercida de un extremo es mayor a velocidades bajas, esto se debe a que a mayor lentitud, mayor es el número de patas en contacto con el suelo, por lo que incrementa la potencia producida.
  - **Robots de 8 o más patas**
    Como se ha podido observar, según se aumenta el numero de patas el movimiento se vuelve mas lento al tener que realizar un mayor numero de movimientos, pero se aumenta la estabilidad al encontrarse mas puntos en contacto con el suelo en cada momento.



(a) Robot de una pata



(b) Robot de 2 patas

(c) Robot de 4 patas

(d) Robot de 6 patas

(e) Robot de 8 patas

Figura 1: Robots con diferentes números de patas

- **Según el número de grados de libertad de las patas:**
  - ◇ **Robots de 1 grado de libertad:** La construcción de estos robots es sencilla ya que las patas serán totalmente rígidas y solo habrá un tipo de movimiento proporcionado por el actuador. La configuración típica de estos robots es hacer que las patas centrales tengan la capacidad de subir y bajar mientras que las de los extremos de moverse hacia adelante o hacia detrás.
  - ◇ **Robots de 2 grados de libertad:** Los robots que poseen este tipos de patas, pueden subir y bajar, además de moverse hacia adelante o hacia detrás. También se pueden inclinar hacia adelante o hacia atrás.
  - ◇ **Robots de 3 grados de libertad o más:** Estos robots pueden hacer lo mismo que los robots de 2 grados de libertad pero mejor ya que pueden inclinarse hacia los lados, no solo hacia adelante o hacia atrás. También pueden subir escaleras, mantenerse en equilibrio ya que pueden mover su centro de gravedad inclinándose hacia los lados, etc. Esto depende de el número de grados de libertad.

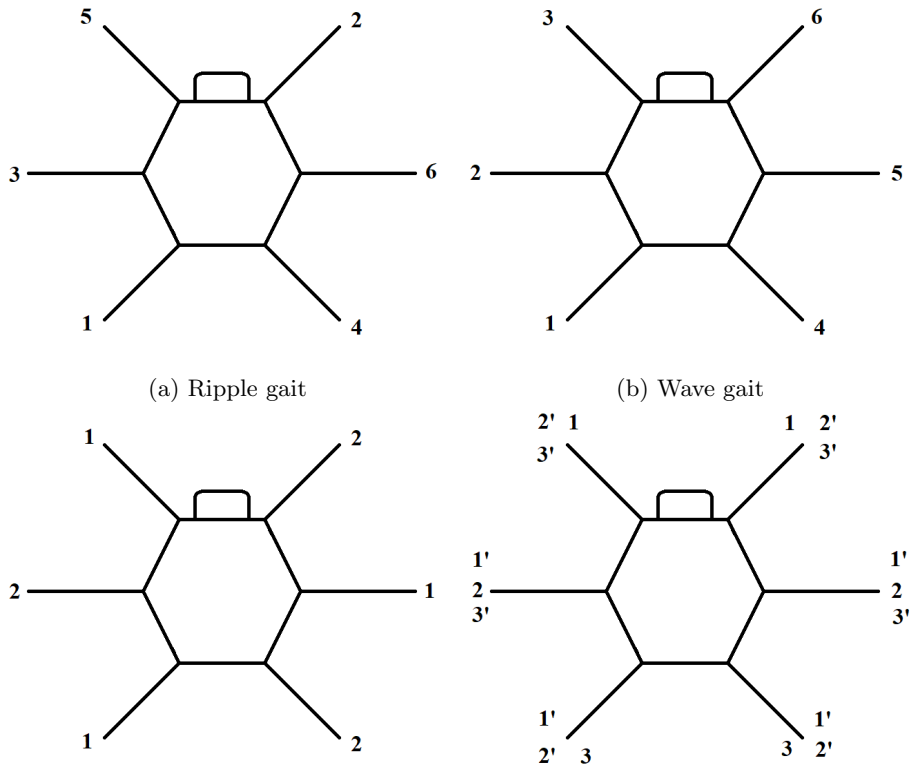
2.2. Tipos de marcha de un robot hexápodo

Los robots hexápodos pueden desplazar sus patas de distintas formas y estos suelen seguir algunos patrones de movimiento relacionado con los insectos. Cada vez que se empieza realizar un movimiento de las patas del robot, por ejemplo mover las patas delanteras de la posición actual hacia delante, es considerado un ciclo. Por lo tanto es común mencionar los ciclos al hablar de las marchas de movimiento de los hexápodos.

- **Marcha ondulada o ripple gait:** En este tipo de marcha se mueven solo dos patas a la vez. El movimiento se inicializa moviendo una pata y cuando esta llega a la mitad del ciclo

se inicia el movimiento de la segunda. Una vez esta pata también llegue a la mitad se iniciará el movimiento de la tercera pata. De esta forma se realizan 2 ciclos a la vez, pero uno de ellos tiene un retardo respecto al otro. El orden a seguir para el movimiento de las patas es el mostrado en el esquema de la Figura 2a.

- **Marcha de onda o wave gait:** En este tipo de movimiento se mueve una sola pata a la vez. El movimiento sigue el orden de la Figura 2b. Dependiendo de la calidad de los motores y el peso del robot, puede ser uno de los tipos de marcha más rápidos.
- **Marcha de tripode o tripod gait:** Junto con la anterior, es el tipo de marcha más utilizada. En este tipo de marcha se mueven 3 patas a la vez y por tanto se realizan 2 ciclos para mover todas las patas. Es el movimiento más lento de los que hemos visto pero también el más estable. Se puede observar el orden en el que se realizan los movimiento en la Figura 2c.
- **Marcha cuadrúpeda o quadruped gait:** La marcha es realizada levantando primero las patas delanteras hasta dejarlas suspendidas en el aire. Entonces las 4 patas restantes se mantienen apoyadas mientras se inclinan hacía la dirección deseada, hasta que se posan las que estaban levantadas. A continuación se repite el mismo proceso con las patas intermedias y posteriormente con las finales. Este tipo de marcha no se observa en insectos pero es relativamente rápida y estable. También es muy utilizada para subir o bajar obstáculos.
- **Marchas libres:** Se pueden encontrar otro tipo de marchas diseñadas específicamente para terrenos especiales mas complejas y difíciles de implementar. No siguen unos ciclos periódicos, si no que el movimiento lo realizan y modifican según las circunstancias del terreno. Dependiendo del objetivo se pueden necesitar más o menos patas, mover cada una a puntos específicos, movimientos mas rápidos o mas lentos, etc.



(a) Ripple gait (b) Wave gait

(c) Tripod gait (d) Quadruped gait

Figura 2: Tipos de marchas de los robots hexápodos

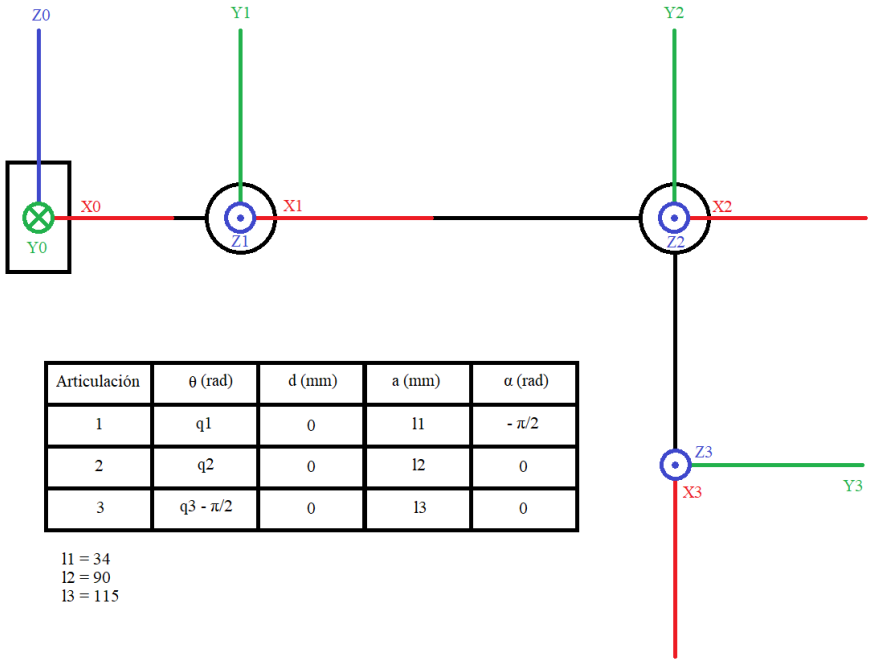
3. Tabla de Denavit–Hartenberg y Cinemáticas del robot

Para obtener la geometría del robot y estudiar su movimiento, se calculan la tabla Denavit-Hartenberg y las cinemáticas directa e inversa. En este caso, se mostrarán los cálculos y representaciones de una sola pata, puesto que podemos hallar las del resto de paras fácilmente usando matrices de transformación con los resultados obtenidos de la primera pata.

La tabla D-H contiene distintos parámetros con el objetivo de describir la estructura cinemática de una cadena articulada constituida por articulaciones de un solo grado de libertad. Para ello asigna un sistema de referencia a cada articulación, donde el origen de coordenadas se encuentra sobre esta y el eje Z sobre el eje de acción de la articulación.

Por otro lado, se ha hallado el resultado de las cinemáticas directa e inversa. La directa consiste en obtener las coordenadas en el mundo (X, Y, Z) del extremo final del último eslabón. Estas coordenadas dependen directamente del accionamiento de las articulaciones y de las medidas de los eslabones. La inversa, en cambio, es al contrario, a través de una coordenadas en el mundo se calculan el movimiento o giro de las articulaciones a realizar para colocar el extremo en esa posición.

A continuación se muestra una imagen con la representación Denavit-Hartenberg, y 6 funciones, 3 para calcular la cinemática directa, y 3 para la inversa.



Articulación	θ (rad)	d (mm)	a (mm)	α (rad)
1	q1	0	l1	- π/2
2	q2	0	l2	0
3	q3 - π/2	0	l3	0

l1 = 34
l2 = 90
l3 = 115

Figura 3: Representación Denavit-Hartenberg (Tabla y Sistemas de Referencia)

$$
\begin{aligned}
x &= (l_1 + l_2 \cdot cos(q_2) + l_3 \cdot cos(q_2 + q_3)) \cdot cos(q_1) \\
y &= (l_1 + l_2 \cdot cos(q_2) + l_3 \cdot cos(q_2 + q_3)) \cdot sen(q_1) \\
z &= l_1 + l_2 \cdot sen(q_2) + l_3 \cdot sen(q_2 + q_3) \\
q_1 &= arctan(\frac{y}{x}) \\
q_2 &= \frac{\pi}{2} - arctan(\frac{L}{z}) + arctan(\frac{l_3 \cdot sen(q_3)}{l_2 + l_3 \cdot cos(q_3)}) \\
q_3 &= arcocos(\frac{L^2 + z^2 - l_2^2 - l_3^2}{2 \cdot l_2 \cdot l_3})
\end{aligned}
$$

Los parámetros l_1, l_2 y l_3 hacen referencia a la longitud de cada eslabón mientras que la variable L se refiere a la longitud de la pata vista desde encima. Los valores se muestran a continuación.

$$
\begin{aligned}
l_1 &= 34(mm) \\
l_2 &= 90(mm) \\
l_3 &= 115(mm) \\
L &= \frac{x}{cos(arctan(\frac{y}{x}))}
\end{aligned}
$$

4. Construcción del robot

Como se comentó en apartados anteriores, el robot se vende despiezado junto con una guía de montaje. Además de los componentes incluidos en la caja, también se requiere una Raspberry Pi 3 o superior para conectarla al robot e implementar en ella ROS. En este caso, se ha usado una Raspberry Pi 4 B. También se han usado 2 conjuntos de 2 baterías recargables del modelo 18650 de 3000 mAh con tetón plano y un tetón normal.

4.1. Pasos para la construcción

Los pasos para la construcción del robot hexápodo son los siguientes:

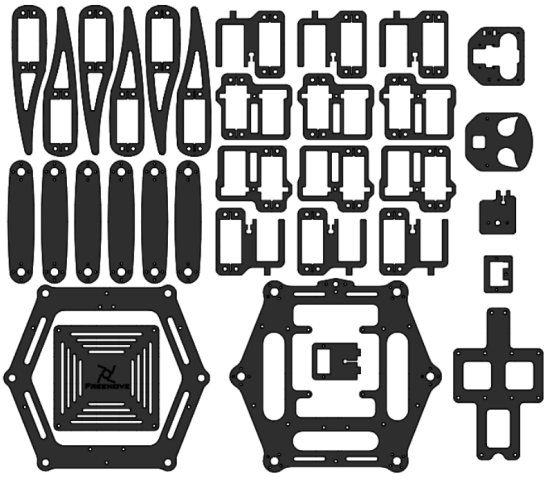
1. **Preparación de la placa**

   Para preparar la Raspberry Pi se ha utilizado el programa VNC, que permite usar la interfaz del sistema operativo al que se ha querido conectar. Una vez enlazado, se prepara el entorno para vincularse mediante ssh al robot y se instalan las dependencias proporcionadas por el tutorial para la correcta calibración del robot.

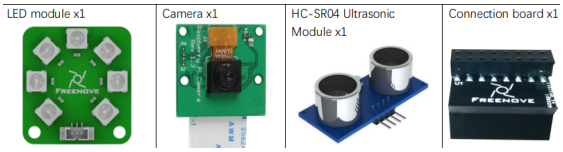
2. **Hardware**

   El hardware del robot está formado por los siguientes componentes:

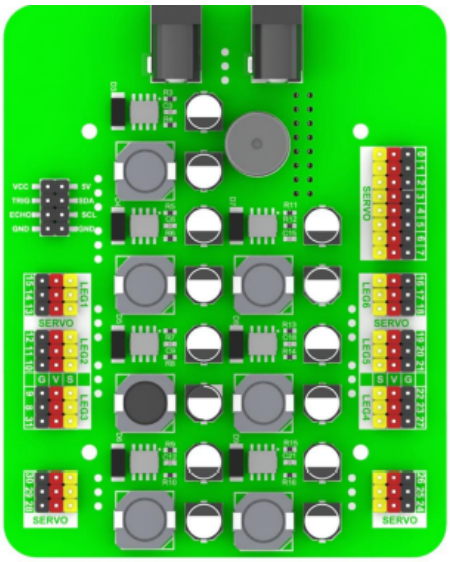
   - Spider Shield para Raspberry Pi
   - MG996R servo
   - S90 servo
   - Partes del robot, que son acrilicas
   - LED module
   - Sensor Ov5647 de 1080p (Cámara)
   - Sensor de ultrasonidos HC-SR04
   - Modulo MPU6050 (Giroscopio)
   - Placa de conexión Freenove



(a) Piezas del robot



(b) Sensores



(c) Spider Shield

Figura 4: Componentes hardware del robot

La alimentación del robot se ha realizado mediante 2 paquetes de 2 baterías 18650 de 3000mAh. Estos packs se han colocado en el tórax, que es la posición habitual, aunque no la única. Existen otros modelos de este tipo de robots con diferente configuración de las baterías. Un ejemplo sería colocar una batería en cada pata, con el objetivo de lograr una mayor eficiencia a la hora de moverse y una disminución del tamaño del robot.

Los servos utilizados son los modelos S90 y MG996R, que se encargan del giro horizontal y vertical de la cabeza del robot y de las articulaciones de las patas, respectivamente. Dichos servos no poseen la característica de obtener una realimentación del giro como los motores que tiene integrados encoders. Esto es un gran defecto en caso de querer realizar un control sobre el movimiento, implementando por ejemplo un controlador PID. Al desconocer la posición, no se puede saber a través del motor si ha llegado al giro destino o no, por lo que tampoco se puede hacer la realimentación con el error entre la posición deseada y la real.

Se han utilizados estos servos porque venían en la caja del propio robot. Por lo que en cualquier momento, se podrían comprar unos servomotores bastante mejores y con capacidad de realimentación. Esto no se ha hecho debido a que tienen un precio bastante elevado.

3. Ensamblaje

En la construcción del robot se han seguido todos los pasos del tutorial, que se puede descargar en Amazon (2), que se muestran detalladamente en el capitulo 2. El resultado final obtenido se puede observar en la Figura 5.

Figura 5: Robot construido

4. **Calibración del hexápodo**

   Para la calibración se han seguido nuevamente los pasos del tutorial especificados en el capitulo 3. Para ello, se ha conectado al robot vía NVC y se han ejecutado los programas para calibrar las diferentes partes del robot. Una vez ejecutados correctamente todos los archivos, se han realizado las pruebas necesarias para verificar todas las calibraciones realizadas, hasta determinar el correcto funcionamiento del robot.

4.2. Problemas con el hardware del robot

Durante el procedimiento de construcción del robot móvil hexápodo, se produjeron varios problemas que ralentizaron el proceso.

El primer contratiempo fue debido a que las baterías eran de una longitud menor al hueco donde se coloca cada una. Para intentar compensar la diferencia de tamaño, se colocó papel de aluminio entre el contacto de la pila y la caja. Como era de esperar, después de unas pocas pruebas, el robot no funcionaba correctamente debido la falta de suministro de energía causado por las pérdidas de electricidad producidas en el aluminio. Finalmente la caja de suministro de energía se quemó, por lo que se tuvo que pedir otra caja a la compañía y esperar varias semanas a que llegara.

Otro imprevisto fue el no posicionar los servomotores en el estado inicial. El problema radicó en que al tener servomotores y no motores corrientes, si se pide a uno de estos girar hasta un ángulo demasiado grande, superando el rango establecido por los fabricantes, esto provoca que el servo no responda o que se sobrecaliente. Este obstáculo dentro de la construcción, se originó debido a que los motores no estaban la posición cero desde un principio, sino que algunos ya estaban girados. La solución fue bastante sencilla, se tenía que ejecutar el fichero Servo.py antes de realizar ningún movimiento para que colocara a todos los servos en su posición inicial.

5. Código de prueba

El código de prueba proporcionado por el fabricante esta realizado en Ensamblador y Python 3. Este programa incluye los ciclos de movimientos de trípode alterno y ciclo de onda. Además de otras

características como inclinar el robot, elevar o bajar la altura del robot, auto balance del robot, etc. También realiza la conexión mediante ssh del robot, la activación de la cámara y reconocimiento de caras.

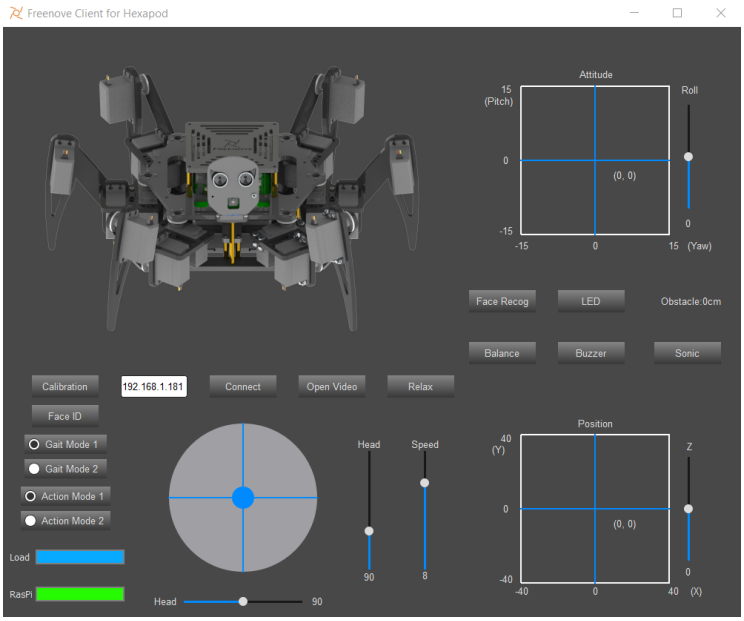


Figura 6: Interfaz de la aplicación

5.1. Código del servidor

El código para crear un servidor donde se conecte el robot está divido en los siguientes archivos en Python:

- **ADS7830.py**
  Este fichero se encarga de obtener el estado de la batería del robot como el porcentaje de batería restante y el voltaje actual de ambas baterías.

- **Buzzer.py**
  Realiza la conexión con un *buzzer* que es un dispositivo capaz de generar una señal de audio cuando se le alimenta con cierto voltaje.

- **Command.py**
  Inicializa una serie de comandos que realizan distintas acciones.

- **Control.py**
  Este fichero es utilizado para calcular la cinemática directa e inversa del robot. Además de cambiar el tipo de movimiento del robot, también da las instrucciones de movimiento de cada pata y autobalancear el robot para mantener el centro de gravedad en su sitio mediante la función PostureBalance.

- **IMU.py**
  Inicializa los valores de acelerómetro y giroscopio y crea un filtro medio, que realiza la media de 100 datos obtenidos con los sensores de movimiento y orientación. También se encarga de actualizar las variables articulares, aceleraciones y ángulos.

- **Kalman.py**
  Implementa un filtro de Kalman como los que se han visto en clase.

- **Led.py**
  Define distintos modos de iluminación para el módulo LED que está incorporado en la parte trasera del hexápodo y determina el modo en que se ilumina. Algunos de los modos son como un foco de teatro que haga un seguimiento o un arco iris distribuido uniformemente en todos los píxeles.

- **main.py**
  Es el archivo main, muestra una ventana en QT y crea un botón de encender/apagar el servidor.

- **PID.py**
  Es un fichero que contine las funciones necesarias para calcular el PID del robot.

- **Server.py**
  Es uno de los ficheros principales en el funcionamiento del servidor. Contiene las funciones para encender y apagar el servidor, de resetearlo si da algún problema, recibir los datos del cliente o enviárselos, realizar la retransmisión de vídeo y recibir las instrucciones para encender, apagar o tomar datos de sensores.

- **ui_server.py**
  Este programa se encarga de crear una interfaz en QT para el servidor en la cual solo se puede pulsa un botón para encender o apagar el servidor.

5.2. Código del cliente

El código para crear un servidor donde se conecte el robot está divido en los siguientes archivos en Python:

- **Calibration.py**
  Permite realizar las calibraciones de los parámetros iniciales del hexápodo.

- **Client.py**
  Es uno de los ficheros principales en el funcionamiento del cliente. Contiene las funciones para encender y apagar el cliente, recibir los datos del servidor o enviárselos y recibir la retransmisión de vídeo y comprobar si es valido.

- **Command.py**
  Realiza la misma función que su equivalente en el código del servidor.

- **Face.py**
  Encargado de detectar caras humanas en las imágenes recibidas.

- **main.py**
  Muestra una ventana en QT en la que se muestran todos los controles disponibles del robot hexápodo.

- **PID.py**
  Realiza la misma función que su equivalente en el código del servidor.

- **Thread.py**
  Realiza la misma función que su equivalente en el código del servidor.

- **ui_client.py**
  Este programa se encarga de crear una interfaz en QT para el cliente en la cual se encuentran todos los controles disponibles del hexápodo.

- **ui_face.py**
  Este programa se encarga de crear una interfaz en QT para mostrar las caras detectadas para poder seleccionarlas.

- **ui_led.py**
  Este programa se encarga de crear una interfaz en QT para seleccionar el color del los LEDs.

5.3. Vídeo de demostración

En el vídeo a continuación se puede observar al robot moviendose por una habitación mientras se utiliza la aplicación del cliente: `https://youtu.be/aFQoScPYtn0`

6. Uso de ROS para controlar un robot hexápodo

Como se ha visto a lo largo de la carrera y también en esta asignatura ROS es un workspace ampliamente utilizado que añade multitud de herramientas para así programar y controlar sistemas robóticos de forma rápida y efectiva. Es por este motivo por el que se ha decidido estudiar la implementación de un robot hexápodo con ROS.

A la hora de desarrollar el entorno de control de dicho hexápodo con ROS se realizó en primera instancia una búsqueda en profundidad de todos los proyectos del mismo estilo publicados en internet y más en concreto en GitHub. Tras estar analizando varios proyectos y al ver que no había ninguno que controlara al hexápodo con éxito se escogieron los dos siguientes:

- **ROS project for a hexapod robot (6):** Este proyecto desarrollado por Peter Leng, trató de controlar el mismo hexápodo que se ha empleado en esta práctica, pero sin éxito aparente, ya que se trata de un proyecto aún en desarrollo. Sin embargo el proyecto le pareció muy interesante a los integrantes del grupo, que mas adelante entrarían en contacto con Peter Lang, para desarrollarlo en conjunto con él.

- **ROS package providing Gazebo simulation of the Phantom X Hexapod (7):** Implementación en Gazebo del famoso robot hexápodo Phantom X.

A continuación se mostrará la estructura de paquetes diseñada para el hexápodo "freenove big Hexapod", además se explicará sus pros y sus contras y se estudiará si realmente es conveniente sobrepasar la curva de dificultad que tiene programar un sistema robótico de estás características en ROS.

6.1. Estructura de paquetes comunes para control simulado y real de un robot hexápodo

Usualmente a la hora de diseñar la estructura de paquetes de ROS se suele llevar un orden general, para así facilitar su lectura y aportar un nivel de orden adecuado a los proyectos. De esta forma se llega al esquema siguiente:

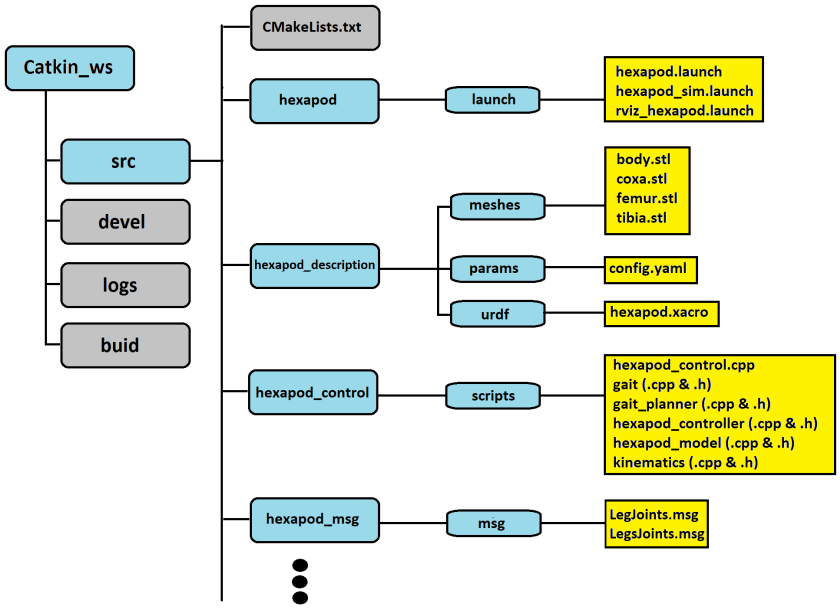


Figura 7: Listado de paquetes comunes para control simulado y real de un robot hexápodo

En este esquema se pueden observar los siguientes paquetes:

- **hexapod :** Este es el principal paquete del proyecto, siendo el que se encarga de llamar con .launch a todos los archivos y demás .launch de cada paquete. En este paquete nos podemos encontrar los siguientes .launch:
  - ◇ hexapod.launch : Se encarga de lanzar todos los paquetes mostrados en la figura 7, además de los necesarios para controlar el robot real.
  - ◇ hexapod_sim.launch : Se encarga de lanzar todos los paquetes mostrados en la figura 7, además de los necesarios para controlar el robot en un entorno simulado.
  - ◇ rviz_hexapod.launch : Este fichero lanza el entorno de visualización de Rviz.
- **hexapod_description :** en este paquete se introduce todo lo relativo al diseño 3D del robot. En la carpeta meshes se añaden los diseños stl de cada eslabón y el cuerpo, mientras que en la carpeta params se introduce un config.yaml donde aparecen todas las constantes fisicas del robot, así como distancia de los eslabones, peso, limites articulares, etc.De esta forma, empleando toda esta información en la carpeta urdf, se diseña el "esqueleto" del robot, codificado en URDF.

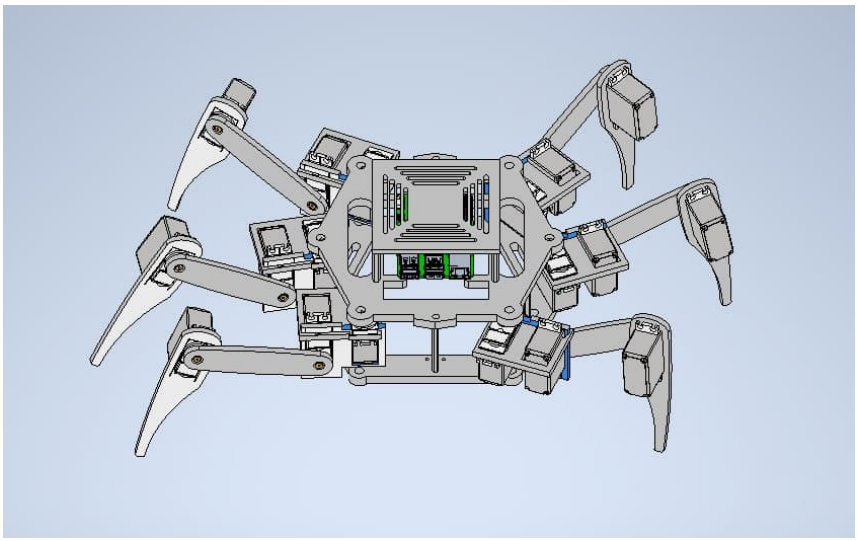


Figura 8: Diseño 3D realizado por un integrante del grupo de prácticas con la aplicación Inventor3D

- **hexapod_control :** En esta otra carpeta se introduce todo el código necesario (programado en C++) necesario para que el robot se mueva, que añadiendo en el CMakeLists.txt un ejecutable se consigue ejecutar con un rosrun, o un .launch. A continuación se explicará brevemente que tarea realiza cada código:
  - ◇ hexapod_control : Este es el fichero main(), donde se ejecuta todo lo necesario para que el robot se mueva dependiendo de su estado objetivo (pararse, andar, sentarse o levantarse).
  - ◇ gait : Definen los tipos de formas de andar que puede tener el robot, implementándose casi en su totalidad las expuestas en el estado del arte.
  - ◇ gait_planner : Realiza todo lo relativo al planificador de trayectorias en cada frame de refresco.
  - ◇ hexapod_controller : Este código realiza el estudio de la maquina de estados del robot, así como la obtención de información articular para así enviarsela al planificador de trayectorias.
  - ◇ hexapod_model : Define la clase hexápodo, la cual al instanciarse recibe toda la información física necesaria para calcular la cinemática del robot, además de esto, define todos los métodos necesarios para emplear esta información (getters y setters).
  - ◇ kinematics : Este fichero implementa la cinemática inversa necesaria para mover cada cadena cinemática de una posición inicial a un punto deseado.
- **hexapod_msg :** Este último paquete define dos tipos de mensajes diferentes, un LegJoint.msg el cual comprende a tres float64 llamados coxa, femur y tibia (articulaciones ordenadas de más cercanas al cuerpo a menos), mientras que el mensaje LegsJoints.msg contiene 6 mensajes de tipo LegJoint.msg.

6.1.1. Simulación

Para lanzar el robot en un entorno simulado se deberá lanzar el paquete hexapod_gazebo, el cual se encargará de preparar todo el entorno se ROS para que se pueda controlar de forma correcta la simulación de Gazebo. Dicho paquete tiene la siguiente estructura de carpetas que harán lo siguiente:

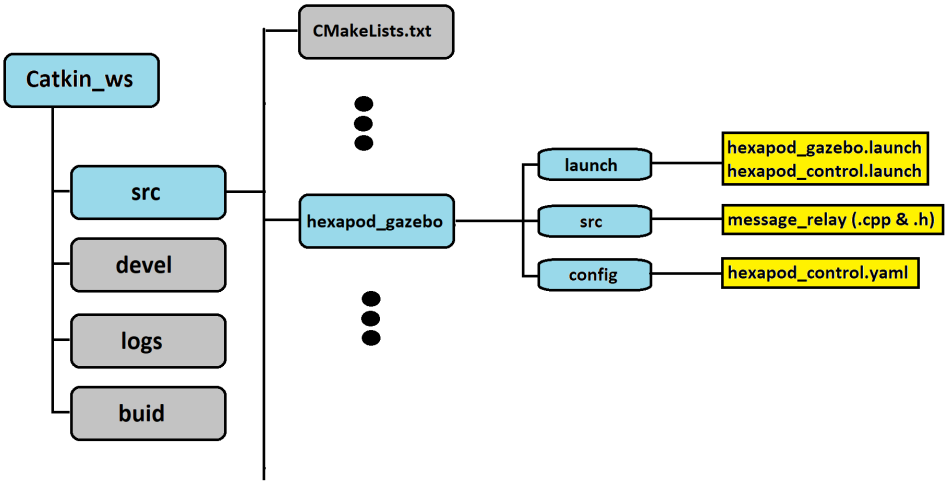


Figura 9: Paquete para realizar una simulación del hexápodo

- **launch :** En esta carpeta se almacenará el .launch encargado de abrir el entorno de Gazebo con el mundo seleccionado y el robot cargado "hexapod_gazebo.launch", además del encargado de cargar los grupos articulares, controladores de cada articulación y la herramienta TF, para así poder referenciar cada parte del robot de forma adecuada .
- **src :** En esta carpeta se almacenará el código message_relay, el cual se encargará básicamente de transladar la información de los tópics creados por los paquetes anteriores a los topics de los controladores de cada articulación. Este código se podrá enviar con un rosrun o un .launch como es de costumbre.
- **config :** En este fichero "hexapod_control.yaml" se define los grupos articulares y sus correspondientes controladores, así como el control PID ejercido.

6.1.2. Control del robot real

Para poder controlar el robot real con ROS se ha creado el siguiente paquete:

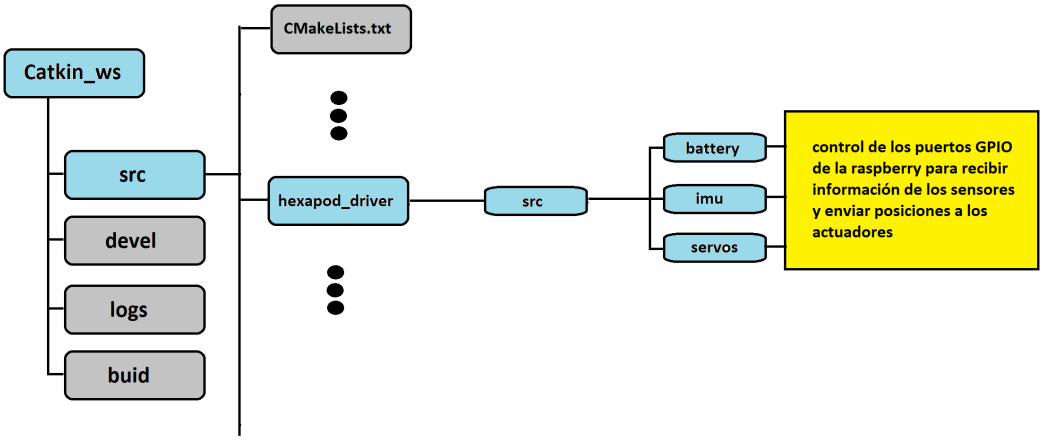


Figura 10: Paquete para realizar un control del hexápodo real

Como muestra la imagen, el robot tendrá un directorio por cada elemento del Hardware que

se quiera controlar o aporte información del entorno. En este caso solo se tendrá como sensores el valor del nivel de la batería que ofrece el hardware del hexápodo "freenove big hexapod"y el IMU instalado, por lo que como se muestra en la imagen habrá 2 directorios para sensores y uno para actuadores. Este último se encargará de transmitir a los servos la información publicada en el topic correspondiente a la posición de cada articulación generado y actualizado por el planificador de trayectorias.

6.2. Experimentación

Con el proyecto diseñado se intentó probar su funcionamiento tanto en el robot real como en el entorno de simulación de gazebo, sin embargo lamentablemente no se consiguió mover de forma correcta el robot en ninguno de los casos. A continuación se mostrará todo el desarrollo realizado por el grupo y todo lo que se ha aprendido a lo largo de este trabajo teórico-práctico.

6.2.1. Prueba con la simulación

Los dos proyectos que se han utilizado como guía simulaban correctamente un robot hexápodo, no obstante, aunque el proyecto de Peter Leng intentaba realizar un control del "freenove big hexapod" el hexápodo diseñado en su URDF no correspondía con este, por lo tanto, no se podía simular a la perfección su funcionamiento.

Por este motivo se decidió diseñar en 3D este robot, mostrado en la figura 8. Una vez hecho el diseño se midió el peso de cada grupo articular body(cuerpo central), coxa(prmer bloque que comprende a las dos primeras articulaciones), fémur(el eslabón intermedio) y tibia(pata). Y tras haber calculado las inercias de cada una de estas piezas se realizó el .xacro que define el hexápodo en URDF. A continuación se muestra el resultado del robot en gazebo comparado con la simulación empleada por el proyecto de Peter Leng:

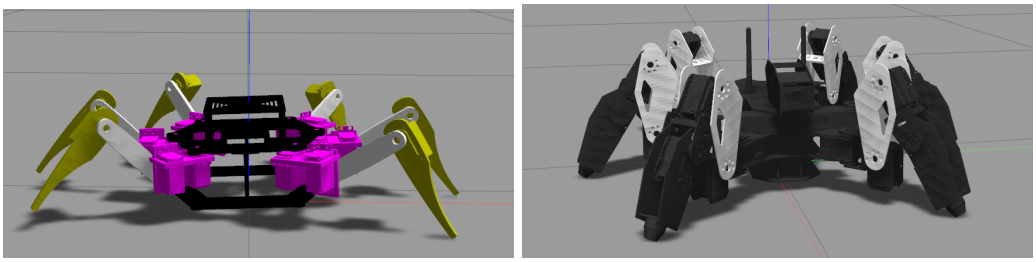


Figura 11: simulación en gazebo del hexápodo freenove y el empleado por Peter Leng

Tras realizar este diseño se pudo controlar manualmente cada articulación mediante *rostopic pub*, sin embargo a la hora de probar el funcionamiento del planificador de trayectorias el movimiento realizado se ha calificado como satisfactorio. De todas formas se considera satisfactorio el resultado obtenido, pues con el acceso al control de cada articulación solo faltaría refinar este último código para así conseguir mover el hexápodo tanto en simulación como en la vida real.

6.2.2. Instalación de ROS en Raspberry

Para probar diferentes paquetes de ROS en el hexápodo se tuvo que instalar este sistema en la Raspberry. Dicha instalación se repitió varias veces para distintas versiones de ROS debido a que

surgían diversos problemas que se explican a continuación.

En primer lugar se probó a instalar en la Raspberry el sistema operativo Ubuntu x64, que es el sistema usual a instalar, puesto que cuenta con una arquitectura de 64 bits y los recursos limitados de esta configuración, aportan una mayor agilidad de cálculo. Sin embargo a la hora de descargar los paquetes necesarios, este distro empezó a dar problemas con diferentes librerías en concreto con "mmal libraries". Tras estar investigando se observó que para el control de los puertos del microprocesador y librerías propias de Raspbian se necesitaba una arquitectura de 32 bits, por lo que se optó por instalar ROS en Raspbian, en concreto en Raspbian Buster (una distribución más eficiente para proyectos de este estilo).

6.2.3. Prueba con el robot real

Tras estar modificando el código base del proyecto "ROS project for a hexapod robot" para hacer corresponder de forma correcta los GPIO de la Raspberry con cada servo se consiguió mover las articulaciones del robot con el comando *rostopic pub*. A pesar de esto, no se consiguió un movimiento del robot por los problemas comentados en el apartado 6.2.1.

6.3. Conclusiones de la experimentación

Las sensaciones de este proyecto podrían definirse como agridulces, debido a diferentes complicaciones explicadas en esta memoria no se ha podido realizar un movimiento acertado en ROS del robot elegido. Sin embargo, se ha avanzado considerablemente, siendo la corrección de fallos del código referente al planificador de trayectorias y su refinamiento, el último paso que faltaría por realizar. Una vez completado este paso, se podría controlar de forma autónoma o teleoperar un robot hexápodo real o simulado en ROS.

Además todo el desarrollo referente al urdf realizado del robot, así como la estructura de paquetes empleada ha sido muy útil para ver en primera instancia un trabajo real que controle un sistema robótico en ROS.

7. Carpeta en Drive

En el siguiente enlace se encuentra una carpeta en Google Drive con el vídeo de la presentación, todos los archivos creados y utilizados, y una copia de esta memoria.

8. Referencias

[1] Enlace del robot a la tienda online Amazon:
`https://www.amazon.es/Freenove-Big-Hexapod-Robot-Kit-Raspberry-Pi-Balancing-Recognition-Ultrasonic/dp/B08M5DXS2P/ref=sr_1_8?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=3KPE9OIZ1HHRK&keywords=freenove&qid=1640883057&sprefix=freenove%2Caps%2C75&sr=8-8`

[2] Enlace de descarga del código de prueba:
`https://git.io/JTtZ5`

[3] Pagina web con información sobre los tipos de marcha: `https://hexapodrobots.weebly.com/types-of-robot-gait.html`

[4] Vídeo con información sobre los tipos de marcha: `https://www.youtube.com/watch?v=JwHKT2Fx7DQ`

[5] Enlace con información Denavit-Hartenberg: https://personal.us.es/jcortes/Material/Material_archivos/Articulos%20PDF/RepresentDH.pdf

[6] Enlace al repositorio de GitHub al proyecto "ROS project for a hexapod robot" de Peter Leng:https://github.com/PeterL328/hexapod

[7] Enlace al repositorio de GitHub al proyecto "ROS package providing Gazebo simulation of the Phantom X Hexapod robot.":https://github.com/HumaRobotics/phantomx_gazebo