



Escuela
Politécnica
Superior

Diseño, construcción y control de un robot hexápodo



Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:

Javier Aparicio Ruiz

Tutor/es:

Jorge Pomares Baeza



Universitat d'Alacant
Universidad de Alicante

Diseño, construcción y control de un robot hexápodo

Autor

Javier Aparicio Ruiz

Tutor/es

Jorge Pomares Baeza

Departamento de Física, Ingeniería de Sistemas y Teoría de la Señal



Grado en Ingeniería Robótica



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Octubre 2022

Preámbulo

El trabajo de fin de grado representa en primera instancia todo el bagaje realizado durante estos cuatro años. Un espejo que reflecta todo el esfuerzo, tiempo y dedicación, así como la madurez, versatilidad y conocimientos adquiridos. Por este motivo decidí en primera instancia diseñar, construir y programar un sistema robótico real, pudiendo emplear el máximo posible de herramientas que nos ha ofrecido esta titulación, cumplir un sueño y lo que mas satisfecho y proyección personal me produce: tener la habilidad de crear con mis propias manos e intelecto, algo que mejore la vida de la humanidad y haga prosperar a la sociedad.

La elección de la creación de un robot móvil ha venido motivada por mi actual atracción por todo lo relacionado con el control de robots, que a mi parecer resulta de lo mas bello dentro del amplio campo de la robótica. Además de la posibilidad de realizar investigación en un futuro con aprendizaje por refuerzo.

Por estos motivos y más los que se expondrán más adelante, el objetivo de este proyecto será el diseño, construcción, programación y control de un robot hexápodo.

*No siempre puedes controlar el viento
pero puedes controlar tus velas*

Dr Robert Chope.

Índice general

1	Introducción	1
1.1	Objetivos	2
1.2	Documentación	3
2	Marco Teórico	5
2.1	Robótica Bio-inspirada	6
2.1.1	Bio-locomoción	7
2.1.2	Locomoción reptante	7
2.1.3	Locomoción bípeda	9
2.1.3.1	Zero-Moment Point (ZMP)	9
2.1.4	Locomoción cuadrúpeda	10
2.1.5	Locomoción hexápoda	12
2.1.5.1	Criterios de estabilidad de un modelo hexápodo	13
2.1.5.2	Estados de estabilidad	14
2.1.5.3	Tipos de marcha hexápoda	14
2.2	Ejemplos de robots hexápodos actuales	17
2.2.1	Elección del tipo de motor	18
2.2.1.1	Motores paso a paso	18
2.2.1.2	Servomotores	19
2.2.1.3	Motores usualmente empleados en robots hexápodos .	21
2.2.2	Robots hexápodos según su configuración	21
2.2.2.1	Robots hexápodos Con servomotores analógicos	21

2.2.2.2	Robots hexápodos con servomotores digitales	22
2.2.3	Hardware usualmente empleado en robots hexápodos	24
3	Metodología	27
3.1	Hardware finalmente empleado	27
3.1.1	Motores utilizados	27
3.1.1.1	Dynamixel technology	27
3.1.1.2	Motores XL430-W250 y 2XL430-W250	31
3.1.1.3	Driver utilizado	32
3.1.1.4	Dinamixel Wizzard	36
3.1.2	Raspberry Pi como PC integrado	36
3.2	Entornos de programación y Software	37
3.2.1	ROS	37
3.2.1.1	Filosofía de ROS	38
3.2.1.2	Diseño interno	38
3.2.2	Ubuntu Server	44
3.2.2.1	herramientas nativas de Ubuntu	45
3.3	Herramientas empleadas	46
3.3.1	Aplicaciones de modelado 3D	47
3.3.2	Robotic toolbox Peter Corke (Python)	47
3.3.3	git	48
3.3.4	Wekan	48
3.3.5	MeshLab	48
3.3.6	Impresión 3D	49
4	Desarrollo	51
4.1	Fase 1: Diseño y construcción del robot hexápodo	51
4.1.0.1	Diseño de las patas de un robot hexápodo	52
4.1.0.2	Diseño del cuerpo	53

4.1.0.3	Diseño de las patas	53
4.1.0.4	Unión de las articulaciones con los ejes motores	54
4.1.0.5	Diseño final	55
4.1.1	Fase 2: Conexionado del robot hexápodo	56
4.1.1.1	Cálculo de las baterías	57
4.1.1.2	Montaje final de las baterías	58
4.1.2	Fase 3: desarrollo del software del robot hexápodo	58
4.1.2.1	Conexión de nodos de ROS mediante SSH	59
4.1.2.2	Esquema de control y ejecución del robot	59
4.1.2.3	Conexión entre el driver U2D2, raspberry y enlace con ROS	61
4.1.2.4	Distribución de paquetes usualmente empleada para sistemas robóticos en ROS	62
4.1.2.5	Problemática encontrada para utilizar los GPIOS de la Raspberry con ROS	64
4.1.2.6	Utilización de servicios con la herramienta systemctl .	65
4.1.3	Fase 4: Control del robot	65
4.1.3.1	Cinemática del robot	65
4.1.3.2	Diseño de la trayectoria de un movimiento usual	68
4.1.3.3	Odometría	70
4.1.3.4	Diferentes tipos de control según el modo operativo de los motores	70
4.1.3.5	Ajuste del baudrate y demás configuraciones de los motores	73
5	Resultados	75
6	Conclusiones	77

Índice de figuras

2.1	Pros y contras de un robot con patas en contraposición con uno dotado de ruedas [2].	5
2.2	Potencia y velocidad alcanzable según su configuración [1].	6
2.3	Tipos de locomoción biológica [1].	8
2.4	Desarrollo de robots bípedos por Honda Honda Motor Company (autor desconocido).	9
2.5	Ilustración del ZMP [19].	10
2.6	Partes mecánicas del robot Big Dog creado por "Boston Dynamics" (autor desconocido).	11
2.7	Estabilidad estática de un hexápedo según el polígono de soportes. Fuente:[16].	13
2.8	fases de la marcha [14].	15
2.9	fases de la marcha de power and return stroke [17].	16
2.10	Tres tipos de marcha hexápoda más comunes [12].	17
2.11	servomotor analógico (autor desconocido).	20
2.12	Hexápedo Frenove (fuente propia).	22
2.13	Hexápodos de la familia zenta (autor desconocido).	23
3.1	conexionado del tipo daisy-chain (autor desconocido).	27
3.2	Controlador interno del modo de posición de los motores dynamixel [7].	29
3.3	Controlador interno del modo de velocidad de los motores dynamixel [7].	29
3.4	Trayectoria realizada según el perfil de velocidad definido [7].	30

3.5	Controlador interno del modo de velocidad de los motores dynamixel [7].	31
3.6	Motores finalmente escogidos [6] [7]	32
3.7	Controlador Arbotix (autor desconocido).	33
3.8	Microcontrolador OpenCM junto a su expansion board (autor desconocido).	34
3.9	Controlador U2D2 (autor desconocido).	35
3.10	Raspberry pi modelo B (autor desconocido).	37
3.11	Aportaciones que ofrece ROS (autor desconocido).	38
3.12	Esquema de plumbing de ROS realizado con la herramienta RQT (fuente propia).	39
3.13	Visualización del robot diseñado con la herramienta Rviz (fuente propia).	41
3.14	Visualización de los frames representados con Rviz del robot diseñado (fuente propia).	43
3.15	Captura de la herramienta Wekan (autor desconocido).	49
3.16	Imagen de la impresora utilizada para realizar la impresión de todas las piezas (fuente propia).	50
4.1	Forma típica de un robot hexápodo [9]	51
4.2	Pata bio-inspirada en contraposición con el diseño final de la pata, con sus motores (imagen Nº 1 [9], imagen Nº2 fuente propia).	52
4.3	Montaje del cuerpo del hexápodo más la tapa que encapsula el controlador (fuente propia).	53
4.4	Diseño de la pata mas el anclaje del botón (fuente propia)	54
4.5	Diseño acople de los ejes con su respectivo eslabón (fuente propia).	54
4.6	Diseño final del hexápodo mostrado en la aplicación Autodesk Inventor (fuente propia).	55
4.7	Primer y último diseño enfrentados (fuente propia).	56
4.8	Hexápodo finalmente montado con la circuitería conectada. (fuente propia).	58

4.9 Código de ejemplo en python para activar un motor (fuente propia)	62
4.10 brazo robótico representado de las extremidades del hexápedo (autor desconocido).	66
4.11 Tabla D-H creada para el cálculo de la cinemática directa e inversa (fuente propia).	67
4.12 Imagen de la representación del sistema creado aportada por el plot de la robotic toolbox (fuente propia).	68
4.13 Explicación de la obtención del punto objetivo según la dirección de movimiento (fuente propia).	69
4.14 Giro sobre sí mismo (fuente propia).	69
4.15 Definición del punto intermedio con una altura llamada hop (fuente propia).	70
4.16 Transparencia de la asignatura de control de robots donde se muestra un bucle de control para un controlador cinemático articular (transparencias de la asignatura de control de robots de el grado de Ingeniería robótica de Alicante).	71
5.1 Imagen del robot H_robix en funcionamiento con la visualización en tiempo real mostrada en un ordenador remoto (fuente propia).	75

1 Introducción

El desarrollo de la robótica ha supuesto uno de los mayores avances de la historia en la industria, siendo los brazos robóticos o robots manipuladores las máquinas mayormente presentes en procesos industriales automatizados. Sin embargo, antiguamente debido a multitud de limitaciones respecto a hardware, software, técnicas de control, etc. Las estaciones robóticas se diseñaban para una única tarea, siendo estáticas y poco flexibles. No obstante, a día de hoy se está empezando a diseñar y programar brazos robóticos con movilidad, debido a la incipiente 4 revolución industrial (industria 4.0), que requiere cada vez mayor flexibilidad, autonomía e independencia, siendo clave la utilización de robots con capacidad de movilidad.

Por otro lado, la robótica de servicios ha evolucionado de forma considerable estos últimos años, empezándose a implantar y normalizar en nuestra sociedad, siendo en este campo mayormente indispensable la capacidad de interactuar y relacionarse con el entorno.

Estos y muchos otros ejemplos denotan la naciente necesidad de dotar a las máquinas de movilidad precisando una locomoción adaptada a su tarea y entorno para poder desplazarse e interactuar en él sin ningún problema. Esto es indispensable ya que la movilidad en un sistema robótico en la gran mayoría de los casos es una herramienta y no un fin, por lo que esta habilidad debe hacerse de forma perfecta para así reducir posibles errores en la tarea principal.

Este trabajo pretende hacer una investigación de los tipos de robots móviles terrestres existentes, para así pretender mejorar o allanar el terreno para hacer estos más eficientes y confiables.

1.1 Objetivos

El objetivo principal de este proyecto es el diseño, construcción y programación de un robot móvil terrestre, para así poder realizar una investigación real de los métodos de control actuales. Para lograr dicho fin, se planteó en primera instancia la siguiente lista de objetivos, de forma que, permitan establecer una serie de pasos e hitos a alcanzar para obtener una versión funcional, para así focalizar finalmente el trabajo en el control.

- Investigación de los robots móviles y elección del más interesante por parte del alumno.
 - Investigación de los diseños reales actuales y estudio de ventajas y desventajas de cada tipo de diseño.
 - Planteamiento inicial del prototipo: elección tanto de hardware, software y diseño grandes rasgos de la estructura del robot.
 - Modelado y diseño de la primera versión del robot.
 - Estudio de la locomoción del robot y la respectiva cinemática directa, inversa y coordinación de cada extremidad.
 - Construcción del robot.
 - Comprobación y posibles ajustes, rediseños, cambios, etc.
 - Estudio de los tipos de control y su implementación.
 - Publicación del proyecto en páginas como GitHub, para así compartir mi trabajo con la comunidad investigadora/docente.
-

1.2 Documentación

Toda la documentación del proyecto se encuentra disponible en el siguiente repositorio de Github: <https://github.com/javi-desp/H-Robix.git>, allí se subirá todo el trabajo desde el inicio. De esta forma se usará la herramienta de versiones de Git en todo momento para tener constancia de cada cambio y versión y hacer más eficiente el desarrollo del código del trabajo.

2 Marco Teórico

La gran totalidad de mecanismos de locomoción ha surgido desde una inspiración biológica. Esto se debe al mismo proceso de validación que la evolución ha ido perpetrando en los seres vivos, siendo los individuos supervivientes o exitosos los que más consiguen adaptarse a un terreno, por ende, los más estables, livianos, y eficientes. Sin embargo, existe una excepción. La rueda, inventada sobre el 4500 a.C. proporciona un sistema con una eficiencia muy elevada en terrenos planos 2.3, y con su debida amortiguación consigue un desplazamiento exitoso en terrenos levemente irregulares, a pesar de esto, los sistemas con ruedas resultan poco precisos y difíciles de controlar, careciendo de la habilidad de responder a las perturbaciones de la marcha causadas por el entorno, es decir, agujeros u objetos en el camino.

<i>Technical Criteria</i>	<i>Wheel Robot</i>	<i>Legged Robot</i>
Maneuverability	X	✓
Transvers ability	X	✓
controllability	✓	X
Terrain Land	X	✓
Efficiency	X	✓
Stability	✓	X
Cost effective	✓	X
Navigation over obstacles	X	✓

Figura 2.1: Pros y contras de un robot con patas en contraposición con uno dotado de ruedas [2].

Llegados a este punto, a la hora de diseñar un sistema locomotor para un robot en cuestión, sería muy interesante emplear un diseño bio-inspirado, sin embargo, imitar

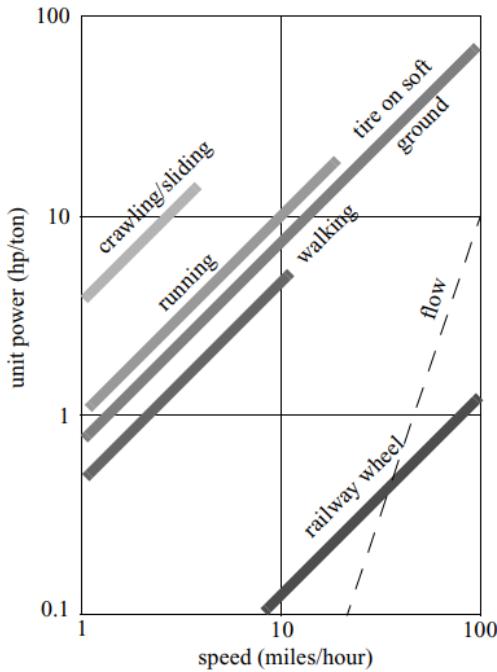


Figura 2.2: Potencia y velocidad alcanzable según su configuración [1].

sistemas biológicos puede resultar extremadamente difícil. En primer lugar la complejidad mecánica de sistemas biológicos se realiza en base a la replicación estructural dada por la división celular y la especialización de las mismas [1]. Gracias a esto se pueden crear sistemas muy pequeños pero con una complejidad mecánica abrumadora. En este aspecto, las técnicas de fabricación humanas de sensores y efectores sigue siendo bastante rudimentaria, por este motivo la replicación de estos sistemas usualmente es bastante pobre y no se consigue ni una flexibilidad ni un control parecido a los sistemas reales.

2.1 Robótica Bio-inspirada

La robótica bio-inspirada no implica únicamente copiar la naturaleza, más bien es un intento de entender los principios básicos del comportamiento de los animales, así

como los aspectos en los que más se ha desarrollado para poder adaptarse y sobrevivir en un entorno. Es por ello por lo que la biología y la etología contienen conocimientos clave para el desarrollo de la robótica. Disciplinas como la neurociencia (neurociencia computacional), o la biomecánica son grandes pilares en los que la bio-inspiración se ha apoyado. Dentro de la Robótica Bio-inspirada, se pueden encontrar varias ramas las cuales se desarrollan en diferentes aspectos: la Bio-locomoción/ Bio-mecánica, neurociencia y biología de sistemas (estudio de nuevas estructuras).

En este proyecto se hará hincapié en la bio-locomoción, y que sistemas actualmente están siendo más estudiados y pueden llegar a tener mayor o menor impacto.

2.1.1 Bio-locomoción

Por los motivos mostrados anteriormente hay un número creciente de investigadores que están empezando a plantear modelos biológicamente motivados [8]. El interés en este área es bastante variado, algunos investigadores están interesados en determinar la base biológica de locomoción de artrópodos, explorando la validez de tales modelos en la simulación, mientras que otros están en la búsqueda de mejores robots andantes.

En la naturaleza existe multitud de tipos de locomoción, como se puede observar en la figura 2.12. Sin embargo, actualmente solo se ha podido crear diseños reales funcionales propias del movimiento animal y algunas excepciones en locomoción propia de insectos. Los sistemas más exitosos son los siguientes: cuadrúpedos, bípedos, hexápodos y reptante.

2.1.2 Locomoción reptante

La locomoción reptante o ápoda es cualquier técnica de desplazamiento, principalmente de deslizamiento o arrastre sobre la superficie, en la que las fuerzas de propulsión

Type of motion	Resistance to motion	Basic kinematics of motion
Flow in a Channel	Hydrodynamic forces	Eddies
Crawl	Friction forces	Longitudinal vibration
Sliding	Friction forces	Transverse vibration
Running	Loss of kinetic energy	Oscillatory movement of a multi-link pendulum
Jumping	Loss of kinetic energy	Oscillatory movement of a multi-link pendulum
Walking	Gravitational forces	Rolling of a polygon

Figura 2.3: Tipos de locomoción biológica [1].

se logran gracias a movimientos o cambios de forma del cuerpo del animal sin la intervención de ningún tipo de extremidades.

Las técnicas de locomoción ápoda artificial puede ser clasificarse en tres grupos:

- **Arrastre con dos puntos de sujeción**
- **Movimiento peristáltico**
- **Movimiento Serpentino**

Pese a ser muy interesantes este tipo de robots, su principal inconveniente es su reducida movilidad y su escaso control, siendo útil únicamente en labores de rescate muy específicas o labores de limpieza en tuberías. Por este motivo se decidió seguir investigando otro tipo de configuraciones.[13]

2.1.3 Locomoción bípeda

La locomoción de robots con piernas, humanoides o similares, no es una novedad para el desarrollo tecnológico. Los primeros trabajos que más expectación popular crearon se iniciaron hace 40 años de la mano de la multinacional Honda Motor Company. Este desarrollo de robots bípedos con un objetivo publicitario fue evolucionando desde su creación con los Honda E Series (1986-1993) hasta su último modelo ASIMO (2011) 2.4 , el cual tenía la capacidad de saltar, correr, bajar escaleras, abrir botellas, hablar en lenguaje de signos... siendo un robot diseñado para el uso en entornos domésticos donde se requiera asistencia. Sin embargo pero en el 2018, la empresa anunció el fin de su desarrollo.



Figura 2.4: Desarrollo de robots bípedos por Honda Honda Motor Company (autor desconocido).

Debido al poco pragmatismo que se le puede dar a un sistema bípedo frente a uno con ruedas en la actualidad, además de la evolución de otras empresas como “Boston Dynamics”, cuyos robots se muestran mas eficientes y exitosos que ASIMO, como el robot Atlas con 28 grados de libertad en total. Honda suspendió esta linea de investigación.

2.1.3.1 Zero-Moment Point (ZMP)

Esta técnica de control fue introducida por primera vez hace más de 35 años [11], y es actualmente una de las más extendidas. Establece un criterio de estabilidad dinámi-

ca permitiendo generar patrones de locomoción. ZMP Se puede definir como el punto pZMP en el suelo tal que el momento neto de las fuerzas externas no tiene componente sobre los ejes horizontales [19]. Cuando pZMP existe dentro del polígono de soporte, el contacto entre el suelo y el pie es estable. Cuanto más cercano esté p al centro de la superficie de soporte, más robustez se conseguirá. Cuando ZMP está fuera del polígono de soporte, el robot se inclina rotando sobre alguno de los bordes de dicho polígono. El criterio de que ZMP exista dentro del polígono de soporte es condición necesaria y suficiente para garantizar la estabilidad dinámica del robot. De forma intuitiva, la condición ZMP asegura que el movimiento del cuerpo será tal que el pie estará plano en el suelo y por tanto no caerá.

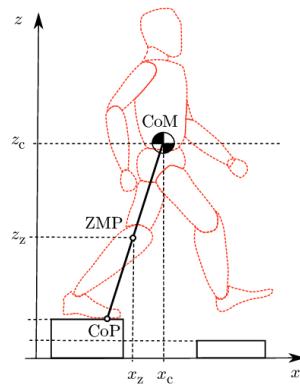


Figura 2.5: Ilustración del ZMP [19].

Debido a la alta complejidad de control de estos sistemas y el ferviente desarrollo que se ha realizado desde hace ya casi 5 décadas, se decidió seguir investigando otro tipo de locomoción con una tecnología menos estudiada y más interesante en el aspecto científico investigador.

2.1.4 Locomoción cuadrúpeda

La robótica cuadrúpeda ha sido el tipo de locomoción que más ha avanzado estos últimos años. Esto ha sido gracias a la multitud de proyectos exitosos que han salido a

la luz, haciendo un gran aporte tanto teórico como práctico. Algunas de las máquinas que mayor expectación han tenido debido a sus recientes y espectaculares resultados. Sin embargo, la locomoción hexápoda lleva estudiándose desde la década de los 60 [2].

Pero no fue hasta 2008 cuando la robótica cuadrúpeda empezó a hacer resonar en los medios, viéndose que efectivamente un sistema robótico con patas podía ser útil en condiciones adversas y terrenos abruptos. Este cambio se debió a la publicación del desarrollo de un cuadrúpedo autónomo avanzado llamado Big Dog creado por "Boston Dynamics". Este cuadrúpedo de 0,91 m de largo y 0,76 m de alto, con un peso de 110 kg era capaz de atravesar terrenos difíciles, correr 6,4 km/h, transportar una carga máxima de 150 kg y subir pendientes de 35 grados. Sin embargo, no se utilizó nunca para el objetivo que en un principio estaba destinado, el campo militar. Esto se debió al fuerte ruido que hacia al moverse por emplear un motor de combustión y accionamientos hidráulicos.

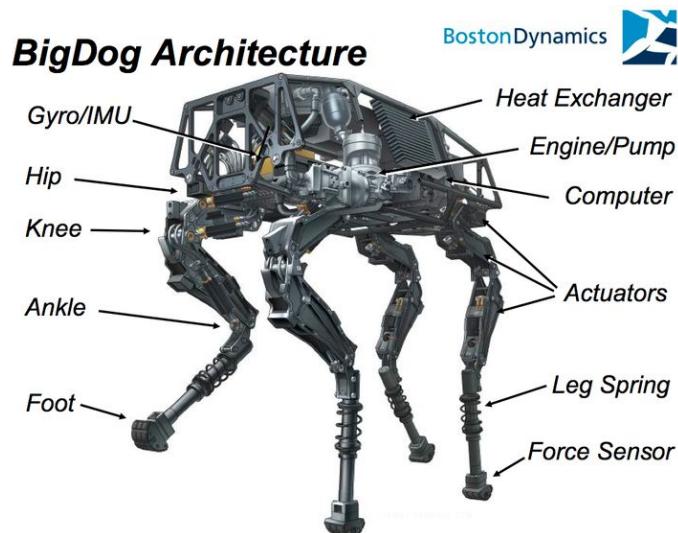


Figura 2.6: Partes mecánicas del robot Big Dog creado por "Boston Dynamics" (autor desconocido).

no obstante la gran publicidad que generó este prototipo le dio una gran visibilidad a esta empresa, cambiando un rumbo al diseño de robótica de servicios y no tan dirigida

al campo militar. Ejemplo de esto es su impresionante nuevo lanzamiento llamado spot (2019), siendo un robot cuadrúpedo de pequeñas dimensiones con un control dinámico bio-inspirado que puede fácilmente subir, bajar escaleras, mantener el equilibrio y adaptarse a fuerzas físicas externas.

Sin embargo, este monopolio tecnológico liderado por Boston Dynamics en este campo se ha terminado, pues actualmente se puede encontrar multitud de robots con un diseño parecido al spot pero de diferentes fabricantes y a precios muy asequibles. Por este motivo, y en vista a los anteriores sistemas locomotores de patas, se decidió investigar finalmente la locomoción hexápoda, siendo la locomoción elegida para la futura concepción del robot que este proyecto plantea realizar.

2.1.5 Locomoción hexápoda

Los robots hexápodos, al igual que los cuadrúpedos, pueden ser utilizados para aplicaciones que precisen un movimiento en terrenos abruptos, poco predecibles, con obstáculos, grandes inclinaciones, etc. Este sistema de locomoción a diferencia de los robots de cuatro patas aparentemente no parece haber cuajado en la industria, no habiendo un gran desarrollo detrás que haya exprimido al máximo las ventajas que puede ofrecer un sistema locomotor con estas características [4]. Algunas de las ventajas que ofrece este sistema son los siguientes:

Este tipo de configuración a pesar de tener una complejidad mecánica superior, gracias la estabilidad que aporta su diseño se consigue obtener un control más simplificado, que requiere un cálculo computacional reducido. Además, este tipo de robots tienen la ventaja de ser más robustos a fallos mecánicos, es decir, si en algún momento alguna pata falla o se estropea, la estabilidad y el funcionamiento del robot no se verá comprometida, a diferencia de los otros sistemas móviles anteriormente citados. Otro gran punto a favor, como se ha mencionado antes, es la estabilidad que ofrece un sistema de

este tipo, pues se sabe que con mayor estabilidad se reducen los posibles puntos críticos que puedan dar a caídas, o movimientos inesperados, protegiendo en mayor medida tanto el hardware del robot como la posible carga que pueda llevar en ese momento.

2.1.5.1 Criterios de estabilidad de un modelo hexápodo

Para mantener la estabilidad un robot hexápodo debe mantener al menos 3 puntos de apoyo, de este modo, 3 o más extremidades deberán estar en contacto con el suelo para asegurar una estabilidad estática. Es importante diferenciar la estabilidad estática de la dinámica [15], esto se debe a que en la estabilidad estática, siempre se debe compensar las fuerzas del centro de gravedad con los apoyos del sistema, haciendo que este punto se encuentre dentro del "polígono de soporte", mostrado en la figura 2.7. Sin embargo, respecto a la estabilidad dinámica este aspecto cambia, ya que jugando con la inercia del sistema se puede realizar un movimiento pese a que el centro de gravedad no este en el polígono de los soportes, o incluso no lo haya (véase el movimiento bípedo).

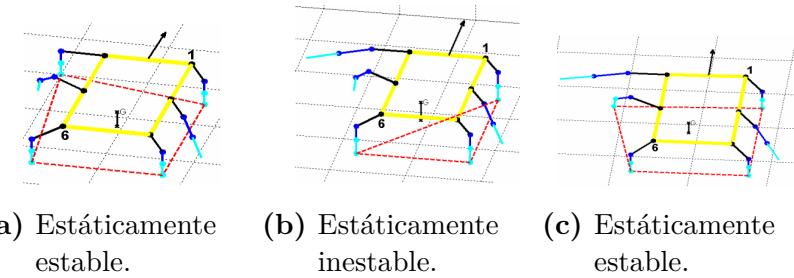


Figura 2.7: Estabilidad estática de un hexápodo según el polígono de soportes. Fuente:[16].

El centro de gravedad de una estructura es el punto de aplicación de la resultante de toda la fuerza de gravedad que actúa sobre las distintas porciones materiales de un cuerpo, usualmente en sistemas simétricos este punto se localiza en el centro. Como puede ser en los robots hexápodos. Esta característica de los robots hexápodos, así como los elevados casos de estabilidad hacen que sea un candidato idóneo para el

movimiento en escenarios con obstáculos, agujeros, flexibles, etc.

2.1.5.2 Estados de estabilidad

Podemos clasificar los estados de estabilidad de un robot hexápodo en tres principales categorías [16] : estable, inestable y críticamente estable. Si el hexápodo en cuestión se encuentra en una situación inestable, este no podrá garantizar la posición, precisando de un movimiento para mantenerla. Mientras que una posición críticamente inestable será cuando el centro de masas del robot este posicionado justo en una arista del polígono de estabilidad, de tal manera que cualquier pequeño movimiento podría convertir el robot en inestable estático y hacer que se caiga.

En el artículo [15], se muestra como calcular el llamado "static stability margin", que define la proyección del centro de masas a la arista más cercana del polígono de soportes. Si este valor es pequeño el robot podrá perder su estabilidad. Este concepto será muy importante y se tendrá que tener en cuenta siempre que se quiera diseñar un algoritmo de locomoción de un robot hexápodo, sin embargo, como se es de suponer, la mayoría de formas de movimiento empleadas en los robots hexápodos son bio-inspiradas y cumplen a la perfección los criterios de estabilidad descritos.

2.1.5.3 Tipos de marcha hexápoda

Los tipos de marcha o "gait", se puede definir como un patrón de movimientos durante la locomoción que al comandarselo a cada miembro del sistema produce un movimiento exitoso. Para comprender este proceso en primer lugar se debe estudiar de forma individual cada miembro, de esta forma se puede dividir el movimiento de cada pata en 2 fases [14] [12]: una fase de postura "stance" y otra de columpio u oscilación "swing", como se puede ver en la imagen 2.10.

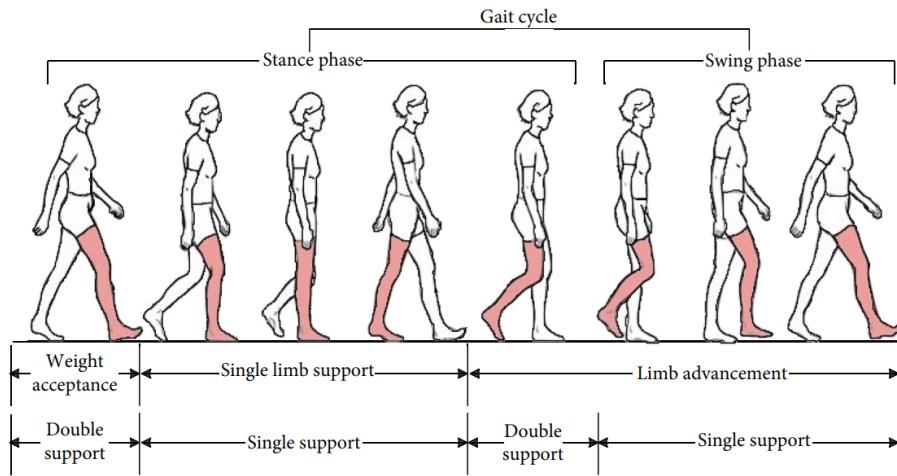


Figura 2.8: fases de la marcha [14].

- **Fase de stance:** abarca todo el tiempo que la pata está en contacto con el suelo.
- **Fase de swing:** En esta otra fase se describe el movimiento que hace el miembro al levantarse y posicionarse más adelante.

De esta forma, al andar se crea un ciclo simétrico, es decir, el tiempo que tarda en hacer una extremidad una trayectoria debe ser la misma a su opuesta. Si este principio no se cumple no se podría asegurar una marcha dinámicamente estable. De esta forma, se puede decir que la extremidad que realiza la fase de swing, soporta el peso y propulsa el cuerpo, por este motivo, esta fase se puede denominar también fase de soporte o de fuerza (support or power phase) [17].

Mientras que en la fase del swing, la pata se levanta hasta la posición inicial de la siguiente fase de postura. Cuando se realiza un movimiento hacia delante la extremidad que realiza esta fase se prolonga hacia la parte delantera del cuerpo. Esta fase se le conoce también como fase de recuperación (recovery or return phase).

Una vez explicado esto, y entendido el funcionamiento de la marcha y las fases de

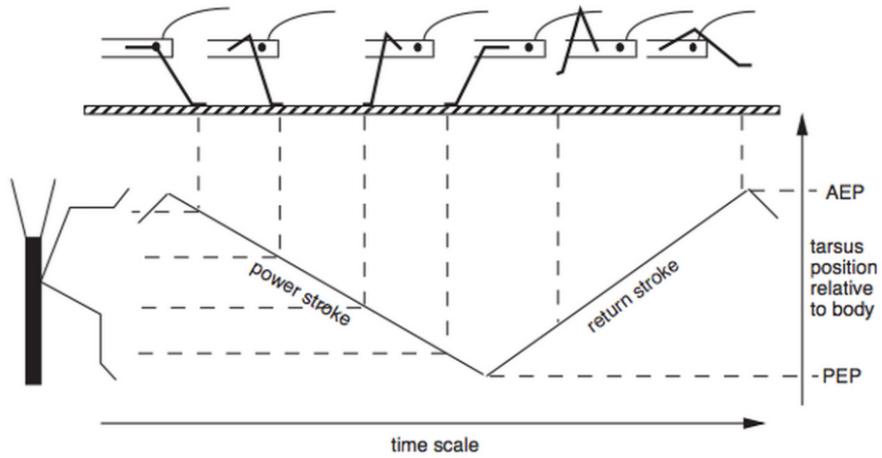


Figura 2.9: fases de la marcha de power and return stroke [17].

fuerza y recuperación, se puede empezar a entender como un robot hexápodo podría moverse. Actualmente existen multitud de marchas y cada programación a cada diseño puede tener un acercamiento único, sin embargo, existen (entre otros) tres tipos de marcha muy comunes de los robots hexápodos, que viniendo de una inspiración biológica mantienen el principio de estabilidad descrito en el apartado 2.7. Estos tipos de marcha o gaits son los siguientes [12]:

- **Wave Gait** Este es el tipo de marcha más lento pero que proporciona el mayor índice de estabilidad. En la marcha de tipo "wave", cada pata se mueve hacia delante de una en una (swing fase), mientras que las cinco demás se quedan quietas proporcionando soporte. Suele empezar con las patas de un lado (izquierdo o derecho) de delante a la parte posterior. De esta forma, combinando el movimiento de cada lado del robot se consigue un movimiento bastante estable. Aunque pueda parecer poco eficiente este sistema se utiliza cuando se requiere de una estabilidad máxima, por ejemplo, en casos que se deba llevar una carga muy delicada.
- **Tripod Gait** el movimiento de tipo "tripod" es el más empleado, debido a su simplicidad computacional y su gran capacidad de movimiento. En esta marcha

la fase de postura la hacen siempre 3 patas, mientras que las otras 3 hacen la fase de swing. Este sistema puede llegar a tener un índice de estabilidad estático reducido, sin embargo, al tratarse de un patron de movimiento que se realiza a altas velocidades este problema no llega a ser considerable.

- **Ripple Gait** Este tipo de marcha es más complicado que los citados anteriormente. Si se observa al gráfico 2.10, se puede reconocer que cada lado es una ola local que consta de fases de swing que no se superponen. El lado opuesto también es una onda local que está exactamente fuera de fase con ella. Si consideramos que la pata trasera izquierda (L3) y la pata trasera derecha (R3) son el comienzo de cada onda local, se puede observar como la onda local derecha (R3-R2-R1) comienza exactamente en el medio de la onda local izquierda (L3-L2-L1) más específicamente, en el medio de la fase de balanceo de la pierna izquierda central (L2). Este tipo de marcha tiene la ventaja de ser más eficiente en términos de estabilidad que la tripod, pero menos veloz.

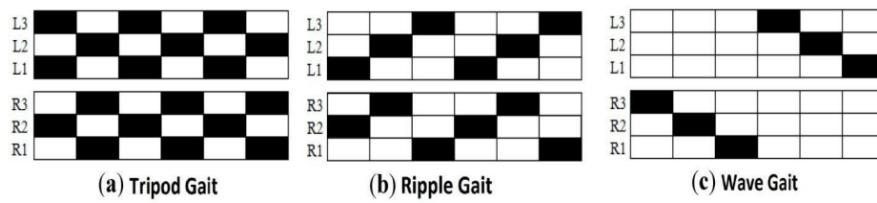


Figura 2.10: Tres tipos de marcha hexápoda más comunes [12].

2.2 Ejemplos de robots hexápodos actuales

Actualmente existen diferentes diseños y aproximaciones de robots hexápodos, que en su mayoría están hechos con fines académicos e investigadores. Por un lado podemos encontrar proyectos personales, en los cuales se ha hecho una aproximación individual y proyectos más serios con el objetivo de vender al público. Principalmente estos robots se

pueden clasificar en dos categorías según su hardware, siendo la elección de los motores fundamental, pues restringe el control final del robot y por consecuencia el movimiento.

Cuando se trata de elegir los motores de un robot con patas, que a la hora de realizar el movimiento debe mantener un peso, se busca que se pueda controlar la posición objetivo, de esta forma, se puede tener un control directo de la trayectoria a realizar de cada extremidad. Además, una especificación fundamental que se va a tener en cuenta va a ser la relación torque (o esfuerzo de torsión) y peso de un motor, ya que se buscará la configuración que mayor potencia ofrezca al sistema en relación al peso final.

Dadas estas necesidades la elección de los motores se restringe a 2 tipos: Servomotores y motores paso a paso.

2.2.1 Elección del tipo de motor

2.2.1.1 Motores paso a paso

El motor paso a paso es un motor de corriente continua sin escobillas en el que la rotación se divide en un cierto número de pasos resultantes de la estructura del motor. Normalmente, una revolución completa del eje de 360° se realiza en 200 pasos, cada uno de $1,8^\circ$. Debido a esta característica el motor paso a paso no gira suavemente, sino que realiza saltos y cruza estados intermedios, por lo que el funcionamiento del motor paso a paso se acompaña de un sonido y vibración.

Normalmente este tipo de motor se elige en tareas que se requiere un movimiento preciso, acompañado de un control sencillo de la posición y velocidad del motor (traducido en micropasos). Sin embargo, la relación torque/peso es bastante pobre, utilizándose normalmente en robots estáticos o robots móviles con ruedas que no requieren levantar su propio peso. Otra ventaja de este tipo de motor es que en el estado estático el mo-

tor no vibra, produciendo vibración únicamente en el estado de movimiento y siendo muy preciso cuando está parado. Además este tipo de motores resultan muy baratos pues por su propia configuración no precisan de sensores para controlar su posición, y duraderos, debido a su configuración sin escobillas.

2.2.1.2 Servomotores

Un servomotor es un motor DC común que cuenta con una configuración que le permite controlar la posición del eje en un momento dado. Normalmente está diseñado para moverse determinada cantidad de grados y luego mantenerse fijo en una posición. Dentro de la categoría de servomotores podemos encontrar dos tipos bien diferenciados según la electrónica de control del servomotor, siendo servomotores analógicos y digitales.

- **Servomotores analógicos**

los servomotores analógicos tienen la estructura mostrada en la imagen 2.11. Estos cuentan con un motor DC, un circuito de control simple que mide la posición del motor con un potenciómetro y finalmente un sistema de engranajes que aumenta el torque final. Este tipo de motores tienen el principal inconveniente que solo se puede hacer un control por posición, sin tener en cuenta perfiles de velocidad, aceleración, torque máximo, etc. Por este motivo el uso de estos motores se reduce únicamente al campo de la educación y el prototipado rápido. Así como el diseño de robots que solo necesitan un control de la posición simple.

Otro inconveniente a tener en cuenta de estos motores es que el control en posición es absoluto y no incremental. Esto significa que dependiendo de la posición inicial del eje cuando este se acople a un transmisor será arbitraria, y precisará de un proceso de calibración, que en muchas ocasiones puede dar a fallos, inexactitudes,

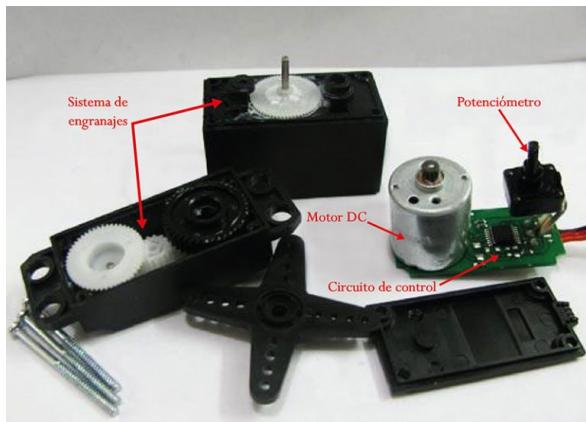


Figura 2.11: servomotor analógico (autor desconocido).

etc.

- **Servomotores digitales**

Los servomotores digitales a diferencia de los analógicos incorporan un microprocesador que analiza la señal del receptor y controla el motor. Es decir, en estos motores existe un sistema embebido que además de controlar gracias a varios sensores características como la temperatura del motor, posición absoluta e incremental, velocidad, aceleración, torque, etc. tiene un sistema que procesa esa información, y envía comandos al motor según en qué modo de control se haya ajustado internamente (posición, velocidad, torque).

De esta forma, el driver interno de un servomotor digital, suele tener un bucle interno (en los modos de posición y velocidad) controlado por un PID, con el que se puede ajustar múltiples variables para así controlar perfectamente la trayectoria realizada. Algunos de los posibles ajustes pueden ser: perfil de velocidad y aceleración, constantes PID, límites de posición, velocidad, aceleración y par, offset del motor, velocidad de transmisión de mensajes con el controlador (PC o PLC) entre otras muchas posibles configuraciones.

Los servomotores industriales siempre son digitales, dada la clara necesidad de tener un control cerrado de los motores. Sin embargo, en el campo de la robótica, aunque casi siempre se emplean digitales a veces se escogen analógicos por problemas debidos al presupuesto, teniendo en cuenta que naturalmente los motores digitales son mas costosos.

2.2.1.3 Motores usualmente empleados en robots hexápodos

Por los motivos mostrados en el apartado anterior, a la hora de escoger los motores de un robot hexápodo, se usa siempre servomotores, sin embargo, existen ejemplos de robots con motores digitales y analógicos.

2.2.2 Robots hexápodos según su configuración

2.2.2.1 Robots hexápodos Con servomotores analógicos

El problema principal de este tipo de configuración escogida es que al tener un bucle cerrado en los motores no se tiene información directa real de en que situación de las articulaciones del robot, por este motivo, hace a los hexápodos con motores analógicos completamente inútiles en tareas de movimiento en terrenos abruptos. Siendo ilógica la construcción de un robot hexápodo si únicamente puede desplazarse en superficies planas.

A pesar de esto, el hecho de que estos robots tengan este tipo de motores hacen que su precio se reduzca drásticamente, llegando a ser bastante económicos. Por este motivo se emplean en el campo de la educación y con un uso más recreativo. Siendo bastante útiles para empezar en el mundo de la robótica o estudiar los tipos de "gaits" o marchas de este tipo de robots.

Dentro de esta clasificación podemos encontrar dos hexápodos muy interesantes, en



Figura 2.12: Hexápodo Frenove (fuente propia).

primer lugar el hexápodo "Hexy robot". Este hexapodo open-source diseñado por la empresa "Arcbotix" presenta la opción de comprar en su web un hexápodo por 200€ o hacerlo con impresión 3D por un precio reducido. Este robot se controla con Arduino Mega y el código y control se pueden encontrar en internet. Por otro lado, podemos encontrar al hexápodo "Freenove big hexapod" de la empresa Freenove, empresa que se dedica a vender robots low-cost educativos. Este robot se controla con una Raspberry Pi, la cual tiene conectada a sus GPIOS una placa con los drivers necesarios para cada servomotor.

2.2.2.2 Robots hexápodos con servomotores digitales

En este otro grupo podemos encontrar proyectos más complejos con un objetivo puesto en la investigación y desarrollo de un sistema robótico más serio que permita realizar un control perfecto del robot y así poder tener un feedback de la posición de cada articulación y el movimiento que se desee.

Uno de los robots hexápodos más conocidos de este tipo de configuración son los de la familia zenta, siendo el mas famoso el "MxPhoenix" por su tamaño y la forma

característica de las patas. Este hexápodo tiene la particularidad de tener botones en las extremidades para así tener un feedback real de cuando las patas se han posicionado en el suelo. En el siguiente vídeo (<https://www.youtube.com/watch?v=rHFuql1cQfU&t=3s>) se puede observar tres tipos de hexápodos diseñados por el mismo autor, en el que se muestran los principales tipos de marcha hexápoda. Este vídeo es muy esclarecedor pues en vista a las pruebas finales se puede denotar que el diseño final del robot será determinante para la eficiencia del movimiento del hexápodo.



Figura 2.13: Hexápodos de la familia zenta (autor desconocido).

Además de estos hexápodos se pueden encontrar otros pero con características muy similares, como el hexapodo llamado "Mildred" (<https://hackaday.io/project/164758/gallery#9f56ed2fe8ff76ec1efe70bd17a6dd46>), o la serie de robots hexápidos creados por la empresa "Tossen Robotics" llamados PhantomX. Empresa que creó el foro "Tossen robotics forum", foro con un gran valor informativo, en el que cualquier persona que decida realizar un diseño de un robot puede acudir a preguntar dudas.

Sin duda, lo más interesante en este aspecto puede resultar la elección de los servomotores digitales de proyectos de este calibre, pues su gran mayoría (por no decir su totalidad), eligen motores "Dynamixel" de la empresa ROBOTIS. Esto se debe a que

los motores dynamixel son los servomotores digitales para uso en robótica (dimensiones reducidas) más empleados en el mundo, siendo relativamente baratos y empleados en proyectos en los que se requiera de un control de trayectorias.

2.2.3 Hardware usualmente empleado en robots hexápodos

Los robots de estas características normalmente cuentan del siguiente hardware:

- **Motores:** Analógicos o digitales
- **Drivers:** En el caso de los motores digitales será preciso la utilización de drivers, es decir un microcontrolador que procese y maneje el envío y recepción de datos de los motores (RX, Tx), y que envíe esta información a la placa de control final, siendo una especie de puente. para la elección de drivers con servomotores se suelen emplear placas Arduino del tipo MEGA u drivers que proporcionen los propios fabricantes de los motores.
- **Placa de control embebida:** Pudiéndose utilizar microcontroladores de tipo Arduino directamente, u placas con una distribución más parecida a la de un ordenador, como una raspberry u ordenadores pequeños o "nettop", que son ordenadores de tamaño reducido con un estilo parecido al de raspberry pero con mayor capacidad de computación. En cada caso y dependiendo del objetivo se elegirá una placa de control, según las funcionalidades del robot, autonomía, complejidad de la tarea, conectividad, etc.
- **Sensorización :** Aunque no es del todo necesario, para un diseño inicial de un prototipo la sensorización es una parte fundamental de la robótica, de esta forma se consigue que el robot tenga la capacidad de reconocer el entorno y así poder actuar en consecuencia. Por este motivo, los robots pueden tener sensores de distancia, Lidar, cámaras, etc. Sin embargo, en este proyecto no se ha decidido profundizar en este campo, pues una vez realizado una teleoperación del robot y hecho un control perfecto, la sensorización y el movimiento dirigido se puede contemplar como otro campo de estudio.

- **baterías y electrónica para su alimentación:** Elementalmente, el robot deberá disponer de una alimentación eléctrica para así poder moverse. Pese a que para realizar pruebas la colocación de baterías no es obligatorio, resulta evidente e imprescindible la liberación de cables, dotando al hexápedo de baterías. En este campo para robots de pequeñas dimensiones en el que el peso es una característica crítica normalmente se emplean baterías de litio LiPO. Siendo necesaria una electrónica, para poder reducir el voltaje de entrada y alimentar de forma correcta todos los elementos del hardware así como otra parte que se encargue de cargar las baterías de forma correcta, equilibrando las cargas de cada célula de la batería.
- **Módulos de transmisión de datos:** Para una correcta teleoperación u un control supervisado de las tareas del robot, se deberá dotar de un módulo de comunicación si la placa de control no cuenta con una ya de serie. La elección del tipo de transmisión de datos vendrá dado principalmente por la cantidad de datos a enviar y la necesidad de un control en tiempo real o no. Pudiéndose utilizar tecnologías como Wifi, bluetooth, LoRaWAN, etc.

3 Metodología

3.1 Hardware finalmente empleado

3.1.1 Motores utilizados

3.1.1.1 Dynamixel technology

Los motores Dynamixel son motores de muy altas prestaciones para robots totalmente programables y que proporcionan mucha información de "feedback" acerca del estado de estos. Alguna de las características de estos motores son los siguientes:

- La conexión entre varios motores Dynamixel se realiza de forma muy simple, gracias a la topología daisy-chain. De esta forma como se puede ver en la figura 3.1 , un motor se puede conectar a otros de forma sucesiva, haciendo que los cables que van a la placa controladora sean reducidos.

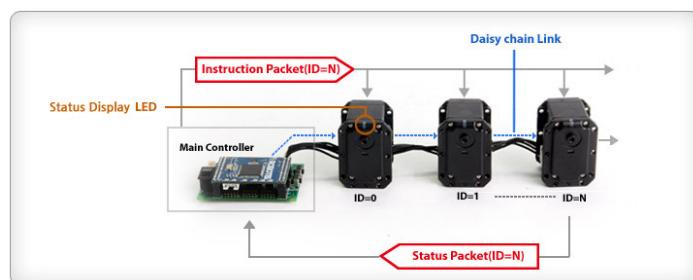


Figura 3.1: conexionado del tipo daisy-chain (autor desconocido).

- Cada motor dynamixel tiene dos ID, un ID de conexión único utilizado para la

conexión entre motores y otro único y modifiable utilizado para ser referenciado en el lado del control de los drivers.

- Estos motores emplean protocolos de comunicación digital de paquetes en la red serie. Soportando TTL y el estándar RS-485 (dependiendo del modelo).
- Tienen un gran feedback del estado de los motores, pudiendo leer la posición actual del motor, la velocidad, la temperatura interna, el torque o la tensión de alimentación entre otras.
- Función de alarma: cuando la temperatura interna, torque (carga), tensión de alimentación, etc. salen de unos márgenes de seguridad para la propia integridad del hardware, los motores entran en estado de paro de emergencia, parpadeando los leds y desconectándose el actuador.
- Estos motores poseen una alta eficiencia debido a la baja corriente de consumo.
- Dependiendo del modo de empleo de los motores Podemos encontrar tres tipos: posición, velocidad o torque (PWM)
 - **Modo de posición:** Esto modo suele ser el más utilizado en el control de trayectorias para la locomoción de robots hexápodos. Esto se debe a la sencillez a la hora de implementar un control directamente en posición. Los motores dynamixel, internamente implementan un bucle interno que realiza la trayectoria de un punto inicial a uno final comandado, teniendo en cuenta esfuerzos externos, torque, velocidad y aceleración utilizada. El esquema de control interno es el que se puede observar en la figura 3.2, que como se puede observar, en primer lugar se asigna un perfil de velocidad y aceleración, y dada estas consignas se suma el error e con un control de tipo PID, teniendo en última instancia un límite de velocidad, aceleración y torque.

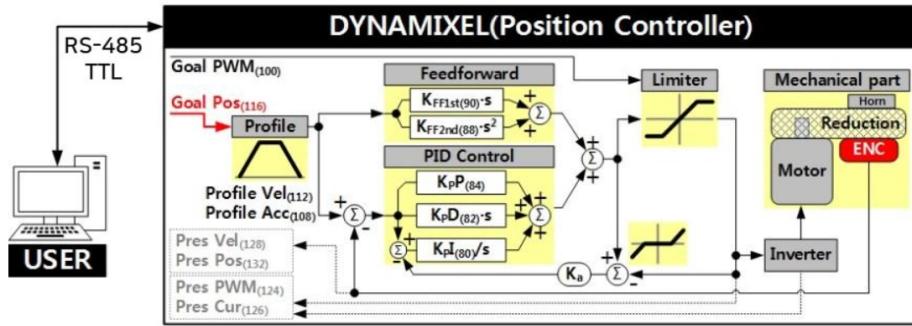


Figura 3.2: Controlador interno del modo de posición de los motores dynamixel [7].

– **Modo de velocidad:** En este modo se controla directamente la velocidad final del motor, muy útil para robots móviles con ruedas o control de trayectorias cinemáticos articulares. Como se puede ver en su esquema de control 3.3, este esquema de control interno es más simple que el del modo de posición, esto se debe a que naturalmente, al realizar un control más “directo” de las consignas del robot, este control se hará en el código de control de trayectorias, teniendo más margen de configuración y control de las trayectorias. En el esquema se puede observar como solo se emplea el esquema de aceleración, siendo la consigna, de tal manera que la consigna se le suma a un control PID, siendo la velocidad objetivo, limitada antes por los límites de aceleración y torque.

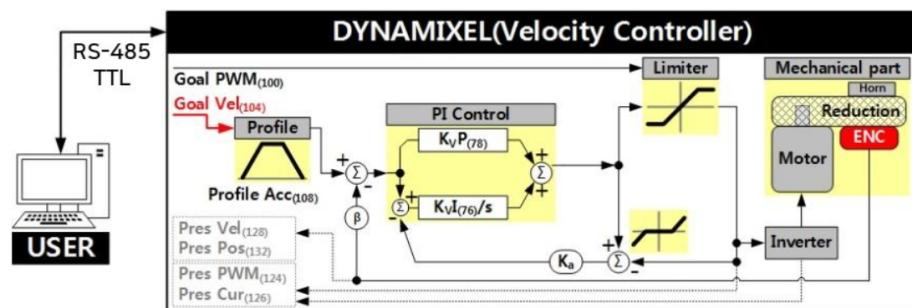


Figura 3.3: Controlador interno del modo de velocidad de los motores dynamixel [7].

– **Modo PWM:** Siempre que se habla del control o el feedback del valor del

torque ejercido o a ejercer en este tipo de motores, no se está hablando de una medida del torque real con un sensor de fuerza. En estos casos el torque hace referencia a la medida de los pulsos PWM de voltaje ejercidos para mantener una posición determinada, o velocidad. De esta forma se puede calcular un valor proporcional al máximo par realizable, obteniendo así una variable que no es exactamente el par pero si es extrapolable a este.

Con el control directo del par se comanda directamente el voltaje PWM a los motores, por este motivo, en estos casos el bucle interno desaparece, siendo el controlador externo de la aplicación el que con un control dinámico calcula variables como la velocidad precisada y posición de una trayectoria.

- Como se ha explicado en la sección de los modos de ejecución, estos motores permiten un control y asignación de perfiles de velocidad y aceleración simétricos. Estos perfiles pueden tener significados distintos según el modo de control de perfiles que esté activado, habiendo dos modos.

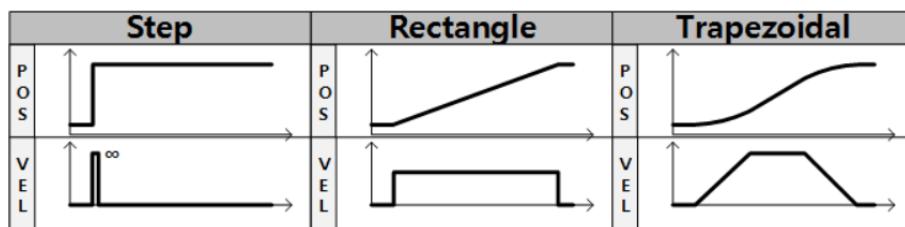


Figura 3.4: Trayectoria realizada según el perfil de velocidad definido [7].

En primer lugar podemos encontrar el modo de configuración de los perfiles basado en tiempo. En este modo, como se puede observar en la figura 3.5, se define los puntos del perfil de velocidad final y del primer tramo en función del tiempo, mientras que en el modo basado en velocidad, estos se configuran respecto a la aceleración y velocidad deseadas.

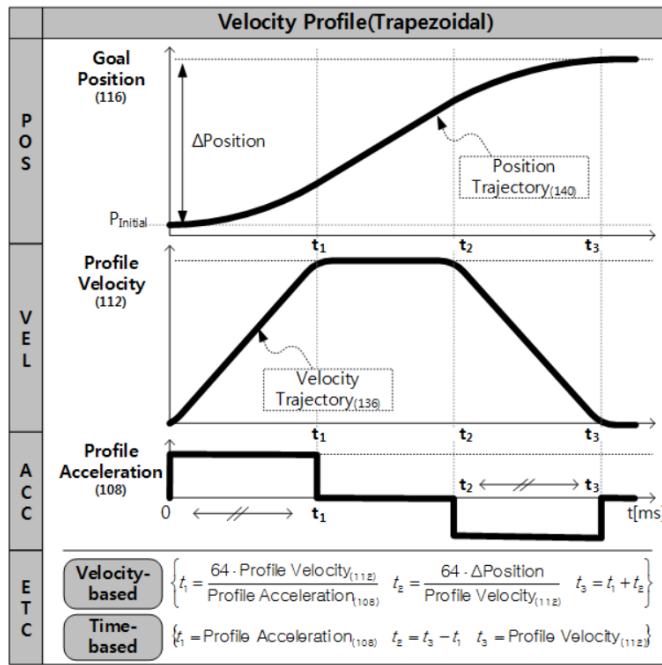


Figura 3.5: Controlador interno del modo de velocidad de los motores dynamixel [7].

3.1.1.2 Motores XL430-W250 y 2XL430-W250

Al final tras estar investigando entre las diferentes versiones de motores dynamixel, la diferencia principal entre unas versiones y otras son el par final operativo, siendo más costosos cuanto más par pueden realizar. Por este motivo, se decidió emplear los motores XL430, cuya potencia es de 1.4 N.m (Newton metro) o 14.276 kg cm (kilogramo por centímetro). Además se eligieron 6 motores XL430-W250, y otros 6 motores 2XL430-W250, siendo el segundo un motor con las mismas especificaciones que el primero pero con dos ejes motores.

Para la elección de estos motores como válidos se tuvo en cuenta el torque producido por estos, y se realizaron diferentes cálculos para comprobar que el sistema pudiera moverse sin ningún problema, es decir, que tuviera potencia suficiente. Para ello se estimó un peso máximo objetivo del robot, siendo este de 3kg. De esta forma, si se tiene el valor del momento que pueden ejercer los motores se puede saber la fuerza

que pueden realizar. Como en este caso los motores elegidos tendrán un momento de 14kg por centímetro, esta fuerza se irá reduciendo según la distancia de los eslabones, teniendo que ser necesariamente (teniendo en cuenta que habrá dos motores que hagan el esfuerzo para levantar al robot, mientras que otro hará el giro), cada pata deberá poder levantar más de 3Kg/6, siendo esto 0.5 kg. De esta forma, se puede decir que los motores podrán levantar sin problema al robot si la distancia de los eslabones es de 28cm.



(a) Motor XL430-W250.

(b) Motor 2XL430-W250.

Figura 3.6: Motores finalmente escogidos [6] [7]

3.1.1.3 Driver utilizado

Como se ha avanzado en el apartado 2.2.3, para poder controlar los motores con el PC integrado se necesitará un controlador o driver específico que haga de función de puente. Esto es necesario ya que para poder mantener estables mensajes de datagramas, baudrates y demás configuraciones de transmisión de datos, debe haber un hardware dedicado para esta misma tarea. Los motores Dynamixel tienen la posibilidad de controlarse con 3 tipos de controladores y cada uno será útil según la tarea a realizar y el hardware empleado en conjunto.

- **Controlador Arbotix**

La empresa interbotix, diseñadora y comercializadora de diferentes robots dotados de motores dynamixel lanzó en 2010 un microcontrolador bastante popular para conectar y controlar motores de este tipo. Este microcontrolador compatible con Arduino tiene las ventajas de ser Open source y de dotar de multitud de librerías con ejemplos que hicieron que esta placa se usara por entonces más incluso que los controladores de dynamixel (ROBOTIS). Además posee entradas y salidas digitales y analógicas.

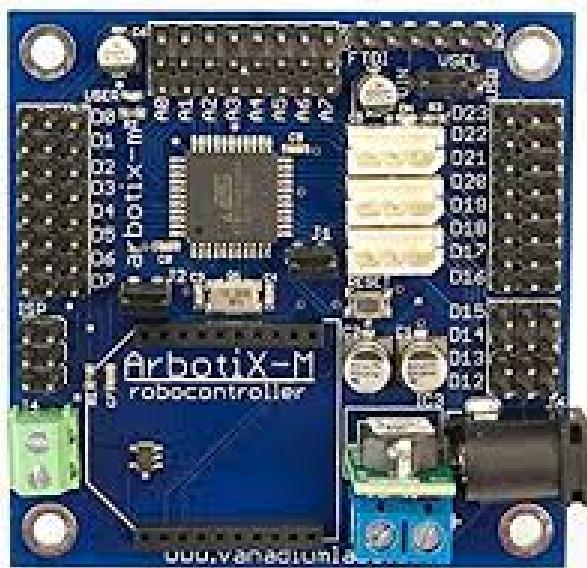


Figura 3.7: Controlador Arbotix (autor desconocido).

Esta placa se programa con el IDE de Arduino, y aunque se pueden hacer programas dentro de la placa, se suele utilizar como bridge, obteniendo la información de los servomotores y enviándosela a otra placa de control como puede ser un PC. Para ello se suelen emplear los pines del Rx y Tx conectados a un cable de

tipo USB.

- **Controlador OpenCM** Este otro microcontrolador para el control de motores dynamixel, se parece bastante al arbotix, pues es una placa Open Source programable en Arduino o con el IDE de ROBOTIS (recomendable). Frente al controlador de arbotix, este presenta un gran número de pines si se acopla con la placa de expansión de este equipo, altamente recomendable, pues además de tener multitud de pines GPIO, para posibles sensores, la capacidad de crear pulsos PWM, etc. También tiene incorporada electrónica para la conexión de 12V junto a un cómodo interruptor para cortar la corriente.

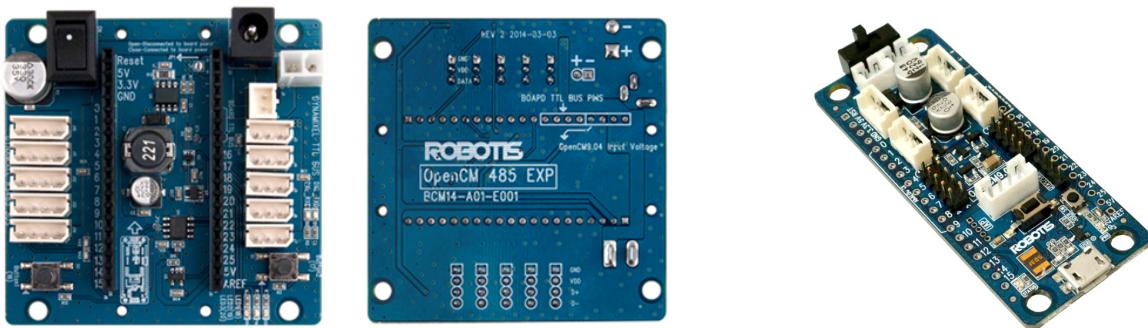


Figura 3.8: Microcontrolador OpenCM junto a su expansion board (autor desconocido).

A diferencia de la placa Arbotix, este microcontrolador es más potente, teniendo un procesador de 32bit ARM Cortex-M3. Por lo que está pensado para programar y controlar directamente el sistema desde este microcontrolador, teniendo la posibilidad de recibir y enviar información con módulos de transmisión de datos.

- **Controlador U2D2**

Finalmente nos encontramos el controlador U2D2, que ha sido el escogido para este proyecto. Esto se debe a que este microcontrolador esta pensado para

conectarse directamente con un PC, teniendo el hardware dedicado para ello y disponiendo de una alta velocidad de transmisión de datos. Como se puede observar en la figura 3.9, esta placa también tiene una placa de expansión, recomendable, pues añade de forma cómoda la electrónica necesaria para alimentar el microcontrolador, así como más entradas para motores.

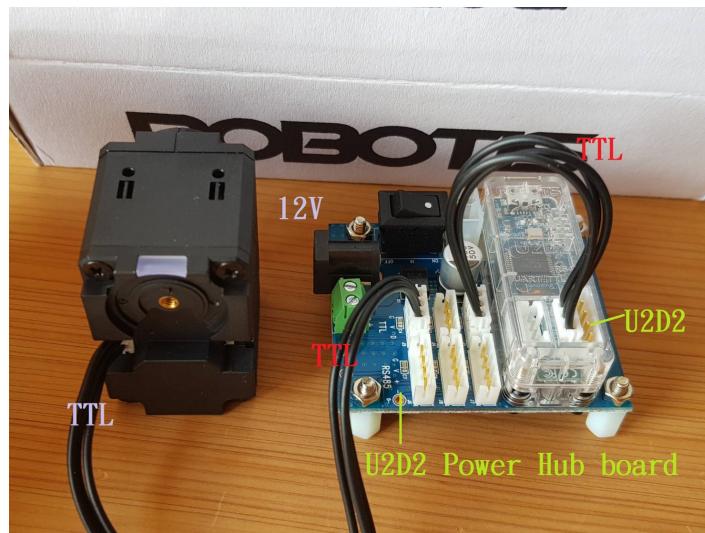


Figura 3.9: Controlador U2D2 (autor desconocido).

De esta forma, esta placa se usará como puente, recibiendo toda la información existente de los servomotores y pudiendo enviarla con una velocidad bastante elevada de transmisión. Para el control existen dos librerías principalmente: "Dynamixel SDK" y "Dynamixel Workbench", siendo la librería SDK una librería a bajo nivel, que controla el tiempo de envío y recepción de datagramas, así como su proceso de pack para envío a diferentes motores conectados en serie. Por otro lado está la librería "Workbench", que es una librería más a alto nivel, donde el control de los motores está por un nivel superior de abstracción, siendo más sencilla la programación y configuración de los motores.

3.1.1.4 Dinamixel Wizzard

Por último, en el apartado de las herramientas utilizadas de dynamixel nos encontramos "Dynamixel Wizzard". Esta herramienta complementaria del controlador U2D2 que no es mas que una interfaz de usuario, la cual lee los puertos de entrada del ordenador y escanea los motores conectados al microcontrolador, listandolos por ID, y pudiendose editar de forma visual parámetros de control, configuración de transmisión de datos, protocolos, modos de ejecución, etc.

Cabe destacar que este proceso se puede realizar de la misma forma con las librerías de control SDK y Wizzard, sin embargo, esta IU, permite cambiar valores como el baudrate sin que se desconecten los servos (haciendo un reload automático), así como hacer cambios con mejor visibilidad y hacer pruebas.

3.1.2 Raspberry Pi como PC integrado

Para la placa de control final se escogió una arquitectura de ordenador, esto se debe a que para poder investigar el control y hacer tareas avanzadas, se necesitará una capacidad de computo elevada, así como memoria RAM suficiente para poder ejecutar múltiples procesos a la vez y tener la posibilidad de teleoperar en tiempo real al robot.

A la hora de escoger el PC integrado habían dos opciones principales, escoger un "nettop ", o una placa reducida del estilo Raspberry. Sin embargo, finalmente se decidió emplear una Raspberry, debido a su reducido precio y a la disponibilidad del momento. En el caso de que no hubiera suficiente capacidad de procesamiento o RAM se procedería a hacer un control teleoperado del sistema, es decir, a enviar la información recibida por la placa a un ordenador con mejores prestaciones, realizar allí los cálculos necesarios y enviarlos de nuevo al ordenador.

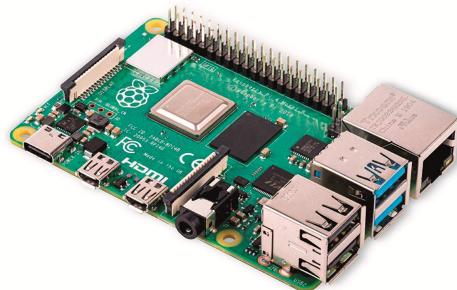


Figura 3.10: Raspberry pi modelo B (autor desconocido).

Respecto a los modelos de Raspberry existen diferentes categorías según sus prestaciones, siendo altamente recomendable utilizar una Raspberry pi 4B para estas tareas, pues de lo contrario, si se decide emplear una de rango inferior es muy posible encontrar problemas debido a la poca capacidad de computo o RAM. Además en este modelo existe la posibilidad de conseguirla con 2,4 y 8 Gb de RAM. Debido al alto precio de la placa de 8Gb de RAM se decidió emplear la de 4, ya que se esa cantidad de RAM para la tarea a realizar es suficiente.

3.2 Entornos de programación y Software

3.2.1 ROS

ROS o Robot Operating System [18], es un middleware open source, que no es más que un software que brinda servicios y funciones comunes a las aplicaciones. Generalmente, se encarga de la gestión de los datos, servicios, la mensajería, la autenticación y la gestión de las API. Aunque no es un sistema operativo como tal, proporciona conjunto de herramientas y utilidades, siendo la abstracción de hardware, el control a bajo nivel de dispositivos, implementación de envío de mensajes mediante procesos y administración de paquetes. En este framework permite programar tanto en C++, Python y Lisp, todo respaldado por la licencia BSD, open source para uso comercial e investigación. Por estos motivos y todas las herramientas que ofrece se ha elegido este

framework para el control del robot hexápodo.

3.2.1.1 Filosofía de ROS

Ros fue diseñado para evitar la reinvención de la rueda, ya que en el campo de la robótica es muy común, pues para poder investigar o resolver soluciones, en primer lugar siempre se debe construir, diseñar y programar un sistema robótico, teniendo que diseñar de cero siempre las mismas herramientas. Además al ser Open Source, ROS tiene un gran respaldo de la comunidad investigadora e incluso empresarial, como puede verse con el ejemplo de "ROS Industrial", pues Ros permite la implementación de paquetes y herramientas desarrolladas por terceros, haciendo que este software este en constante crecimiento y actualización.



Figura 3.11: Aportaciones que ofrece ROS (autor desconocido).

3.2.1.2 Diseño interno

El diseño interno de ROS se puede ver en la figura 3.11, que como describe se puede dividir en "plumbing", herramientas, capacidades y su propia comunidad.

- **"Plumbing"**

El "plumbing" es la arquitectura que define, procesa y realiza la transmisión de datos del sistema, a continuación se listan los elementos principales de este proceso en ROS.

- **Nodos**

Un nodo representa un proceso que ejecuta el gráfico ROS mostrado en la imagen descrito como una circunferencia. Cada nodo tiene un nombre, que

registra con el master de ROS antes de que pueda realizar cualquier otra acción. Los nodos están en el centro de la programación de ROS, ya que la mayoría del código de cliente de ROS tiene la forma de un nodo de ROS que realiza acciones en función de la información recibida de otros nodos, envía información a otros nodos o envía y recibe solicitudes de acciones hacia y desde otros nodos.

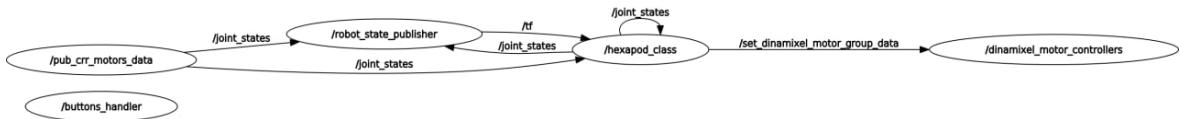


Figura 3.12: Esquema de plumbing de ROS realizado con la herramienta RQT (fuente propia).

– Topics

Los topics son buses con nombre a través de los cuales los nodos envían y reciben mensajes. Los nombres de los topics también deben ser únicos dentro de su espacio de nombres. Para enviar mensajes a un topic, un nodo debe publicar en dicho tema, mientras que para recibir mensajes debe suscribirse. El modelo de publicación/suscripción es anónimo: ningún nodo sabe qué nodos están enviando o recibiendo sobre un topic. Los tipos de mensajes transmitidos sobre un tema varían ampliamente y pueden ser definidos por el usuario. El contenido de estos mensajes puede ser datos de sensores, comandos de control de motores, información de estado, comandos de actuadores o cualquier otra cosa.

– Servicios

Un nodo también puede anunciar servicios. Un servicio representa una acción que puede realizar un nodo y que tendrá un único resultado. Como tal, los servicios a menudo se usan para acciones que tienen un inicio y un fi-

nal definidos, como capturar una imagen de un cuadro, en lugar de procesar comandos de velocidad en un motor de rueda o datos de odómetro de un codificador de rueda. Los nodos anuncian servicios y llaman a servicios entre sí.

- **Parameter server**

El parameter server es una base de datos compartida entre nodos que permite el acceso común a información estática o semi-estática. Los datos que no cambian con frecuencia y, como tales, a los que se accederá con poca frecuencia, como la distancia entre dos puntos fijos en el entorno o el peso del robot, son buenos candidatos para el almacenamiento en el servidor de parámetros. Normalmente, cuando se define la estructura física del robot (posición de actuadores, longitud de los eslabones, etc.), esta información se carga en el parameter server.

- **Herramientas**

La funcionalidad de ROS se ve aumentada por una variedad de herramientas que permiten a los desarrolladores visualizar y registrar datos, navegar fácilmente por las estructuras de paquetes de ROS y crear ficheros que automatizan procesos de configuración y configuración complejos. La adición de estas herramientas aumenta en gran medida las capacidades de los sistemas que utilizan ROS al simplificar y brindar soluciones a una serie de problemas comunes de desarrollo de robótica. Estas herramientas se proporcionan en paquetes como cualquier otro algoritmo, pero en lugar de proporcionar implementaciones de controladores de hardware o algoritmos para diversas tareas robóticas, estos paquetes proporcionan tareas y herramientas agnósticas de robots que vienen con el núcleo de la mayoría de las instalaciones modernas de ROS. Algunas de las herramientas que se van a utilizar en este proyecto son las siguientes:

- **Rviz**

Rviz es un visualizador tridimensional que se utiliza para visualizar robots, los entornos en los que trabajan y los datos de los sensores. Es una herramienta altamente configurable, con muchos tipos diferentes de visualizaciones y complementos. En este proyecto, se tratará de usar para así comprobar el movimiento y las trayectorias de las articulaciones en tiempo real.

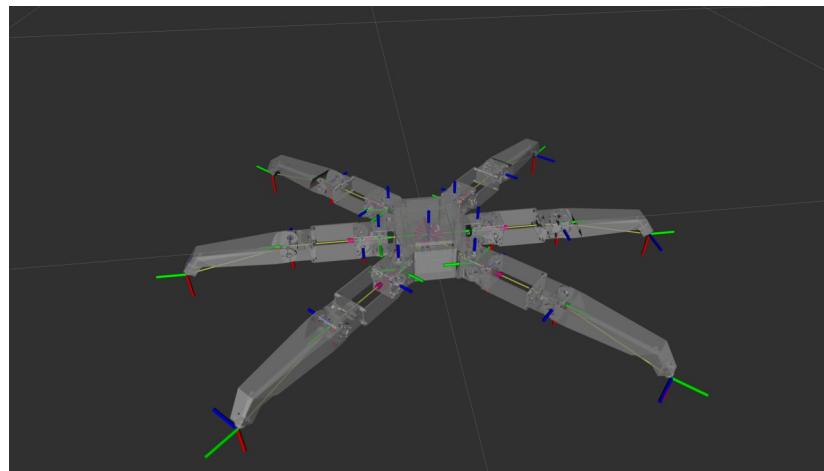


Figura 3.13: Visualización del robot diseñado con la herramienta Rviz (fuente propia).

– Catkin

Catkin es el sistema de compilación de ROS, se basa en CMake y es igualmente multiplataforma, de código abierto e independiente del lenguaje de programación. Esta herramienta ofrece múltiples comandos como puede ser catkin_make, catkin build, catkin_clean, etc. En este caso, catkin_make y catkin build hacen la misma tarea, que es la de compilar un workspace, sin embargo es preferible el uso de catkin build, pues ofrece la funcionalidad de crear un bashlog, que son ficheros donde se almacena la información de cada proceso con su registro en el tiempo. Una herramienta muy interesante para hacer un debug en proceso de ejecución y post ejecución.

- **Rosbash**

El paquete rosbash proporciona un conjunto de herramientas que aumentan la funcionalidad del shell bash. Estas herramientas incluyen rosls, roscd, roscp y rosfind entre otros. Las versiones ROS de estas herramientas permiten a los usuarios usar nombres de paquetes ros en lugar de la ruta del archivo donde se encuentra el paquete. El paquete también agrega tabulador a la mayoría de las utilidades de ROS e incluye rosed, que edita un archivo determinado con el editor de texto predeterminado elegido, así como rosrun, que ejecuta ejecutables en paquetes de ROS.

- **roslaunch**

Roslaunch es una herramienta que se utiliza para lanzar múltiples nodos ROS tanto de forma local como remota, así como para establecer parámetros en el servidor de parámetros ROS. Los archivos de configuración de roslaunch, que se escriben con XML, pueden automatizar fácilmente un proceso complejo de inicio y configuración en un solo comando. Los scripts de roslaunch pueden incluir otros scripts de roslaunch, lanzar nodos en máquinas específicas e incluso reiniciar procesos que mueren durante la ejecución.

- **Representación de sistemas de referencia o ”frames”**

Esta herramienta proporciona una funcionalidad muy interesante en la robótica, que es la referencia dinámica de los sistemas de referencia o frames de diferentes partes del robot, de esta forma, mediante transformaciones geométricas, se puede referenciar perfectamente todo del robot, de una forma sencilla y automática. Para realizar esta tarea se emplea tanto ”tf” como ”tf2”, que es la segunda generación de esta herramienta, que ofrece más funcionalidades.

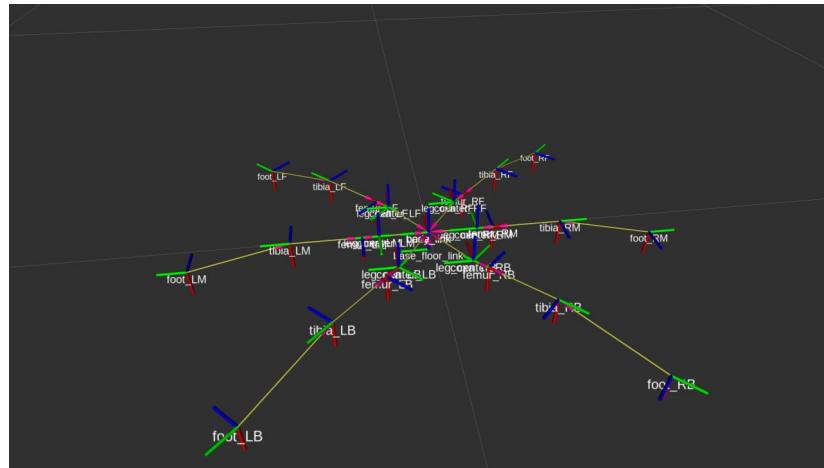


Figura 3.14: Visualización de los frames representados con Rviz del robot diseñado (fuente propia).

– Herramientas de simulación

Además ROS ofrece varias herramientas de simulación 2D Stage, y 3D Gazebo, siendo la segunda muy interesante para simular sistemas robóticos, así como su movimiento, uso de visión artificial, evitación de obstáculos, etc.

Gazebo, para poder simular un sistema robótico necesita definir la posición de los eslabones y articulaciones, así como su masa y matrices de inercia, para poder tener en cuenta fuerzas externas como la gravedad o de colisión. Además en esta herramienta se deben definir los controladores de los motores con el paquete "ROS controller", los cuales con un .yaml se define si van a ser de esfuerzo, posición, velocidad y su valor de control PID.

– URDF

Unified Robotics Description Format es una forma de modelar sistemas multicuerpo programado en XML utilizada en el mundo académico y la industria. URDF es especialmente popular entre los usuarios de ROS, pues define a la perfección la estructura de un robot para tareas de simulación, visualización, análisis y diseño de control. De esta forma el URDF o llama-

do comúnmente como esqueleto del robot, define la referencia en el espacio de todas sus articulaciones, tamaño de eslabones, masa, matriz de inercias, controladores, etc. Como es natural, esta herramienta es imprescindible para emplear tanto Rviz como Gazebo.

- **Capacidades**

El último elemento de ROS que lo hace muy completo es la existencia de paquetes que ofrecen funcionalidades. Ejemplo de esto pueden ser el paquete de navegación, que implementa herramientas de como SLAM, el paquete de control de trayectorias "joint_trajectory_controller" que implementa generadores de trayectorias cúbicos, quínticos de splines, etc. Así como controladores cinemáticos y dinámicos. El famoso paquete MoveIt, para robots manipuladores y muchos más.

3.2.2 Ubuntu Server

Además de haber elegido el dispositivo del control central, (en este caso una Raspberry), de debe escoger el sistema operativo (OS) que se adecúe mejor a las características así como las necesidades del proyecto. En este caso, como se va a querer implementar ROS para poder controlar el robot hexápedo va a ser necesario un OS que sea compatible con ROS, por lo que tenemos dos opciones principalmente: emplear una distro (o distribución de linux) o el sistema operativo que ofrece Raspberry (Raspbian).

En primer lugar raspbian es el Sistema Operativo específico de Raspberry, por lo que su instalación es bastante sencilla, sin embargo, aunque parezca extraño, la raspberry 4, que es la que se va a emplear en este proyecto tiene una arquitectura de procesador de 64 bits, sin embargo Raspbian es de 32 bits. Aunque ya se está empezando a desarrollar una versión de Raspbian de 64 bits aún no está disponible, y la elección de una arquitectura inferior al tamaño de bits del procesador radica en una bajada de la eficiencia y capacidad de computo, por este motivo se decidió emplear Ubuntu

server, que es la versión de Ubuntu sin la aplicación del escritorio, siendo más eficiente y menos costosa de ejecutar.

En concreto, se va a utilizar la versión Ubuntu Server 20.04.3 LTS, que actualmente, además de la versión 22.04, se trata de las últimas versiones disponibles. Se ha elegido este distro, debido al tiempo que lleva ya lanzado, por lo que resulta más robusto que la reciente versión 22.04. La versión 20.04 tiene soporte completo de actualizaciones de seguridad y mantenimiento hasta el año 2025, además, de un soporte adicional de tres años para actualizaciones de seguridad.

3.2.2.1 herramientas nativas de Ubuntu

Además de emplear Ubuntu se han utilizado múltiples herramientas nativas que han hecho que se pueda hacer una programación con mayor estilo de la máquina. Estas son las herramientas que se han empleado:

- **systemctl y journalctl**

La herramienta systemctl es una utilidad que se encarga de examinar y controlar el sistema systemd y el administrador de servicios. Es una colección de bibliotecas, utilidades y daemons de administración de sistemas que funcionan como sucesores del daemon System V init. Los nuevos comandos systemctl han demostrado ser bastante útiles para administrar los servicios de un servidor. Proporciona información detallada sobre servicios systemd específicos y otros que se utilizan en todo el servidor. Por lo que, gracias a esta herramienta se pueden definir servicios que se lancen cuando el sistema se inicie o cuando eventos ocurran.

Además existe una herramienta muy ligada a systemctl llamada journalctl, el cual permite hacer una lectura de las salidas de texto, activación/desactivación, etc. Ordenados por tiempo para así hacer un debug perfecto de la máquina.

En definitiva, esta herramienta será imprescindible para que una vez iniciada la máquina, esta se active de forma independiente, lanzando los roslaunch de los necesarios e instanciando los códigos que hagan falta para que la estación robótica este lista y preparada para moverse.

- **SFTP y SSH**

SSH o Secure Shell es un protocolo de comunicación de red que permite que dos ordenadores se comuniquen y compartan datos. Una característica inherente de ssh es que la comunicación entre las dos computadoras está encriptada, lo que significa que es adecuado para su uso en redes inseguras. De esta forma esta herramienta es muy útil para acceder a ordenadores remotos y poder controlar su terminal.

Por otro lado, SFTP (Protocolo de transferencia de archivos SSH) es un protocolo de archivo seguro que se utiliza para acceder, administrar y transferir archivos a través de un transporte SSH cifrado. La diferencia de SFTP y SSH es que SFTP permite abrir la herramienta de Ubuntu de archivos y navegar con la interfaz visual sobre estos, además de abrir archivos con editores de texto para programar o cambiar código, así como descargar o subir archivos de una forma sencilla y rápida.

3.3 Herramientas empleadas

En este apartado, se listan no todas pero muchas de las herramientas que se han empleado y se ha apoyado para hacer este proyecto realidad.

3.3.1 Aplicaciones de modelado 3D

A la hora de hacer el diseño la estructura del robot se han utilizado 2 aplicaciones diferentes para el diseño 3D de elementos. En primer lugar se ha utilizado Autodesk Inventor Professional, que es una herramienta de autodesk para hacer modelos 3D. Esta herramienta es de alta calidad, permitiendo una interfaz de diseño muy robusta y fácil de emplear. Sin embargo el problema de esta herramienta es que principalmente es de pago, por lo que resulta bastante restrictiva, y hubo un momento que no pudo emplearse. Cuando se llegó a este momento, se decidió emplear FreeCAD. FreeCAD es una herramienta de diseño 3D Open Source, que implementa multitud de herramientas para realizar este cometido, con el añadido de que al ser abierto, la comunidad puede crear plug-ins y añadir nuevas características y funcionalidades. El punto negativo de este software sin duda es la menor facilidad del uso del software así como su poca robustez, "crasheandose" el programa de vez en cuando o siendo excesivamente lento para hacer algunas tareas.

3.3.2 Robotic toolbox Peter Corke (Python)

A la hora de realizar la cinemática directa e inversa del robot hexápodo, se empleó esta herramienta para verificar que los cálculos estuvieran bien realizados. Esta toolbox es la versión de Python de la famosa Robotic Toolbox de Matlab, la cual añade multitud de funcionalidades y algoritmos propios del control de robots. Debido a que la mayoría del código de programación va a ser escrito en Python esta librería resultó muy útil en el desarrollo del proyecto.

Cabe destacar que en el propio GitHub del proyecto, existe una subcarpeta llamada wiki, en la cual hay una documentación bastante detallada y aclaradora de como emplear dicha librería.

3.3.3 git

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

Esta herramienta resultó bastante útil a la hora de realizar el proyecto, pues gracias a Git-Hub, se creó un repositorio online público donde se han ido subiendo las sucesivas versiones y actualizaciones del código, haciendo que la subida del código al ordenador del hexápodo fuera muy simple pues únicamente se debía descargar la nueva versión del repositorio que previamente había sido modificada en un ordenador de sobremesa. Además su funcionalidad de control de versiones en algunas ocasiones fue un salvavidas, pues se pudo recuperar códigos antiguos que funcionaban a diferencia de las sucesivas nuevas versiones que se fueron añadiendo.

3.3.4 Wekan

Wekan es una aplicación de código abierto basada en el concepto Kanban un término de origen japonés que literalmente significa “tarjeta” o “señalización”. Este es un concepto normalmente relacionado con el uso de tarjetas (post-it y otros) para indicar el progreso de los flujos de producción en proyectos.

3.3.5 MeshLab

MeshLab es un software de código abierto para el procesamiento y edición de mallas triangulares no estructuradas en 3D. Este sistema tiene varias herramientas para la edición, optimización, inspección y 'rendering' de malla. En este caso se utilizó esta

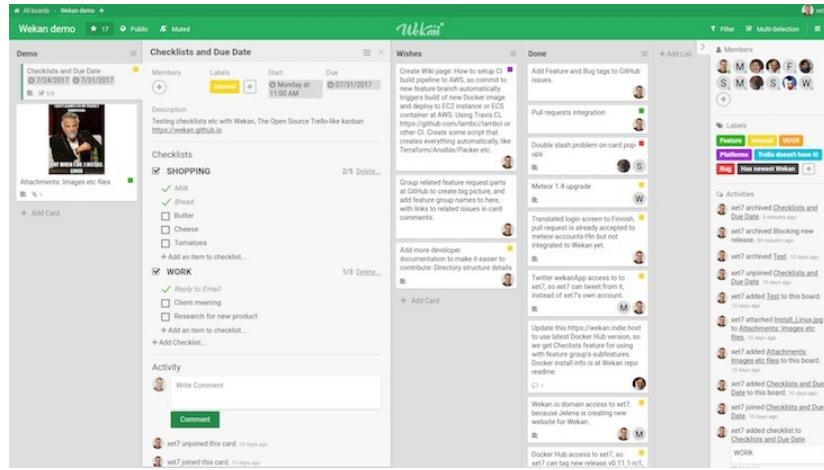


Figura 3.15: Captura de la herramienta Wekan (autor desconocido).

herramienta para calcular las matrices de inercia de los modelos STL de las partes del robot hexápedo.

3.3.6 Impresión 3D

para crear las piezas de los eslabones del robot, así como la estructura, el enclosure y demás partes físicas se ha empleado tecnología de impresión 3D, pues esta tecnología ofrece una gran versatilidad y un prototipado rápido y barato, siendo idóneo para la creación de diseños de robots en contraposición de el mecanizado de piezas metálicas que en su mayoría suelen ser más pesadas.

Para la impresión de las piezas se usaron dos tipos de materiales principalmente, el PLA y el PETG. En los primeros prototipos y piezas del robot se imprimió todo con PLA, esto se debe a que aunque es un material con poca resistencia mecánica y bastante flexibilidad tiene la ventaja de que es biodegradable, por lo que por el bien del planeta y del ecosistema se debe primar este material a diferencia de otros que tardan mucho más tiempo en degradarse. Sin embargo, para el diseño final se empleó PETG, pues aunque es más difícil de imprimir tiene mejores especificaciones técnicas que el PLA, siendo más resistente a golpes, torsión, menos elástico y con un punto de fusión

elevado. De esta forma este material es idóneo para hacer post-procesado de piezas, limado e incluso la realización de agujeros o cortes.

Además se utilizó filamento flexible o TPU para hacer unas gomas de acople ?? y las patas del hexápodo.



Figura 3.16: Imagen de la impresora utilizada para realizar la impresión de todas las piezas (fuente propia).

4 Desarrollo

El desarrollo del proyecto debido a que consta de partes muy diferenciadas se puede dividir en diferentes apartados o fases. A continuación se listarán todas estas fases explicando el desarrollo y la experimentación realizada hasta llegar al último diseño funcional del robot.

4.1 Fase 1: Diseño y construcción del robot hexápodo

Una vez estudiado el estado del arte, así como los diseños y proyectos que se han realizado y se puede tener constancia, se empezó con la fase de diseño de la estructura del robot. Para esto se tuvo en cuenta una gran cantidad de diseños de robots hexápodos, y se cogió lo mejor de cada uno, teniendo en cuenta colocación de la electrónica, motores, forma de los eslabones, cuerpo central. Una vez terminado este proceso se empezó a diseñar las diferentes partes que tiene un robot hexápodo.

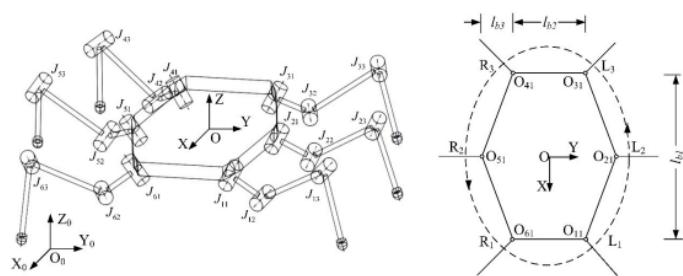


Figura 4.1: Forma típica de un robot hexápodo [9]

Como se puede ver en la imagen 4.3, la forma de un hexápodo suele cumplir siempre

dicha estructura, siendo en primer lugar simétrica únicamente en el eje X del robot. Mientras que las distancias l_b1 , l_b2 , l_b3 son proporcionales a la distancia final del fémur de las patas 4.2, siendo un hexágono regular cuando esta es reducida y pareciéndose a un rectángulo cuando es más alargada. A continuación se explicará el por qué y la estructura de diseño tanto de las patas, el cuerpo y demás partes interesantes que han aportado novedades al estado del arte.

4.1.0.1 Diseño de las patas de un robot hexápodo

La pata de un hexápodo, según el modelo biológico se divide en 4 partes principalmente [9], siendo en primer lugar el tarsus, seguido de la tibia, el fémur y por último la coxa. Esta estructura se puede ver en la imagen 4.2, donde además se ha añadido el diseño último realizado, representando además las partes de la pata bio-inspirada. Véase como la tibia realmente forma parte también del tarsus, siendo su unión rígida.

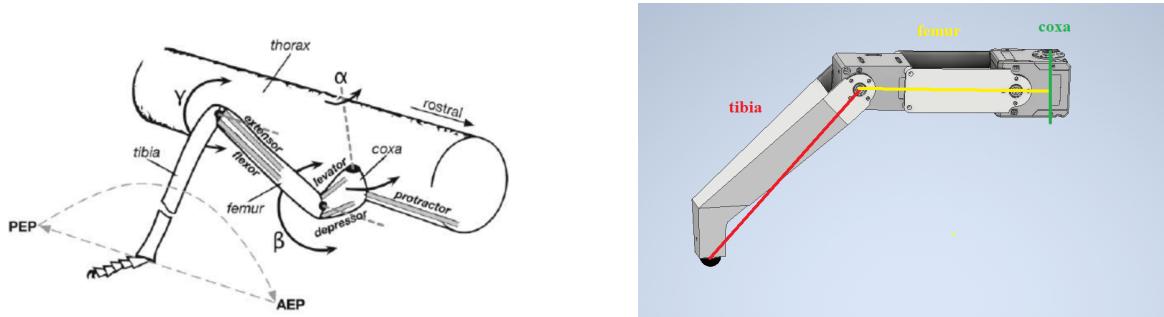


Figura 4.2: Pata bio-inspirada en contraposición con el diseño final de la pata, con sus motores (imagen N° 1 [9], imagen N°2 fuente propia).

Además se puede observar como en la figura 4.2, el coxa es el propio motor, por lo que se reduce a cero esta distancia, siendo una configuración que reduce peso, pues en vez de emplear dos motores se utiliza uno, reduciendo además un eslabón.

4.1.0.2 Diseño del cuerpo

Para el diseño del cuerpo se creó un compartimento, con forma de caja hexagonal, de esta forma, en la unión de la superficie inferior y superior pueden ir anclados los servomotores, dejando dos compartimentos, el del interior de la caja, (para todo lo relacionado con los drivers de los motores y su alimentación) y la parte superior de esta, en la cual se colocaría tanto las baterías como el PC, en este caso la Raspberry. Además se realizó una tapa, de forma que tapara a la vista todo el cableado y dejara un aspecto visual agradable. Como se puede observar en el diseño final de la figura, se ha añadido un agarre en la parte de arriba para poder coger el hexápedo y desplazarlo de forma cómoda, así como un hueco para colocar un interruptor que apague el robot.

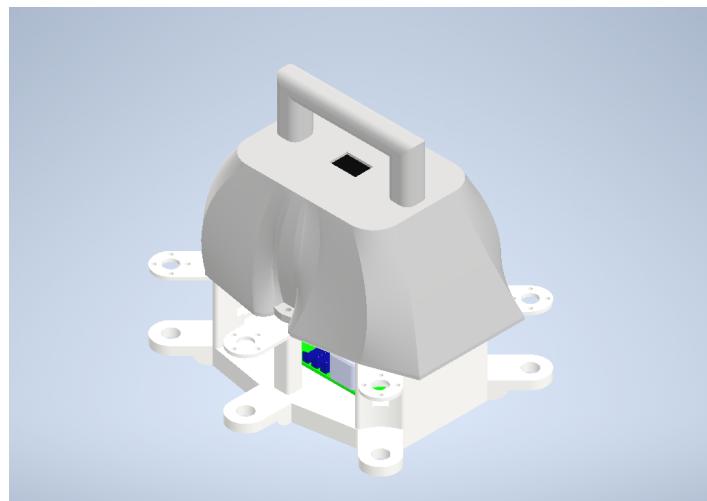


Figura 4.3: Montaje del cuerpo del hexápedo más la tapa que encapsula el controlador (fuente propia).

4.1.0.3 Diseño de las patas

Para poder sensorizar el extremo de las patas y así poder detectar el suelo y dotar de la capacidad de moverse en terrenos abruptos al robot hexápedo se decidió colocar switches en los extremos de las patas, de esta forma, la pata o tibia, se ha diseñado con un agujero central, así como un hueco para colocar los interruptores. Además en

la parte de los interruptores se imprimió con filamento flexible, una terminación con forma semiesférica.

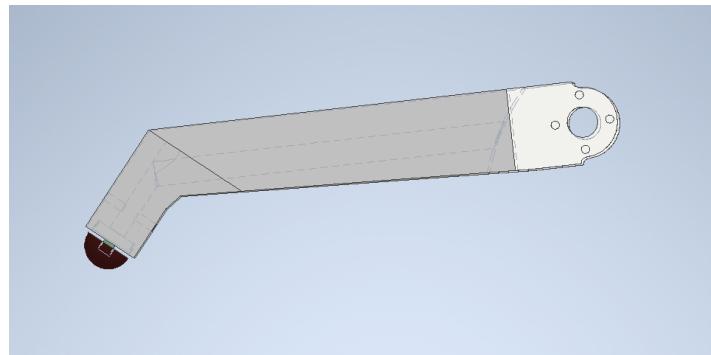


Figura 4.4: Diseño de la pata mas el anclaje del botón (fuente propia)

4.1.0.4 Unión de las articulaciones con los ejes motores

Debido al hecho de que los motores XL430-W250 y 2XL430-W250, en el eje motor, solo dispone de movimiento de un lado se tuvo que diseñar una forma de agarrar ambos lados del motor a los eslabones de forma que permitieran el giro sin ningún tipo de rozamiento. para ello se realizaron diferentes prototipos hasta el último que fue el que se muestra en la figura 4.5

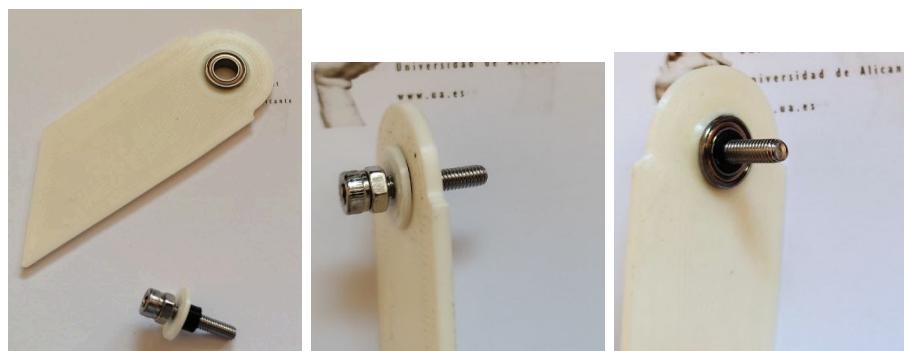


Figura 4.5: Diseño acople de los ejes con su respectivo eslabón (fuente propia).

Como se puede observar, el funcionamiento es el siguiente: Hay un tornillo que se acopla al eje no motriz de motor, a continuación se coloca una arandela impresa en

3D, y a continuación un rodamiento radial, que se sujeta al tornillo gracias a una goma impresa en 3D con un material flexible, de esta forma, el tornillo se introduce en la goma y la goma en el eslabón a conectar, colocando finalmente otra tuerca y un tornillo final para asegurar que no se mueva la unión. Finalmente, gracias a esto se consiguió conectar el eje del motor a ambos lados del motor, haciendo que la distribución de fuerzas sea homogénea y eliminando posibles roturas por esfuerzos debido a momento.

4.1.0.5 Diseño final

Sumando todos los diseños así como rediseños y cambios en diferentes versiones se ha acabado diseñando la estructura robótica mostrada en la figura 4.6, que es el fruto de la 4 versión modelada e impresa del robot.

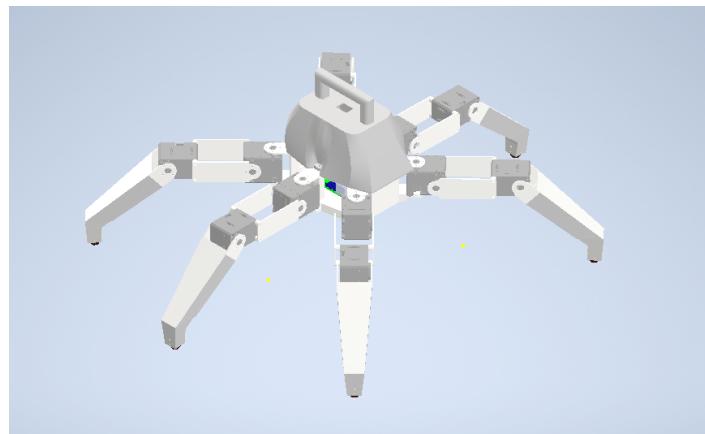


Figura 4.6: Diseño final del hexápedo mostrado en la aplicación Autodesk Inventor (fuente propia).

Sin embargo, esta no fue la única versión del robot, sino la V3.2, ya que se tuvo 3 versiones muy diferenciadas en las que se cambiaron aspectos principales del robot, así como las patas, el cuerpo, en enclosure. Obteniendo finalmente el diseño de la figura 4.6, con leves modificaciones, por eso la 3.2. A continuación se muestra la primera versión de todas en contraposición con la última, con montaje eléctrico incluido.



Figura 4.7: Primer y último diseño enfrentados (fuente propia).

Una de las modificaciones finales substanciales que se realizaron fue la adición de un pequeño ventilador de 5V en la tapa del enclosure del hexápodo, debido a que al realizar varias pruebas se pudo detectar que al no estar ventilado llegaba a temperaturas algo elevadas. Una vez realizado este ajuste se consiguió tener una temperatura ambiental en el espacio del ordenador (generador de mayor temperatura).

4.1.1 Fase 2: Conexionado del robot hexápodo

Una vez montado el robot hexápodo, habiendo atornillado y acoplado todas las articulaciones de forma correcta, y colocado el driver de los motores en el compartimento interior del bloque central, se procedió a realizar el conexionado del robot, para ello se soldaron cables a los 6 diferentes botones colocados en cada pata, y estos a los GPIOS de la raspberry, utilizando una clavija secundaria para no soldar directamente en los pines de la placa. habiendo hecho esto, se procedió a continuación a realizar el conexionado de alimentación del sistema.

Para ello, el cable de alimentación, que debe de ser de 12V, se conectó por un lado al driver de los motores dynamixel, que precisa de 12V para su alimentación y por otro lado a un convertidor DC-DC, para reducir los 12V de entrada a 5 y así poder alimentar de forma correcta a la Raspberry. además se commuto el positivo principal de la conexión a un interruptor, para así poder apagar o encender el sistema de una forma sencilla.

Una vez conectado todo de forma correcta, se empezó a trabajar con el robot hexápodo haciendo pruebas conectado a una fuente de alimentación variable. Y gracias a esta herramienta, ya que se limita el voltaje a 12 voltios, se pudo obtener la intensidad pico cuando el robot se movía y por tanto la potencia gastada pico, siendo esta de 25W.

4.1.1.1 Cálculo de las baterías

Como ya se explicó en el apartado 2.2.3, las baterías normalmente empleadas para este tipo de robots son las Lipo, y dependiendo de su numero de celdas (cada una de 3,7V), se puede obtener diferentes voltajes, por lo que en este proyecto se colocó una de 3 celdas (11,1V), sin embargo, como se disponía de 2 baterías de este mismo tipo de 3000mAh se colocaron en paralelo para así sumar su capacidad y disponer de 6000mAh. Por lo tanto, teniendo el voltaje empleado, la capacidad y la potencia consumida se puede realizar un cálculo estimado de la autonomía del sistema.

De esta forma, siguiendo las siguientes fórmulas y calculos se consigue el tiempo en horas de autonomía estimada, teniendo en cuenta que se gasta siempre 25W y que el sistema tiene unas perdidas del 50%.

$$I = P/V$$

$$I_{real} = \frac{100 \cdot I}{100 - Perdidas}$$

$$I_{real} = 3.962A$$

De esta forma, si tenemos en cuenta:

$$Capacidad_{real} = I_{real} \cdot Horas \cdot 1000 = 3962mAh$$

$$\text{tiempo de autonomía} = \frac{23000}{3962} = 1,51H$$

Con las dos baterías de 3000mAh se obtiene una autonomía de 1 hora y media, lo cual es bastante aceptable.

4.1.1.2 Montaje final de las baterías

Finalmente, se diseñaron y se imprimieron unas torretas justo encima de la base, para así colocar abajo las baterías, una junto a otra y encima la Raspberry. como muestra la imagen 4.8.



Figura 4.8: Hexápodo finalmente montado con la circuitería conectada. (fuente propia).

4.1.2 Fase 3: desarrollo del software del robot hexápodo

Pese a que ya se empezó a escribir el código y a diseñar diferentes funcionalidades del hexápodo antes de realizar el montaje e imprimir las piezas, no fue ya cuando el hexápodo estaba 100% listo, cuando se empezó a desarrollar esta parte en profundidad.

Como se explicó en el apartado 3.2.1, Se ha elegido el framework de ROS para controlar y programar el robot hexápodo, por todas las herramientas y funcionalidades

que ofrece. Pero aunque ROS es muy potente, también tiene una curva de aprendizaje elevada, y más cuando se pretende crear un robot completamente nuevo e introducirlo en ROS. A continuación se explicará paso a paso que se hizo y que se necesitó aprender y resolver para poder tener funcional ROS como OS en la Raspberry.

4.1.2.1 Conexión de nodos de ROS mediante SSH

En primer lugar se buscaron formas de teleoperar ROS, debido a que desde un principio se iba a controlar el sistema robótico con una raspberry y no se esperaba que este tuviera mucha capacidad de computo para realizar controles de marcha y trayectoria complejos. Tras estar investigando se descubrió que ROS ofrece una funcionalidad de envío de mensajes con SSH. De esta forma, se crea un ROS node Master en el origen del controlador, en este caso en la raspberry, y un ordenador externo conectado a la misma red wifi, si se configuran diferentes parámetros en el .bashrc (lado del ordenador) y en el .profile (lado de la Raspberry pi) se puede hacer que ambos ordenadores compartan los mismos topics y se redirijan los mensajes que publiquen los nodos en estos. De esta forma se ha conseguido utilizar aplicaciones como Rviz, en un ordenador remoto mientras que en la raspberry esta localizado en ROS master pues es allí donde se comanda la consigna a los motores y se recibe información del driver U2D2.

4.1.2.2 Esquema de control y ejecución del robot

Como se ha ido adelantando en el apartado anterior, habrán dos líneas de ejecución si se desea teleoperar el robot hexápodo. Por un lado tendremos la Raspberry pi 4B, la cual hará las siguientes funciones:

- **Recepción del feedback de los motores**

En primer lugar se creará un servicio, con su respectivo fichero .srv, el cual cuando sea llamado haga un request al controlador U2D2 de los valores solicitados, pudiéndose requerir, tanto la velocidad, posición y torque realizados en ese momento, teniendo la posibilidad de recibir uno o todos a la vez. De esta forma se

llamará a este servicio con la frecuencia que se crea conveniente para hacer un control aceptable. Se debe tener en cuenta que la recepción de información de los motores tendrá un delay, por este motivo cuanta más información se requiera de los motores más tardará el servicio en dar respuesta.

- **Realizar comandos a los motores**

En segundo lugar se deberá comandar a los motores, consignas objetivo, dependiendo, obviamente del modo de funcionamiento, pudiéndose controlar tanto la posición, como la velocidad y el torque. Para realizar esta tarea se creará un nodo que se suscriba al topic de los comandos, de forma que cuando llegue un nuevo dato se ejecute la función que tiene ligada el nodo. De esta forma cuando ROS detecte que se ha obtenido un nuevo valor en el topic, rápidamente se llamará al código correspondiente para comandar esos datos a los diferentes motores. Para realizar esta tarea se deberá crear un tipo de mensaje .msg, el cual defina tanto el ID del robot a comandar como el valor en binario enviado.

- **Configuración de los motores**

Además de lo mencionado anteriormente, este sistema deberá tener una opción de configuración de los motores, para que de forma remota se pueda cambiar tanto el modo de operación, reiniciar los motores en caso de paro, cambiar los perfiles de velocidad y aceleración, límites de posición, velocidad y aceleración y múltiples configuraciones más que serán necesarias para hacer un control exitoso del sistema.

- **Recepción de las entradas GPIO de la Raspberry** Por último, el ordenador "local" deberá lanzar un código que lea en una frecuencia determinada la entrada de las inputs de los GPIO de la Raspberry, para poder tener tener el feedback de la posición contra el suelo del robot. Por lo que se creará un topic, y se estará

publicando de forma local continuamente su estado.

De esta forma, como se ha definido la estructura del control del robot, el hexápedo generará el Máster de ROS, sin embargo, en un ordenador remoto, se leerá el feedback completo de los motores y sus sensores, y respecto a todo esto se realizará el control del movimiento, empleando herramientas como RviZ, que no podrían emplearse en el ordenador local.

4.1.2.3 Conexión entre el driver U2D2, raspberry y enlace con ROS

Como ya se avanzó en el apartado 3.1.1.3, Dynamixel ofrece 2 librerías para poder hacer un control de los motores, Dynamixel SDK y Dynamixel Workbench. Siendo el segundo más sencillo de utilizar pues se pueden importar funciones y crear archivos .yaml con lo que configurar los motores, sin embargo, tras estar investigando y recibiendo algunos fallos, se detectó que el motor escogido 2XL430-W250 no es compatible con esta biblioteca, debido a que al haber dos ejes controlados por un mismo controlador interno emplean un mismo pin que se emplea en dicha librería, pudiéndose controlar únicamente un motor. Por este motivo se tuvo que utilizar Dynamixel SDK.

El principal problema de Dynamixel SDK es que simplemente ofrece herramientas de bajo nivel para poder transmitir información a los servos, herramientas para abrir los puertos y únicamente enviar información mediante ese canal a nivel de bytes. Además en la pagina web oficial y en la documentación existen códigos de ejemplo de su uso, y algunas funciones implementadas, sin embargo están desarrolladas de una forma muy pobre y hay muy pocas funcionalidades implementadas.

Por este motivo, se tuvo que realizar una librería propia en la que en cada método se modificaran configuraciones de los motores, un ejemplo de esto podría ser hacer un enable/disable de un motor. En este caso como se muestra en la siguiente imagen 4.9. Puede parecer no tan complicado, pero este proceso se complica cuando se deben enviar

diferente longitud de bytes, y realizar a mano todo este código es un proceso laborioso y costoso.

```
def torque_motor_group(self, array_ids, state):
    for id in range (6):
        # Enable Dynamixel Torque
        dxl_comm_result1, dxl_error1 = self.packetHandler.write1ByteTxRx(self.portHandler, array_ids[id], self.ADDR_TORQUE_ENABLE, state)

        if dxl_comm_result1 != self.COMM_SUCCESS :
            print("%s" % self.packetHandler.getTxRxResult(dxl_comm_result1))
            quit()
        elif dxl_error1 != 0 :
            print("%s" % self.packetHandler.getRxPacketError(dxl_error1))
            quit()
        time.sleep(0.005)
    if state:
        print("Torque enabled - DYNAMIXEL motor group with ids = ",array_ids)
    else:
        print("Torque disabled - DYNAMIXEL motor group with ids = ",array_ids)
```

Figura 4.9: Código de ejemplo en python para activar un motor (fuente propia).

Además, el proceso de comandar y recibir información se realizó en C++, ya que como se trata de una funcionalidad que se va a estar empleando constantemente y se va a requerir de la máxima velocidad y eficiencia, se eligió este lenguaje de programación frente a Python, lenguaje en el que se ha programado prácticamente todos los módulos y los códigos.

4.1.2.4 Distribución de paquetes usualmente empleada para sistemas robóticos en ROS

Finalmente el hexápodo se ha llamado H_robix, por lo que sus consecuentes paquetes tendrán este mismo nombre seguido del nombre que haga referencia a su funcinalidad. Este tipo de nomenclatura se ha empleado ya que en ROS, se utiliza como estándar para crear robots e introducirlos en ROS. De esta forma se hace mucho mas sencillo entender la distribución de paquetes y su uso.

- **h_robix_control**

Los paquetes de control siempre hacen referencia a todo lo relativo con la comunicación con el hardware real del robot, así como motores, sensores, displays, etc. El objetivo de ROS en todo momento es abstraer todos estos elementos de forma

que una vez realizado los paquetes y hecha la comunicación de forma exitosa, únicamente se deba referir tanto a los topics como a los nodos creados.

Este paquete tiene las siguientes subcarpetas las cuales tienen código referente a la configuración de los motores, recepción y envío de información de los motores, y uso de los GPIO del robot. Además en estos paquetes siempre están los roslaunch del sistema. En este caso estará el del driver del hardware y el del proceso de teleoperación.

- **launch**
- **msg**
- **srv**
- **src**
- **h_robix_description**

En los paquetes definidos como description, siempre se define el urdf, además de incluir los ficheros roslaunch necesarios para lanzar la herramienta de visualización. Además como es natural, en este paquete se encontrarán los archivos .stl que modelen la forma del robot.

 - **launch**
 - **meshes**
 - **urdf**
- **h_robix_gazebo**

En los paquetes con el subnombre de gazebo siempre se define tanto el launch del simulador como los ficheros .yaml de configuración de los controladores de cada articulación (dentro de config). Gazebo al igual que Rviz utilizará el urdf del robot, leyéndolo del paquete de description.

- **config**
 - **launch**
-
- **h_robix_movement** En este otro paquete estará todo el código referente a la configuración y al movimiento del robot, dentro de src, con el fichero en este caso llamado hexapod_class.py. Además se han incluido otras carpetas que contienen tanto información y ejemplos del uso de tf así como los cálculos de la cinemática inversa/directa de robot empleando la librería de la robotic toolbox de Peter Corke.
- **kinematics**
 - **src**
 - **tf_examples**
-
- **h_robix_sh_codes** En este último paquete creado, se incluirán diferentes archivos .sh para el funcionamiento del sistema. En este caso este paquete contiene únicamente el .sh de la creación del punto de acceso móvil wifi, al cual se conectará el robot y hasta que no reciba señal no se encenderá.
 - **rqt_virtual_joystick** Finalmente se ha añadido un paquete encontrado en internet creado por la comunidad de ROS. Este paquete implementa un joystick virtual que se usará para comandar posiciones objetivo al robot hexápedo.

4.1.2.5 Problemática encontrada para utilizar los GPIOs de la Raspberry con ROS

Un problema serio que se tuvo al estar programando el robot, fue la utilización de los pines GPIO de la Raspberry sin tener el acceso sudo. Finalmente se pudo resolver con una librería poco conocida que al parecer hace el bridge correspondiente para saltar la necesidad de tener acceso de súper usuario. la librería finalmente empleada fue pigpiod.

4.1.2.6 Utilización de servicios con la herramienta systemctl

Para que la herramienta de roslaunch en el ordenador local se lanzara siempre se creó un servicio con la herramienta systemctl para que así cuando el ordenador se conectara a internet ejecutara el comando de roslaunch y así encienda los motores, como el inicio de control de los GPIOS.

4.1.3 Fase 4: Control del robot

4.1.3.1 Cinemática del robot

Para poder convertir la posición articular de los motores de los robots en cartesiana y viceversa se tuvo que calcular tanto la cinemática directa como inversa de las extremidades del robot, siendo la misma para todas las patas.

Para realizar esta tarea existen dos diferentes aproximaciones, una es calcularla por métodos geométricos y otra por Algebraicos empleando algoritmos basados en las tablas de Denavit Hartenberg. Por un lado la utilización de D-H, tiene la ventaja de ser automático, sin embargo a la hora de realizar la cinemática inversa existe el problema de las indeterminaciones, que lo hacen un proceso más costoso, pues para hayar una de las infinitas posibles soluciones se debe iterar y esto resulta mucho menos eficiente que una cinemática inversa geométrica, que puede dar también varias soluciones, pero que en este caso por ejemplo, realizando la cinemática inversa de un brazo de 3 grados de libertad, (véase la figura 4.10) únicamente se tendrán dos soluciones, la solución codo arriba y codo abajo, por lo que será fácil elegir la deseada, siendo siempre codo arriba.

Por este motivo, aunque se ha realizado la cinemática por D-H también, a la hora de la verdad en el código implementado se ha utilizado la cinemática geométrica. A continuación se explicará que metodología se ha empleado para obtener dichas cinemáticas.

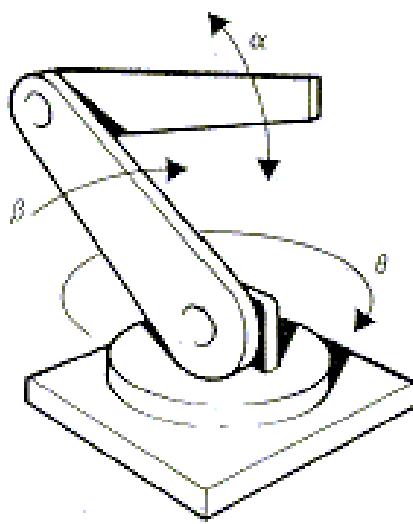


Figura 4.10: brazo robótico representado de las extremidades del hexápodo (autor desconocido).

- **Cinemática calculada por el método geométrico**

Este proceso se basa en encontrar un número suficiente de relaciones geométricas en las que intervendrán las coordenadas del extremo del robot, sus coordenadas articulares y las dimensiones físicas de sus elementos. Para así obtener la posición final en cartesiana según las posiciones articulares (cinemática directa) y las posiciones articulares con la posición en el mundo cartesiano deseada (cinemática inversa).

Este tipo de aproximación se puede realizar en brazos robóticos de pocos grados de libertad, pues al ir aumentando la complejidad se ve elevada, haciendo que emplear otros métodos de resolución sean mas interesantes.

En este tipo de resoluciones la cinemática directa siempre es mucho mas sencilla de calcular, pues simplemente se obtiene sumando los senos/cosenos de los ángulos de las articulaciones, sin embargo, el proceso de la búsqueda de la cinemática inversa es un proceso mucho más complicado y laborioso, que requiere de altos

conocimientos de relaciones geométricas para poder resolver el sistema.

En el GitHub, en el paquete de movimiento se puede encontrar en el código del "hexapod_class.py" un método que corresponde con la cinemática directa e inversa calculada.

- **Cinemática calculada por el método Algebraico**

La tabla de D-H es una forma de definir la forma de brazos robóticos de una forma normalizada, de tal forma que para conseguir las cinemáticas con este método se pueden emplear otros tipos de representaciones como puede ser el propio URDF del robot. Sin embargo, para calcular la cinemática del robot se empleo Denavit Hartenberg, pues la herramienta que se iba a emplear para dicho cálculo iba a ser la toolbox de robotics de Peter Corke de Python. De esta forma, esta librería implementa multitud de algoritmos para el control de robots.

θ_j	d_j	a_j	α_j
q1	0	0.024	-90.0°
q2	0	0.1	0.0°
q3	0	0.1505	0.0°

Figura 4.11: Tabla D-H creada para el cálculo de la cinemática directa e inversa (fuente propia).

Por lo que una vez representado el brazo robot, únicamente se deben emplear los comandos fkine para la cinemática directa e ikine para la inversa. Como este método al ser algebraico puede dar infinitos resultados existen diferentes algoritmos de cinemática inversa, para iterar X puntos al rededor de la posición escogida, para así obtener un punto accesible sin indeterminaciones, estando las siguientes: ikine_a, ikine_LM, ikine_LMS, ikine_min.

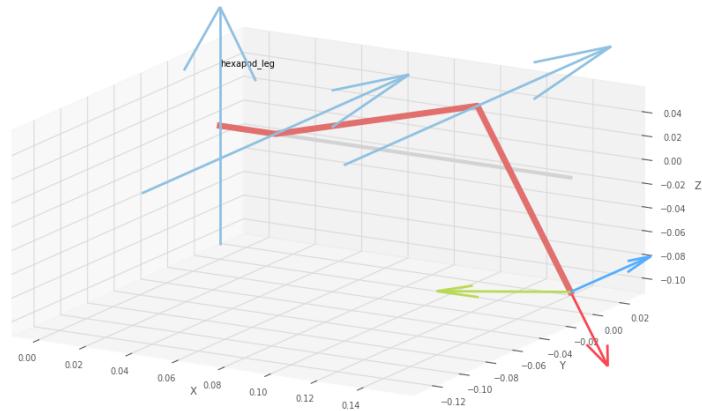


Figura 4.12: Imagen de la representación del sistema creado aportada por el plot de la robotic toolbox (fuente propia).

4.1.3.2 Diseño de la trayectoria de un movimiento usual

Para generar una trayectoria para una extremidad, se planteó un enfoque distinto al usualmente empleado. En este caso lo que se va a realizar es definir el espacio de trabajo de cada articulación, este naturalmente será el de un brazo de 3 GDL, teniendo forma de esfera segmentada por el centro siendo este la base del robot.

Una vez definido este espacio se va a obtener el espacio a cierta altura, considerada la altura usual de andado (donde estarán las patas tocando el suelo en referencia a la posición de la base del robot). Sin embargo, el hecho de posicionar el extremo de las patas muy alejadas del centro del robot, hace que por momento, se pueda realizar muy poca fuerza, no siendo útil para levantar el peso del robot, por otro lado, acercar mucho la articulación al centro de la base puede repercutir en choques con la propia estructura. Por estos motivos, se definió una esfera en el eje central del espacio de trabajo, en la que las patas siempre estarán posicionadas en esa zona.

De esta forma, como se puede ver en la figura 4.13 si en el joystick de teleoperación se indica que se quiere ir en una dirección concreta y con un módulo concreto, el punto objetivo será igual al punto en la misma posición de la circunferencia generada en la

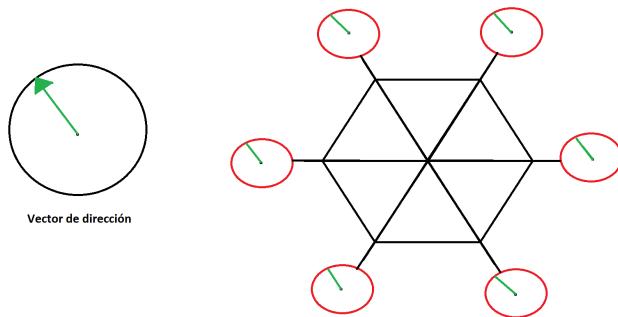


Figura 4.13: Explicación de la obtención del punto objetivo según la dirección de movimiento (fuente propia).

base home de cada pata. De esta forma se consigue una generación de trayectorias bastante simple que permite el movimiento en cada dirección e incluso el giro sobre sí mismo, siendo este si el vector de dirección e cada pata es tangente a la circunferencia del centro del robot.

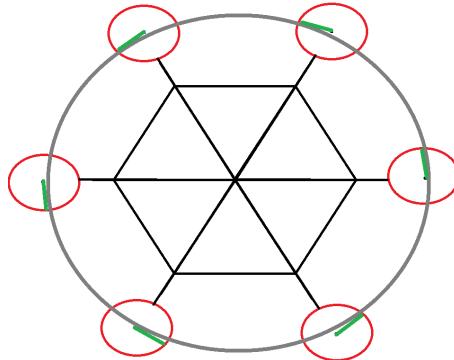


Figura 4.14: Giro sobre sí mismo (fuente propia).

Además cabe destacar que la trayectoria creada sera una compuesta por dos puntos, el punto final calculado como se acaba de explicar y un punto intermedio con un llamado "hop" o elevamiento de la pata, para así evitar u esquivar posibles elementos en el suelo.

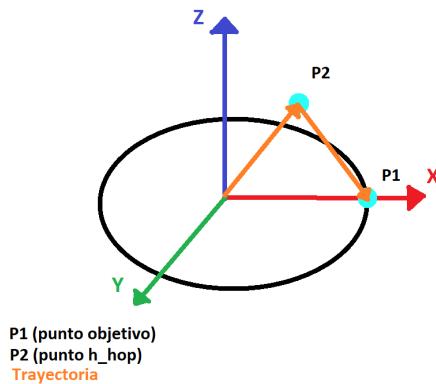


Figura 4.15: Definición del punto intermedio con una altura llamada hop (fuente propia).

4.1.3.3 Odometría

Teniendo la cinemática directa e inversa calculada y referenciada los puntos de los extremos del robot, calcular el desplazamiento de este es muy sencillo, pues será el módulo de desplazamiento del vector explicado en la sección de generación de trayectorias.

4.1.3.2.

4.1.3.4 Diferentes tipos de control según el modo operativo de los motores

Según el modo de ejecución de los motores se pueden emplear varios tipos de control, siendo los siguientes:

- **Control en posición directo** Este control es el más utilizado, pues es el más simple de realizar, ya que realmente no implica realizar ningún tipo de control, pues el control de trayectorias se realiza dentro del hardware de los motores. Pudiendo modificar valores como la rampa de velocidad y aceleración así como los valores del PID que lo modulan. Este tipo de control tiene resultados buenos, sin embargo, tiene la problemática de no ser reactivo, esto significa que en el caso de tener sensores en las patas este tipo de control lo que deberá hacer es ir reduciendo la posición final de las patas hasta que se cumpla la condición de que se detecte la colisión. Esto supone realizar trayectorias no continuas en el

espacio, haciendo que se hagan continuamente trayectorias reposo a reposo continuamente.

Además este problema también se observa cuando se realiza la trayectoria descrita en el apartado 4.1.3.2. Si se quiere realizar una trayectoria pasando por 2 puntos, el sistema lo que hará es ir a un punto, realizar un reposo y continuar hasta el último punto, siendo muy poco eficiente y pudiéndose mejorar con creces.

- **Control en velocidad** En este modelo de control, normalmente empleado con controladores cinemáticos, con un feedback tanto de la posición como de la velocidad se consigue hacer un control comandando la velocidad. Sin embargo, para hacer un control de este tipo no vale como en el de posición, que únicamente se debe enviar las posiciones objetivo, sino que se debe hacer una interpolación de la trayectoria, pudiéndose usar técnicas como splines, interpoladores polinómicos o 3-4-3. De forma que estos algoritmos generen una matriz de puntos que sobre una ley temporal definen la trayectoria a realizar, siendo el controlador el que con un control del error va aumentando o reduciendo la velocidad, según como se ha descrito en el interpolador.

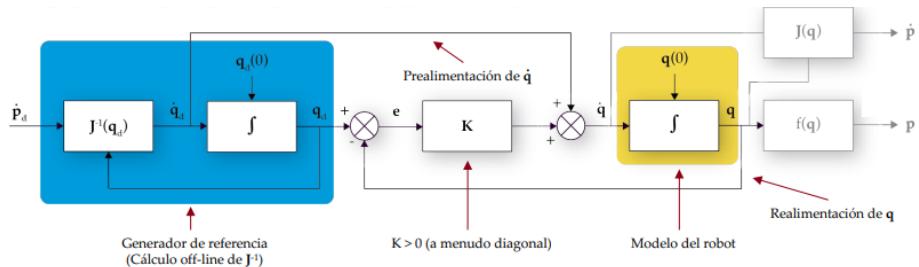


Figura 4.16: Transparencia de la asignatura de control de robots donde se muestra un bucle de control para un controlador cinemático articular (transparencias de la asignatura de control de robots de el grado de Ingeniería robótica de Alicante).

- **Control en torque**

Este último bucle de control lo que ofrece es tener un control absoluto del movimiento del robot, haciendo que se anule el controlador interno de los motores. Este tipo de control normalmente no se emplea, ya que precisan de un controlador dinámico que modele la estructura física del brazo robótico a controlar, sin embargo en este caso será bastante útil debido a un problema con el hardware.

El problema en cuestión es que aunque en las especificaciones de la pagina web de dynamixel sale publicado que el par máximo de los motores es de 1.4Nm, esto solo se puede realizar durante momentos, pues si se prolonga un alto torque los motores entran en alarma debido a una sobrecarga, causada por un sobrecalentamiento de los motores. Por este mismo hecho, se tiene el actual problema que el robot no es capaz de levantarse con un control en posición normal, y aunque no se ha terminado de probar un control en velocidad se cree que tampoco sería exitoso.

La forma de abordar el problema sería el siguiente: realizar un control en torque que cuando se requiera de un aumento de la posición en el eje Z cuando el robot esta en el suelo (peor condición) realice una tarea de levantamiento en el que de forma organizada todos los motores realicen un torque máximo durante un instante de tiempo muy reduciendo, pudiendo usar así el torque máximo sin la posibilidad de entrar en fallo. Debe destacarse que los motores según el momento que pueden realizar son capaces de levantar el peso del robot (con baterías de 2,3 kg aproximadamente). Ejemplo de esto fueron múltiples pruebas que se realizaron al principio del diseño del software en las cuales si se levantaba al robot con un control en posición comandando micro aumentos de la posición el robot podía levantarse sin problemas, pero que de otra forma le era imposible.

4.1.3.5 Ajuste del baudrate y demás configuraciones de los motores

Tras estar realizando pruebas se pudo comprobar como al modificar el baudrate de los motores se podía obtener información de estos a una velocidad mucho mayor, siendo la velocidad más alta que no ocasionaba ningún error de lectura 3000000.

5 Resultados

Finalmente, se ha conseguido realizar un control en posición en diferentes marchas, empleando las explicadas en el apartado 2.1.5.3 siendo trippod, wave y ripple. El resultado del movimiento así como todo el código implementado se puede encontrar en el siguiente repositorio de GitHub: <https://github.com/javi-desp/H-Robix.git>. Sin embargo, un gran problema ha sido el hecho de poder levantar desde el suelo al hexápodo, que no se ha podido resolver hasta el momento. Este problema se explica en el apartado 4.1.3.4, así como las ideas para solucionarlas, que serían principalmente reducir el tamaño de las articulaciones o realizar un control dinámico que controle de forma exacta el par efectuado por los actuadores.

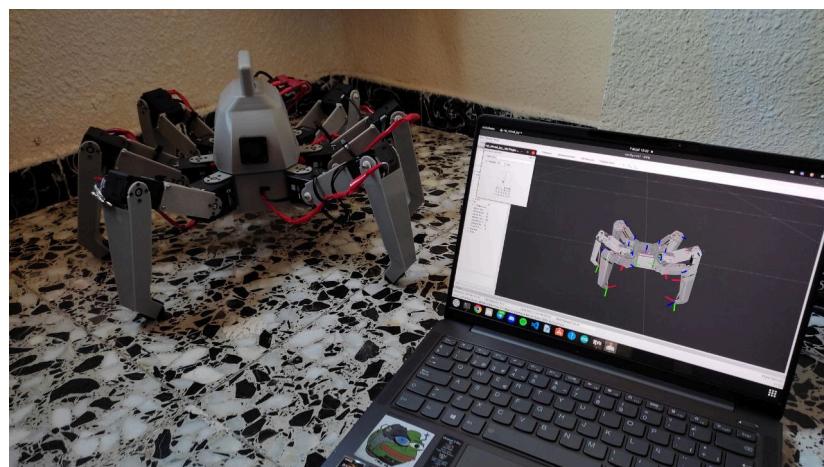


Figura 5.1: Imagen del robot H_robix en funcionamiento con la visualización en tiempo real mostrada en un ordenador remoto (fuente propia).

Sin embargo, no se debe poner el punto de vista en este último punto, pues se ha conseguido controlar de forma perfecta los motores con una frecuencia de actualización muy reducida, así como la simulación y visualización en 3D del diseño del robot e implementado en ROS. Siendo uno de los primeros proyectos que realizan esta tarea en todo el mundo. Además se han propuesto técnicas de control de trayectorias novedosas en este campo y diferentes diseños 3D del robot que bajo el punto de vista del autor resultan muy interesantes y que aportan mucho valor al proyecto.

6 Conclusiones

Como conclusión se puede decir que pese a algunos inconvenientes encontrados por la marcha se ha conseguido construir un robot hexápodo real e implementarlo en ROS, tarea que muy poca gente ha realizado, y que no está nada guiada. Aunque solo se ha conseguido hacer un control por posición básico, este proyecto deja un sistema robótico plenamente funcional para así estudiar controles como los mencionados en el apartado 4.1.3.4. Así como posibles algoritmos de aprendizaje por refuerzo. Además de todo esto, diseñar, construir programar y además controlar un robot y más siendo sobreactuado como un hexápodo ha sido una tarea muy didáctica, pues se ha podido utilizar multitud de conocimientos aportados en la titulación así como aprender muchos otros nuevos por cuenta propia.

Por todos estos motivos, pese a terminar el proyecto con un sentimiento agridulce por las múltiples funcionalidades que no se han podido implementar, este proyecto ha sido todo un reto, y ver como se puede crear algo de la nada y más sin tener mucha información a la que referirse hace sentirse a uno mismo orgulloso.

Bibliografía

- [1] *Introduction to Autonomous Mobile Robots*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2 edition, 2011.
- [2] Priyaranjan Biswal and Prases K. Mohanty. Development of quadruped walking robots: A review. *Ain Shams Engineering Journal*, 12(2):2017–2031, 2021.
- [3] François Ubald Brien. Midred the hexapod. <https://hackaday.io/project/164758-mildred-the-hexapod>. [Online; accessed 19-July-2008].
- [4] Rodney A. Brooks. The behavior language; user's guide. *MIT A.I. Memo 1227*, page 34, 1990.
- [5] Tossen Robotics Company. Forum Tossen Robotics. <http://forums.trossenrobotics.com/content.php>. [Online; accessed 19-July-2008].
- [6] Dynamixel. Manual 2XL430-W250 . <https://emanual.robotis.com/docs/en/dxl/x/2xl430-w250/>. [Online; accessed 19-July-2008].
- [7] Dynamixel. Manual XL430-W250 . <https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/>. [Online; accessed 19-July-2008].
- [8] Cynthia Ferrell. A comparison of three insect-inspired locomotion controllers. *Robotics and Autonomous Systems*, 16(2):135–159, 1995. Moving the Frontiers between Robotics and Biology.

- [9] Dariusz Grzelczyk, Bartosz Stańczyk, and Jan Awrejcewicz. *Power consumption analysis of different hexapod robot gaits*, pages 197–206. 12 2015.
- [10] Kåre Halvorsen. Zenta robotics blog. <http://zentasrobots.com/mx-phoenix-hexapod/>. [Online; accessed 19-July-2008].
- [11] Samuel Fernández Iglesias. Locomoción bípeda del robot humanoide nao. 1:1–114, 2009.
- [12] Mindaugas Luneckas, Tomas Luneckas, Jonas Kriauciūnas, Dainius Udris, Darius Plonis, Robertas Damaševičius, and Rytis Maskeliūnas. Hexapod robot gait switching for energy consumption and cost of transport management using heuristic algorithms. *Applied Sciences*, 11(3), 2021.
- [13] Mario Andrei Garzón Oviedo. Estrategias bio-inspiradas para locomoción de robots Ápodos. 1:1–153, 2011.
- [14] Ping Qi. Task offloading and scheduling strategy for intelligent prosthesis in mobile edge computing environment. *Wireless Communications and Mobile Computing*, 2022:1–13, 01 2022.
- [15] Firas Raheem and Hind Khaleel. Static stability analysis of hexagonal hexapod robot for the periodic gaits. 1414, 09 2014.
- [16] Mircea Nițulescu Sorin Mănoiu-Olaru. Basic walking simulations and gravitational stability analysis for a hexapod robot using matlab. 1:12, 09 2011.
- [17] 1995) (Wilson, 1966) for (Ferrell. Insects and Robots. <https://insectsandrobots.weebly.com/>. [Online; accessed 19-July-2008].
- [18] RyuWoon Jung TaeHoon Lim YoonSeok Pyo, HanCheol Cho. *ROS Robot Programming*. 2017.

- [19] Alessio Zamparelli, Nicola Scianca, L. Lanari, and Giuseppe Oriolo. Humanoid gait generation on uneven ground using intrinsically stable mpc. *IFAC-PapersOnLine*, 51:393–398, 01 2018.