Universidad Carlos III de Madrid

Escuela Politécnica Superior

Grado en Ingeniería Electrónica

Industrial y Automática

Trabajo Fin de Grado

OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA

Autor: Mario Almagro Cádiz

Tutor: Alberto Valero Gómez

Leganés, Septiembre 2012

AGRADECIMIENTOS

Me gustaría agradecer su incansable perseverancia a mis padres, que nunca desistieron en recordarme que el proyecto tenía fecha de presentación, además de todas las veces que me recogieron o llevaron a la nave de la universidad.

Debo dar las gracias por todo el apoyo y la ayuda prestada en los momentos más difíciles en este proyecto, así como por sus innumerables ideas, a mi tutor Alberto.

Me gustaría agradecer también su apoyo y motivación, sobre todo en aquellas tardes interminables de impresión, a una persona muy especial. Gracias Isa.

Por último, quisiera nombrar a mi compañero, Alejandro. Espero que tenga suerte tanto en su vida profesional como sentimental.

Gracias a todos.

RESUMEN

El presente documento recoge el desarrollo y la documentación del proyecto "*OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA*". En este se realiza una explicación detallada de la nueva herramienta de diseño de componentes mecánicos *OOML*, implementada siguiendo el paradigma *Open Source*, y de su validación en el diseño de *El Mars*. Para ello, se disponen gráficos y capturas de pantalla, además de proporcionar planos y comentarios, ampliando la información transmitida.

Se puede dividir el proyecto en varios puntos:

1) Inicialmente, se introduce al lector en la corriente *Open Source*, destacando sus ventajas y su aplicación, cada vez más extendida en robótica, así como el cambio asociado en el modo de trabajo.

2) Además, se aborda la herramienta *OpenSCAD* como medio para generar archivos *STL*, junto con la tecnología de las impresoras 3D, mediante las cuales se implementará el prototipo descrito.

3) Tras esto, se justifica la aparición de esta herramienta y se discute sus aportaciones al ámbito científico, así como sus inconvenientes.

4) En los siguientes apartados se introduce una breve explicación sobre el manejo y la utilización de *OOML*, del mismo modo que se analiza su estructura y diseño.

5) Por otro lado, se evalúa *OOML* desarrollando un prototipo a partir de esta biblioteca, destacando su complejidad mecánica y el conjunto de opciones que se han barajado. Se detallan las características logradas de este diseño, su capacidad de adaptación al terreno, así como su mecanizado en el mundo físico.

6) Se detallan todos los procesos para la implementación del prototipo: la impresión del mismo, la instalación de la electrónica y la programación necesaria para el control.

7) Por último, este documento finaliza con una serie de ideas sobre posibles las mejoras y actualizaciones, tanto para la herramienta de programación *OOML* como para los diseños mencionados, que podrán ser parte de un trabajo futuro.

This document describes the development and the documentation of "OOML C++ LIBRARY FOR MECHANICAL DEVELOPMENT APPLIED TO ROBOTICS" project. Here it's made a detailed explanation of the new mechanical components design tool OOML, implemented by following the Open Source paradigm, and its validation in the design of Mars. To this end, it's included pictures and screenshots, besides providing planes and comments, extending the transmitted information.

The project can be divided into several points:

1) *First, reader is introduced into the Open Source current, emphasizing its advantages and its application, increasingly widespread in robotics, well as the associated change in the working mode .*

2) *Also, the OpenSCAD tool is covered as a way to generate STL files, along with 3D printer technology, whereby the prototype described is implemented.*

3) *After, the appearance of this tool is justified and their contributions to the scientific field are discussed, as well as its disadvantages.*

4) *In the following sections, a brief explanation about the management and OOML use is introduced, in the same way its structure and design is analyze.*

5) *By the way, OOML is evaluated by developing a prototype from this library, highlighting its mechanical complexity and the set of options has been considered. The achieved features, its ground adaptability, as well as its machining in the physic world are detailed.*

6) *All processes for implementation of the prototype are detailed: its printed, the installation of the electronics and the necessary programming for the control.*

7) *Finally, this paper concludes with a number of ideas about the improvements and actualizations for the programming tool OOML and the mentioned designs, which may be part of a future work.*

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

1 INTRODUCCION

Este proyecto tiene como fundamento dos propósitos:

1) El desarrollo de una nueva herramienta de diseño mecánico, basada en el lenguaje *C++*, que facilite a los usuarios el intercambio de diseños, además de adquirir toda la potencia de este lenguaje.

2) La implementación de un prototipo considerablemente complejo usando esta herramienta, demostrando así su potencialidad.

El creciente aumento de la presencia de las impresoras 3D ha impulsado la distribución y evolución de diseños 3D, con la aparición de repositorios como *thingiverse*. Este efecto plantea la necesidad de disponer de una herramienta que facilite la comprensión y modificación de los diseños para optimizar la colaboración. Bajo esta necesidad aparece *OOML*.

Oriented-Object Mechanical Library (*OOML*) consiste en un conjunto de bibliotecas de libre distribución. Estas se han elaborado conforme al modelo de la *OOP* (programación orientada a objetos) y bajo una licencia *Open Source*, actuando de manera eficaz como un impulsor de la comprensión, modificación y distribución de diseños mecánicos.

Estas bibliotecas utilizan una estructura de clases, permitiendo la definición de piezas parametrizables con la intención de facilitar al usuario la construcción de diseños más complejos. De esta forma se puede definir las piezas más simples (como ruedas, uniones, rodamientos, servos, etc…) y reutilizarlas para otros diseños, pudiendo adaptarlas de una forma rápida y sencilla.

Oriented-Object Mechanical Library se apoya en la plataforma *OpenSCAD*, que consiste en un lenguaje e interfaz para el diseño 3D mediante código; de hecho, actúan como un traductor entre el lenguaje *OpenSCAD* y *C++*. De este modo, la herramienta permite al usuario emplear toda la funcionalidad de *C++*, y posteriormente, utilizar la plataforma *OpenSCAD* para exportar el diseño a un formato estándar, como por ejemplo *STL*, *DFX*, etc.

En este proyecto se pretende demostrar, además, la utilidad de esta herramienta a través del diseño *Mars*. Este simboliza el compendio que se puede alcanzar entre complejidad y claridad. La mecanización se realiza a través de las impresoras 3D presentes en la universidad, dada su rapidez y su escaso coste.

El Mars se ideó como un vehículo para terrenos accidentados. La estructura se asemeja a la del *Mars Rover Curiosity*. Su adaptación al terreno se efectúa con un sistema de dos rodamientos concéntricos y un sistema mecánico auto-regulador

basado en un resorte con 3 grados de libertad. Además, incorpora la posibilidad de equiparlo con un sistema electrónico sensor-actuador para mantener la base horizontal.

El control se efectúa mediante un microcontrolador *Arduino*, una interfaz para ordenador bajo la plataforma *GNU/Linux* y la comunicación a través de dos dispositivos *bluetooth* (emisor y receptor).

2 OBJETIVOS

A continuación se enumeran los objetivos del proyecto:

- Desarrollar y documentar una herramienta con la capacidad de diseñar piezas mecánicas.
- Facilitar la reutilización de diseños.
- Promover la colaboración científica a través de la fabricación robótica.
- Implementar el prototipo *Mars*, basado en un sistema cinemático complejo, a través de esta herramienta.
- Demostrar la capacidad y potencialidad de *OOML* mediante el prototipo *Mars*.
- Alcanzar un sistema suficientemente eficaz para el movimiento por terrenos abruptos.
- Desarrollar un método de teleoperación y control para dicho sistema.

3 MOTIVACION

Durante los últimos años se ha producido una gran evolución en la investigación robótica a causa de la cada vez más extendida corriente *Open Source*.

Inicialmente este movimiento ha sido aplicado exclusivamente al software, dada la relativa facilidad de reutilización; sin embargo, el creciente esfuerzo de estandarización ha repercutido en la expansión de este paradigma hacia el hardware y, más recientemente, la mecánica.

Como se ha comentado, la constante expansión de las impresoras 3D está potenciando la aparición de repositorios de piezas mecánicas, con una comunidad en pleno crecimiento. Sin embargo, las herramientas convencionales de diseño gráfico no están ideadas para la reutilización, por lo que frenan el avance.

A principios de 2011 apareció *OpenSCAD*, una plataforma de diseño 3D con una diferencia fundamental de las herramientas de diseño convencionales: el diseño no utiliza una interfaz gráfica convencional, sino que se realiza exclusivamente con código, característica que facilita la modificación.

Aún con esta ventaja añadida, los diseños no siguen una metodología: el lenguaje es estructurado, de modo que cada individuo trabaja de un modo distinto. Surge así una elevada dificultad en el proceso de reusarlos.

Este aspecto se atenúa con la elaboración de *OOML*, y podrá observarse en el diseño expuesto, realizado con esta herramienta: *El Mars*. El uso del paradigma de la programación orientada a objetos pretende mitigar este problema, del mismo modo que surgió como una respuesta a la amplia extensión del *Open Source software*, facilitando la reutilización.

4 SOLUCION PROPUESTA

La difusión del modelo *Open Source* en el ámbito de la mecánica está colisionando fuertemente con las técnicas actuales y las herramientas más utilizadas para el diseño gráfico.

Una de las posibles soluciones que está teniendo más éxito es la utilización de código para el diseño; de esta forma, se pretende facilitar la distribución de las piezas a semejanza del software en sí mismo.

Una de las razones del gran triunfo del modelo *Open Source* en el software fue la aparición del paradigma de la programación orientada a objetos (*OOP*), que surgió como una poderosa herramienta para la reutilización de código.

Bajo esta premisa, se ha considerado utilizar esta misma herramienta para el modelo *Open Source* en la mecánica. La combinación del diseño a través de código y el paradigma *OOP* adquiere una elevada eficiencia en la colaboración entre individuos.

Este proyecto defiende una plataforma de diseño gráfico con el empleo de un lenguaje de programación, más concretamente *C++*, basada en el modelo de programación orientada a objetos. Se pretende dotar a los usuarios de un mayor control sobre los diseños con una capacidad comprensión menos compleja como consecuencia de una avanzada estructuración.

La elección del lenguaje *C++* se debe a:

- Es un lenguaje estandarizado.
- Permite emplear el paradigma *OOP*.
- Además de ser un lenguaje muy potente, proporciona más versatilidad que otros lenguajes.
- Existe un elevado número de bibliotecas ya creadas para distintas funciones, por lo que se puede aprovechar la resolución a muchas cuestiones.
- Es uno de los lenguajes más populares.

5 INTRODUCCIÓN A CONCEPTOS

5.1 EL MODELO OPEN SOURCE

La idea principal de *Oriented-Object Mechanical Library* consiste en impulsar, de forma muy efectiva, la capacidad del usuario de entender y alterar la mayor parte de un diseño 3D, así como su posterior distribución. Este proceso sigue las pautas del movimiento *Open Source*.

La aparición de *OOML* supone un nuevo logro en el marco *Open Source*, que tanto ha revolucionado estos últimos años. Por ello, se ha incluido el correspondiente apartado en relación a este paradigma.

5.1.1 CORRIENTES OPEN SOURCE Y FREE SOFTWARE

A menudo se confunden los términos *Open Source* y *Free software*, por lo que se ha introducido una pequeña aclaración: mientras que el primero hace referencia a una técnica de desarrollo, *Free Software* supone más bien una filosofía.

Ambos conceptos están relacionados con el cúmulo de ideas en torno a la cooperación entre usuarios y la libertad de distribución y modificación. Sin embargo, *Free software* aboga por la eliminación de toda barrera, ya sea económica o legal; *Open Source* no descarta ambas necesidades.

De hecho, bajo esta última premisa surgen las licencias de tipo *CopyLeft*, que apoyan la libre distribución con el mantenimiento de ciertos derechos de autor, como el reconocimiento legal de los mismos en usos posteriores.

No es pretensión de este proyecto afrontar este proceso de forma ideológica, por lo que se centrará en el término *Open Source,* el cual avala estas bibliotecas.

A continuación, se introduce al lector en la breve historia de la aparición del *Open Source* en el ámbito del software. Este transcurso ejemplifica los cambios que se producen en torno a los medios y las técnicas, que inevitablemente tienden a adaptarse a esta corriente.

5.1.2 MARCO TECNOLÓGICO

En la década de los 70, surge el concepto de L*icencia Copyleft*, en contraposición a la *Licencia Copyrigth*. A diferencia de esta última, la licencia *Copyleft* permite la difusión de los proyectos asociados a esta, así como la modificación y la distribución de los mismos. Esta libertad es permitida a través del sustento y reconocimiento de los derechos de autor en cualquier producto derivado del original.

Se ha debatido innumerables veces acerca de los inconvenientes de estas licencias. Hay discrepancias sobre la cuantía de código reutilizado necesaria para la repercusión legal en un gran proyecto, por lo que en muchos casos se producen conflictos que terminan en la intervención de un tribunal.

Bajo estas premisas aparecen los primeros trabajos, como *BASIC* del *Doctor Li-Cheng Wang* (donde podría haber surgido el nombre de estas licencias) o la primera licencia *Copyleft*, *GPL* (*General Public LIcense*) de *Richard Stallman*.

Figura 1. Símbolo para la licencia *General Public License,* de *Richard Stallman.*

Este tipo de licencia permite la libertad de distribución, uso y modificación en torno a los productos asociados a ella. Siguiendo este modelo, aparecen organizaciones, como *Creative Commons* [1], con el objetivo de infundir este concepto en la sociedad a través de las leyes de cada país.

A principio de los años 80, *Richard Stallman* comenzó a desarrollar el proyecto *GNU* y la fundación *Free Software Foundation* en el *MIT*. La pretensión de ese proyecto era la construcción de un sistema operativo totalmente libre. Así, *Stallman* empezó a desarrollar todas las herramientas necesarias bajo su propia licencia, *GPL*, para esta tarea: compiladores, editores, debugger, etc.

En 1991, *Linus Tovalds* implementó en Filandia su primera versión del kernel *Linux*. Recibió el apoyo de personas de todas partes del mundo, quienes hicieron posible la aparición del sistema *GNU/Linux* [2].

De forma paralela, en los años 80, el grupo de investigación de ciencias de la computación de la universidad de California mejoró el sistema *Unix*, que junto con un grupo de aplicaciones, se convirtió en el sistema *BSD Unix*. Este comenzó a distribuirse bajo la licencia *Unix AT&T*.

En 1992, *Bill Jolitz* diseño la última parte de *Net/2* a través de un kernel, completando el sistema *BDS Unix*. De esta forma lo liberó completamente, cambiando el tipo de licencia.

Ambos proyectos se han ido desarrollando como consecuencia del esfuerzo y la dedicación de numerosos individuos, quienes han logrado reunir interesantes herramientas bajo una misma plataforma.

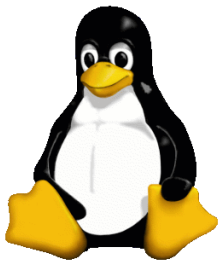


Figura 2. Símbolo del kernel de *Linux.*

Una de las principales causas del éxito de ambos sistemas ha sido la expansión de internet y la enorme influencia que ha tenido la aparición de los repositorios como medio de difusión de información y datos.

5.1.3 MODELO

Una de las cuestiones más importante que afecta a la *Open science* considera la viabilidad económica como un modelo de negocio. Surge la desconfianza de que se mantenga, bajo este modelo, la posibilidad de producir beneficios económicos a partir de las ideas, métodos y protocolos.

Este proyecto no considera esta cuestión; en su lugar, se enfrenta a la idea de que *Open science* represente un modelo de optimización de la investigación: el hecho de

que el proceso de compartir datos e ideas entre comunidades sea una técnica más efectiva para evolucionar hacia el conocimiento.

Las instituciones con financiación son libres de desarrollar investigaciones no económicas, lejos de la presión del mercado: tienen la oportunidad de centrarse en la investigación en sí misma, ausentes de la competitividad económica empresarial.

Su propósito no es vender un producto, sino generar un conocimiento necesario en algún ámbito, con el propósito de elaborar un producto. Este es el caso de las universidades públicas o los centros de investigación. ¿*Open Science* puede ser un modelo válido para realizar investigación en instituciones semejantes?

En el artículo *Open Science in practice: Researcher perspectives and participation* [3], los autores indagaron sobre los beneficios percibidos por los investigadores, las instituciones de investigación y los organismos de financiación en torno a la utilización de métodos *Open Science*.

Las conclusiones de este trabajo resumen las ventajas de la corriente *Open Science* en 5 cuestiones principales:

1. Velocidad y eficiencia en la investigación.
2. Aumento de las capacidades para identificar nuevos temas de investigación.
3. Incremento de la eficacia y la calidad.
4. Innovación, intercambio de conocimiento e impacto.
5. Grupo de investigación y desarrollo de carrera.

El autor concluye que muchos de los beneficios previstos para los métodos *Open Source* están relacionados con el hecho de que permita no solo el acceso a las comunidades de investigación, sino también la participación de las personas menos familiarizadas con estas, como consecuencia de la eliminación de muchas de las barreras que aíslan a estas comunidades.

Puede afirmarse que una gran visibilidad para los investigadores, bajas barreras para la colaboración y conjuntos de datos más reutilizables son ventajas destacables para considerar este paradigma.

El modelo de comunidad es inherente al de *Open Science*; de hecho, la mayor parte de la filosofía *Open Source* está basada en el principio de permitir la participación a tanta gente como sea posible.

Inicialmente, los desarrolladores eran atraídos por detalles técnicos; sin embargo, acorde a *New trends from libre software that many change education* [4], participar en una comunidad *Open Source* es un continuo proceso de aprendizaje, donde no solo una tecnología es considerada, sino otros factores como grupos de trabajo y comunicación.

Figura 3. Logo del sistema operativo *Ubuntu*.

Probablemente, el futuro ingeniero pertenecerá a una o varias de estas comunidades. Por tanto, es importante para ellos saber cómo desarrollar su trabajo dentro de estas comunidades. Además, deberían aprender cómo crear, guiar e inspirar esas comunidades.

Hasta ahora, *Open Science* ha sido extensamente aplicado al desarrollo del software con el *Open Source* software. El ejemplo más obvio es el sistema operativo *GNU/Linux*.

El impacto de estos sistemas, como *Ubuntu*, muestra la viabilidad de semejantes proyectos, no solo para pequeñas y especializadas comunidades, sino también para el público en general.

La comunidad científica ha sido beneficiada por un importante incremento de productividad a causa de este efecto.

La comunidad *Open Source* ha sido afectada con un impresionante crecimiento con la introducción del paradigma de la programación orientada a objetos (*OOP*). De hecho, es inherente a los objetivos de la *OOP* la colaboración, reutilización y modificación de código, permitiendo que grandes proyectos y desarrollos independientes puedan ser unidos.

Recientemente, ha surgido un movimiento similar orientado a la electrónica, llamado *Open hardware*. El *Open hardware* comparte toda la información relacionada con el

diseño y desarrollo del hardware. Siguiendo este paradigma aparece el proyecto *Arduino*.

Figura 4. El robot *Khepera III* pertenece a una familia de robots destinados como herramientas de educación o prototipado para la comprobación de algoritmos en la investigación.

Los proyectos *Open Source* han generado un profundo impacto en la comunidad robótica, al igual que está ocurriendo con el *Open hardware*. Para los diseños mecánicos de robots hay aún pocas comunidades. De hecho, la mayoría de los investigadores robóticos prueban sus algoritmos en simulaciones o con pequeños conjuntos de plataformas comerciales como *Pioneer robots* o *Khepera* [5].

Las impresoras 3D están cambiando este panorama. En otros apartados se mostrará como las impresoras 3D son la herramienta óptima para impulsar el diseño abierto de objetos físicos.

5.1.4 APLICACIÓN

Durante años la comunidad *Open Source* ha estado muy presente en el ámbito de la robótica; en la mayoría de las aplicaciones robóticas actuales participan proyectos como *Player/Stage/Gazebo*, *ROS*, *OpenCV*… pertenecientes a esta comunidad.

Recientemente, el mundo de la electrónica ha acogido esta tendencia, llegando a definir una nueva corriente conocida como *Open hardware*.

Con ella han surgido proyectos como *Arduino*, uno de los más populares y con una gran comunidad a su alrededor. Este consiste en una plataforma de electrónica abierta para la creación de prototipos. En este proyecto se empleará un microcontrolador *Arduino Uno*, basado en esta tecnología, para el control del prototipo *Mars*.

Figura 5. Microcontrolador *Arduino Uno*.

OOML fue planteado bajo el paradigma *Open Source*, proporcionando un medio efectivo de colaborar fácilmente en el diseño de componentes mecánicos, en el que se permite alterar estos diseños para su evolución de forma sencilla.

Combina tanto las ventajas de la programación orientada a objetos, como el poder del lenguaje *C++*, permitiendo al usuario crear diseños más complejos y, a su vez, mantener la claridad y transparencia de la estructura para su fácil comprensión.

De este modo, las piezas son completamente abiertas, capacitando a los usuarios su distribución e impulsando, de este modo, la evolución global de estas, a partir de la habilidad de las personas de las distintas partes del mundo.

El objetivo de esta herramienta consiste en permitir al usuario la reutilización, modificación y distribución de diseños robóticos.

Las piezas pueden ser mecanizadas mediante los procesos clásicos (arranque de viruta, abrasión…), o bien, a través de las nuevas impresoras 3D de forma más económica**.**

Como se ha mencionado, en apartados posteriores se profundizará en la evolución de estos dispositivos; sin embargo, es importante destacar la analogía existente entre la contribución de dichas impresoras al diseño mecánico y la aportación de los ordenadores personales a la comunidad *Open Source*: personas de todas partes y lugares comparten código, colaborando en un esfuerzo común para desarrollar proyectos *Open Source*.

Con las impresoras 3D, los diseños mecánicos pueden ser rápidamente replicados, de forma muy económica, al mismo tiempo que adaptados a otras necesidades.

5.2 OPENSCAD

El diseño de piezas mecánicas tradicionalmente se ha llevado a cabo mediante el empleo de programas gráficos *CAD* que siguen el paradigma *WYSIWYG* (*what you see is what you get*). Como divergencia de la corriente principal, aparece *OpenSCAD* [6].

OpenSCAD no sigue el paradigma *WYSIWYG*, como la mayoría de las aplicaciones *CAD* de diseño 3D; el diseño en esta plataforma está enfocado hacia la funcionalidad, acogiendo el paradigma *WYGIWYM* (*what you get is what you mean*).

En el documento, el usuario define el contenido acorde a su significado, en lugar de diseñar su apariencia describiendo el contenido en el editor de forma estructurada. Esto implica el conocimiento previo del contenido semántico, anterior a la edición.

En el caso de un objeto 3D, es necesario tener una mínima idea de como será para comenzar a definirlo.

En la *Figura 6* se puede observar un engranaje definido bajo este paradigma. Para ello, primero se parte de una idea sobre la forma del objeto que se quiere desarrollar, y posteriormente se especifica los cilindros que compondrán el cuerpo principal del engranaje, junto con los dientes y los huecos.

El editor que sigue este paradigma necesita disponer de un sistema de exportación, con el objetivo de generar el archivo final en el formato indicado, a partir del contenido descrito siguiendo la estructura indicada.

La ventaja principal de este paradigma es la total separación entre presentación y contenido: los desarrolladores son capaces de concentrar sus esfuerzos en estructurar y redactar el documento, en vez de enfrentarse a la apariencia del mismo. Esta es abandonada, pues será resuelta de forma automática por el sistema de exportación.

Otra ventaja considerable es la facilidad de exportación de un mismo contenido a diferentes formatos.

OpenSCAD construye sus diseños bajo este paradigma, partiendo del contenido en sí mismo. Se puede definir como un compilador 3D que lee archivos *script*; estos describen un objeto permitiendo la construcción de su modelo 3D, y proporcionando la posibilidad de exportación a diferentes formatos estándar de objetos 3D (STL, OFF, DXF).

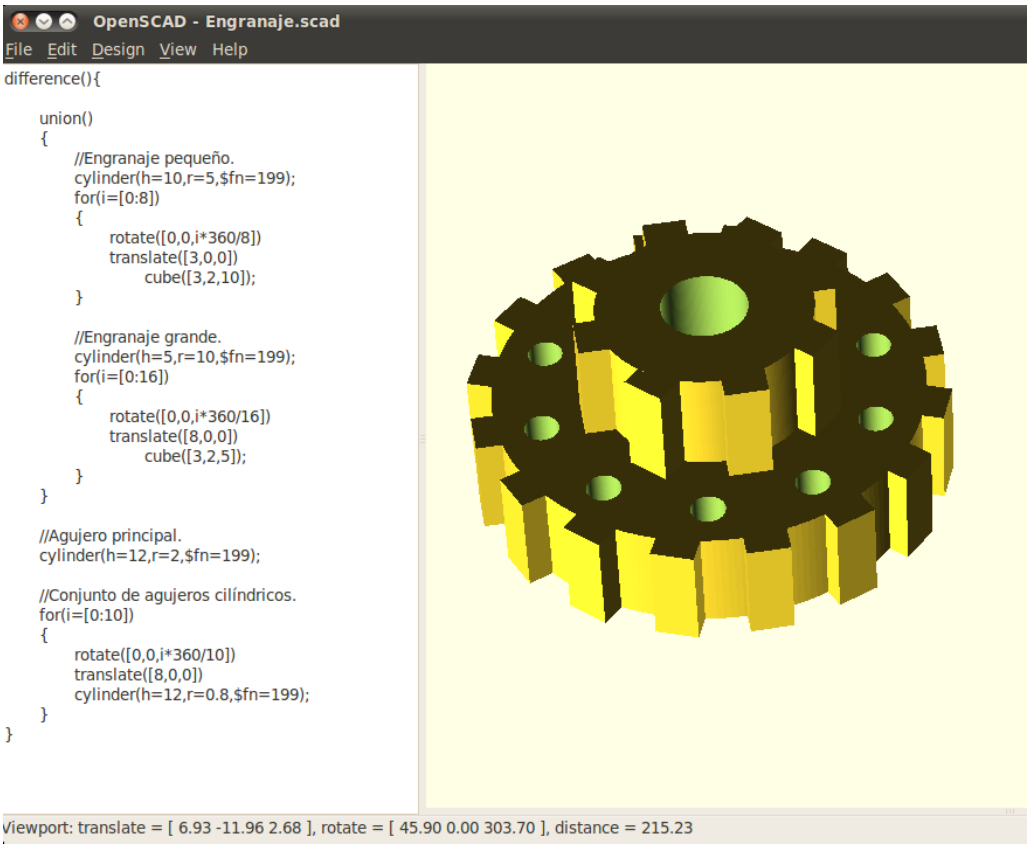


Figura 6. Ejemplo de programación centrada en el contenido bajo la plataforma *OpenSCAD*.

Esta técnica aporta al desarrollador un control pleno sobre el proceso de modelado, además de proveer de una mayor facilidad para cambiar el proceso paso a paso o mediante parámetros configurables.

El diseño de objetos 3D mediante el empleo de código implica la aparición de posibilidades que hasta ahora solo estaban presentes en la comunidad del software.

Actualmente los objetos 3D pueden ser compartidos a través de internet (normalmente mediante repositorios), y posteriormente, modificados, mejorados y reusados, como ha sucedido en el ámbito del software durante muchos años.

Hoy en día, *OpenSCAD* es el único programador popularmente conocido que integra una interfaz y un lenguaje destinados al modelado de objetos mecánicos; no obstante, la idea de diseñar objetos 3D a través de código es presentada también en *PovRAY*: no para el diseño mecánico, sino para la visualización y edición de video.

En el siguiente apartado, se justificará la decisión de renunciar a la plataforma *OpenSCAD* y elaborar una nueva herramienta de programación para el diseño de objetos mecánicos.

Además, se argumentará por qué se entiende esta herramienta, OOML, como la evolución lógica de *OpenSCAD*. Esta evolución consiste en la introducción del paradigma de la programación orientada a objetos en el diseño de objetos físicos.

5.3 STL

Hasta ahora, se ha tratado de mostrar la razón que subyace en la mejora de la investigación a través de las comunidades de conocimiento libre. Se ha explicado también cómo, en las comunidades de software y electrónica, este modelo tiene éxito, compartiendo fuentes de diseño.

Recientemente, la posibilidad de replicar objetos 3D ha causado el inicio de este movimiento en la investigación del diseño mecánico.

El primer requerimiento para compartir un objeto o proceso es la posibilidad de digitalizar la información que este describe. Para la digitalización de objetos existen muchos formatos que permiten la descripción 3D.

Actualmente, el formato estándar más popularmente utilizado es el formato *Standard Tessellation Language* (*STL*).

El formato *STL* consiste en un archivo centrado en la estereolitografia *CAD*, creada por sistemas 3D. Su aplicación deriva hacia el prototipado rápido y la fabricación asistida por ordenador.

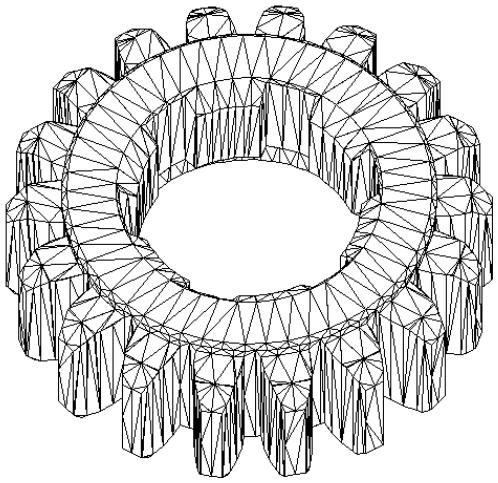


Figura 7. Representación visual del formato *STL*.

Un archivo *STL* describe una superficie no estructurada triangulada por la unidad normal y los vértices (ordenados por la regla de la mano derecha) de los triángulos, usando un sistema de coordenadas tridimensional cartesiano. Cualquiera puede reproducir un objeto físico teniendo el archivo *STL*.

5.4 IMPRESORA 3D

5.4.1 INTRODUCCION

Bradshaw Et. Al. [7] han realizado recientemente un estudio sobre las económicas impresoras 3D. Este estudio analiza brevemente la historia de las impresoras 3D: partiendo de los últimos años de la década de los 70, hasta la actualidad.

En estos más de 30 años se ha logrado mejorar las técnicas de extrusión, así como abaratar los materiales utilizados y los costes de mantenimiento. De esta forma, se ha conseguido la disponibilidad de impresoras 3D asequibles para la mayoría de las personas con el salario medio occidental.

Las impresoras personales 3D permiten la impresión de complejas piezas de ingeniería de forma semi-automática, partiendo exclusivamente del archivo que contiene el diseño (generalmente el formato *STL*).

La ventaja más destacada de estos dispositivos es la posibilidad de compartir los diseños a través de internet, permitiendo que cualquier individuo con acceso a otra impresora tenga la capacidad de replicar el mismo objeto físico.

Los estudios sobre el desarrollo del paradigma *Open Source* en el ámbito del software han sido estudiados intensamente, con resultados bastante favorables; sin embargo, relativamente poco es conocido acerca de la viabilidad de este mismo modelo para el diseño mecánico. Este hecho no carece de importancia pues las impresoras 3D están ofreciendo nuevas posibilidades en el proceso de compartir objetos físicos.

La definición de la geometría de los objetos físicos mediante código (la digitalización de la geometría) permite a los individuos compartir sus propias piezas, evolucionarlas y construirlas empleando una impresora 3D. De este modo, una comunidad descentralizada puede colaborar en la producción y mejora de piezas mecánicas, basadas en diseños digitales, con ayuda de internet y sin apenas barreras legales o económicas.

Bruijn, cofundador del proyecto *Reprap* e investigador del modelo *Open Source*, muestra una considerable mejora en el diseño de los objetos físicos por parte de los usuarios que comparten código y tienen acceso a una impresora 3D [8].

Con el acceso a una impresora 3D, las modificaciones son relativamente fáciles de replicar para cada individuo. Actualmente, al igual que ha ocurrido con el software durante años, existen repositorios on-line de objetos 3D donde la gente puede descargar o compartir diseños.

5.4.2 ESTADO ACTUAL

A continuación se nombran las impresoras *Open Source* más destacadas por su popularidad.

El modelo original fue el proyecto *Reprap* [9], iniciado por Adrian Browyer en 2004. El objetivo de este proyecto era desarrollar una máquina *Open Source* auto-replicable.

En mayo de 2007 se finalizó el primer prototipo llamado *Darwin*, y pocos días más tarde, el 29 de mayo, se logró realizar la primera réplica. Desde entonces, la comunidad *Reprap* (centrada en la impresora *Reprap* original y los diseños derivados) ha crecido exponencialmente.

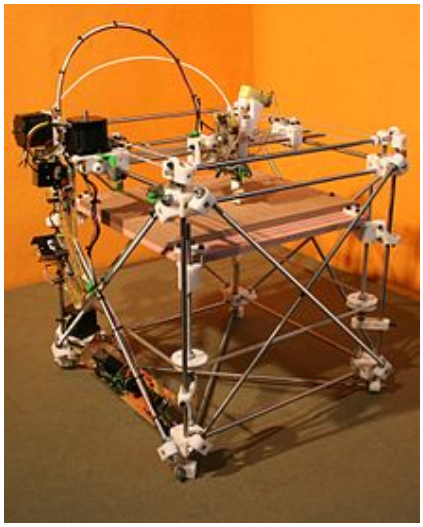


Figura 8. Primera impresora 3D, *Darwin*.

El número actual de impresoras que se estima es de unas 4500. La segunda generación de impresoras *Reprap*, llamada *Mendel*, fue finalizada en septiembre de 2009.

Algunas de las principales ventajas de las impresoras *Mendel* sobre las impresoras *Darwin* son: un área de impresión mayor, una mejor eficiencia, la facilidad de montaje, su precio más económico y son más ligeras y fáciles de transportar.

Inicialmente, tanto *Mendel* como *Darwin* no fueron diseñadas para el público en general, sino para personas con alguna formación técnica. Debido al carácter *Open Source* del proyecto *Reprap*, surgieron pequeñas empresas destinadas a la venta de esas impresoras, por lo que también se dedicaron a mejorarlas.

La primera compañía fue *Makerbot Industries* [10], que envió un primer lote en abril del 2009. A finales de ese mismo año, habían enviado cerca de 500 dispositivos completos. Después de operar durante un año, habían vendido alrededor de 1000 unidades.

Tras este éxito, apareció la impresora *Thing-O-Matic*, anunciada en septiembre de 2010. Esta superaba a las demás por la gran facilidad de construcción y uso causada por su sencilla estructura, además de su bajo coste de adquisición (por solo 950 dólares).

Su modelo actual ha sido llamado *The Replicator*, y se distribuye ya montado. Es la primera impresora 3D con un extrusor doble, permitiendo usar un material de soporte o imprimir en dos colores. Su precio ronda los 1900 dólares.

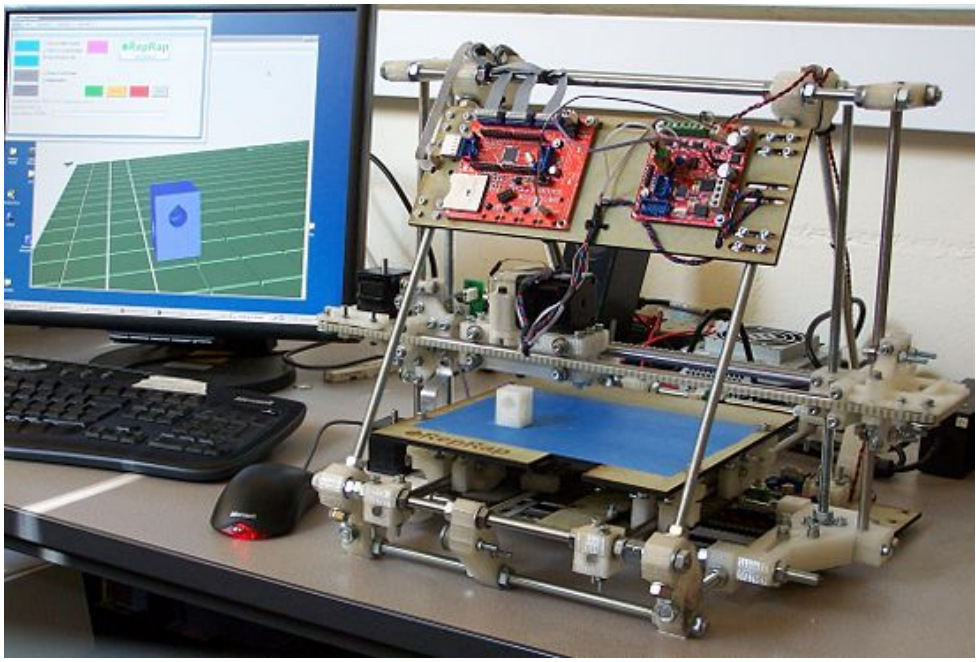


Figura 9. Impresora 3D modelo *Mendel*.

En los últimos años, han surgido otras impresoras, inspiradas por *Reprap*, siguiendo el comercio impulsado por *Makerbot*. Algunas de estas son: la impresora *PrintrBot* (que solo cuesta 450 dólares) o *UltiMaker* (alrededor de los 1000 dólares).

La distribución de estos archivos a través de los repositorios implica únicamente la posibilidad de compartir un diseño, no de alterarlo: modificar un archivo STL es demasiado complejo.

Con el propósito de impulsar la colaboración, los desarrolladores necesitan herramientas que describan sus diseños, y a su vez, sean capaces de generar el archivo en formato STL correspondiente.

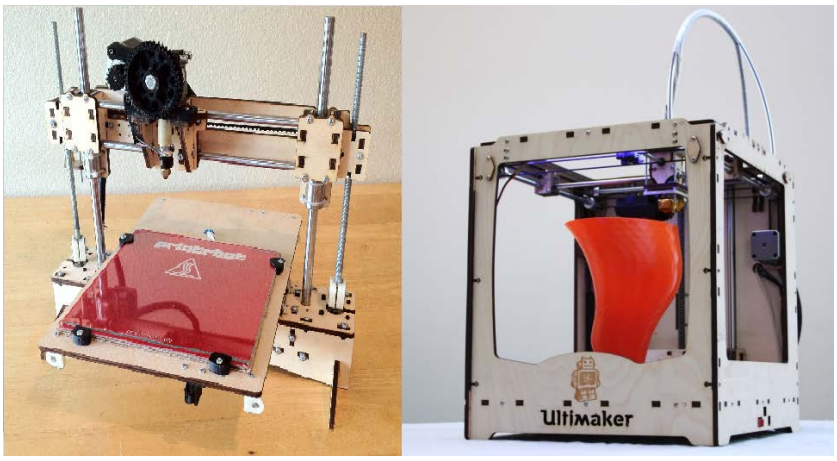


Figura 10. Impresoras *PrintrBot* y *UltiMaker*.

Puede observarse una analogía con el código fuente y las bibliotecas binarias o ejecutables en el desarrollo del software.

5.5 OOP

La programación orientada a objetos, OOP, tiene como fundamento la simulación del comportamiento de los objetos físicos. Así, sería lógico pensar que el empleo del paradigma OOP en el diseño de objetos físicos es un método efectivo.

A continuación, se va a presentar las principales propiedades de la programación orientada a objetos, junto con su analogía respecto a la mecánica, para manifestar su eficiencia:

- *Encapsulación y modularidad*. Los objetos físicos tienen muchas propiedades; algunas de ellas pueden ser consideradas independientes, mientras que otras dependen entre sí.

  Por ejemplo, el radio de un engranaje y el tamaño de sus dientes determinarán el número y la posición de estos. Por tanto, simplificando solo se necesita emplear tres parámetros para la definición de un engranaje: el grosor, el radio y la anchura del diente. El resto puede ser procesado por un ordenador.

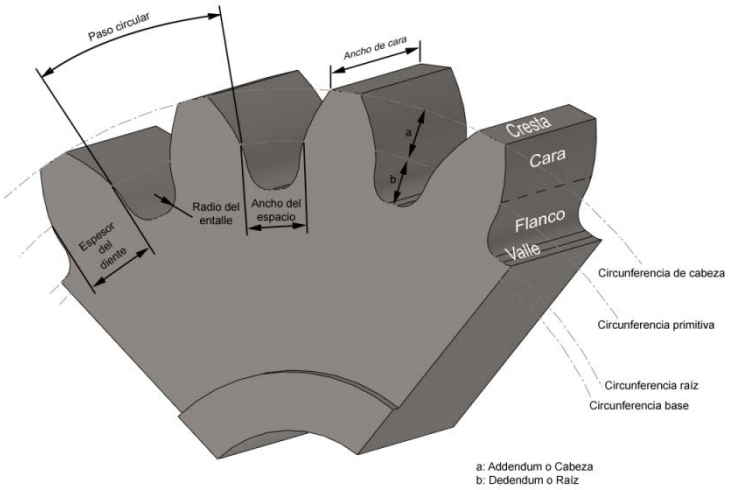


Figura 11. Parámetros de un engranaje.

De este modo, se consigue encapsular la información que el diseñador no necesita fijar para la definición del engranaje.

- *Herencia y composición*. La herencia de clases está basada en la relación *IS-A*. En el ejemplo anterior es fácil ver está relación entre objetos físicos. Una rueda dentada *IS-A* (*es una*) rueda.

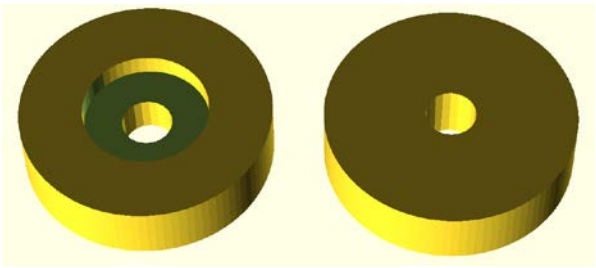


Figura 12. Una Servo wheel y una Simple Wheel siguen siendo una rueda.

En este caso, podemos definir dos clases: una primera, que sea rueda; y una segunda, rueda dentada, que hereda de la primera.

La composición es incluso más clara. Un reloj está compuesto de engranajes, al igual que un coche de ruedas.

- *Polimorfismo*. El polimorfismo representa el hecho de que cada objeto proporcione los métodos que utiliza sobre los demás, incluso si el método es común a todos los objetos. Esta propiedad es una gran ventaja en el diseño de objetos físicos.

  El polimorfismo permite definir una sólida estructura de clases para declarar objetos mecánicos.

- *Abstracción*. Todas las ruedas son conceptualmente la misma, pero cada una tiene diferentes parámetros. Esto ilustra el hecho de que la abstracción es una metodología adecuada para definir clases a partir de objetos físicos.

  Como su nombre sugiere, la programación orientada a objetos utiliza todas estas propiedades para imitar objetos físicos, así como sus interacciones.

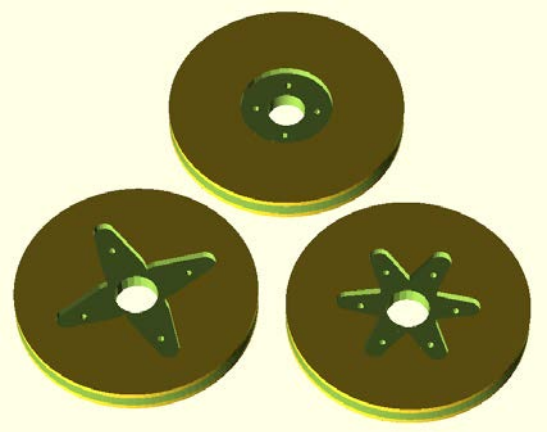


Figura 13. Tres Servo Wheel. Cada rueda tiene diferentes parámetros, pero conceptualmente son la misma.

Se puede concluir que la opción más lógica es usar el paradigma OOP para el diseño de objetos.

6 OOML

6.1 INTRODUCCIÓN

Considerando lo mencionado en apartados anteriores, la disponibilidad de una herramienta bajo el modelo *Open Source*, que permita el diseño de objetos físicos a través del paradigma *OOP*, impulsa el diseño en el ámbito de la robótica, a la vez que mejora la colaboración y reusabilidad en la comunidad.

Experimentalmente puede comprobarse la complejidad de compartir diseños a través de la plataforma *OpenSCAD*. Esta proporciona una herramienta *Open Source*, basada en código; sin embargo, cada usuario tiene una manera muy distinta de diseñar, por lo que conlleva una moderada complejidad entender el código de los demás y adaptarlo.

Otro aspecto a considerar, es la ausencia, en el lenguaje *OpenSCAD*, de algunas importantes funcionalidades presentes en otros lenguajes, como *C++*: bucles *while*, variables, funciones, etc.

Bajo estas cuestiones, surge la idea de definir objetos físicos como objetos programables, y convertir las operaciones geométricas en métodos pertenecientes a los objetos.

De esta idea parte el desarrollo de *Oriented-Object Mechanical Library*, *OOML* [11]. Existen tres premisas sobre las que se asienta esta herramienta:

- Los objetos físicos son objetos programables.
- Las operaciones geométricas son métodos propios de los objetos.
- Los parámetros de los objetos físicos son atributos.

Durante el desarrollo inicial surge la idea de la semántica: los objetos geométricos primitivos son diferentes de las piezas mecánicas.

En la etapa final, una vez las técnicas de la semántica y la orientación a objetos son definidas, el concepto de interfaz de objetos surge de forma natural. Este estandariza la forma de crear las piezas y los objetos primitivos, y el modo en que se accede a ellos mediante el uso del polimorfismo de *C++*, como consecuencia de que todo objeto heredado incluya la misma interfaz.

La primera versión de *OOML* fue lanzada en julio del 2011; la segunda versión, en febrero del 2012. Junto con la biblioteca, ha sido desarrollada una página web con toda la información necesaria, incluyendo los tutoriales necesarios para comenzar su uso.

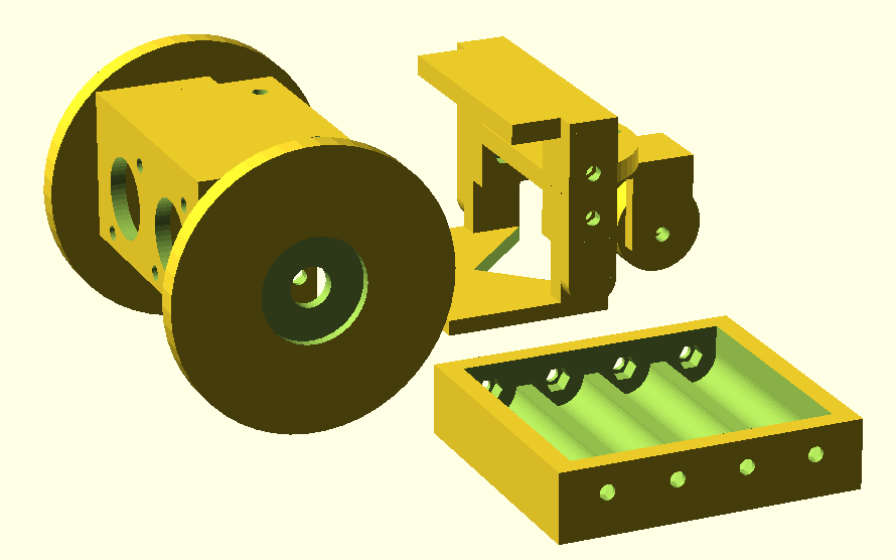


Figura 14. Diseño MiniSkyBot desarrollado bajo la plataforma OOML.

La utilización de código para el diseño de objetos físicos ha causado un gran impacto en la comunidad desde la aparición de *OpenSCAD*. Ya hay más de 2716 diseños desarrollados con *OpenSCAD* en *Thingiverse* (datos extraidos el 15 de marzo de 2012).

OOML es bastante más reciente, y únicamente incluye 14 diseños, pero se espera que ese número se incremente considerablemente.

En la *Figura 14* podemos observar el impacto del diseño *Open Source*. El diseño aquí presente es el robot *Mini-Skybot*, una versión del robot *Skybot*. El robot *Skybot*, mecanizado en aluminio y plástico con métodos tradicionales existe desde hace más de 5 años. Se han impartido un gran número de cursos en universidades e institutos y este robot no ha evolucionado mecánicamente en todo ese tiempo.

En el año 2011 el prototipo *Mini-SkyBot*, diseñado en *OpenSCAD*, fue impreso y publicado en el repositorio *Thingiverse* [12]. En menos de un año, un gran número de robots derivados de este diseño inicial han surgido, alrededor del mundo.

El *ProtoBot,* diseño imprimible con una sola pieza, ya ha sido diseñado usando *OOML* en febrero del 2012 (con la participación de Alberto Valero, Nieves Cubo y Mario

Almagro). Desde entonces ya hay réplicas en otros lugares y se han aplicado numerosas modificaciones.

6.2 OOML FRENTE A OPENSCAD

Las principales características de *OOML* pueden enumerarse en los siguientes puntos:

- Su función es el diseño de objetos 3D a través del lenguaje *C++.*
- Incluye todas las operaciones necesarias para la manipulación de los objetos en el espacio.
- La estructura se basa en el paradigma de la programación orientada a objetos.
- Utiliza la orientación a la semántica. El empleo de una categorización semántica de las clases permite la aplicación del paradigma *WYGIWYM* (*what you get is what you mean*) de forma más fácil y precisa.
- Sigue el modelo *Open Source*: permite usarlo, modificarlo y compartirlo.
- La compilación del diseño genera un archivo en lenguaje *OpenSCAD*. Posteriormente, es posible construir el archivo STL mediante el uso de esta plataforma.

La ventaja principal que aporta *OOML* en contraposición con *OpenSCAD* no reside tanto en el paradigma de diseño, pues ambas plataformas comparten el mismo paradigma (*WYGIWYM*), sino en su estructura basada en la programación orientada a objetos.

Una posible analogía sería la relación entre el lenguaje *C* y *C++*. Para pequeñas aplicaciones, el lenguaje *C++* no parece proporcionar ninguna ventaja respecto al lenguaje *C*; sin embargo, en proyectos más grandes, *C++* ofrece mayores beneficios en torno a la escalabilidad y reusabilidad de código, a través de la metodología de la orientación a objetos.

A continuación se muestra una breve comparación entre ambas plataformas:

	OpenSCAD	OOML
Operaciones geométricas	✓	✓
Orientación a objetos	×	✓
Orientación a la semántica	×	✓
Open Source	✓	✓
Objetos primitivos	✓	✓

Tabla 1. Comparación entre las características de ambas plataformas.

6.3 OOP

La biblioteca *OOML* aplica el modelo *OOP* (programación orientada a objetos) para el diseño mecánico. Los principios de este paradigma se basan en la idea de la interacción entre procesos aislados; es decir, la programación imita las propiedades de los objetos físicos y la interacción entre estos. Por lo tanto, las piezas mecánicas, diseñadas como objetos, siguen todas las características de la *OOP*:

- Encapsulación. Los atributos y métodos no necesarios para la definición de una pieza permanecen ocultos al usuario, restringiendo su acceso a los demás componentes.

- Envío dinámico. Cuando un método es invocado en una pieza, esta determina por sí misma qué código se ejecuta, buscando el método en el tiempo de ejecución en una tabla asociada con el objeto.

- Herencia. Las piezas pueden heredarse de otras siguiendo una relación *IS-A* (es un/una). Por ejemplo, una *servo wheel IS-A wheel*, conservando sus métodos y atributos como: *set_radius()*, *set_thickness()*, *set_axis_radius()*.

- Reusabilidad. Las piezas pueden utilizarse para formar otras, mediante adición o composición.

6.4 ORIENTADO A LA SEMÁNTICA

En *OpenSCAD,* todos los objetos tienen el mismo significado semántico pues cada uno es del mismo tipo; es decir, todos los tipos de objetos derivan de un mismo objeto abstracto, compartiendo ciertas características. De este modo, todos los objetos son compatibles entre sí, permitiendo la aplicación de operaciones geométricas a cualquier conjunto.

OOML está orientado a la semántica, de modo que introduce distintos tipos de objetos. Distinguimos tres clases principales:

- *Primitive objects* (o *components*): Son entidades geométricas sin significado mecánico, como un cilindro o una esfera.

- *Parts*: Son objetos más complejos con significado mecánico, como una rueda o un engranaje.

- *Packages*: La combinación de un conjunto de piezas puede formar un objeto más complejo. De este modo, se puede construir una *Castor Wheel* mediante la composición de una rueda y el soporte.

Esta división permite la adición de nuevas funcionalidades a las diferentes clases de objetos. En las clases *primitive objects*, se pueden aplicar todas las operaciones geométricas; la clase *part* incluye conexiones para formar objetos más complejos a través de la unión de varias de estas, utilizando la función *attach*. La clase *package* proporciona la función para generar piezas de forma separada.

Aplicando la metodología de la programación orientada a objetos, todos esos objetos comparten la misma interfaz de programación: todos heredan de una clase superior, abstracta, que provee de las mismas características. Así, se logra disponer de unas bibliotecas fáciles de usar.

6.5 ESTRUCTURA

En la segunda versión de *OOML*, lanzada en febrero de 2012, se ha decidido restructurar todas las bibliotecas con el propósito de, por un lado, emplear un código más estándar que facilite su entendimiento, y de otra forma, optimizar el código *OpenSCAD* generado para obtener archivos *STL* más simples.

Se ha dividido la organización de las bibliotecas en tres áreas: el apartado *core*, que contiene las clases indispensables para el correcto funcionamiento del conjunto de bibliotecas; el apartado *component*, que ubica los objetos más sencillos y simples, con la posibilidad de añadir otros; y el apartado *parts*, que alberga los diseños más avanzados que pueden ser usados para elaborar prototipos de forma más rápida.

Todos los objetos derivan de la clase *AbstractObject*, que contiene las características generales de un objeto. A partir de esta clase se crean los objetos básicos: como los de tipo *Component*, que comprenden todos aquellos objetos 3D (esferas, cubos, cilindros…) y sus modificaciones (toroides, poliedros…); la clase *Circle* o *Square* alberga círculos y cuadrados 2D que pueden ser extruidos tanto circular como linealmente; con la clase *Polygon2D,* estas bibliotecas permiten la definición de un polígono en el plano X-Y mediante coordenadas, y su posterior extrusión en la ordenada Z.

Mediante este sistema de distribución de tipos y propiedades se consigue una ordenación sencilla de comprender y alterar. De esta forma, el diseño persigue la constante modificación del mismo para un desarrollo conjunto de la comunidad. Con este propósito, se ha creado un apartado exclusivo para el diseño de piezas más desarrolladas, que puedan ser empleadas para su uso directo. Así, en el apartado p*arts* se ha elaborado una serie de diseños ya implementados: clase *FutabaS3003*, clase *ArduinoUNO*, clase *ServoWheel*, clase *NineBattery*, etc

6.6 OPERACIONES GEOMÉTRICAS

OOML incluye todas las operaciones geométricas necesarias para manipular objetos y crear otros nuevos:

- Translación. Mueve un objeto desde un punto a otro a través de desplazamientos en cada uno de los ejes cartesianos, manteniendo la orientación.

- Rotación. Modifica la orientación de un objeto trasladándolo en torno a un eje. La rotación se realiza con incrementos angulares en cada uno de los ejes cartesianos.

- Intersección. Devuelve un objeto formado por la intersección entre otros dos.

- Adición. Retorna el objeto resultante de la suma de dos objetos.

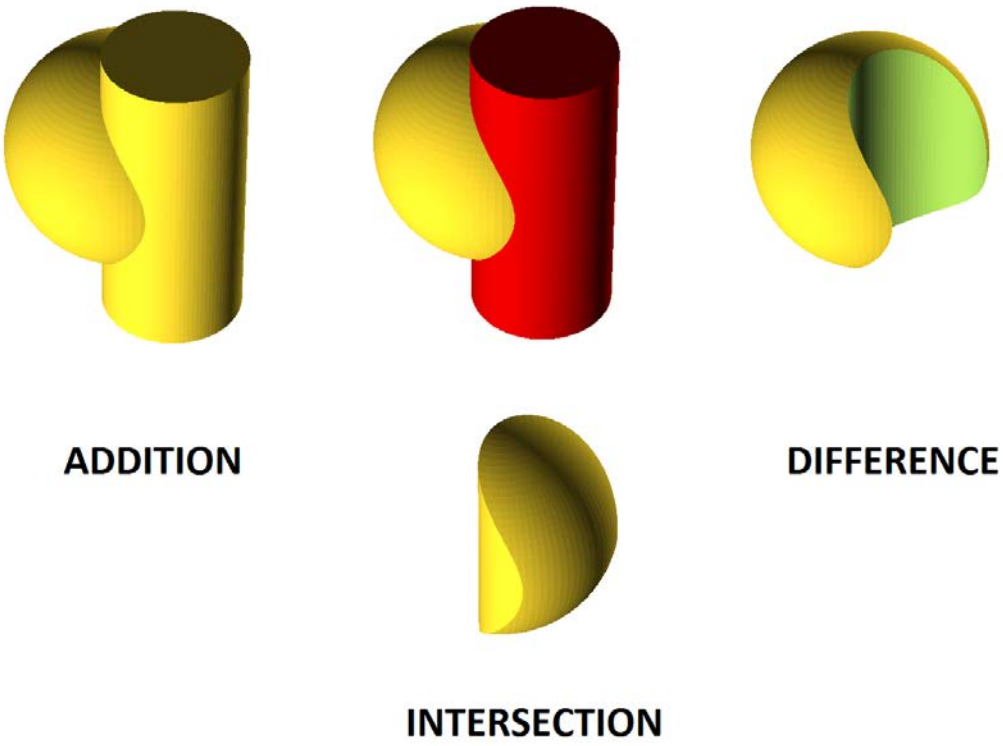


Figura 15. Principales operaciones geométricas.

- Diferencia. Substrae un objeto de otro, y devuelve el objeto resultante.

- Escalado. Permite alterar el tamaño de un objeto con distintos factores respecto a cada uno de los ejes cartesianos.

En la *Figura 15* se puede observar las principales operaciones geométricas relativas a dos o más objetos, mientras que en la Figura 16 y *Figura 17* se representa la mayor parte de las operaciones geométricas que se emplean sobre un único objeto, aplicadas en distintos objetos para mejor claridad en la *Figura 16*, y sobre un mismo cilindro, de color amarillo, en la *Figura 17*.

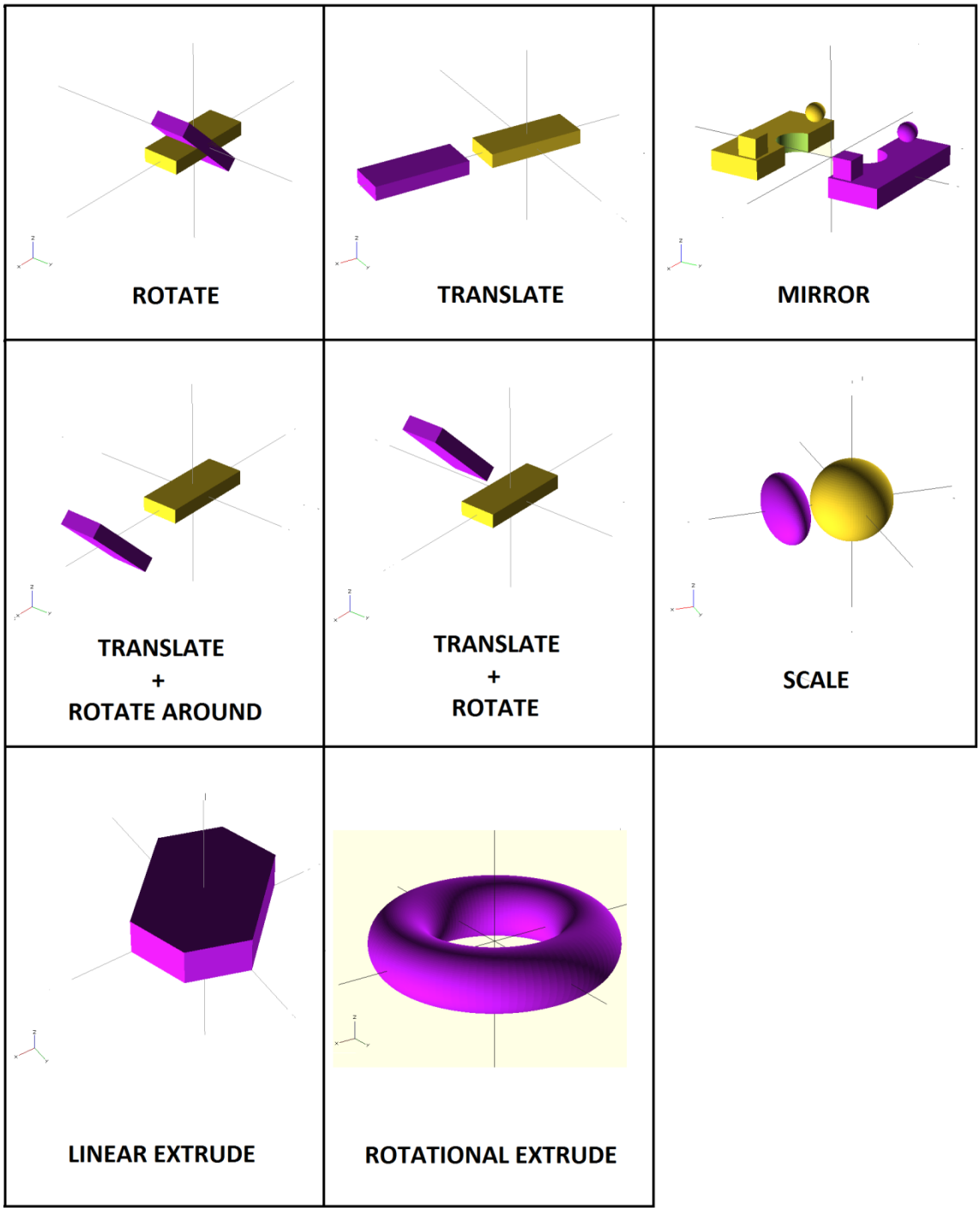


Figura 16. Principales operaciones geométricas.

Se adjunta el código correspondiente a los objetos de la *Figura 17* como el *Anexo 1*.

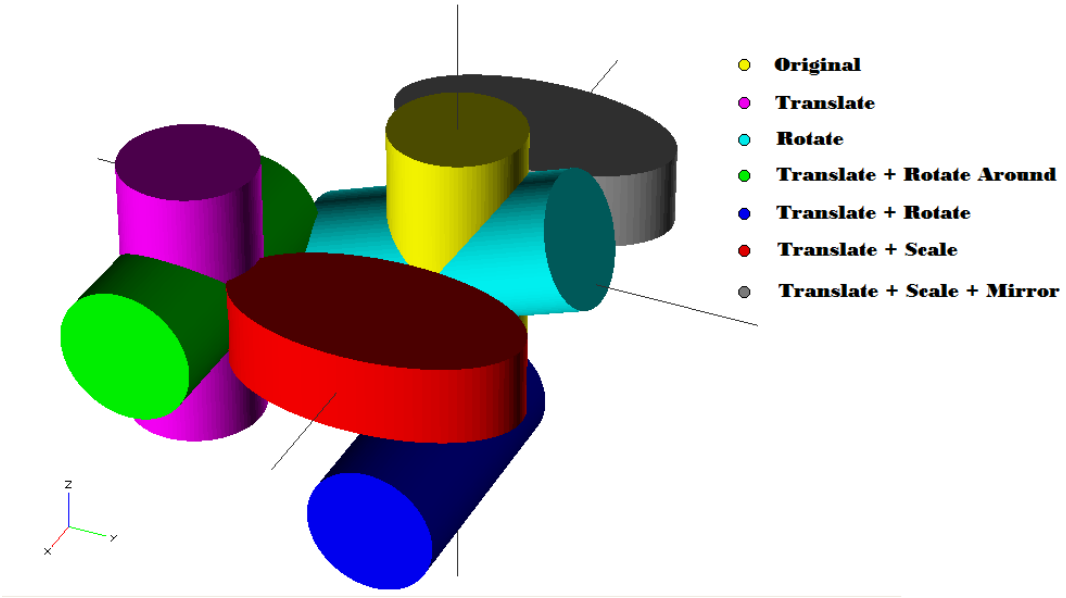


Figura 17. Operaciones geométricas aplicadas sobre un cilindro. A cada resultado de una operación se le asigna un color distinto.

6.7 WIKI

Puesto que el objetivo de *OOML* es que los propios usuarios sean los que desarrollen las piezas a reutilizar, es importante que se produzca una buena difusión de dicha herramienta. Partiendo de esta base, se ha desarrollado una página web, en formato *página wiki*, con toda la información y los archivos necesarios para su utilización.

Esta contiene un tutorial detallado de cómo descargar e instalar la herramienta, diversos tutoriales sobre cómo utilizar la herramienta y desarrollar diseños propios, y distintos apartados describiendo su estructura.

De forma adicional, incluye una sección administrada por la herramienta *Doxygen*, de forma que permite una visión más detallada de la biblioteca.

En la *Figura 18* se puede observar el aspecto general de este sitio web.

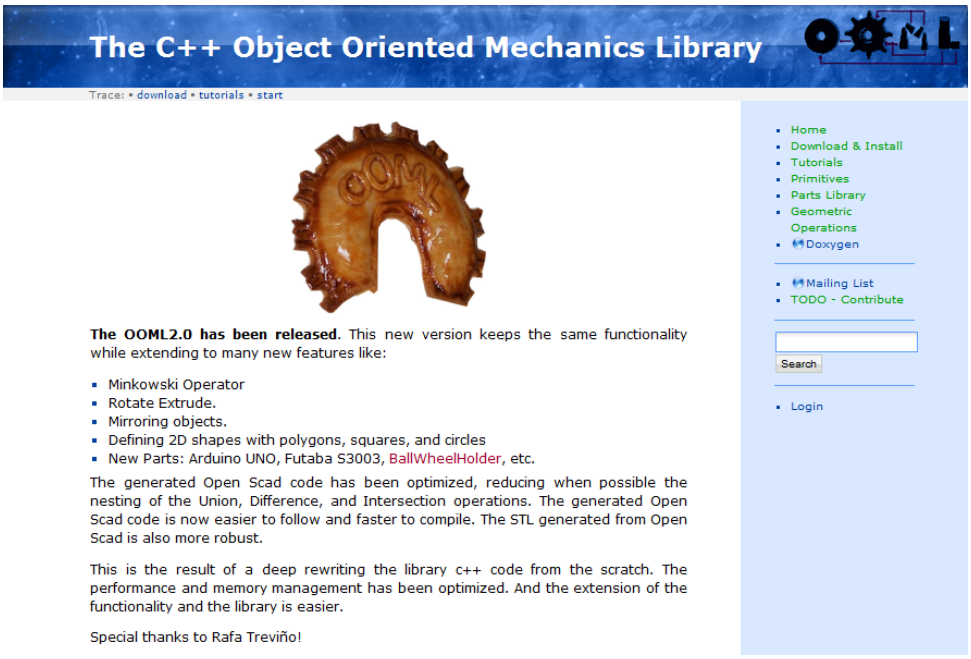


Figura 19. Página de inicio de la *wiki*.

Figura 18. Tutoriales *OOML* en la *wiki*.

Para consultar toda la documentación mencionada y visitar este sitio web se puede seguir este enlace: http://iearobotics.com/oomlwiki/doku.php?id=start

6.8 INTRODUCCIÓN AL MÉTODO DE EMPLEO

La utilización de estas bibliotecas carece de complicación alguna. Para comenzar, es necesaria su descarga, para la que se recomienda emplear *Subversion*, una herramienta de control de versiones.

El repositorio que contiene los archivos es: http://svn.iearobotics.com/oom/trunk/

Tras su descarga, se debe compilar y construir los archivos especificados en el documento "*CMakeList.txt*". Para ello, se utiliza la herramienta *CMake*, que interpreta la información sobre la compilación que contiene este archivo.

Para iniciar la programación se recomienda emplear un entorno de desarrollo de software integrado, tales como *Visual Studio*, *Eclipse*, *QtCreator*, etc. En cada diseño que se vaya a implementar debe añadirse, además de las bibliotecas correspondientes a los objetos, la biblioteca "*IndentWriter.h*". Esta se emplea para invocar el objeto *IndentWriter*, que será el que recoja el código *OOML* y lo transforme en código *OpenSCAD*.

De esta forma, cada diseño suele ir acompañado de su transcripciçon a un archivo mediante un párrafo del estilo:

```
IndentWriter writer;

writer << diseño;

ofstream os("diseño.scad");

os << writer;

os.close();
```

Para más información se puede consultar la sección de "Download & Install" en la wiki de OOML, siguiendo el próximo enlace:

http://iearobotics.com/oomlwiki/doku.php?id=download:start

6.9 IMPLEMENTACIÓN COMO EJEMPLO

A continuación se expondrá uno de los muchos ejemplos de utilización de las bibliotecas *OOML* que se pueden encontrar en el siguiente link:

http://iearobotics.com/oomlwiki/doku.php?id=tutorials:start

En este ejemplo, se crearán representaciones de árboles de forma aleatoria, llegando a representar un bosque. Para ello se comenzará definiendo un objeto árbol, que necesitará los parámetros: altura y radio.

Se adjunta el ejemplo en el *Anexo 2*.

Este objeto se compone del tronco y la copa. El tronco está formado por un cilindro, al que se dotará el color marrón, de radio y altura especificados mediante los parámetros de entrada. La copa consiste en un cono de color verde, con una altura igual que el tronco, y un radio en la base cinco veces superior al del tronco. Por último, se desplazan ambos objetos en la medida justa para apoyarse el uno sobre el otro.

En cuanto al proceso principal, se define el objeto "*writer*", que será el que traduzca los códigos (el paso de *OOML* a *OpenSCAD*) y el objeto fichero de salida "*os*".

Mediante el objeto "*ground*" se crea la representación del suelo.

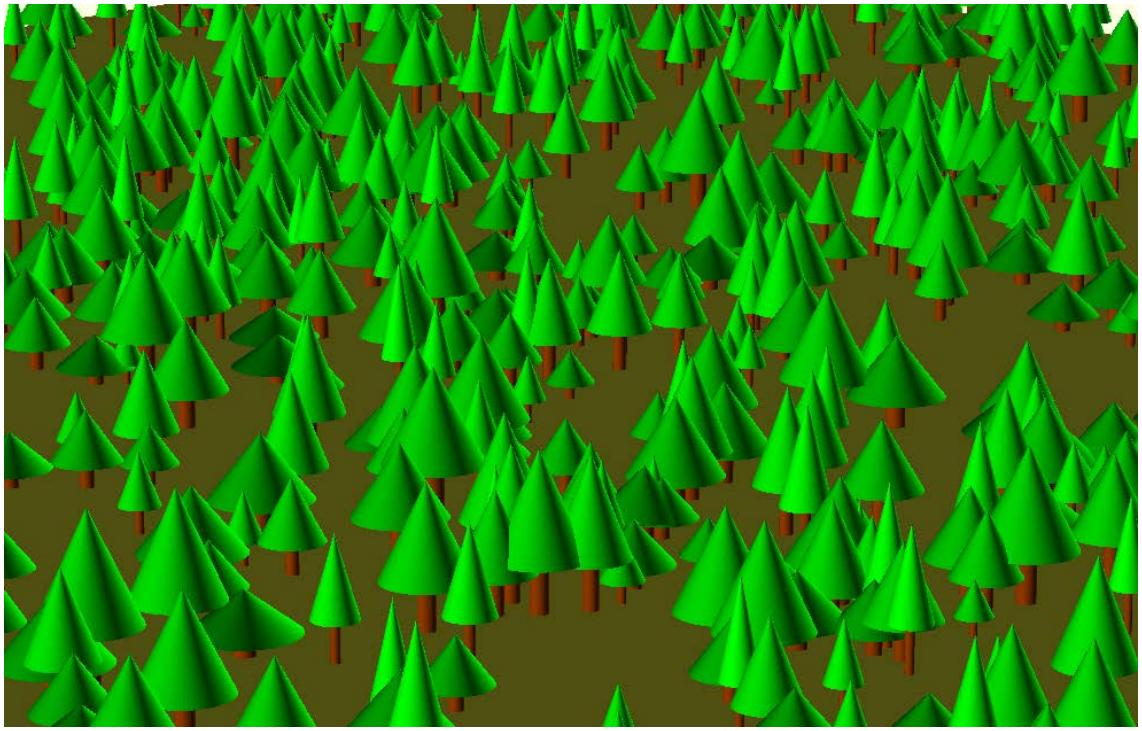


Figura 20. Representación de un bosque utilizando figuras geométricas con OOML. A partir de conos, cilindros y rectángulos, se puede utilizar algoritmos seudo-aleatorios para simular efectos naturales, como la distribución de un bosque.

Se utiliza un bucle "*for*" de 1000 ciclos (por ejemplo) para declarar 1000 árboles: se asignan dos valores aleatorios a las variables "*radius*" y "*height*", que serán los parámetros del árbol; se llama al componente "*pine*" (el objeto árbol) pasándole los parámetros mencionados; se vuelven a asignar dos valores aleatorios, a las variables "*x*" e "*y*"; y por último, se desplaza el árbol las coordenadas marcadas por esas dos variables y se añade al escenario junto con el suelo, retornando al comienzo del bucle.

Finalmente, una vez implementados los 1000 árboles, se transmite el código al objeto "*writer*", que a su vez aplica decodifica el objeto en el lenguaje *OpenSCAD*, y se transfiere al archivo "*forest.scad*" mediante el objeto "*os*".

7 MARS

7.1 INTRODUCCIÓN

Uno de los principales objetivos de este proyecto ha sido la implementación de un sistema de tracción complejo a través de la herramienta *OOML*. Este sistema de tracción se ideó para terrenos accidentados, más concretamente, destinado a la simulación de terrenos marcianos.

Con este fin, diversos estudiantes de la universidad han participado en la elaboración de una maqueta del terreno de Marte, de tamaño 4x3 m^2.

Este proyecto forma parte de un compendio de proyectos relacionados entre sí, asociados al planeta rojo. El proyecto desarrollado por Naiara Escudero Sánchez pretende proporcionar el sistema de navegación; el proyecto realizado por Alejandro Aguilar Romero consiste en la implementación de un brazo robótico; y el presente proyecto provee de un sistema motriz, así como el soporte del brazo robótico.

El mecanizado del diseño se lleva a cabo mediante las impresoras 3D disponibles en la universidad. La electrónica empleada consiste en un microcontrolador *Arduino Uno*, 8 servos *FutabaS3003*, una batería lipo *Turnigy* y una *Protoboard* de tamaño reducido.

El diseño resultante incluye la funcionalidad de una base auto-regulable, a la que se añadirían dos sensores de inclinación y un microcontrolador *ArduPilot*.

En principio, el proyecto comenzó a realizarse con la primera versión de *OOML*, más compleja de utilizar, por lo que se experimentó con el diseño del robot *MiniSkybot* [13] (desarrollado por Juan González Gómez). Este ya había sido implementado a través de la plataforma *OpenSCAD*.

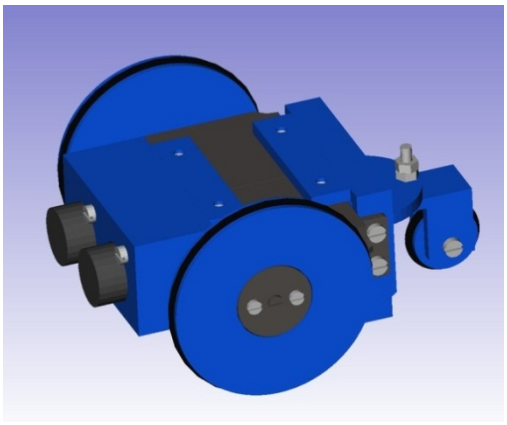


Figura 21. Imagen del diseño MiniSkybot, por Juan González Gómez.

Con conocimientos básicos del lenguaje *C++* se consiguió reproducir el diseño en el entorno *OOML* en menos de tres días, y en muchas menos líneas de código que con la herramienta *OpenSCAD*. Se adjunta el código del diseño como anexo (*Anexo 3*).

Partiendo de la experiencia adquirida con la programación de este prototipo, se comenzó a diseñar distintos prototipos para la navegación por la maqueta marciana.

7.2 SISTEMAS PROPUESTOS

En un comienzo, se propuso una serie de diseños para la tracción, cada uno con características diferentes que les conferían unas mayores prestaciones en unos u otros ámbitos. Entre los primeros diseños constan los siguientes prototipos:

- Sistema de tracción a cuatro ruedas con un rodamiento, para una mayor estabilidad. Confiere al diseño una gran simplicidad, facilitando posteriormente su duplicación.

- Sistema de cuatro ruedas omnidireccionales, basadas en el diseño de rueda *Mecanum Wheel* de *Bengt Ilon*.

  Estas permiten una mayor libertad de movimientos, potenciando el sistema de control sobre el dispositivo. De esta forma, se posibilita una mejor maniobra bajo circunstancias adversas como caminos estrechos u hondonadas.

- Sistema de tres ejes, con dos rodamientos y seis ruedas; cuatro a tracción y las otras dos conectadas con alguna mediante una estructura tipo Oruga.

  El conjunto de rodamientos aporta una mayor estabilidad, pues permite una alta adaptación al terreno.

  Motorizando únicamente cuatro ejes se consigue un menor gasto en material y energético, por lo que la batería alarga su duración.

  La estructura de tipo Oruga confiere una elevada adherencia al terreno, mejorando la tracción.

- Sistema de 3 ejes, con dos rodamientos concéntricos y seis ruedas; todas ellas motorizadas.

Los dos rodamientos concéntricos proporcionan, al igual que el modelo anterior, una excelente estabilidad por la alta adaptación al terreno.

La motorización de las seis ruedas compensa en gran medida la adhesión al terreno, ejerciendo una mayor fuerza sobre este.

Solo se han desarrollado dos de estos diseños por razones prácticas: aquellos que se ha considerado proveen de mayores propiedades a la estructura final.

7.3 REQUERIMIENTOS PARA EL TERRENO

El escenario de Marte consiste en un terreno erosionado, muy accidentado, compuesto de materiales granulares extensamente diseminados. Teniendo esta situación en cuenta, el mejor modelo posible sería aquel que cumpliera todas las propiedades expuestas a continuación:

- Gran adaptabilidad al terreno, para una mejor estabilidad.
- Simplicidad estructural, que mejora no solo la implementación, sino también su posterior mantenimiento.
- Fuerte adhesión al terreno, para un mejor control y un menor desgaste de los servos.
- Facilidad de control.
- Alta maniobrabilidad.
- Posibilidad y facilidad de réplica.

En un principio, se escogió el modelo de ruedas omnidireccionales por la versatilidad de movimiento que capacitaba. Sin embargo, dada la baja adhesión y la implementación exclusiva a través de las impresoras 3D, no ha sido posible su correcto funcionamiento; es decir, el hecho de que el diseño sea completamente imprimible conlleva una elevada tasa de defectos que impiden su realización.

A continuación se ha incluido un apartado que contiene referencias a este prototipo.

7.4 MECANUM WHEEL

Fueron diseñadas por primera vez en 1973, por el ingeniero sueco *Bengt Ilon* [14]. Su estructura se compone de un soporte cilíndrico sobre el que se aplicará la torsión, y un conjunto de elementos conectados a lo largo del contorno de este soporte. Estos están constituidos por un rodamiento y una estructura de contacto conectada a este, que será la que ejerza la adhesión al terreno.

Los elementos mencionados se organizan en torno al soporte principal de forma tangencial y distribuida, formando un ángulo de 45º respecto al plano perpendicular al plano tangente en el punto de unión.

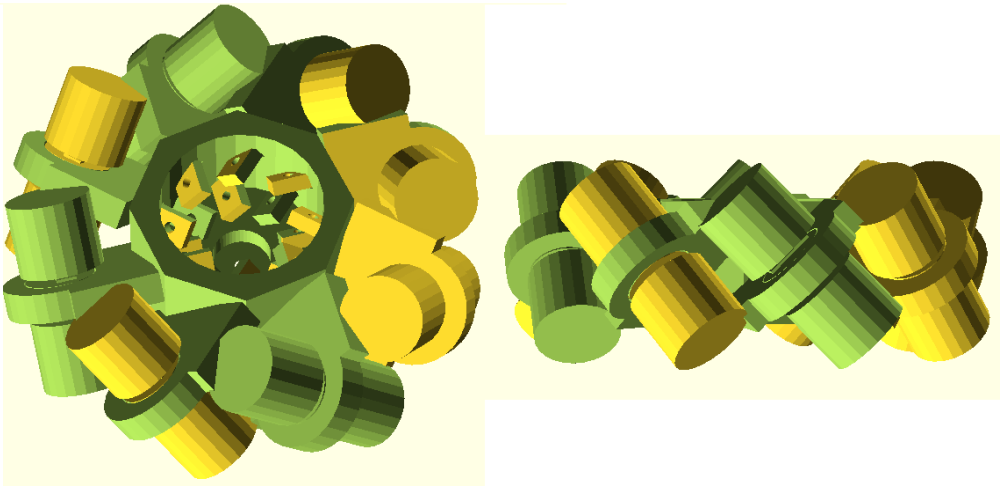


Figura 22. Primer diseño del prototipo Mecanum Wheel.

Su funcionamiento se basa en la alternancia de Pares de Fuerza (T_f). La disposición de los elementos, en una orientación de 45º, provoca la distribución de la fuerza aplicada sobre el terreno tanto en el eje paralelo a la rueda como en el perpendicular, de forma uniforme, tal y como se indica en la *Figura 23*.

De esta manera, girando 4 ruedas en el mismo sentido, se contrarrestan las fuerzas laterales, por lo que el movimiento resultante será vertical, tal y como se muestra en la *Figura 24*.

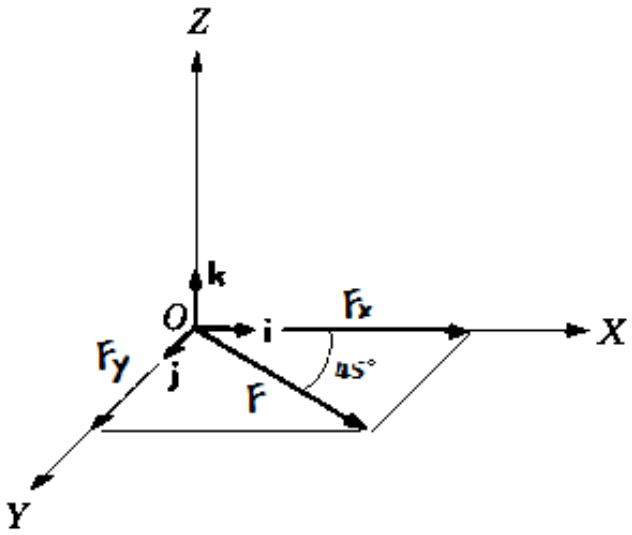


Figura 23. Distribución de fuerzas.

Si las ruedas de cada lado giran en sentidos opuestos, se anulan las fuerzas verticales, y el movimiento es rotativo.

Por último, es posible determinar los sentidos de las ruedas delanteras y traseras contrapuestos, lo cual permite un movimiento lateral.

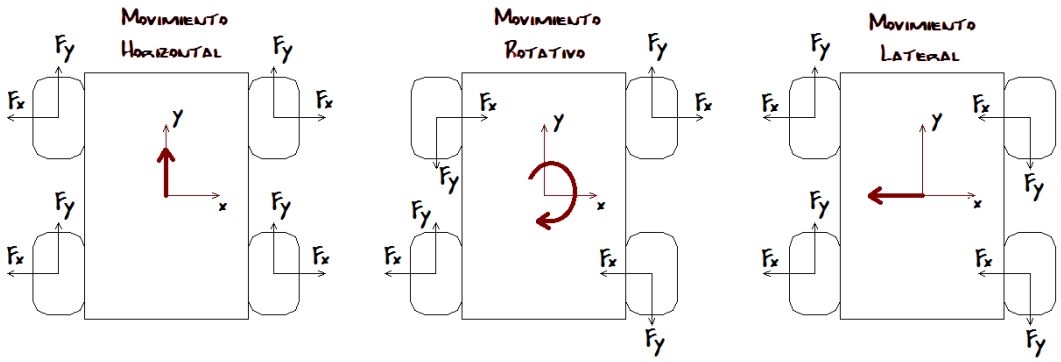


Figura 24. Representación del movimiento de una estructura basada en el comportamiento de cuatro ruedas tipo Mecanum Wheel.

7.5 ELECCIÓN DEL SISTEMA

Como se mencionaba en los apartados anteriores, la primera elección se descartó: las ruedas deslizaban en las pendientes como consecuencia de su escasa fricción contra el terreno; y los rodamientos no distribuían las fuerzas de forma uniforme, por lo que el movimiento lateral era prácticamente nulo.

En cuanto a los otros diseños, la tracción sobre cuatro ruedas presenta una baja estabilidad y adhesión, además de una limitada maniobrabilidad, por lo que se desestimó su implementación.

Se optó por un sistema de doble rodamiento, pues a pesar de la complejidad se observó que la adhesión y la adaptabilidad eran las propiedades más críticas en un terreno de ese estilo. Los dos últimos diseños conferían un sistema más óptimo en semejantes condiciones.

Finalmente se escogió el último de todos, el modelo con las seis ruedas motorizadas. La razón se centra en la mejor adhesión al terreno que presenta la motorización de dos ruedas más. A su vez, emplear dos ejes en una rueda tipo Oruga facilita en gran medida esa adhesión al terreno, pero sacrificando una parte considerable de la adaptabilidad y flexibilidad.

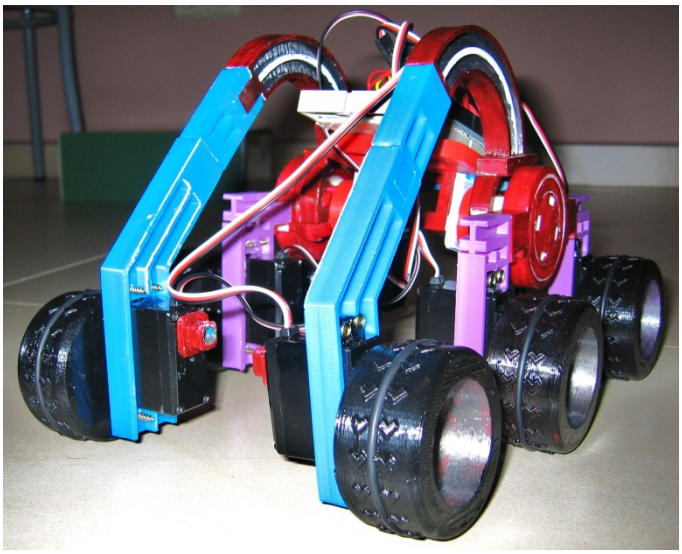


Figura 25. Diseño Mars impreso e implementado.

Por tanto, el modelo elegido es el sistema a tracción de seis ruedas motorizadas, que incluye un conjunto de dos rodamientos concéntricos, permitiendo una mayor flexibilidad de sus ejes.

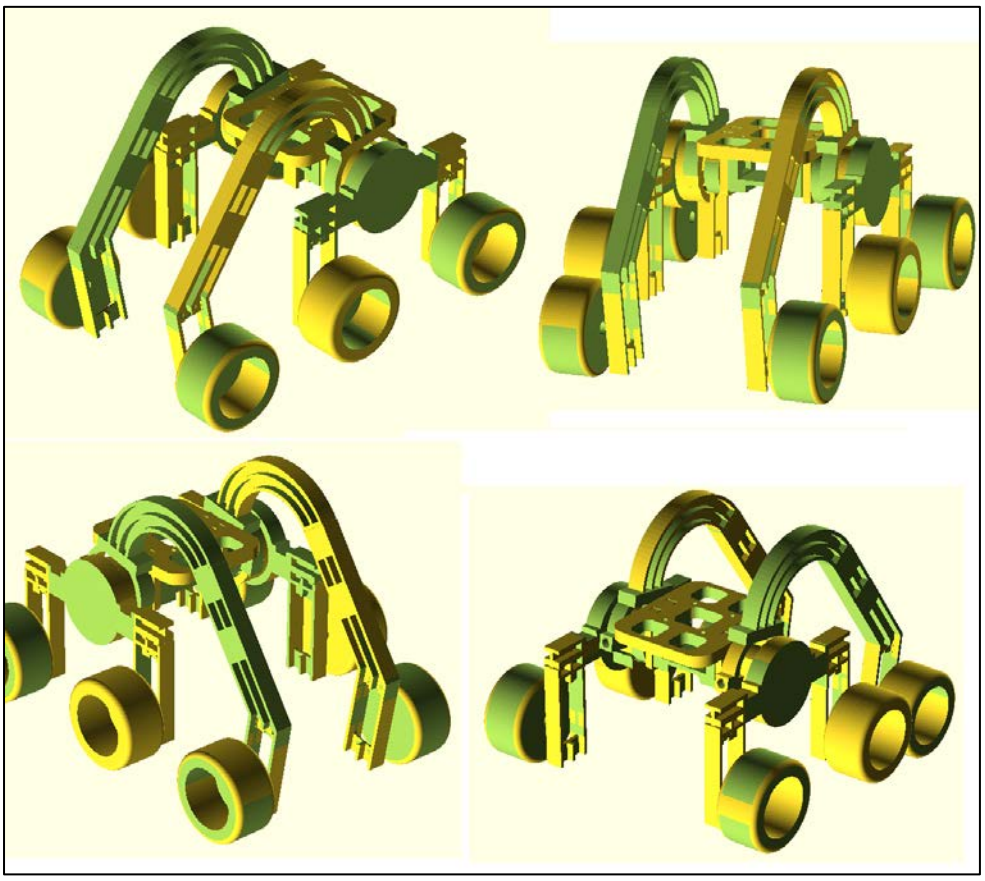


Figura 26. Prototipo virtual del modelo Mars, visualizado con OpenSCAD. Debido a las limitaciones de construcción por la superación del máximo número de elementos que soporta la plataforma, se ha simplificado el modelo. Por ejemplo, las estructuras de adhesión de las ruedas han sido suprimidas.

7.6 MODELO CURIOSITY

El funcionamiento del sistema está inspirado en el modelo *Mars Rover Curiosity*, robot de exploración construido por la NASA [15]. Este fue enviado a Marte con el propósito de recoger muestras geológicas.

Las seis ruedas motorizadas proporcionan un par elevado, en torno a 19.2 kg · cm, lo que permite adquirir suficiente potencia para alcanzar grandes pendientes y superar tramos con poca fricción.

El modelo no permite el giro de los ejes, es decir, imposibilita la modificación de la orientación de las ruedas, por lo que se debe emplear un control basado en velocidades diferenciales para determinar la dirección.

Cada lado se basa en dos sistemas balancín: el primero comprende las dos ruedas traseras, mientras que el segundo abarca la rueda delantera con el conjunto de las dos ruedas anteriores. De esta forma, cada rueda puede permanecer en un nivel distinto de altura (sin importar la irregularidad del terreno).

De forma adicional, se ha incorporado al diseño un muelle que comprime la rueda delantera contra el terreno. Así, se mantiene la movilidad de esta rueda mientras se conserva una fuerte adhesión que favorece la superación de obstáculos.

Este diseño libera el eje central, donde se sitúa la lógica de control; por ello ha sido necesaria la incorporación de un servo adicional que sustente dicho eje.

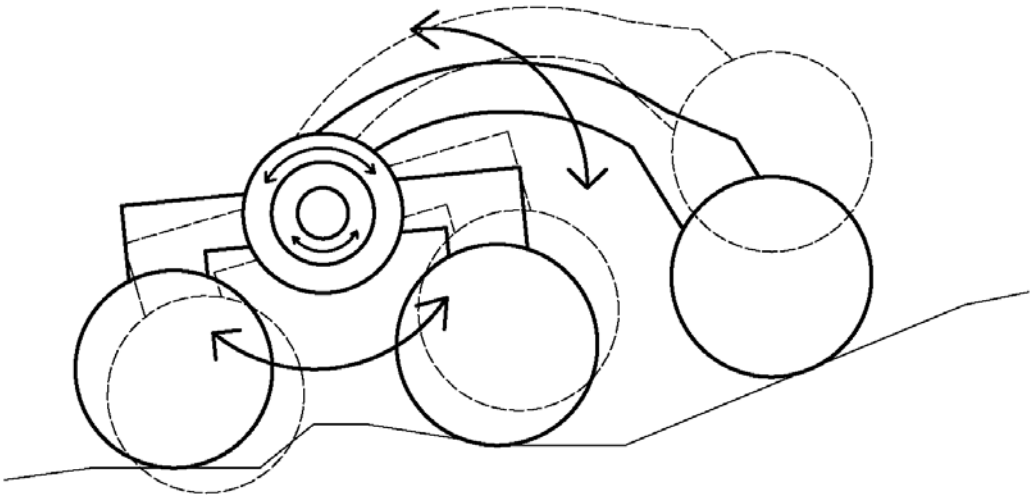


Figura 27. Representación del movimiento de los ejes.

7.7 PROTOTIPO DEFINITIVO

El prototipo se compone de 16 tipos de piezas, con un total de 29, todas ellas diseñadas para la impresión, por lo que no son el resultado del modelo más óptimo para el movimiento ni poseen la forma más aerodinámica. Sí poseen, no obstante, una

gran efectividad en la réplica y una escasa dificultad con el ensamblaje con otras piezas. Se adjuntan los planos de las piezas con el *Anexo 4*.

A continuación se realiza una breve descripción de cada pieza:

- Pieza tipo 1: Rodamiento.

  Es la pieza principal que sustenta el prototipo y proporciona el sistema de ejes. Se compone de un conjunto de doble rodamiento, ambos concéntricos entre sí. Este sistema, formado por tres cilindros con numerosas esferas reunidas entre los mismos, aporta la posibilidad de que realicen distintos movimientos las ruedas delanteras, las ruedas traseras y la base.

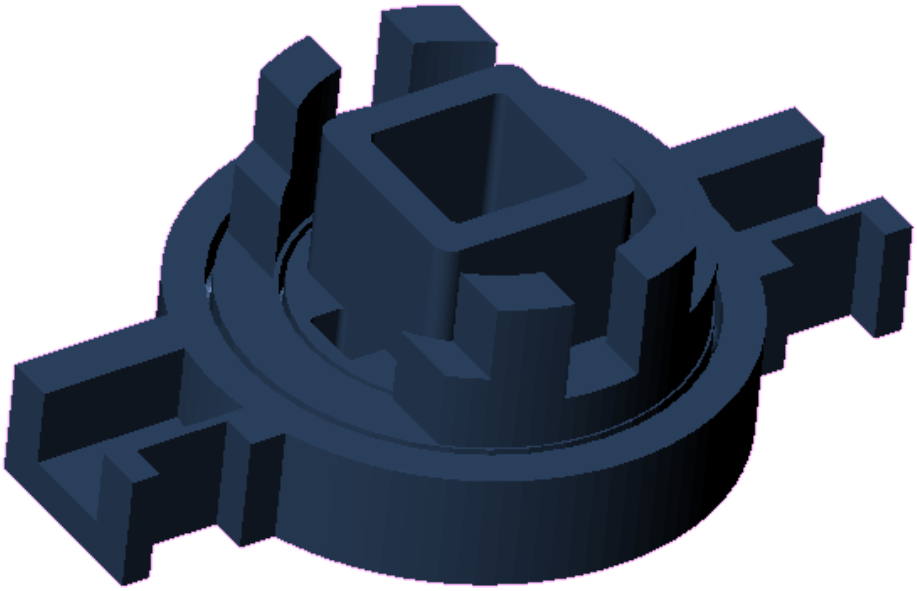


Figura 28. Rodamiento del Mars.

  Esta pieza forma prácticamente el sistema central del eje, pues facilita la unión de 6 piezas. De esta forma, se ha diseñado de forma robusta para la resistencia del par y la fuerza que concentra.

  Se dispone de dos piezas de este tipo, cada una correspondiente a un eje.

- Pieza tipo 2: Soporte.

Esta pieza sustenta uno de los servos destinado a la torsión de una rueda, y va conectada a una pieza del tipo 1. Se ha intentado emplear el menor material posible de plástico para su elaboración, por lo que se intenta usar el servo como parte del diseño, a la vez que muchas partes se realizan en forma de "*T*".

El hueco central está destinado a la inserción de un servo, con la parte de la corona más próxima a la parte inferior (aquella que no contiene la pestaña de conexión). Este se mantiene unido mediante tornillos y tuercas, que son introducidas a través de los agujeros presentes tanto en el servo como en la pieza.

Conforme a la distribución de los salientes y hendiduras, puede ser necesario desmontar la parte inferior del servo para desplazar (hacia el lado contrario a la corona) el recubrimiento de los hilos que transmiten las señales de alimentación y control. Esta elección se debe a que ese recubrimiento servirá de sustentación a la pieza 16, tal y como se indicará en ese apartado.

En cuanto a la conexión con la pieza de tipo 1, el saliente superior es introducido en una hendidura de medidas similares.

Se utiliza una pieza de este tipo por cada eje.

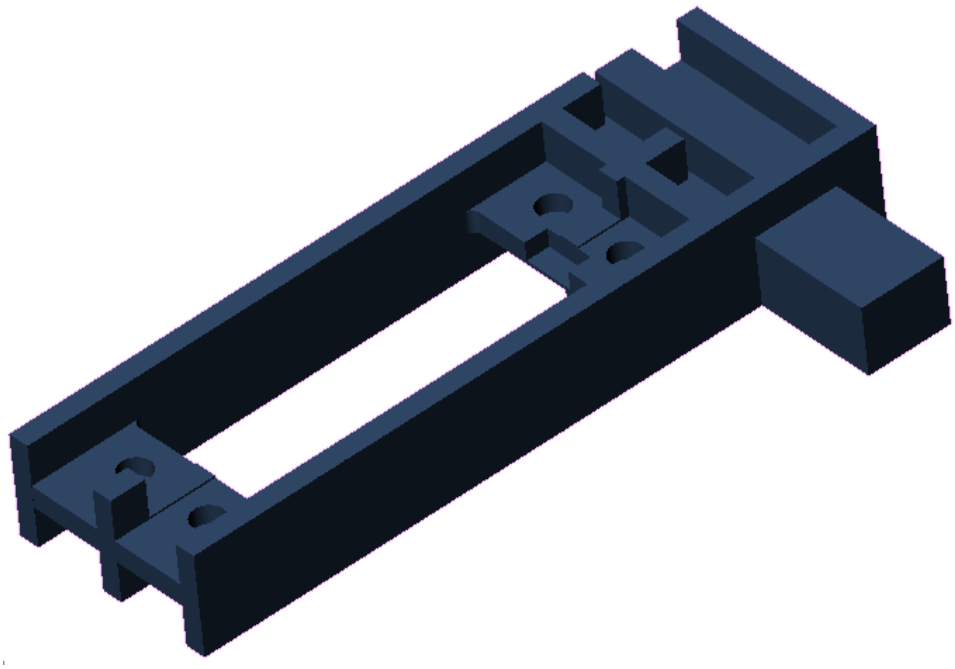


Figura 29. Soporte 1 para un servo del Mars.

- Pieza tipo 3: Soporte.

  Es totalmente simétrica a la pieza anterior, pues se emplea con la otra rueda trasera.

  Se emplea una pieza de este tipo por cada eje (dos en total).

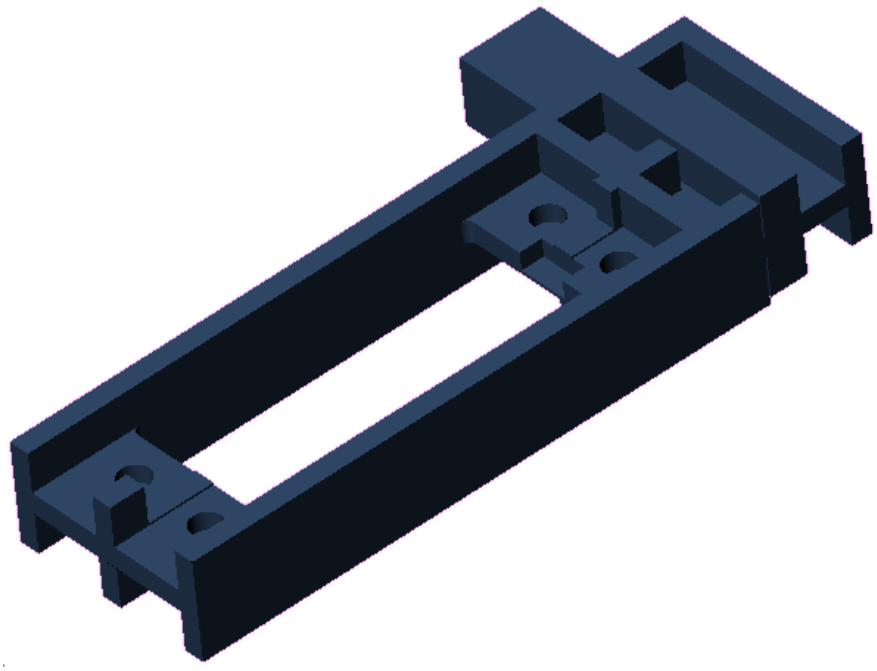


Figura 30. Soporte 2 para un servo del Mars.

- Pieza tipo 4: Conector.

  Transmite el movimiento de uno de los rodamientos al extremo de la rueda delantera. Se conecta, por tanto, con la pieza de tipo 1.

  El saliente a la derecha (según la *Figura 31*) se conecta a la pieza 6 de la misma forma que la conexión entre pieza 2 y 1.

  La parte izquierda (según la *Figura 31*) se extiende a lo largo de la parte superior de la pieza 1, de forma concéntrica, encajando los extremos rectangulares entre dos salientes.

De la misma forma que las piezas 2 y 3, se ha diseñado el tronco de la pieza en forma de "*T*", ahorrando material sin sacrificar la resistencia de la misma.

En cuanto al hueco cilíndrico, sirve como conexión a un muelle, el cual ejerce la fuerza necesaria para mantener la rueda contra el suele y adquirir una mejor adhesión.

Sólo se emplea una pieza de este tipo.

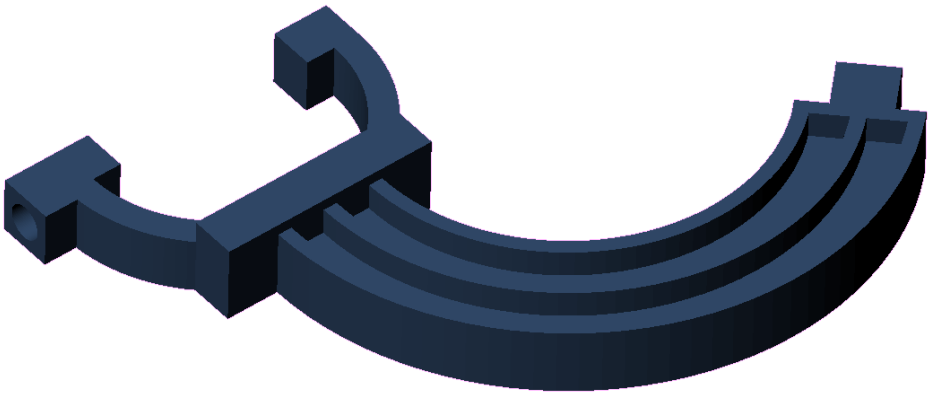


Figura 31. Conector 1 en forma de pinza del Mars.

- Pieza tipo 5: Conector.

  Es el resultado a aplicar una transformación de simetría a la pieza anterior, por lo que conserva las mismas propiedades y funciones.

  Se usa una única pieza.

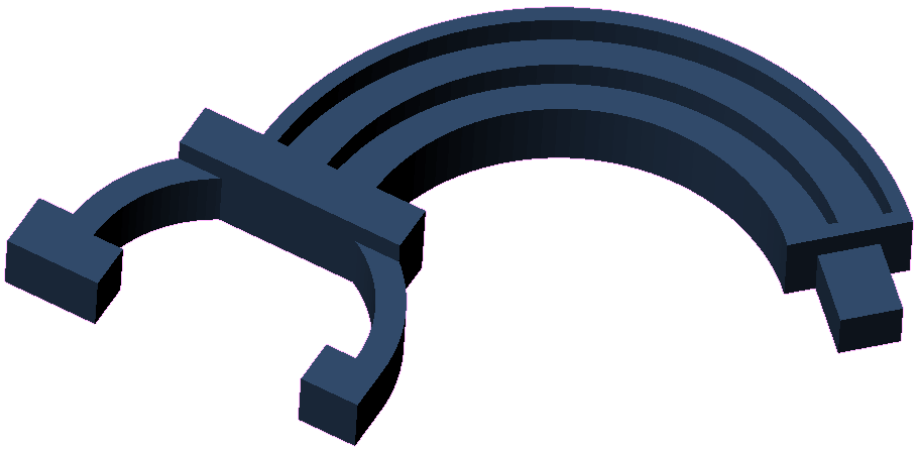


Figura 32. Conector 2 en forma de pinza del Mars.

- Pieza tipo 6: Conector.

  Debido a las limitaciones dimensionales que impone la impresora 3D en la fabricación de piezas, se deben imprimir algunas partes del diseño en varias piezas. De esta forma, se ha diseñado esta pieza como parte del conjunto piezas 4, 6 y 7.

  Se conecta a la pieza 7 mediante el saliente, y a la pieza 4 mediante la hendidura, como se ha indicado anteriormente.

  De nuevo, se intenta ahorrar material en el diseño mediante la disposición en forma de "*T*".

  Se utilizan dos piezas de este tipo; una en cada eje.

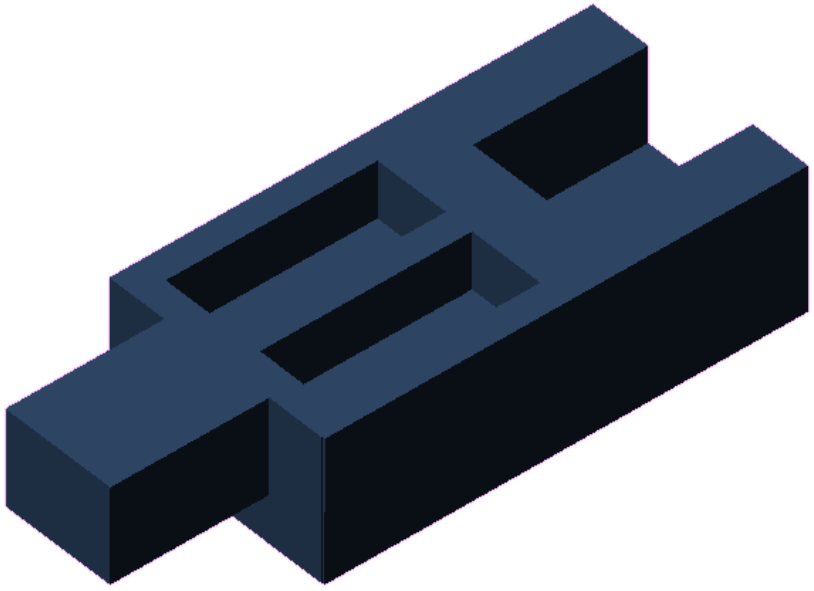


Figura 33. Conector del Mars.

- Pieza tipo 7: Soporte.

  Esta pieza sostiene el servo que impulsa la rueda delantera. Al igual que la pieza de tipo 2, el servo se inserta en el hueco, con la corona próxima al extremo, de forma que los agujeros de la pieza y el servo queden alineados.

Se vuelve a utilizar la estructura en forma de “T”, y la hendidura rectangular para la unión con la pieza 6.

Se emplea una pieza de este tipo.

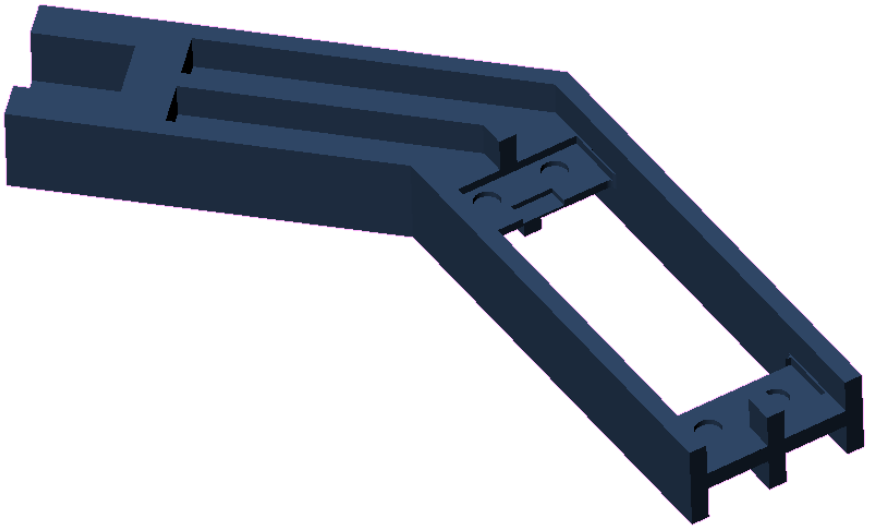


Figura 34. Soporte 3 para un servo del Mars.

- Pieza tipo 8: Soporte.

  Totalmente simétrica a la pieza 7, contiene las mismas funciones y propiedades que esta.

  Su uso se destina al soporte de una de las ruedas delanteras.

  Una única pieza de este tipo es usada en el diseño *Mars*.

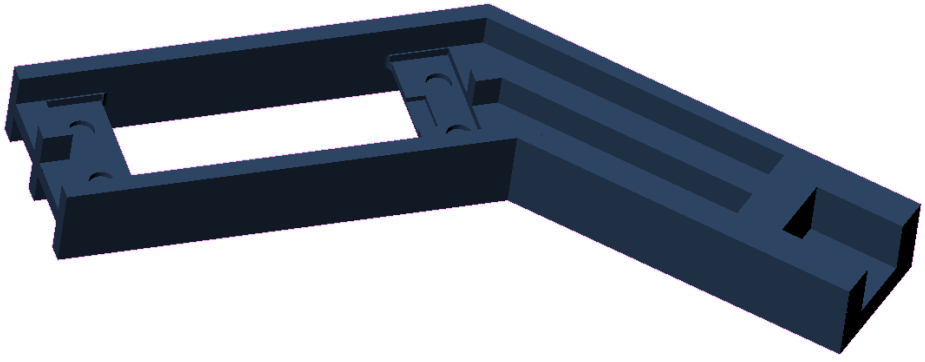


Figura 35. Soporte 4 para un servo del Mars.

- Pieza tipo 9: Rueda.

  Este diseño proporciona la movilidad al prototipo. Contiene un hueco en el medio para la conexión de una corona de un servo, y dotar de un movimiento circular a la pieza.

  Para aumentar la adhesión al suelo, se han diseñado una serie de salientes triangulares, distribuidos uniformemente, que aumenten la fricción.

  Además, se ha introducido una hendidura con forma de toroide, para el empleo de juntas tóricas, que contribuyan al rozamiento; al mismo tiempo, se han curvado los extremos para mejorar el movimiento circular.

  Se utilizan 3 piezas de este tipo (todas en un mismo eje).

Figura 36. Rueda tipo 1 del Mars.

- Pieza tipo 10: Rueda.

  Se ha elaborado este diseño mediante la función *mirroredCopy(0,0,1)*, que produce una copia del diseño anterior, cambiando la simetría del mismo según los parámetros introducidos.

Al igual que la anterior, cabe la posibilidad de utilizar juntas tóricas para una mayor adhesión.

Tres de estas piezas se emplean en el otro eje.

Figura 37. Rueda tipo 2 del Mars.

- Pieza tipo 11: Conector.

  Esta pieza circular se utilizar para conectar las piezas 1 y 12, y transmitir el movimiento que libera la base del eje.

  Se conecta al rodamiento (pieza 1) mediante el hueco rectangular del centro de la pieza, y utiliza cuatro hendiduras en la parte inferior (según la *Figura 38*) para su unión con la pieza 12.

  Se emplean dos piezas en el diseño *Mars*.

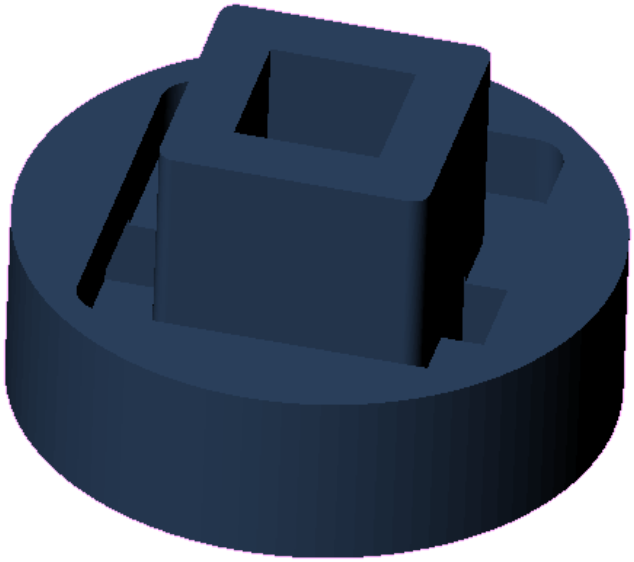


Figura 38. Conector circular tipo 1 del Mars.

- Pieza tipo 12: Conector.

  Debido a restricciones de fabricación de las impresoras 3D, resulta medianamente complejo conectar las piezas 1 y 13 (el rodamiento y la base) manteniendo en cierto modo sus formas originales. Por ello, se han diseñado las piezas 11 y 12, que eliminan cualquier complejidad estructural que se hubiese añadido a los diseños.

  Se utilizan dos piezas de este tipo.

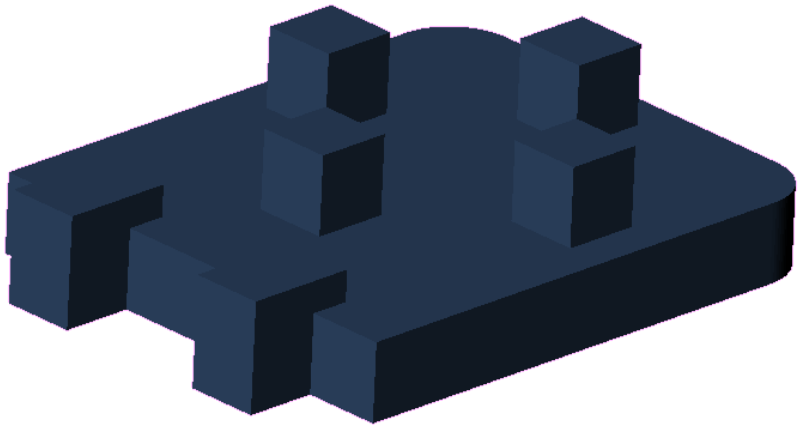


Figura 39. Conector a la base del Mars.

- Pieza tipo 13: Base.

  La base del robot ejerce varias funcionalidades: consolida ambos ejes, robusteciendo el diseño; y actúa como soporte para la batería y el controlador *Arduino UNO*.

  La batería se mantiene sujeta en el centro de la estructura, mediante su inserción. A su vez, el microcontrolador se atornilla en el medio, haciendo coincidir los agujeros de la placa con los de la pieza.

  Los huecos más pequeños a ambos lados de la estructura que actúa como soporte para la batería son utilizados como el medio de unión entre la base y las dos piezas de tipo 12.

  El resto de huecos cumple la funcionalidad de eliminar material y peso del diseño.

  Se utiliza una única pieza de este tipo.

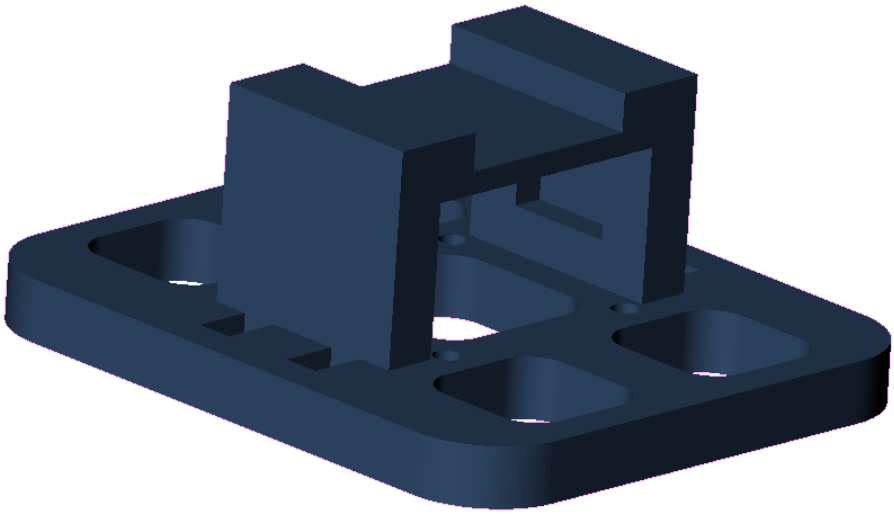


Figura 40. Base del Mars.

- Pieza tipo 14: Soporte.

  Esta pieza adicional se utiliza para la sujeción de un servo que oriente la base, a través de un sensor de posición.

Su conexión al robot se realiza mediante las hendiduras situadas en el rodamiento (pieza 1), y teniendo en cuenta la posición de la corona del servo. Este se introduce en la estructura y se fija mediante tornillos y tuercas.

Se puede emplear una o dos piezas de este tipo.

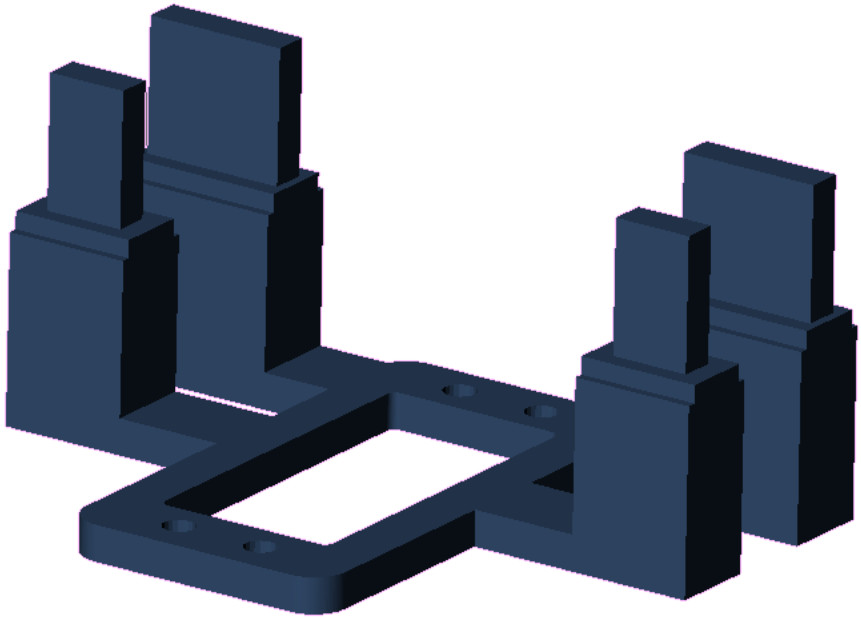


Figura 41. Soporte adicional para un servo del Mars.

- Pieza tipo 15: Conector.

  La utilización de la pieza anterior implica la introducción de esta. La corona del servo adicional es fijada al hueco central de la pieza 15, transmitiendo su movimiento al cilindro central del rodamiento (pieza 1), mediante los salientes rectangulares que aparecen en la *Figura 43*.

  Se emplea una pieza de este tipo por cada pieza tipo 15 utilizadas, es decir, se pueden llegar a utilizar 0, 1 o 2 piezas.

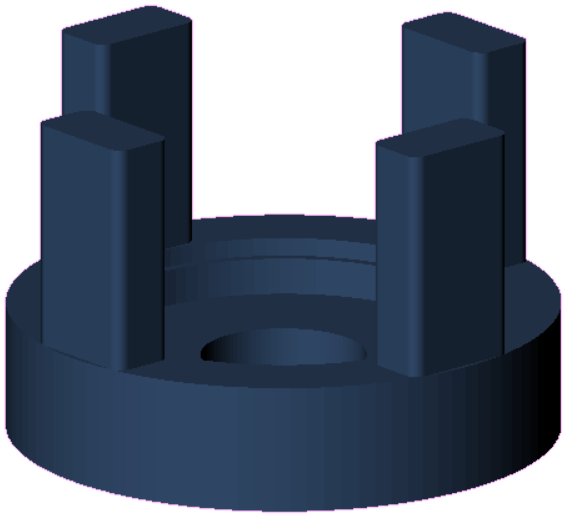


Figura 42. Conector circular tipo 2 del Mars.

- Pieza tipo 16: Soporte.

  Esta última pieza se utiliza para la introducción de un muelle en el diseño. Este muelle se inserta, por un extremo, en la hendidura cilíndrica en el centro de la estructura. El otro extremo estará sujeto a la pieza 4 o 5, tal y como se indicaba anteriormente.

  De esta forma, el muelle tiende a presionar cuando es contraído, impidiendo que la rueda delantera permanezca levantada. Además, ejerce una ligera fuerza contra el suelo, mejorando la adhesión.

  Esta estructura se conecta al diseño mediante el recubrimiento de los cables del servo, y la parte superior de la pieza de tipo 2.

  Se emplean dos piezas de este tipo: una por cada eje.

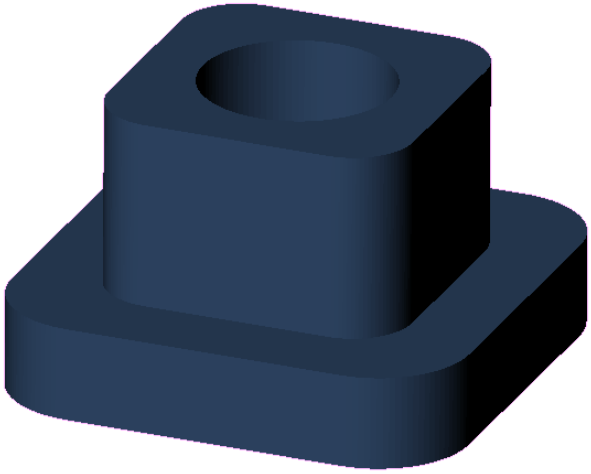


Figura 43. Soporte para muelles del Mars.

El montaje final se ha realizado con acetona para fijar las piezas de forma definitiva. Además, se ha decidido dar un color uniforme al prototipo con pintura para mejorar el aspecto visual del mismo.

A continuación se expone el prototipo junto con la maqueta del terreno marciano.

Como se ha mencionado en el apartado anterior, la cinemática de este diseño es muy amplia. En la *Figura 44* se puede observar los efectos del sistema cinemático: cada lado del diseño mantiene un movimiento independiente del otro, al igual que la rueda delantera no está limitada por las traseras.

El código *OOML* del diseño completo se adjunta con el Anexo 5.

Figura 44. Versión definitiva del prototipo Mars.

8 IMPLEMENTACIÓN

8.1 IMPRESIÓN

Una vez generados los archivos *SCAD* mediante la herramienta de programación *Qt Creator* y las bibliotecas *OOML*, se exportan a través de *OpenSCAD* a archivos de tipo *STL*.

Para controlar la impresora 3D se emplea la plataforma *ReplicatorG* [16], bajo el entorno *Processing*. En esta plataforma, es posible utilizar los archivos *STL* para generar archivos en formato *GCode*.

Los primeros contienen la información de las piezas mediante la combinación de triángulos; el formato *GCode* contiene las coordenadas de los puntos objetivos, las trayectorias a seguir y la velocidad de las mismas.

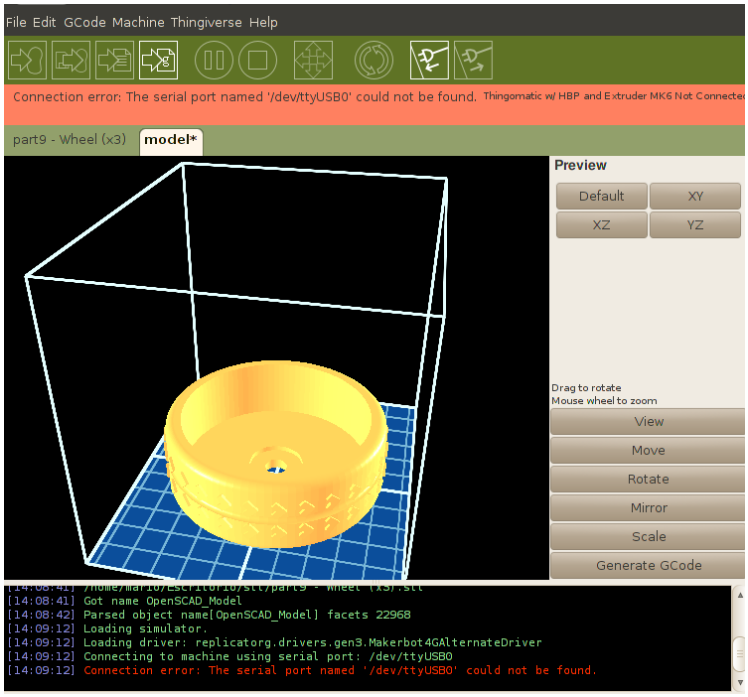


Figura 45. Captura del programa Replicatorg, que actúa de interfaz entre el usuario y una impresora 3D.

Para la impresión de cada pieza podemos modificar distintos parámetros que repercutirán en el código *GCode*, y por tanto, en la pieza final. Calibrando de forma adecuada la posición inicial en z del extrusor y la temperatura de fusión del mismo, se pueden destacar entre los parámetros más importantes: la velocidad de extrusión, el

tanto por ciento de densidad de plástico y la utilización de una capa inicial para una mayor adhesión a la base de la impresora.

La producción se realiza mediante la fusión de plástico, y su deposición sobre la base. De esta forma, las piezas se van construyendo capa a capa, por lo que existen ciertas limitaciones en la fabricación. Por ejemplo, no puede fabricarse una '*T*' tridimensional comenzando por la base, sino que habría que imprimirla al revés.

En total, a una velocidad de 35 mm/s del extrusor se tarda 39 horas y 45 minutos en imprimir todas las piezas del diseño.

8.2 ARDUINO

Arduino [17] consiste en una plataforma tecnológica de electrónica bajo el paradigma *Open Source*, que comprende tanto *software* como *hardware*. Su uso es muy amplio, pero está enfocada a la construcción de prototipos.

Esta tecnología se concentra en un*a gamma* de controladores, entre los que destaca *Arduino UNO*, y un software para la programación de los mismos.

La placa *Arduino UNO* es la que se empleará en este proyecto. Esta se basa en el microcontrolador *ATmega328*, con un oscilador de 16MHz, 6 entradas analógicas y 14 pines digitales de entrada o salida.

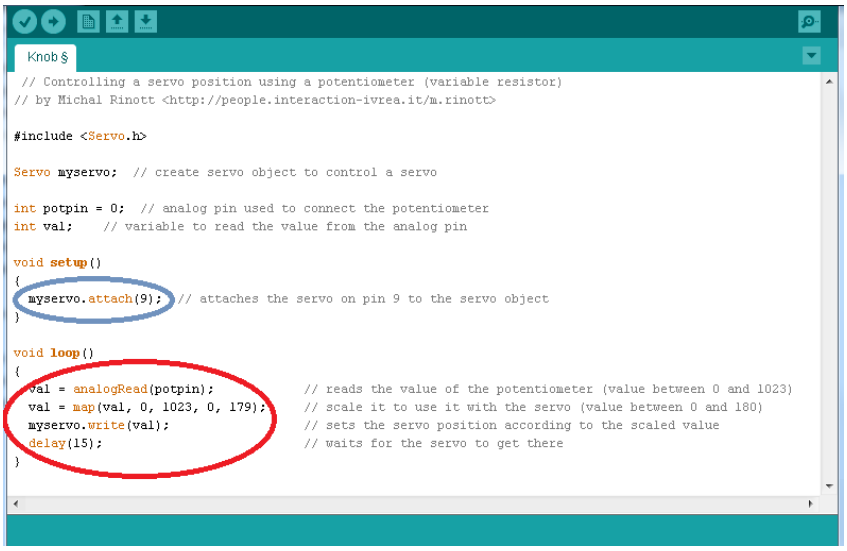


Figura 46. Ejemplo de una programación en la aplicación *Arduino*. El código se divide en dos partes: la parte azul o función *setup* contiene las inicializaciones; la parte roja o función *loop* contiene las líneas que se van a ejecutar periódicamente.

Cabe la posibilidad de usar 6 de esos pines digitales para transmitir una señal *PWM*. El voltaje de trabajo es 5V.

En cuanto al software, utiliza un entorno de desarrollo basado en la plataforma *Processing*, con un lenguaje de programación basado en *Wiring*.

La transmisión de un programa a la placa se realiza de forma sencilla: el código implementado se compila dentro de la interfaz de *Arduino*; se conecta la placa mediante un cable *USB*; y por último, se selecciona *Cargar* para transferirlo al microcontrolador.

8.3 SERVOS

Los servos utilizados se corresponden con el modelo *Futaba S3003* [18], que contienen las siguientes características:

	Alimentado a 4.8V	**Alimentado a 6V**
Torque (kg/cm)	3.2	4.1
Velocidad (sec/60º)	0.23	0.19
Altura (mm)	36	
Anchura (mm)	20	
Longitud (mm)	41	
Peso (g)	37.2	

Tabla 2. Características del servo *Futaba S3003*.

Un servo consiste en un motor de corriente continua con una placa de control y reductoras incorporadas. De esta forma, mejora el par y es posible un control más preciso en la posición del mismo.

Para la mejorar la precisión, se utiliza un potenciómetro como sensor de posición. Esto significa que el movimiento está limitado a un rango de 180º (de -90º a +90º).

Así, se ha decidido emplear 6 servos para la tracción del robot, consiguiendo un mayor par que con motores de continua. Sin embargo, es necesario trucar los servos para conseguir un giro de 360º, sacrificando el control de los mismos.

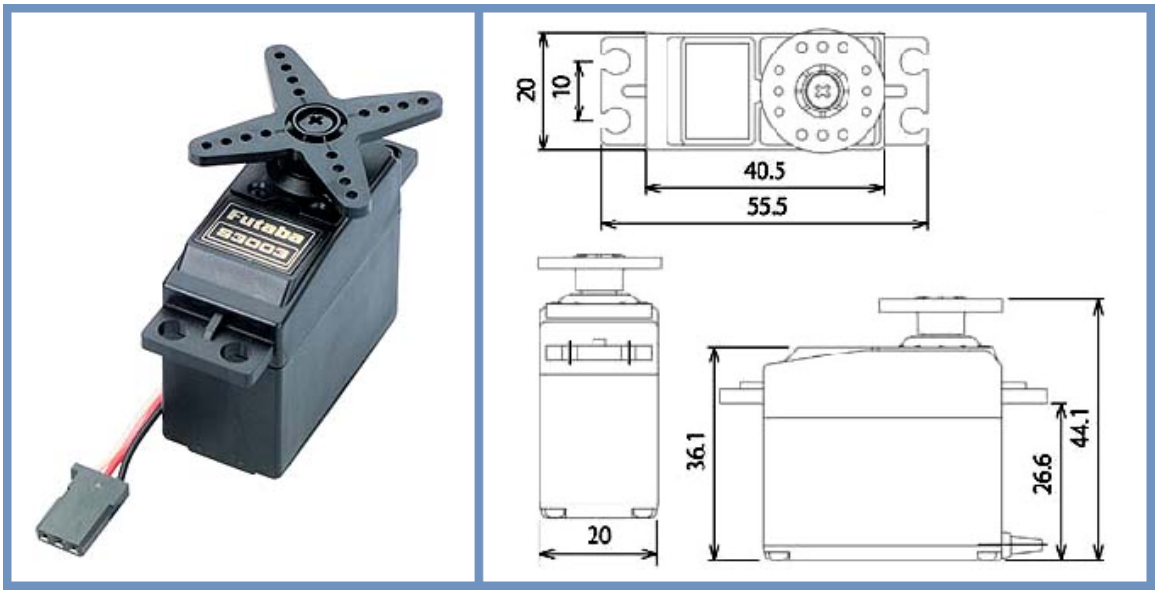


Figura 47. Plano e imagen del servo Futaba S3003.

Para ello, se describe a continuación el proceso utilizado:

- Se debe abrir el servo desenroscando los tornillos de la parte inferior y apartar la tapa.

- Se levanta ligeramente la placa de control (no se debe tirar de forma intensa pues un tornillo impide su separación del servo), y con un pequeño destornillador o *trimmer* se desenrosca un pequeño tornillo situado en un lateral, sujetando el potenciómetro.

- Tras desatornillar, se separa la placa junto con el potenciómetro (teniendo en cuenta que la placa sigue unida al motor mediante los cables de alimentación), y con unos alicates se secciona el potenciómetro de la placa.

- En su lugar, se sueldan dos resistencias en paralelo del mismo valor, siendo estos de 2k2 o 2k7. En este proyecto se han utilizado resistencias de 2k2 (2200 ohmios).

- Se vuelve a colocar todo en su sitio, salvo el potenciómetro y los tornillos.

- Se separa la parte superior dejando al descubierto los engranajes, y se debe localizar el engranaje situado en la parte superior del potenciómetro, y cortar la pestaña que limita el movimiento del mismo.

- Se cierra por último el servo enroscando los tornillos.

Una vez trucados, el control del servo cambia: el servo comienza en una posición inicial fija (por ejemplo 73º), de la que se puede desplazar a pesar de que esta no varíe virtualmente.

La posición a la que se encuentra el servo es simulada mediante la señal de referencia (la resultante entre las resistencias en paralelo), que en principio es desconocida. Esta señal no varía su valor, por lo que respecto a la lógica del servo siempre se sitúa en la misma posición.

Como las dos resistencias dan una señal continua, aunque el servo se mueva al introducir una posición final distinta de la inicial, nunca logrará alcanzar esa posición en la lógica de control, por lo que la velocidad que adquiere es también continua.

De esta forma, primero es necesario detectar cual es la posición inicial y permanente del servo. Para ello se prueban distintas señales hasta conseguir que el servo no ejerza movimiento.

Una vez conocida la posición del servo, este se puede controlar en velocidad introduciendo mayores o menores posiciones (en grados) a través del entorno *Arduino*, pues el servo intentará alcanzar esa posición en tiempos distintos según lo cercana de la misma.

Se ha comprobado experimentalmente que el incremento o decremento unitario de la escala de grados en torno a la posición virtual del servo produce una variación casi lineal de la velocidad en un rango de 20 grados. A partir de un incremento de 10 grados positivo o negativo, la velocidad alcanza su valor máximo debido a las limitaciones físicas de los servos.

8.4 BATERÍA LIPO Y CONEXIONADO

Para asegurar cierta independencia del diseño *Mars*, se ha decidido el uso de una batería que proporcione toda la energía necesaria para una completa autonomía. Con ello, se ha logrado eliminar la dependencia de cables que suministren el voltaje desde una fuente de alimentación fija.

Figura 48. Imagen de una batería *Turnigy 2200 mAh 2S Lipoly Pack*.

Se ha utilizado una batería *Turnigy Lipoly Pack* de dos celdas y 2200 mAh de amperaje. Esta batería cumple las siguientes propiedades:

Capacidad (mAh)	2200
Configuración	2S1P / 7.4 V / 2 celdas
Constante de descarga	25 C
Pico de descarga (10 s)	35 C
Peso (g)	138
Tamaño (mm^3)	114 x 33 x 17
Tipo de conector	JST-XH

Tabla 3. Características de la batería *Turnigy 2200 mAh 2S Lipoly Pack*.

Ha sido necesaria una modificación del conector entre la batería y el microcontrolador para una perfecta adaptación. Para ello, se ha soldado el conector de la batería (tipo

JST-XH) a un conector Jack mediante dos cables de hilo, uniendo los lados positivos y negativos de forma que la tensión se perpetúe a través de ambos.

Todas las conexiones se realizan sobre una *protoboard* negra de 4.3x3.4x0.8 cm^3 y 29 g. Esta contiene 170 puntos de conexión.

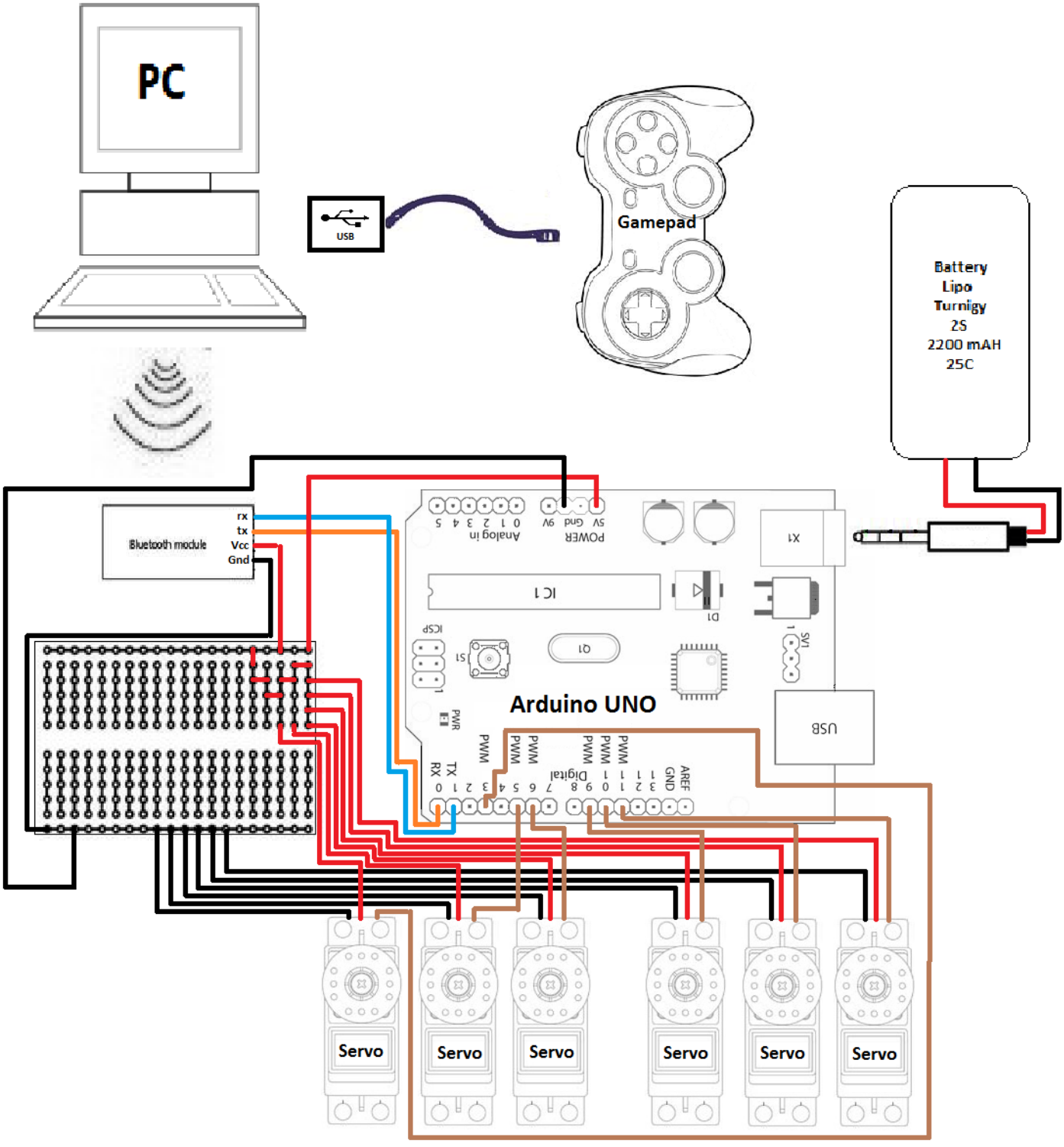


Figura 49. Esquema de conexionado.

En la *Figura 49* se muestra un esquema de las conexiones entre los distintos elementos del sistema electrónico: el *gamepad*, el módulo *bluetooth*, la batería, el microcontrolador *Arduino* UNO, la *protoboard* y los servos.

8.5 COMUNICACIÓN PUERTO SERIE (BLUETOOTH)

El prototipo es controlado mediante una interfaz gráfica bajo la plataforma Ubuntu. Para ello, se utiliza la comunicación puerto serie, combinada con una asignación de códigos a cada acción. Esta comunicación es transmitida mediante un dispositivo *bluetooth*; a su vez, los comandos son recibidos por un receptor *bluet9ooth* con protocolo de comunicación Rx-Tx mediante el microcontrolador *Arduino UNO*. Posteriormente, el microcontrolador tratará de descifrar esa letra en código *ASCII* para ejecutar la acción correspondiente.

ASCII	**Código**	**Control**
119	W	Movimiento hacia adelante
115	S	Movimiento hacia atrás
100	D	Giro hacia la derecha
97	A	Giro hacia la izquierda
120	X	Cese de movimiento
101	E	Movimiento hacia delante y giro hacia la derecha
113	Q	Movimiento hacia delante y giro hacia la izquierda
99	C	Movimiento hacia atrás y giro hacia la derecha
122	Z	Movimiento hacia atrás y giro hacia la izquierda
109	M	Incremento de la velocidad lineal
110	N	Decremento de la velocidad lineal
106	J	Incremento de la velocidad angular
104	h	Decremento de la velocidad angular
120	r	Reinicialización de las variables

Tabla 4. Código para la transmisión de movimiento.

El control desde el ordenador (Cpu) se efectúa a través de una aplicación especializada en la transmisión de datos al puerto *USB* junto con un interfaz gráfico para efectuar las acciones, y la opción de emplear un *gamepad* para un manejo más fluido. Dicha aplicación se ha realizado con la herramienta de diseño de software *Qt Creator*, facilitando en gran medida la implementación de la parte gráfica.

La configuración del puerto empleado, así como la acción de transmitir información, se ha llevado a cabo mediante la biblioteca externa a la herramienta *Qt Creator*, *QextSerialPort* [19].

A continuación se expone un resumen sobre el funcionamiento de la aplicación sin tener en cuenta el *gamepad*:

- Configuración y apertura del puerto serie a utilizar, por defecto el *"/dev/rfcomm0".*
- Comprobación de la apertura del puerto serie, y comunicación al usuario de la respuesta.
- En espera de la interacción del usuario con la interfaz (presión sobre alguna dirección, modificación de la velocidad, etc).
- Con la acción iniciada correspondiente, emisión de una letra a través del puerto serie (*bluetooth*), la cual encierra la codificación. Los códigos enviados se muestran en la *Tabla 4*.

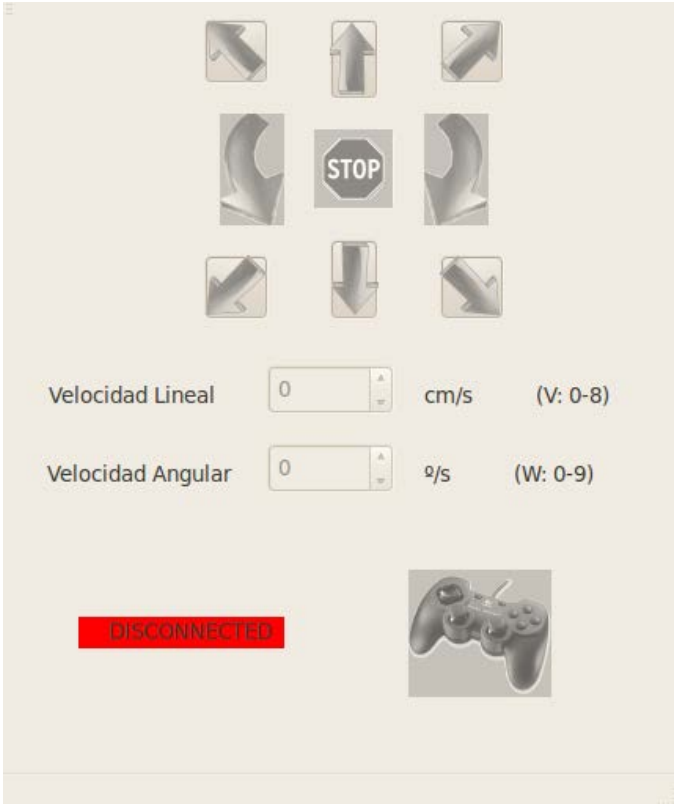


Figura 50. Panel de control sin conexión en el puerto.

- En espera de nuevo de otra interacción del usuario.

De esta forma, el programa inicializa tanto un objeto *MainWindows*, que invoca una ventana que a su vez contendrá el interfaz, como el puerto serie a utilizar con los parámetros: velocidad de 9600 baudios, paridad nula, 8 bits de longitud, 1 bit de parada y flujo de tipo hardware.

En caso de producirse un error en la apertura del puerto, se anulan toda interacción con el usuario, y se comunica el estado de "*Disconnected*", tal y como se indica en la *Figura 50*.

Si no se ha producido ningún problema con el puerto serie, aparece una ventana como en la *Figura 51*, a la espera de alguna acción.

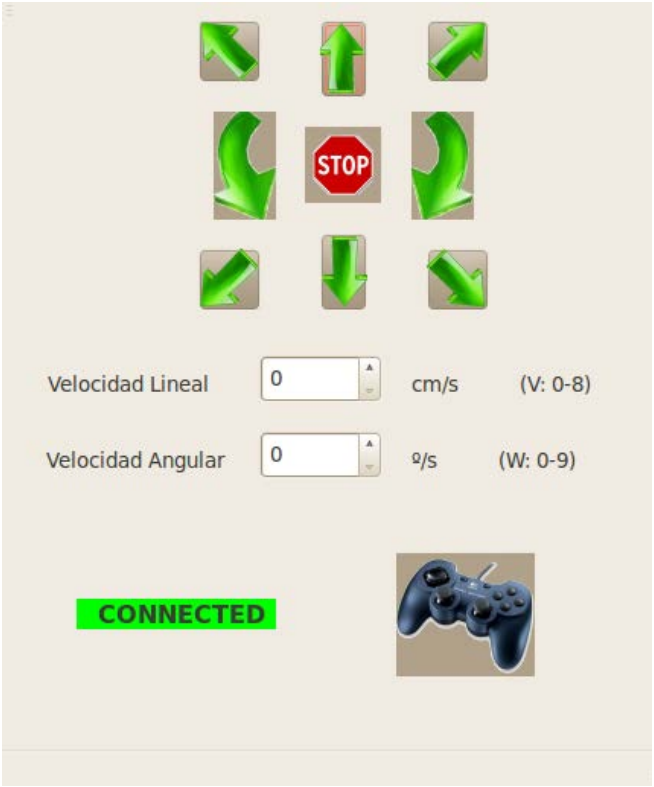


Figura 51. Panel de control en estado activo.

Cada interacción del usuario con uno de los objetos contenidos en *MainWindows* llama a una subrutina con su correspondiente proceso:

- Tanto los botones de dirección como los de parada ejecutan la actualización del estado de movimiento, representado por la variable "*move"*, comprobando previamente si el movimiento seleccionado se corresponde con el movimiento actual.
  Si el movimiento actual no coincide con el movimiento seleccionado, se produce la transmisión del código asignado al movimiento escogido, tal y como se especificaba en la tabla anterior.

  Debido a la menor velocidad de la transmisión Rx-Tx, es necesario esperar unos milisegundos para la correcta transmisión. Así, tras la función "*write(const QByteArray &data)"*, que envía en código binario la letra especificada, se invoca la función "*wait(float seconds)*" con el argumento "*0.2*", por lo que la aplicación espera 0.2 segundos en los que la transmisión termina de ejecutarse.

- Los cambios de los valores de las velocidades lineales y angulares conllevan la comprobación de si se ha producido un incremento o un decremento del valor, tras lo cual transmiten el código asociado al mismo.

  Se vuelve a llamar a la función "*wait(float seconds)*", y se comprueba el estado de la variable que contiene la información sobre el movimiento actual. Teniendo en cuenta esta información, se retransmite el código de ese movimiento para que el cambio en la velocidad sea efectuado por la placa.

  La aplicación vuelve a realizar una espera de 0.2 segundos para que la comunicación se lleve a cabo de forma correcta.

- El cierre de la aplicación (más concretamente la destrucción del objeto *MainWindows*) lleva asociado la transmisión de la orden "*parada*", y el cierre del puerto serie.

De forma adicional, se ha introducido la posibilidad de emplear un *joystick* para el control de la velocidad y la dirección del prototipo. Para ello, se incluye una acción que anula todas las interacciones posibles del usuario con el interfaz, y activa la lectura del puerto serie al que está conectado el *gamepad*.

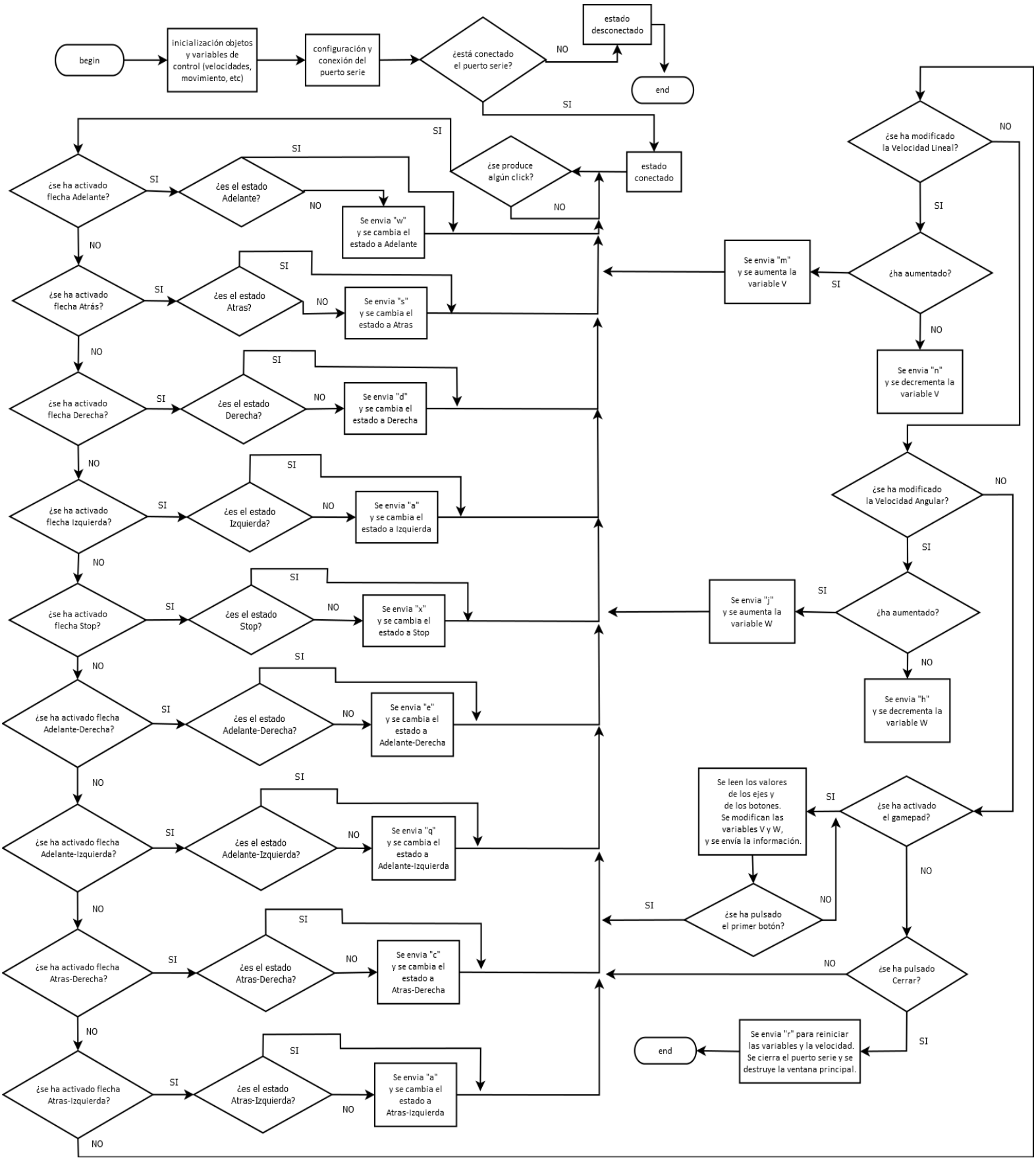


Figura 52. Diagrama de flujo de la aplicación.

Para ello, se utilizan las bibliotecas *Joystick.h* [20], que contienen la invocación del objeto *Joystick* y todas las propiedades asociadas a este que configuran y controlan el *gamepad*.

Únicamente se realiza la lectura de los ejes horizontal y vertical de uno de los joystick, así como el primer botón para indicar la finalización de dicha rutina. Inicialmente se calibra mediante el archivo *"joy_calibration.dfu"*, configurado previamente. De esta forma, cada eje alterna valores entre -180 y 180, facilitando en gran medida el control.

El proceso que sigue la rutina del *gamepad* se puede resumir en:

- Calibrar el *gamepad* e inhabilitar la interfaz gráfica.
- Leer los valores de los ejes y el botón.
- Convertir esos valores en una dirección a seguir y unas velocidades lineal y angular.
- Comparar dichas velocidades con las actuales, y enviar el código a través de *bluetooth* de aumentar o disminuir el parámetro correspondiente.
- Enviar el código asociado a la dirección.
- Volver a tomar valores del *gamepad* en el caso de no haberse activado la variable del botón 1.

En la *Figura 52* se muestra un diagrama de flujo que esboza el proceso principal de la aplicación.

Todos los archivos correspondientes a la aplicación de comunicación se detallan en el *Anexo 6*.

8.6 CONTROL

Como se comenta en el apartado relativo a la tecnología *Arduino*, la programación de estos microcontroladores se divide en dos partes: la inicialización de las variables en la función *setup()* y el proceso a desarrollar dentro del bucle *loop*.

Por tanto, el microcontrolador comienza declarando las variables *val**X*** y *myservo**X***, correspondientes, en ese orden, a los valores de posición en grados que deben adquirir

los servos para generar un movimiento nulo, así como a los objetos servos, que se asocian a una salida digital *PWM* de la placa.

Además, también se declaran las variables *scv* y *scw*, que determinan la velocidad lineal y angular que adquiere el prototipo, y se especifica la velocidad de transmisión a través de los canales *Rx*-*Tx*.

Cabe mencionar el hecho de que los servos estén trucados mediante dos resistencias de 2K2. Esto significa que los valores iniciales que se transmiten a los servos, respecto a la posición inicial de los mismos, deben corresponderse con el valor de la señal constante que recibe a través de las resistencias, la cual simula la posición real.

De esta forma, el control del servo responderá con resistencia a cambiar de posición, y solo se necesita transmitir un incremento o decremento de los grados para producir un incremento o decremento de la velocidad, que se traduce en un movimiento hacia uno u otro lado.

En cuanto al proceso principal, sigue una configuración sencilla y periódica:

- Se comprueba que el dispositivo *bluetooth* está activo.

- En caso afirmativo, una variable de tipo entero guarda el valor recibido por el canal *Rx*.

- Se compara el valor de la variable anterior con todas las órdenes que contienen un proceso asociado, y en caso de existir se imprime por pantalla y se ejecuta las líneas correspondientes.

- Se elimina cualquier información contenida en la entrada del canal de transmisión, y el ciclo se vuelve a repetir.

Todos los caracteres que se reciben desde el ordenador a través del dispositivo *bluetooth* se identifican mediante su similar numérico en código *ASCII*.

Cada acción, relativa a una orden recibida, se puede equiparar con uno entre dos tipos: dotación o eliminación de movimiento y alteración de las variables de control.

Todo lo relativo al incremento o decremento de la velocidad angular y lineal consiste en aumentar o disminuir *scv* o *scw* en una unidad, omitiendo ciertas cifras que no causan cambios significantes en la velocidad: tales como 1, en *scv* y *scw*; o 3, en *scw*.

Respecto a los mensajes de movimiento, la acción "*parar*" inicializa los valores de posición de los servos; "adelante" y "atrás" transmiten a los servos una combinación del valor de la posición inicial y la señal de control de la velocidad lineal *scv*; "derecha" e "izquierda" alteran ese valor a transmitir para que sea el resultado de la posición inicial y la señal *scw*, que representa la velocidad angular; el resto de acciones (las que son la combinación de más de un movimiento) utilizan ambas señales de control.

Figura 53. Órdenes sobre el control para la placa Arduino Uno.

Al tratarse de un control diferencial, los cambios de dirección se realizan dotando a las ruedas de un lado con una velocidad mayor a las del otro lado. Así, las órdenes "adelante" o "atrás" incrementan la velocidad de todas las ruedas en la misma proporción y en un mismo sentido; "derecha" e "izquierda" emplean velocidades opuestas, pero iguales; y el resto de movimientos usa un mismo sentido pero distintas velocidades.

Como se deduce, no se puede alcanzar la velocidad lineal máxima al mismo tiempo que se realiza un giro. Para mantener ambas condiciones, yendo a la velocidad máxima

lineal, al ser el control diferencial, las ruedas de un lado tendrían que superar la de ese valor.

Se ha otorgado prioridad al giro sobre la velocidad lineal, de forma que si se supera la velocidad máxima a la que puede ir el servo, se concede una menor velocidad al lado correspondiente, sacrificando así la velocidad lineal en conjunto.

El archivo de Arduino correspondiente al control se detalla en el *Anexo 7*.

9 INCIDENCIAS

Se han producido escasas incidencias que merezcan ser nombradas. Entre ellas destaca la imposibilidad de impresión de la pieza tipo 1, que contiene el doble rodamiento, en determinadas versiones del software *Replicatorg*. Esto se debe a un *bug* que impide la elaboración de ciertas capas del objeto.

Por otro lado, la apertura de los servos y la modificación de la señal de referencia han producido ciertas alteraciones en la señal de posición del servo que, aun no siendo significativas, si logran causar algunas molestias. De esta forma, la posición tanto inicial como indefinida de estos debería ser constante; sin embargo varía en algunos grados causando un movimiento inicial no deseado que suele conducir a una nueva calibración.

10 CONCLUSIONES

Se ha presentado la herramienta *OOML*, una biblioteca para el diseño mecánico centrada en la elaboración de robots de prototipado rápido. La expectativa en torno a esta herramienta es que posibilite la colaboración entre los investigadores de todo el mundo en la creación de nuevos modelos robóticos, facilitando el intercambio de sus experiencias y compartiendo sus diseños.

De esta forma, se dispone de una herramienta que se caracteriza por una extrema sencillez en su empleo, causada por el uso del lenguaje *C++* y la orientación a la semántica, así como por una potente rapidez en la elaboración de diseños debido a la reutilización de código que permite la *OOP*.

De forma adicional, se puede considerar su alta eficacia en la implementación de diseños complejos, pues esta se realiza a través de la combinación de las formas más simples.

Como resultado, se han desarrollado por distintos usuarios numerosas clases correspondientes a piezas reutilizables para numerosos diseños, entre las que destacan un microcontrolador *Arduino UNO*, un portapilas, una batería, un servo, una rueda loca, etc…

El trabajo pendiente consiste en incrementar el número de objetos primitivos y piezas en la biblioteca. Será la comunidad la que enriquezca la biblioteca, y por lo tanto, es necesaria una intensa labor de difusión para tener un impacto en la comunidad de investigación.

Como posibilidad se baraja la ampliación de la biblioteca para la creación de prototipos exclusivamente a través de la elección de sus piezas (reflejadas en clases). Así, sería posible elegir las partes de una lista de objetos e indicar su conexión en el diseño para una implementación semiautomática, en la que no haya que realizar operaciones.

Un trabajo más ambicioso consistiría en emplear algoritmos evolutivos para conseguir el desarrollo de diseños hasta su forma más óptima.

Se ha presentado también la implementación del diseño *Mars*, efectuado bajo esta herramienta, consistente en un vehículo de tierra para relieves accidentados. Se ha demostrado la capacidad de *OOML* de simplificar la programación, así como su potencialidad presente en la programación orientada a objetos.

Cabe destacar que se ha logrado diseñar, elaborar e implementar un prototipo robótico con escasos materiales, a muy bajo coste y exclusivamente bajo licencias *Open Source* (se ha empleado *software* y *hardware* libre). Puede verse detalles del presupuesto en el *Anexo 8*.

Por lo tanto, una vez más se demuestra que el modelo *Open Source*, además de tener cabida en el mundo científico, arrastra consigo innumerables ventajas que hacen de él un óptimo método de trabajo.

11 REFERENCIAS

[1] Sitio web oficial: http://creativecommons.org/ Fecha de consulta: 25/08/2012

[2] Más información en http://www.gnu.org/gnu/thegnuproject.es.html Fecha de consulta: 25/08/2012

[3] A. Whyte and G. Pryor, "Open Science in practice: Researcher perspectives and participation", International Journal of Digital Curation, vol. 6, no. 1, pp. 199-213, 2011. [Online]. Disponible: http://www.ijdc.net/index.php/ijdc/article/view/173 Fecha de consulta: 25/08/2012

[4] G. Robles, J. M. Gonzalez-Barahona, and J. Fernandez, "New trends from libre software that may change education", in *IEEE Engineering Education 2011* (EDUCON), IEEE Education Society. Amman, Jordan: IEEE Education Society, 04/2011 2011. [Online]. Disponible: http://www.educon-conference.org/educon2001/ Fecha de consulta: 25/08/2012

[5] Sitio web del principal distribuidor del robot *Khepera*: http://www.k-team.com/ Fecha de consulta: 25/08/2012

[6] Sitio web de la herramienta *OpenSCAD*: http://www.openscad.org/ Fecha de consulta: 25/08/2012

[7] S. Bradshaw, A. Browyer and P. Haufe, "The Intellectual Property Implications of Low-Cost 3D Printing", SCRIPTed 5, 2010. [Online] Disponible: http://www.law.ed.ac.uk/ahrc/script-ed/vol7-1/bradshaw.asp Fecha de consulta: 25/08/2012

[8] E.de Bruijn, "On the viability of the open source development model for the design of physical objects. Lessons learned from the RepRap project", November 2010.

[9] R. Jones, P. Haufe, E. Sells, P. Iravani, V. Olliver, C. Palmer and A. Bowyer, "RepRap-the replicating rapid prototyper", Robotica, vol.29, no. 01, pp. 177–191, Jan. 2011. [Online]. Disponible: http://www.journals.cambridge.org/abstractS026357471000069X Fecha de consulta: 25/08/2012

[10] Sitio web de la empresa *MakerBot Industries*: http://www.makerbot.com/ Fecha de consulta: 25/08/2012

[11] *OOML* cuenta con una wiki propia para su difusión: http://iearobotics.com/oomlwiki/doku.php Fecha de consulta: 25/08/2012

[12] *Thingiverse* consiste en un conjunto de repositorios de objetos tridimensionales, muchos de ellos disponibles para su impresión. Dirección web: http://www.thingiverse.com/ Fecha de consulta: 25/08/2012

[13] Diseño realizado por Juan Gonzalez-Gomez. Para más información visitar: http://www.ieearobotics.com/wiki/index.php?title=Mini-Skybot Fecha de consulta: 25/08/2012

[14] Olaf Diegel, Aparna Badve, Glen Bright, Johan Potgieter and Sylvester Tlale, "Improved Mecanum Wheel Design for Omni-directional Robots", Proc. 2002 Australasian Conference on Robotics and Automation. Mechatronics and Robotics Research Group. Institute of technology and Engineering, Massey University, Auckland, 27-29 November 2002. Disponible: http://ftp.mi.fu-berlin.de/Rojas/omniwheel/Diegel-Badve-Bright-Potgieter-Tlale.pdf Fecha de consulta: 25/08/2012

[15] Sitio web oficial: http://www.nasa.gov/ Fecha de consulta: 25/08/2012

[16] Herramienta de control y procesamiento para impresoras 3D. Más información en: http://replicat.org/ Fecha de consulta: 25/08/2012

[17] Más información de esta tecnología en http://www.arduino.cc/es/ Fecha de consulta: 25/08/2012

[18] Para consultar el *datasheet* del servo *Futaba S3003* entrar en http://www.cntl.kyutech.ac.jp/robocar/2010/references/kurolab/datasheet/ET-SERVO-S3003.PDF Fecha de consulta: 25/08/2012

[19] Esta biblioteca está contenida, así como toda la información respecto a su instalación y empleo, en la siguiente dirección: http://code.google.com/p/qextserialport/ Fecha de consulta: 25/08/2012

[20] La biblioteca *Joystick.h* se puede descargar del repositorio https://github.com/avalero/arducpp/tree/master/cpp/gamepad Fecha de consulta: 25/08/2012

12 ANEXOS

12.1 ANEXO 1

```
Component objetoFinal = Cylinder::create(3,12).color(0.8,0.8,0)
+ Cylinder::create(3,12).rotate(-70,0,0).color(0,0.8,0.8)
+ Cylinder::create(3,12).translate(7,-9,0).color(0.8,0,0.8)
+ Cylinder::create(3,12).translate(7,-9,0).rotate(90,45,90).color(0,0,0.8)
+ Cylinder::create(3,12).translate(7,-9,0).rotateAround(90,45,90,7,-9,0).color(0,0.8,0)
+ Cylinder::create(3,12).scale(1.2,2.2,0.3).translate(8,0,0).color(0.8,0,0)
+ Cylinder::create(3,12).scale(1.2,2.2,0.3).translate(8,0,0).mirror(1,0,0).color(0.5,0.5,0.5);
```

12.2 ANEXO 2

```
#include <components/Cube.h>
#include <components/Cylinder.h>
#include <core/IndentWriter.h>
#include <core/Difference.h>
#include <core/Union.h>
#include <core/Intersection.h>

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include <iostream>
#include <fstream>

using namespace std;

//-- A simple pine tree... or a kind of.. :-)
Component gen_pine(double trunk_height, double trunk_radius)
{
        Component trunk = Cylinder::create(trunk_radius*2,
trunk_height).color(0.5,0.2,0);
        Component top = Cylinder::create(trunk_radius*10 , 0 , trunk_height).color(0,1,0);
        Component pine = trunk.translate(0,0,trunk_height/2) + top.translate(0,0,trunk_height);
return pine;
}

int main(int argc, char **argv)
{
         IndentWriter writer;

    // Open a file where it's going to generate the OpenSCAD code.
    ofstream os("forest.scad");

    srand(time(0));

    Component ground(Cube::create(100,100,1));

    //-- Create a forest
    Component scenario = ground.translate(50,50,-0.5).color(0.5,0.5,0.1);

    for (int i=0; i<1000; i++)
    {
      double radius = float(rand())/RAND_MAX+0.5;
      //cerr << "radius="<<radius << endl;
```

```
        double height = 20.0*rand()/RAND_MAX+10;
        //cerr << "height="<<radius << endl;

        Component pine = gen_pine(height, radius);

        double x = 100.*rand()/RAND_MAX;
        double y = 100.*rand()/RAND_MAX;

        pine.translate(x,y,0);

        scenario = scenario + pine;
      }

      writer << scenario;

            os << writer;
            os.close();

      return 0;
}
```

12.3 ANEXO 3

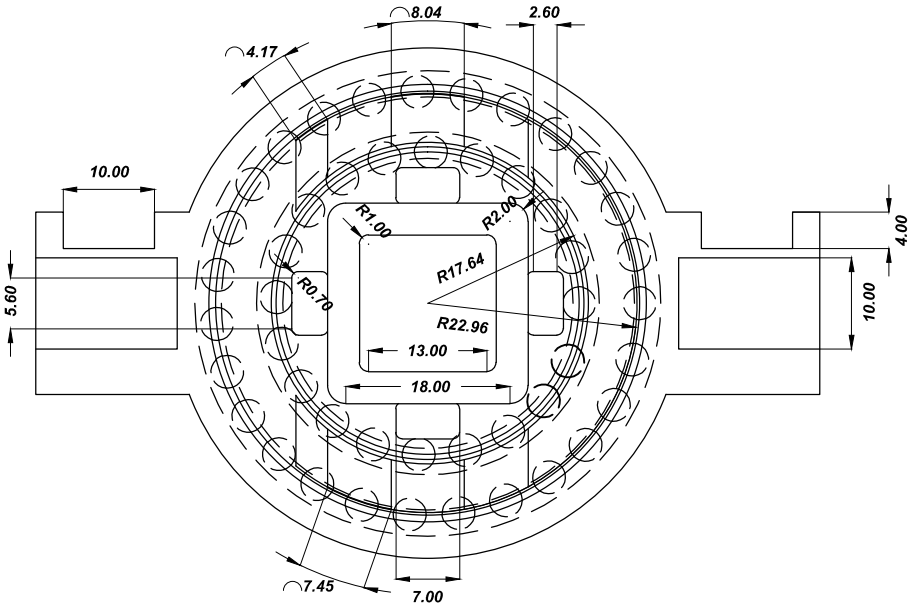
Se adjunta a este proyecto las piezas correspondientes al prototipo *MiniSkybot*, ubicadas en la carpeta "*Diseño MiniSkybot*".

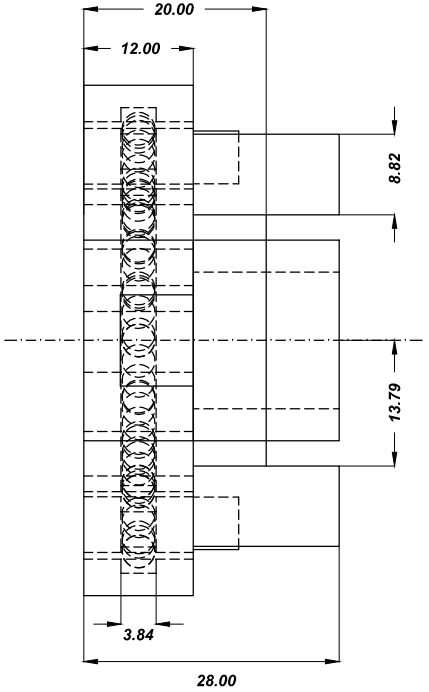
12.4 ANEXO 4

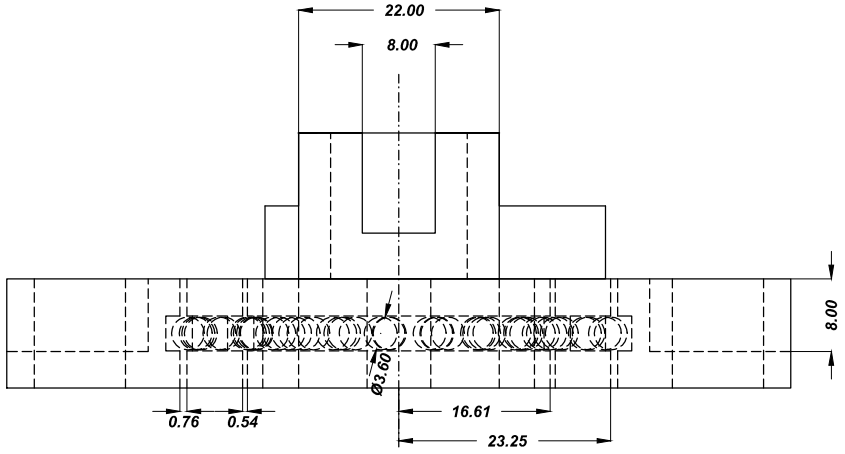
En el *Anexo 4* se presentan los planos correspondientes a las piezas del diseño *Mars*, de1 forma que se puedan transmitir toda la información necesaria para su reproducción. Se adjuntan también los archivos *dwg*, elaborados mediante la herramienta *Autocad*.

A continuación se muestra una lista de los planos adjuntos:

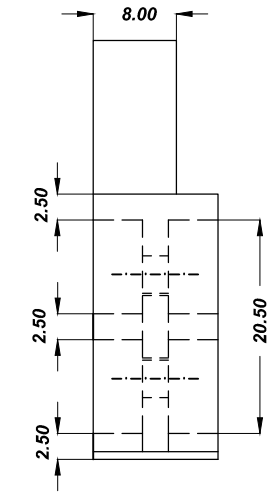
- Pieza tipo 1: Rodamiento.
- Pieza tipo 2: Soporte.
- Pieza tipo 3: Soporte.
- Pieza tipo 4: Conector.
- Pieza tipo 5: Conector.
- Pieza tipo 6: Conector.
- Pieza tipo 7: Soporte.
- Pieza tipo 8: Soporte.
- Pieza tipo 9: Rueda.
- Pieza tipo 10: Rueda.
- Pieza tipo 11: Conector.
- Pieza tipo 12: Conector.
- Pieza tipo 13: Base.
- Pieza tipo 14: Soporte.
- Pieza tipo 15: Conector.
- Pieza tipo 16: Soporte.

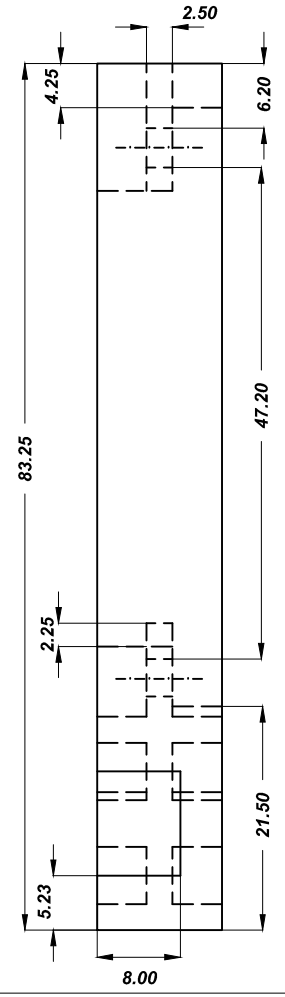


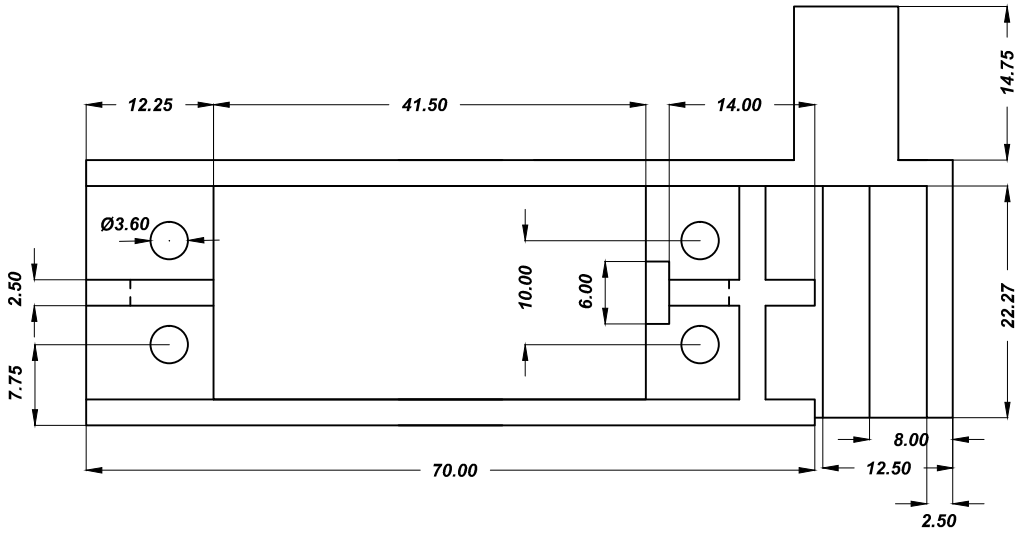




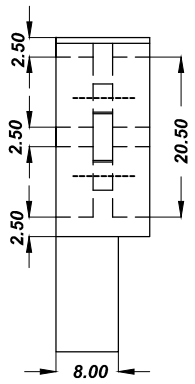
	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 4/1	PIEZA TIPO 1: RODAMIENTO			PLANO Nº 1

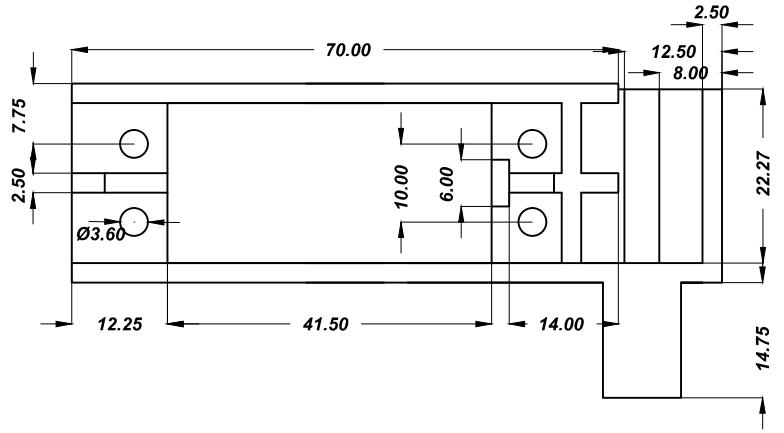


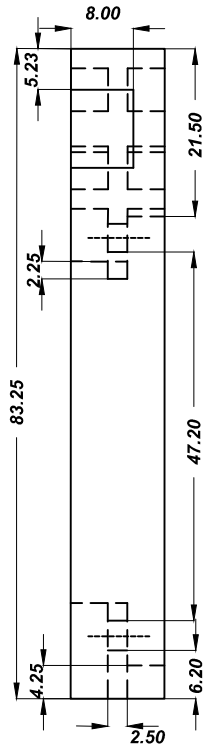




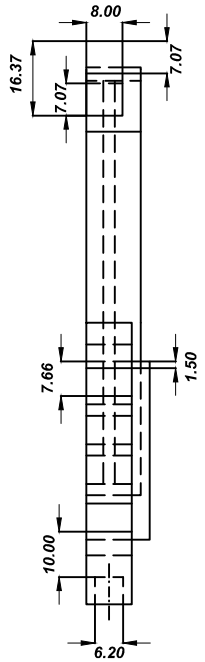
	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 4/1	PIEZA TIPO 2: SOPORTE			PLANO Nº 2

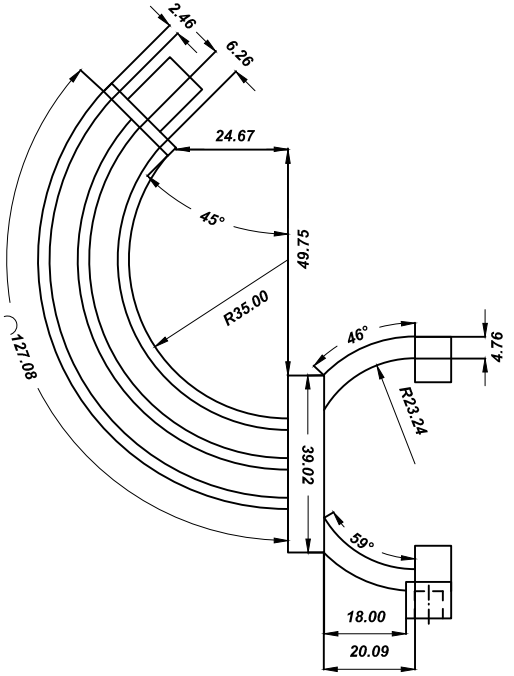


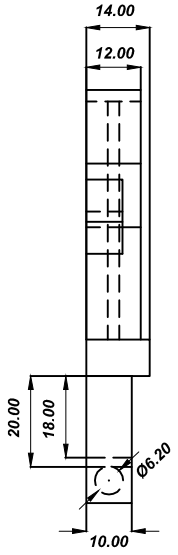




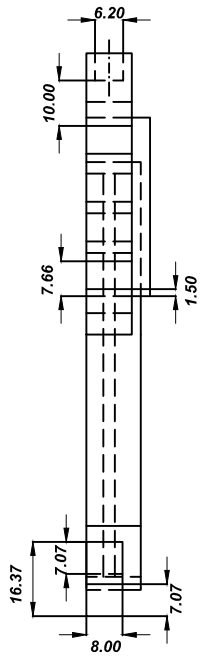
	Fecha	Nombre		OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 3/1	PIEZA TIPO 3: SOPORTE			PLANO Nº 3

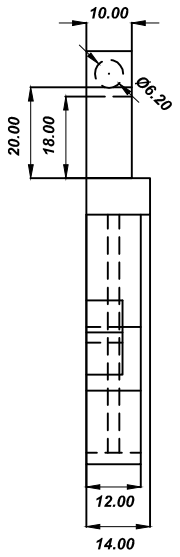


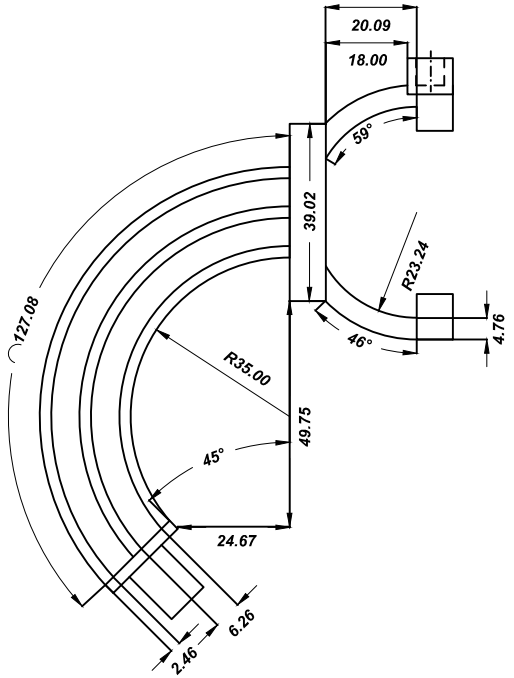




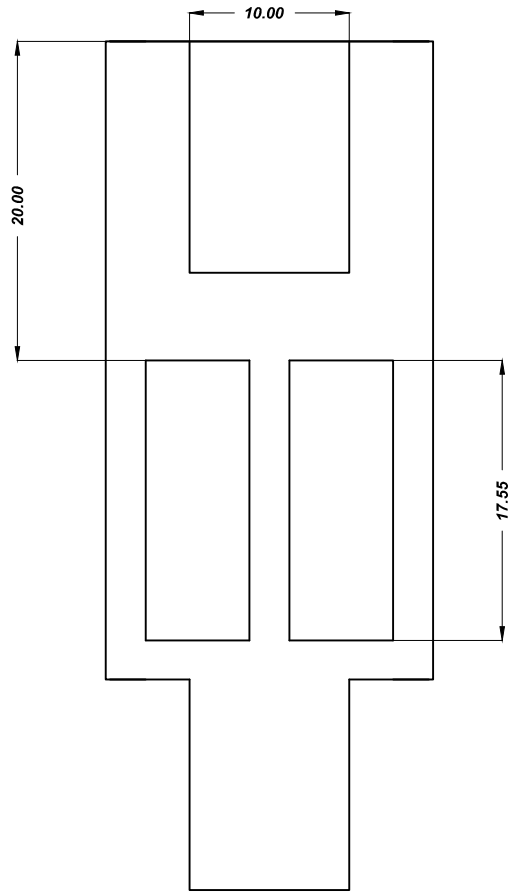
	Fecha	Nombre		OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 2/1	PIEZA TIPO 4: CONECTOR			PLANO Nº4

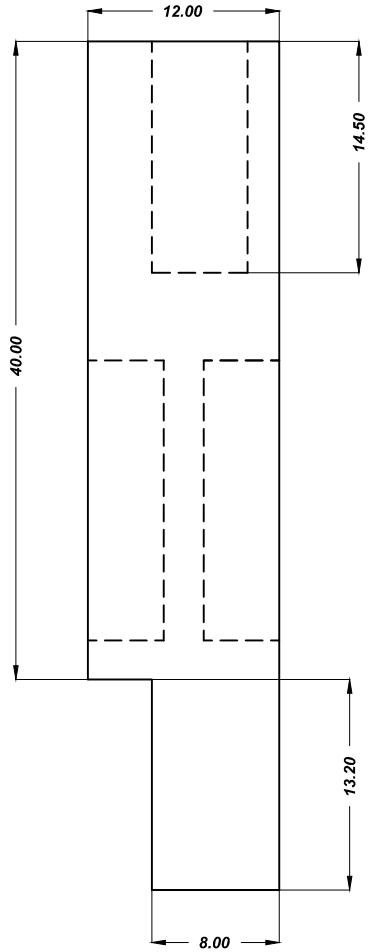


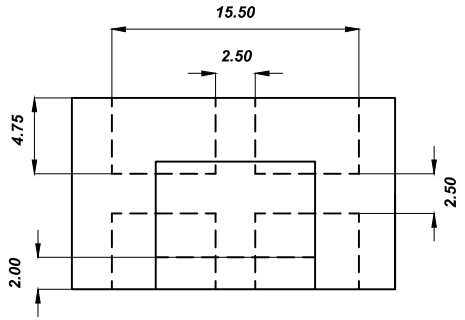




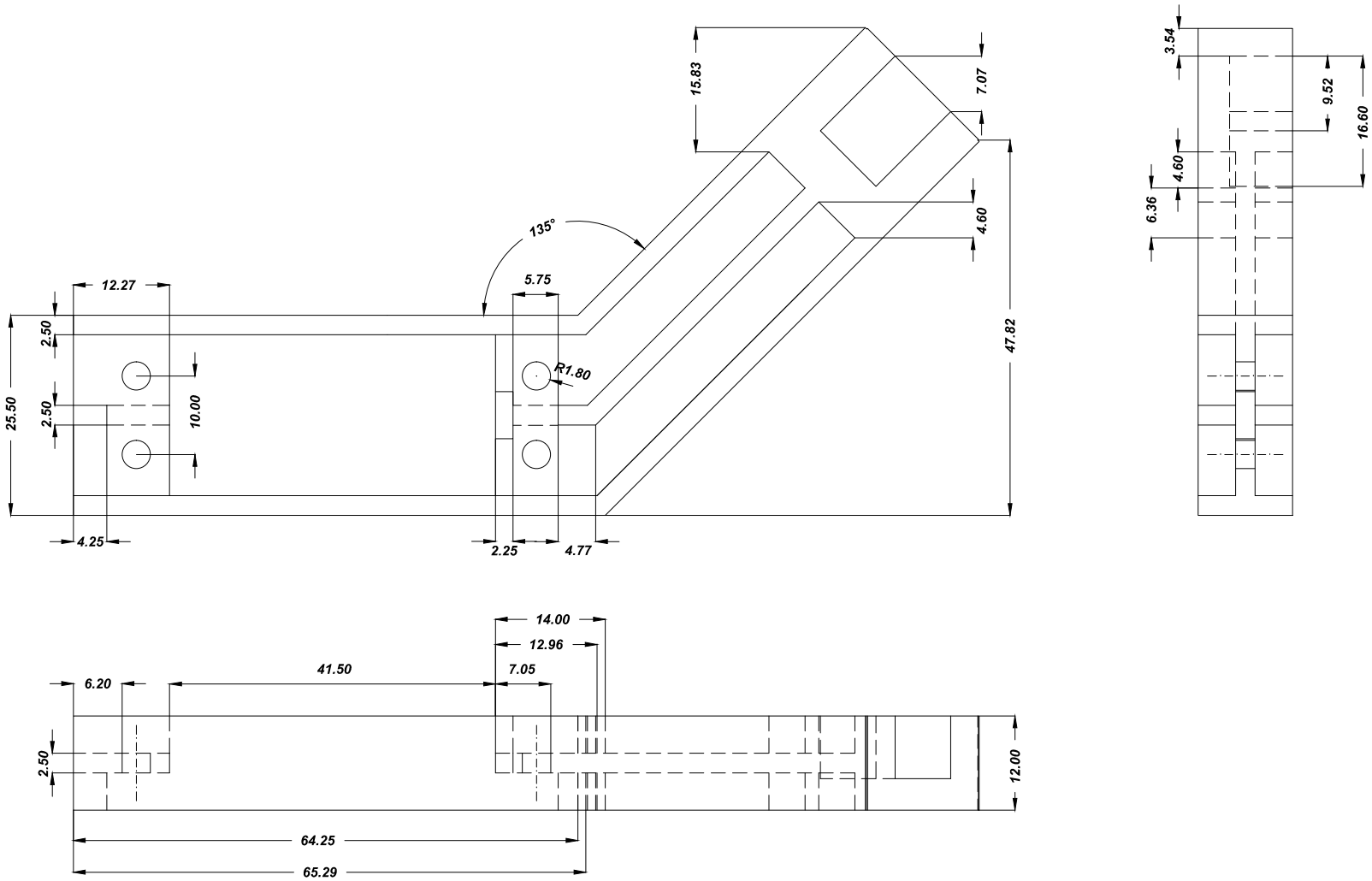
	Fecha	Nombre		OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 2/1	PIEZA TIPO 5: CONECTOR			PLANO Nº5



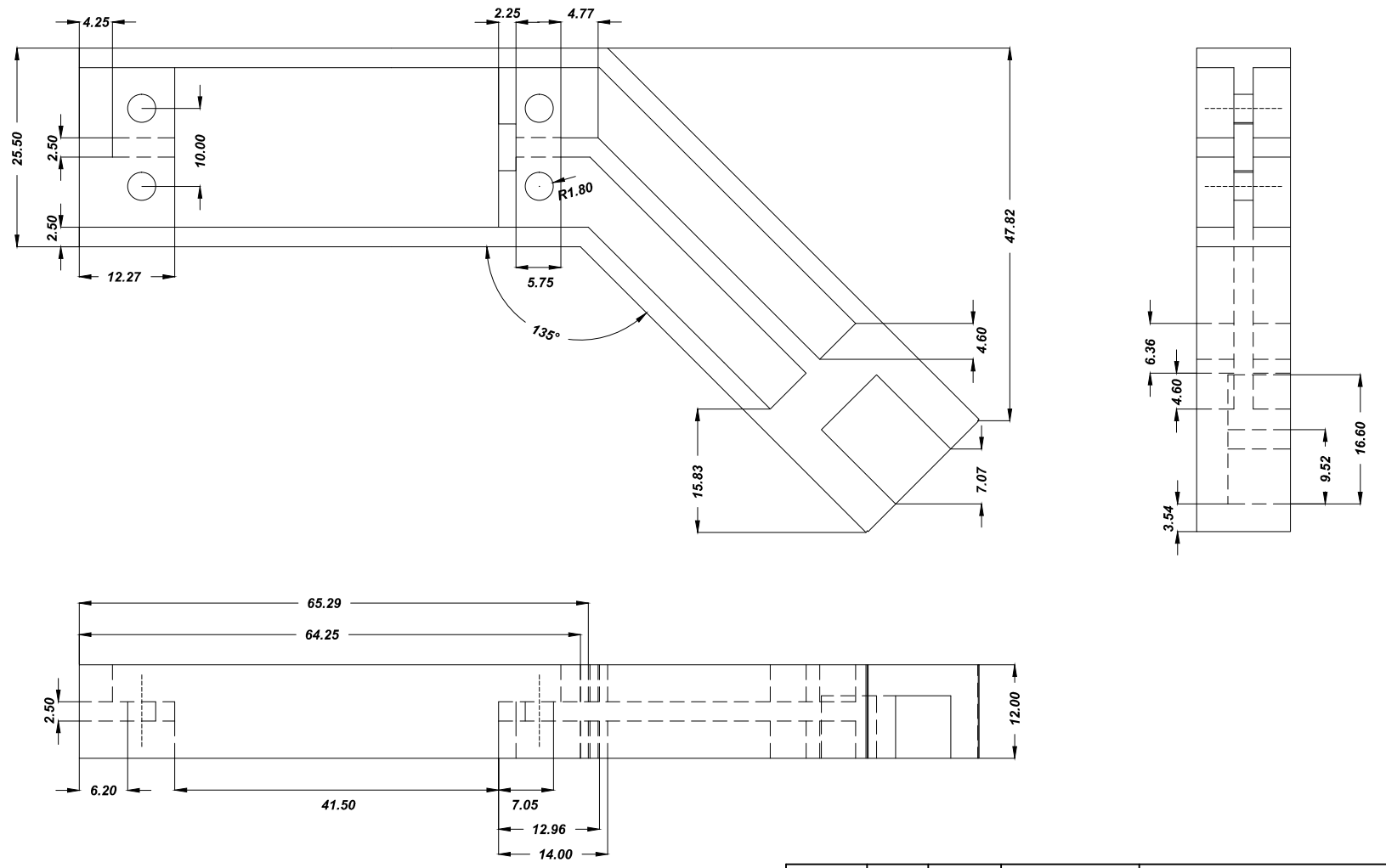




	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 7/1	PIEZA TIPO 6: CONECTOR			PLANO Nº6

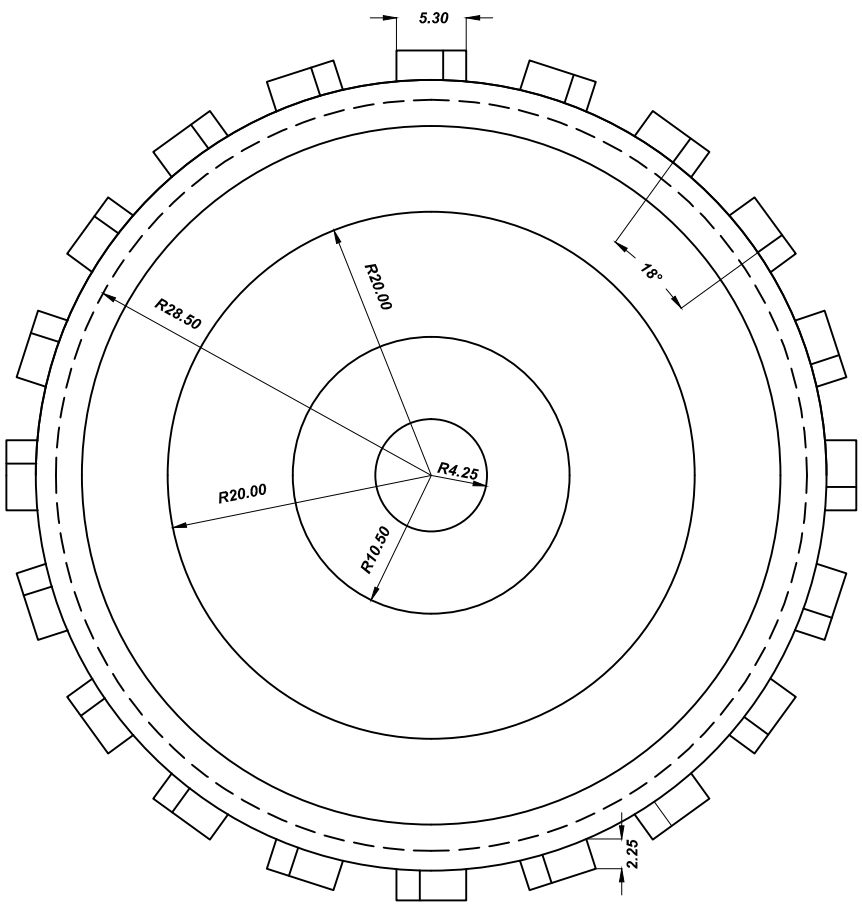


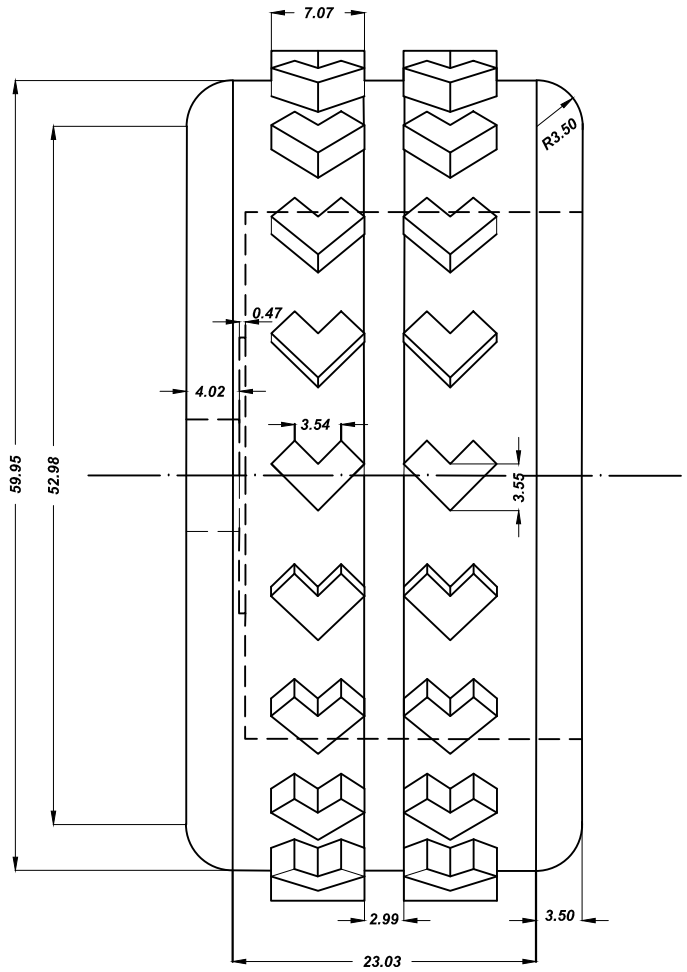
	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 4/1	PIEZA TIPO 7: SOPORTE			PLANO Nº 7



	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		

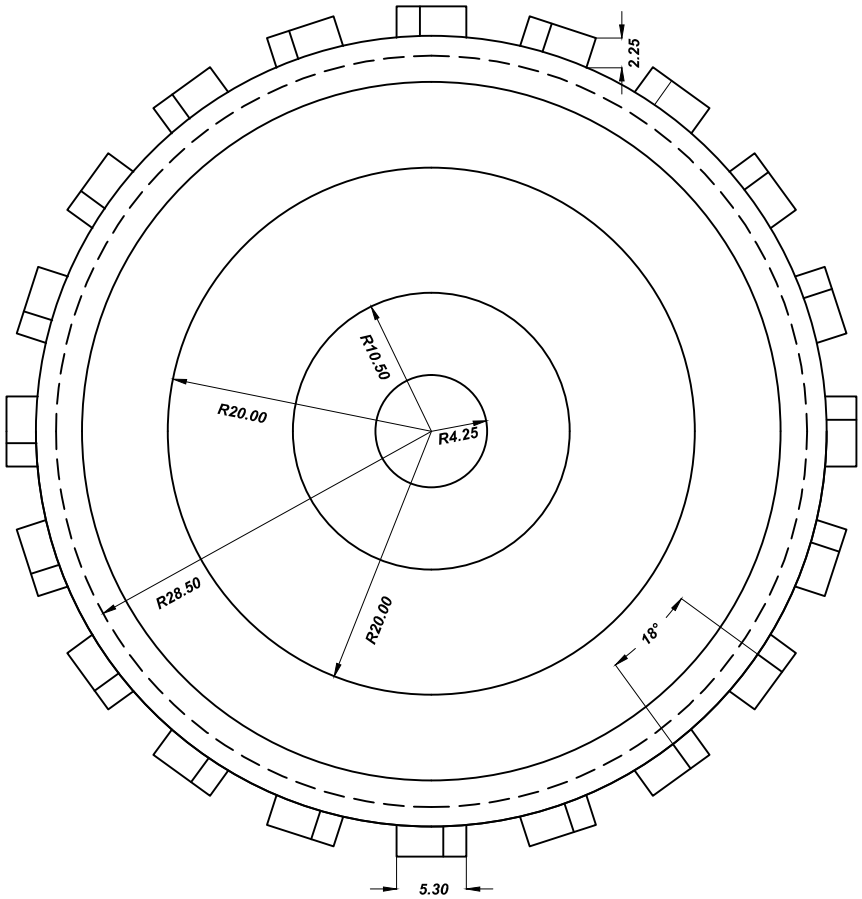
Escala		
4/1	PIEZA TIPO 8: SOPORTE	PLANO Nº 8

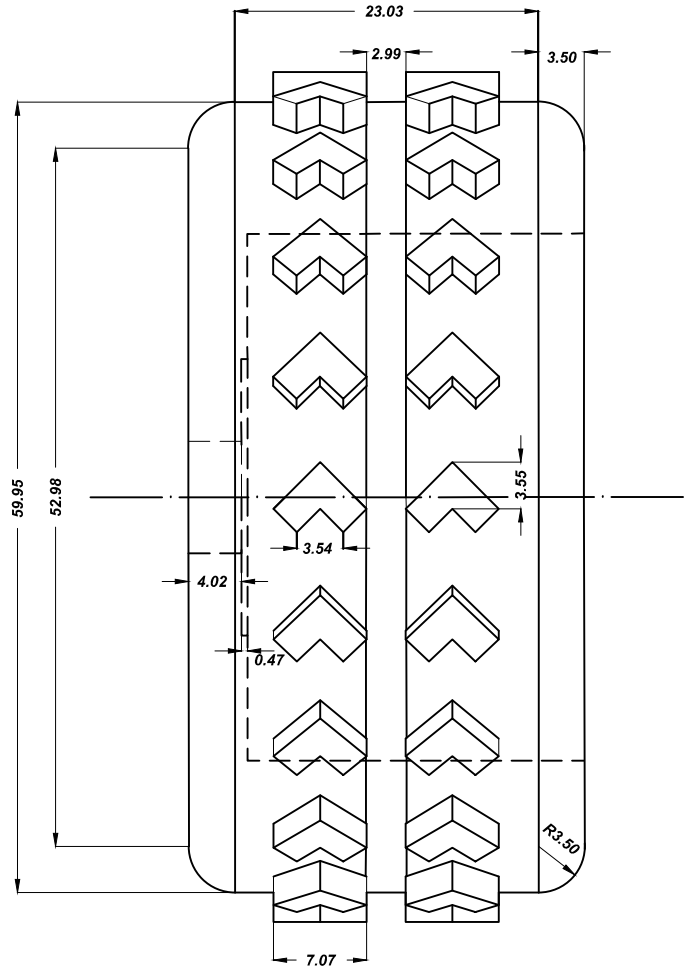




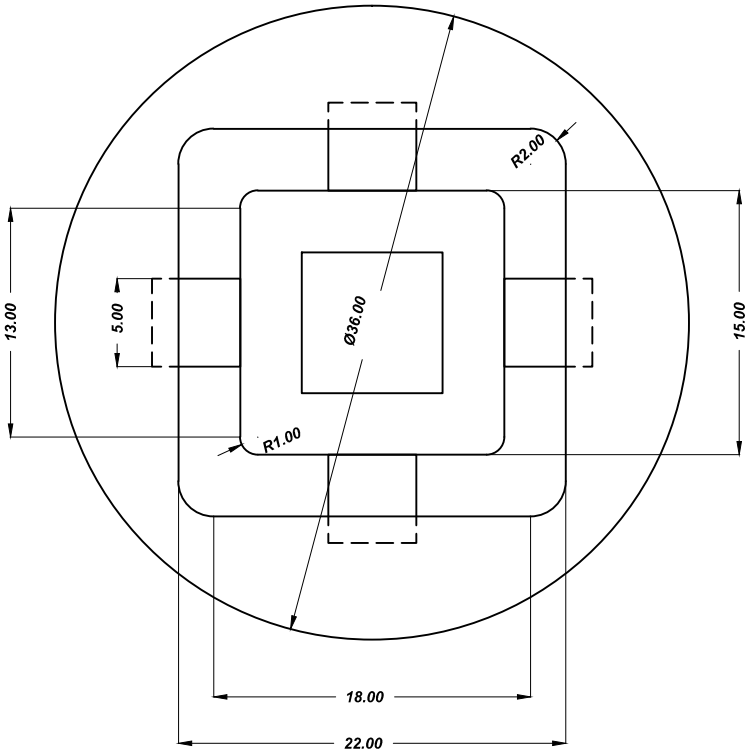
	Fecha	Nombre		OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	
Comprobado	Abril '12	A. Valero		
S.normas		UNE		

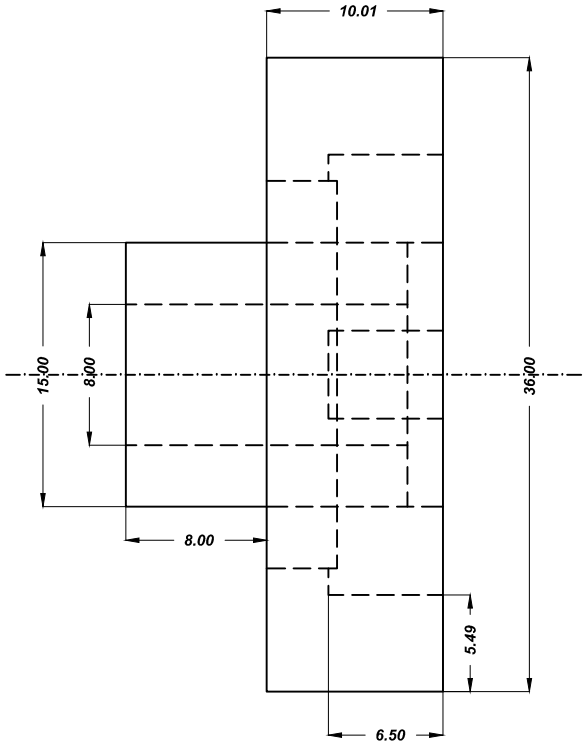
Escala		
6/1	PIEZA TIPO 9: RUEDA	PLANO Nº9



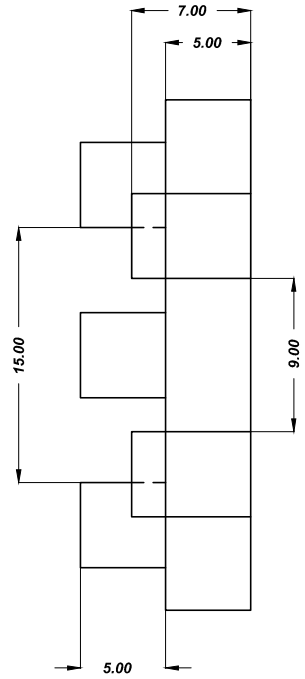


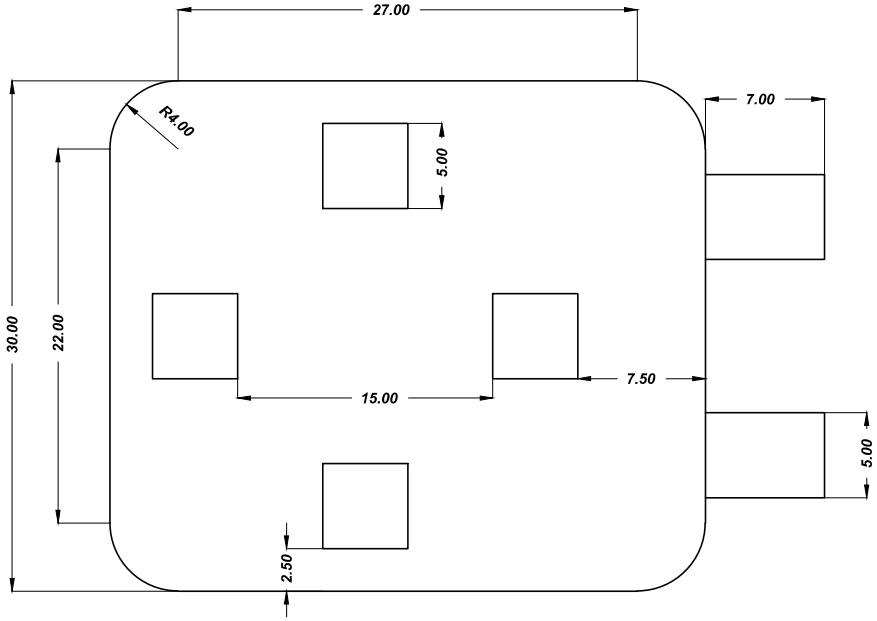
	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 6/1	PIEZA TIPO 10: RUEDA			PLANO Nº10



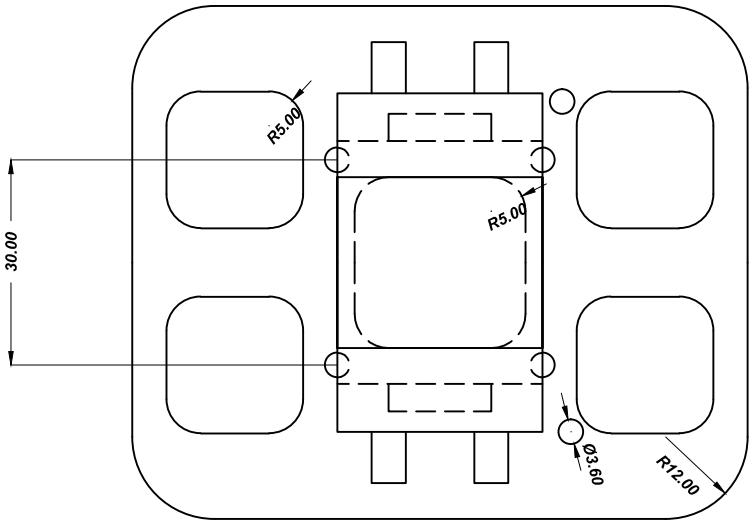


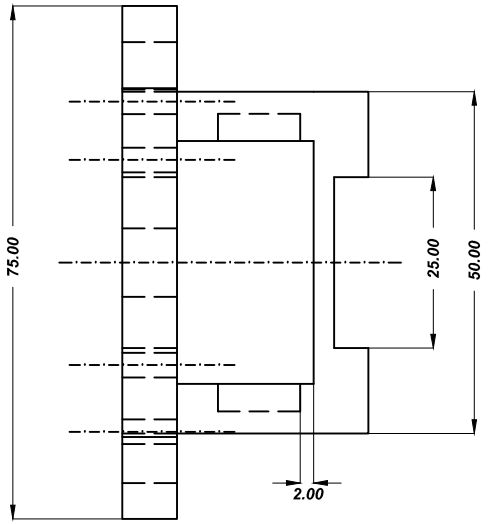
	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 8/1	PIEZA TIPO 11: CONECTOR			PLANO Nº11

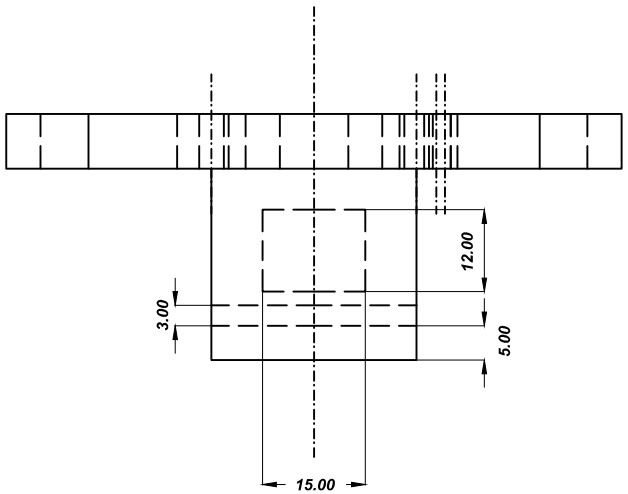




	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 8/1	PIEZA TIPO 12: CONECTOR			PLANO Nº12

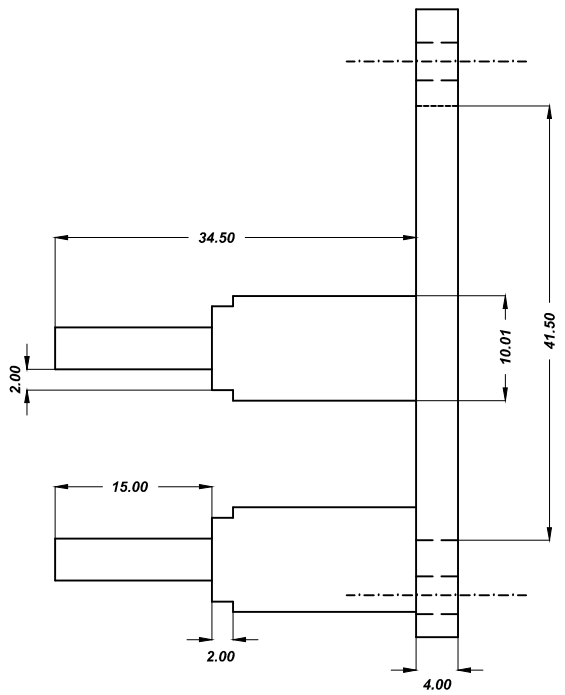


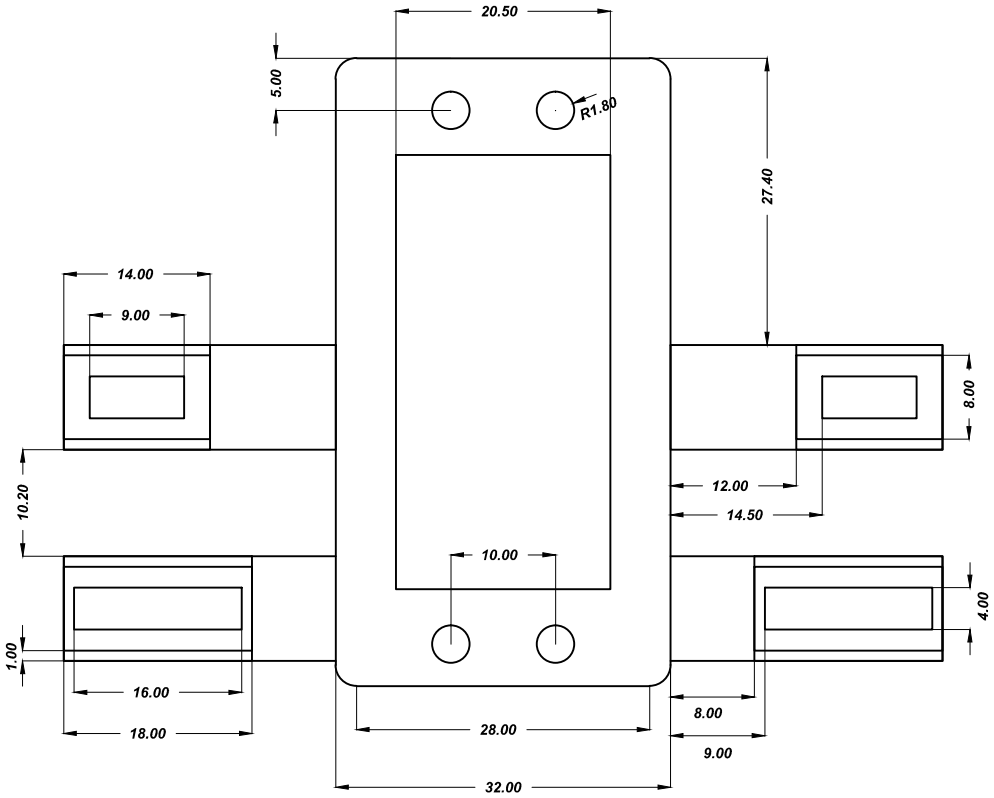




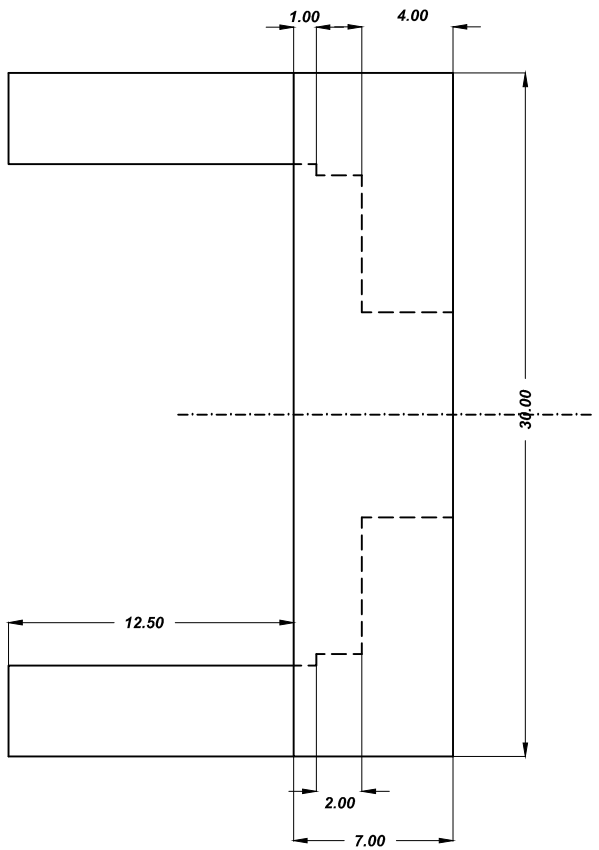
	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		

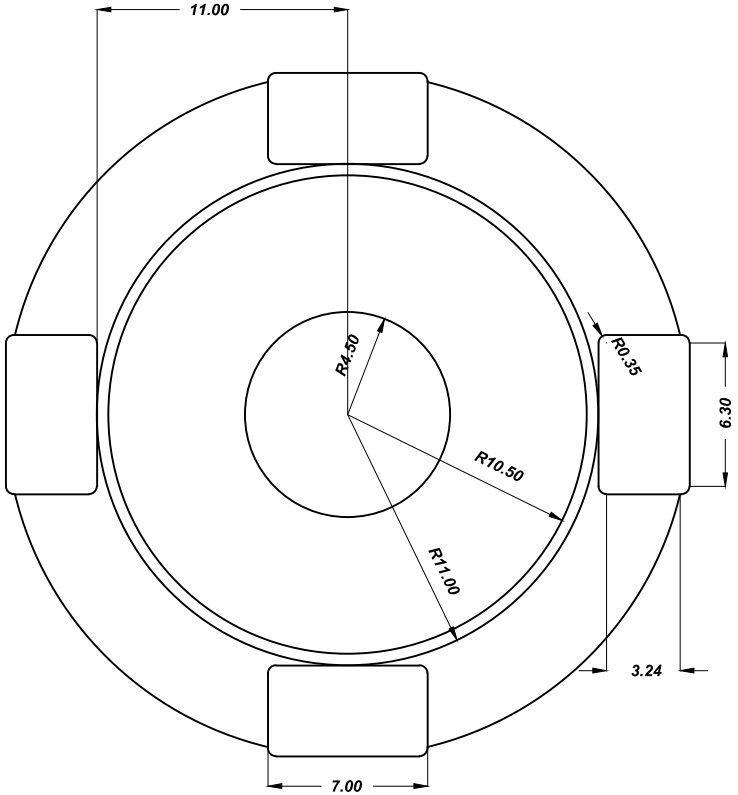
Escala		
3/1	PIEZA TIPO 13: BASE	PLANO Nº13



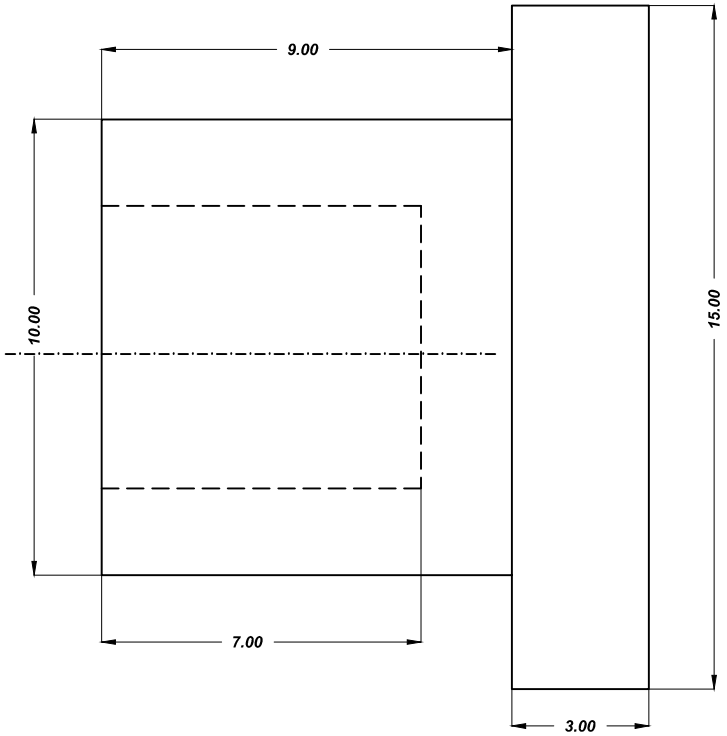


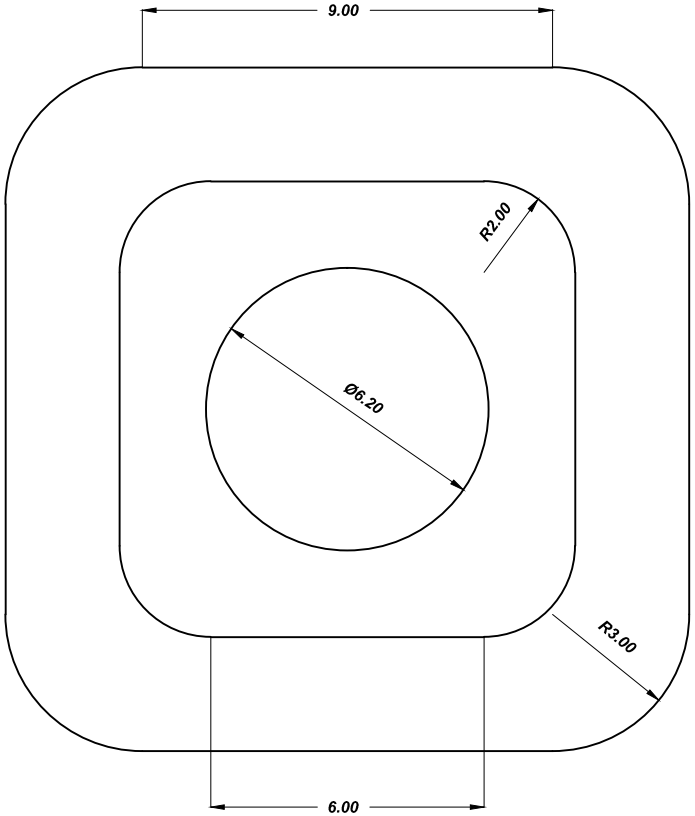
	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 5/1	PIEZA TIPO 14: SOPORTE			PLANO Nº14





	Fecha	Nombre		OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 10/1	PIEZA TIPO 15: CONECTOR			PLANO Nº15





	Fecha	Nombre		
Dibujado	Abril '12	M. Almagro	Mario Almagro Cádiz Ingeniero Electrónico	OOML: UNA BIBLIOTECA C++ PARA EL DESARROLLO MECÁNICO APLICADO A LA ROBÓTICA
Comprobado	Abril '12	A. Valero		
S.normas		UNE		
Escala 20/1	PIEZA TIPO 16: SOPORTE			PLANO Nº16

12.5 ANEXO 5

El archivo *OOML* que contiene el diseño *Mars* está situado en la subcarpeta "*Fichero OOML*", dentro de la carpeta "*Mars*".

El código está comentado, por lo que se puede seguir la construcción de cada diseño paso a paso.

12.6 ANEXO 6

En la subcarpeta “Aplicación de Control”, dentro de la carpeta “Mars”, están contenidos todos los archivos correspondientes a la aplicación de comunicación, que incluye el código en *C++*, un ejecutable (bajo *Linux*), resultado de la compilación de dicho código, y las bibliotecas de la comunicación puerto serie y el *gamepad*.

12.7 ANEXO 7

El código que distribuye las señales del microcontrolador *Arduino UNO* se organiza en el archivo "*Control.pde*", localizado en la subcarpeta "*Aplicación para Arduino UNO*", dentro de la carpeta "*Mars*".

12.8 ANEXO 8

A continuación se organizan todos los materiales utilizados con sus costes asociados para determinar el presupuesto necesario para este proyecto.

Componente	Unidades	Coste
Servo Futaba S3003	6	48.00 €
Batería Turnigy 2S Lipo Pack	1	8.00 €
Módulo Bluetooth de Arduino	1	1.58 €
Adaptador Bluetooth	1	7.54 €
Mini-protoboard	1	3.25 €
Arduino Uno	1	21.00 €
Junta Tórica	6	6.48 €
Pintura Spray	3	6.60 €
Muelle	4	2.10 €
Tornillo	14	1.10 €
Tuerca	14	0.40 €
Cable Hilo	60 cm	2.00 €
Plástico ABS	270 g	10.32 €
TOTAL		**118.37 €**

Tabla 5. Lista de materiales y costes.

La tabla anterior muestra el conjunto de materiales utilizados para la elaboración del prototipo *Mars*, tal y como se muestra en la *Figura 44*. Así, se concluye que el precio mínimo para su implementación es de 118.37 €.