Noname manuscript No.
(will be inserted by the editor)

A New Paradigm for Open Robotics Research and Education with the C++ OOML

Alberto Valero-Gómez · Juan González-Gómez · Rafael Treviño

Received: date / Accepted: date

Abstract For many years robotics has been benefited from the open source community. Software community projects like Player, Stage, Gazebo, ROS, or OpenCV are present in most robotic applications. In recent years this trend has also been initiated among electronic and mechanical developments (open hardware). The Arduino development platform is a good example of a successful hardware project with a great community of developers and users around it. The apparition of personal 3D printers is bringing the open source philosophy to the fabrication of physical things as well. This new technology is in need of new designing tools to take advantage of it. In this paper we are presenting the C++ Object Oriented Mechanics Library (OOML), a tool to design mechanical components, taking into account the needs and requirements of these emerging technologies. These designs can be easily shared, reused, and modified. The OOML brings together the advantages of 1) modelling things through code, 2) the object oriented programming paradigm, and 3) the power of C++. In the OOML, mechanical parts are described as geometrical combinations of basic primitives. Once a part is defined, fabrication files can be generated in order to print, or mechanize it. Models could also be used for simulation, visualization, structural analysis, etc.

1 Introduction

Richard Stallman said in 2008: "maybe in 10 years most users, common or not, will use free software and will have the freedom and control of their own computing" [18]. In this article we are presenting a new coding tool for roboticists that will introduce the mechanical design of robotic parts into the open source community. Perhaps personal manufacturing will allow that in 2018 researchers can share their robots' mechanical designs just as easily as software is shared today.

The advances in personal manufacturing introduced by personal 3D printers let us foresee this future in engineering research. But this new technology seems to request a change of paradigm in the design process of physical objects, so that they can be easily shared over

Alberto Valero-Gómez
BitBrain Technologies
E-mail: alberto.valero.gomez@gmail.com

the internet, re-used and evolved. This is the goal of the *C++ Object Oriented Mechanics Library* (OOML): allow researchers or consumers to reuse, re-share, and modify physical designs modelled with C++. Parts can then be mechanized using the classical tools for fabrication, or with the new personal (and low cost) open source 3D printers. The topic of 3D printers will be discussed later in this paper. Nevertheless it is important to highlight in this introduction that what personal 3D printers are bringing to the mechanical design of things is deemed to be analogous to what personal computers brought to the open source community [2][11]: people all around the world sharing code and being able to contribute in a joint effort to build open source projects. With 3D printers, mechanical designers can clone rapidly and cheaply others' designs, use them, or adapt them to their requirements [17]. The raising of personal manufacturing is the reason to develop this new tool called OOML. The contribution of this paper is more than a library implementation, it is a proposal for a change of paradigm in the design of physical things. We have tested this proposal with the OOML, but other implementations are possible. In the following sections we will justify this affirmation.

This paper is organized as follows. First of all we will justify the impact of 3D printers on engineering research and the convenience of a new paradigm for mechanical design. Taking the model of the Object Oriented Programming (OOP), as a powerful tool for code sharing and reusability, we will justify why this paradigm fits in the design of physical things. Afterwards we will present our contribution: the *C++ Object Oriented Programming Library*, an open C++ library that applies the OOP paradigm to the design of mechanical parts. A section presenting the developments achieved up to the date in OOML will justify the expected impact of this tool in the research and pedagogical community, and the advantages that it may bring to the roboticists community. A conclusions section will close this article.

2 On Personal 3D Printers and Mechanics

The origin of open source printers was the RepRap Project[10], started by Adrian Bowyer in 2004. The aim of this project was to develop an open source self-replicating machine. In May 2007 the first prototype, called Darwin, was finished and some days later, on May 29th, the first replication was achieved. This first personal 3D printer was based on the computer numerical control (CNC) Cartesian machines design. Since then, the RepRap community (original RepRap machines and derived designs) has been growing exponentially[6]. The second RepRap generation, called Mendel, was finished in September 2009. Josef Prusa simplified the Mendel design, originating the most widely spread printer to date, the Prusa Mendel. The Air (a stylized modification of the Prusa Mendel) was developed in 2011. Also in 2011 the RepRap Wallace and the PrintrBot changed significantly the existing designs, but they were still based on CNC Cartesian machines. In 2012 a completely new design saw light, called Rostock Delta, based on a parallel robot. Currently there are ongoing projects for making new printers based on robotic arms. The current estimated population is around 20000 machines (considering only open source printers). Fig. 1 shows several models of RepRap printers.

Initially, Darwin and Mendel were not designed for the general public but for people with some technical background. Since the reprap project is open source, small companies were created to start selling these 3D printers, as well as derived designs. The first company was Makerbot Industries, who shipped a first batch of their Cupcake CNC in April 2009. By the end of 2009 they had shipped nearly 500 complete kits. After operating for a year they had sold about 1000 kits. Afterwards came the Thing-O-Matic printer, announced in

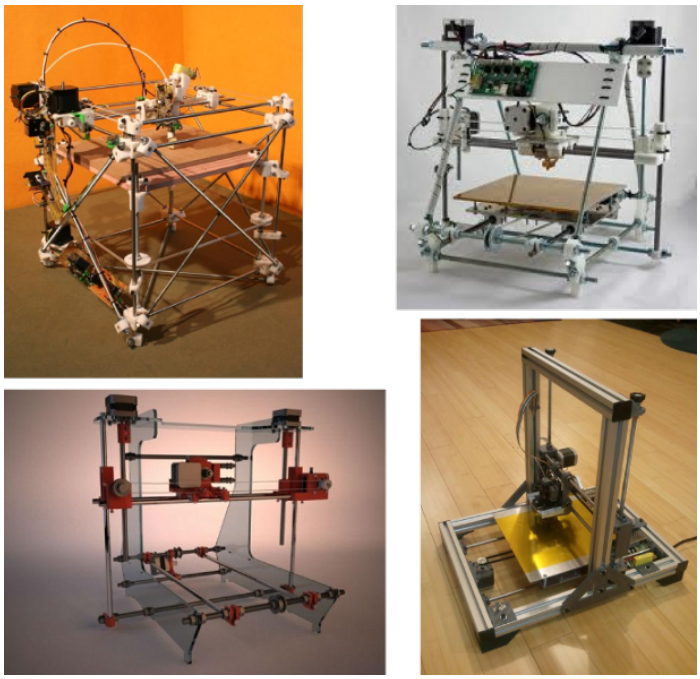


Fig. 1: Pictures of some RepRap 3D Printers developed by the community. From left to the right: Darwin, Prusa Mendel, Prusa Air, and Aluminium. Pictures taken from http://reprap.org

September 2010, easy to build and use, for $1300. Their current model is called The Replicator, it comes fully assembled and it is the first personal open printer with dual extruder, allowing the use of support material or printing in two colors. Its price is around $1900. Other open printers have arisen inspired by the RepRap models and following the commercial impulse of the MakerBot. These are the PrintrBot (that costs just $450), the UltiMaker (around $1000) or the MakerGear (around $1300). Figure 2 shows four of the most important commercial open source 3D-printers.

The flourishing of 1) open source 2) affordable 3) personal 3D printers has had an impressive impact on *do-it-yourself* manufacturing [3][4]. With 3D printers individuals can print complex engineering parts from design files at low cost and short times. Bruijn shows that a considerable improvement on physical things is proposed by people sharing parts over the internet and having access to 3D printers [6]. With a 3D printer, these modifications are relatively easy for others to replicate. As it has been the case with software for many years, currently, there are also on-line repositories of parts, where people can download and upload their designs; the most famous is Thingiverse.com. Although initially 3D printers were thought only as prototyping machines, that is not their only use. Fig. 3 shows a extruder system composed of printed and non printed parts. Printing these parts takes around 3 hours with a Prusa Mendel printer, and the associated cost in ABS plastic is around 1 Euro. The shown mechanism is not just a prototype, but a real final product, ready to be used. [2] discusses whether 3D printers can be used for production or not. TeenAgeEngineering.com is

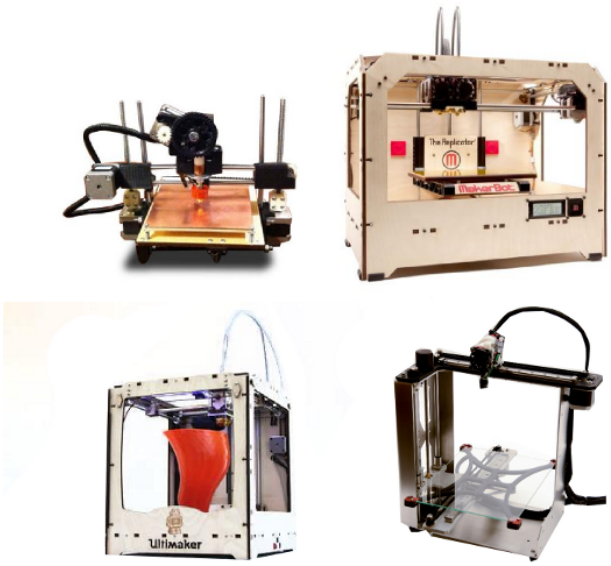


Fig. 2: Pictures of some open source commercial 3D printers. From left to the right: PrintrBot, MakerBot, UltiMaker, and MakerGear. Pictures taken from the corresponding company website

the first company offering 3D models of the parts included in their products so that their clients may manufacture and substitute them if necessary. They also provide improvements of the machined parts to update their products (as it is done with software updates).

These aspects, together with the possibility of sharing the designs over the internet, have not left indifferent the engineering community nor the open source community, which sees a vast spectrum of new possibilities in this technology. While open source software development has been studied extensively, relatively little is known about the viability of the same development model for a physical object design.

Currently many research groups are investigating the possibility of introducing personal 3D printers as a powerful tool for research and collaboration, studying their potentials and limitations. In [9][16] 3D printers are used to manufacture and test evolutionary physical designs. In [8] an educational low-cost printable robot is presented. Figure 4 shows the flowchart of fabricating a PrintBot (Printable Robot). Modular robots can easily be printed and expanded [15]. In bioengineering they are used to fabricate reticulations for tissue cultivation [7][13][14]. Specific research focused on 3D printers technology is also being carried on in these last 30 years [5][12].

3 A New Paradigm for the Design of Physical Things

The impact of open source 3D printers on the research community has been shown in the previous section. The apparition of new technologies carries along the necessity of developing new tools that take advantage of their particularities, for example, the apparition of rewritable CDs and CD recorders required of programs capable of burning data into CDs.

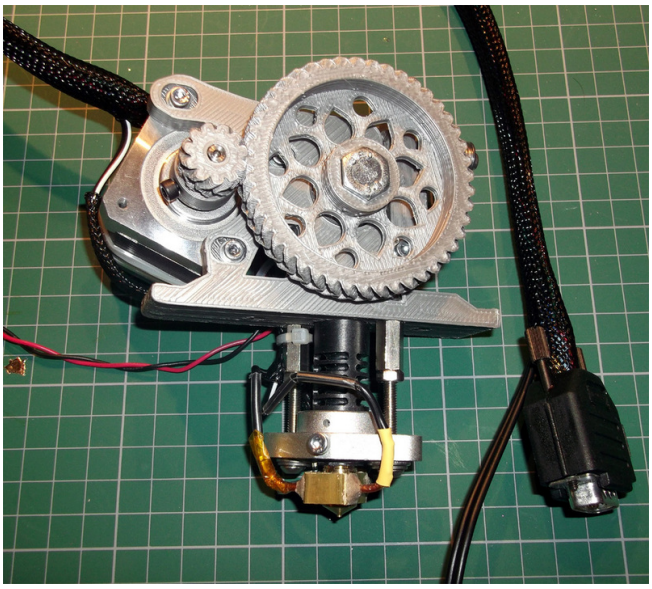


Fig. 3: Extruder designed by RichRap. Image courtesy of Thingiverse.com

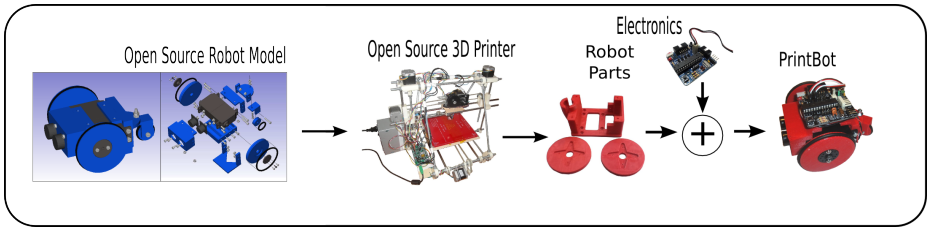


Fig. 4: PrintBot Design and Fabrication Flowchart

3D printers allow for rapid prototyping, manufacturing and sharing of 3D objects. Consequently, the tools for designing physical things should evolve in order to take advantage of the full potential of this new technology. These tools should allow to:

- Rapid design of physical objects.
- Easy and efficient modification and reutilization of existing objects.
- Possibility of sharing and co-developing physical models in an efficient way and oriented to a distributed community.

3.1 Rapid Design of Physical Objects

Designing mechanical parts has been traditionally made using graphical CAD programs, following the WYSIWYG paradigm (What You See Is What You Get). Diverging from the main stream of 3D design OpenSCAD was born. Unlike most 3D CAD applications, OpenSCAD does not follow the WYSIWYG but a variation called WYGIWYM (What You Get Is

What You Mean). In a WYGIWYM editor, the user writes the contents in a structured way, marking the content according to its meaning, its significance in the document, instead of designing its appearance. The main advantage of this paradigm is the total separation of presentation and content: designers can concentrate their efforts on structuring and writing the document, rather than concerning themselves with the appearance of the document, which is left to the export/compile system. There are well known examples that follow this paradigm, such as Latex (used for writing documents) and Verilog or VHDL (used to model electronic systems). XML is another example in which the visualization of data in independent of the data itself.

For mechanical design the WYGIWYM paradigm relies on the separation of the logical definition of the thing from its renderization. One of the major research directions in the field must be the development and study of abstract, human-oriented models to specify the structure of objects and to manipulate them. These models must permit the user to concentrate on a logical representation of data that resembles his vision of the reality modelled by the data much more closely than the physical representation of the object.

OpenSCAD offers a basic scripting language consisting of a set of geometric primitives: cube, cylinder, sphere, etc. and operations to manipulate and/or combine them: union, difference, rotation, translation, etc. Our classroom experience in the last two years has shown us that designing things by describing their geometry reduces significantly the learning time of the tools. In this scenario, learning a program like AutoCad, Catia, or SolidEdge is a disadvantage from the productivity point of view, as the required learning time might not provide proportional results in terms of quality design. Instead, designing a relatively complex thing with OpenSCAD or OOML has proven to be about one day of work. Figs. 5 and 6 show some things designed and printed by our students after few hours of lesson.

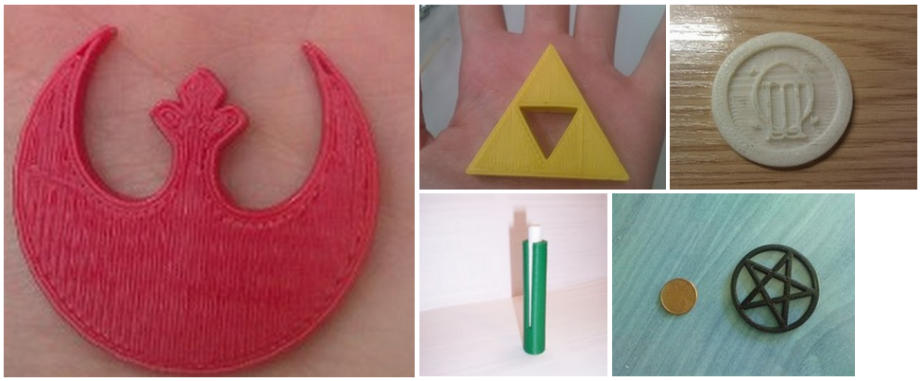


Fig. 5: Designs made in OpenSCAD after one hour lesson. All source code hosted in Thingiverse.com

3.2 Easy and Efficient Modification and Reutilization of Existing Designs

Designing 3D objects through code opens a new possibility that until now was only present in the software community. As it has been the case with software for many years, tools like

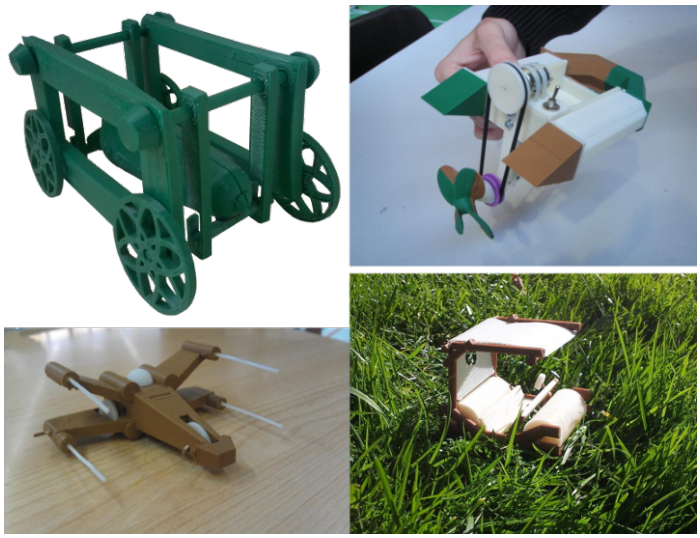


Fig. 6: Designs made in OpenSCAD and OOML after three hours of lessons. All source code hosted in Thingiverse.com

OpenSCAD or OOML allow to code 3D objects, share them on the internet, modify existing designs, improve them, and reuse already designed parts.

Before focusing on the motivation for moving forward from OpenScad to the C++ OOML it would be necessary to highlight an advantage of modelling 3D things through code in terms of reutilization and under the light of the open source philosophy.

A key aspect in reutilization is the possibility to parametrize the models. In classical mechanics parts are defined using production drawings that include the dimensions of the parts. Using the drawings the parts can be machined. If a part needs to be adjusted (more thickness, bigger axis, etc.) all dimensions need to be updated and the production drawings remade. In the age of computing this work-flow is somehow obsolete. Tools like OpenSCAD or OOML can set the dimensions of the models dependent of a set of parameters. Changing these parameters the model dimensions or shape will change accordingly. A well designed model is capable to produce a large variety of valid particular parts. An example of a well defined part can be a Servo Wheel. This part can take three parameters: radius, thickness and servo type. Once the user has set these three parameters the Servo Wheel will 1) check if the radius and thickness are coherent with the servo type; 2) if dimensions are right a wheel of radius and thickness and without axis will be created; 3) finally the model of the chosen servo will be subtracted from this wheel. This will result in the desired servo wheel. This work-flow let us anticipate the utility of the object oriented paradigm to implement this functionality.

Another key is that the WYGIWYM paradigm separates the model, created through code, from the visualization of the modelled thing. This has a broader horizon of advantages. Once the physical thing is modelled, all its geometric properties are stored and have a geometric meaning: dimensions. The model could also include other properties, like materials, tolerances, etc. Materials could be also coded, including their density, flexibility, fuse point, etc., reaching a limitless level of detail or specialization. This means that we may perfectly define the physical thing, without taking into consideration what it is modelled for. We

may never want to visualize it, or fabricate it. The reutilization of this model has no limits. It can be used in simulations, it can be used to make structural or resistance analysis, it can be used for collision detection, renderization... or for generating a STL file to be printed in an open source 3D printer.

When working with graphical CAD programs this utilization of the models can be done through the ”export” option, exporting the designed model into a format legible by other programs. Otherwise, the CAD program itself could include some of these functionalities, such as structural analysis. However, working in this way we depend directly on the CAD program, that may (or may not) give the user the export capabilities or the desired functionalities. If the CAD program does not give this functionality and the encoding of the physical thing is proprietary, the designed model cannot be used for other purposes.

Coding the physical thing immediately gives the possibility to program other tools that work with this model for many different goals, and the model will remain the same, or else enlarged with new properties, attributes, or methods. This is a major advantage of coded models of physical things over proprietary encoded CAD designs.

Another potentiality is extending the properties of modelled things, for example adding part flexibility or material. An open source coding tool allows to easily add this property by extending the tool itself. Here we can already anticipate the power of using the object oriented paradigm, specially thought for extending existing classes through composition and inheritance. Then, if instead of a program, like OpenSCAD, a library is provided to model the things, like the OOML, the extension is even easier from the programming point of view.

Finally, coded physical things can be evolved in the same way that software is evolved, providing a multiplicity of designs derived from an initial one or combining different designs (just reusing their code). Fig. 7 shows this evolution for the extruder of the RepRap Mendel 3D printers family. The extruder is the most important part of a printer, and the printing quality, life time, and ease of use depend on it. The result of having the extruder coded in OpenSCAD has driven to many variations and improvements. Such a diversification does not exist in Thingiverse.com for other parts that were designed with graphical CAD programs, even when these programs and the designed things were open source.

3.3 Sharing Physical Models

The third requirement we listed is offering the possibility of distributed collaboration on a common mechanical project. This aspect needs small justification. Sharing binary files over the internet does not allow to collaborate or maintain a versioning system in an efficient way. There are many online repositories of binary files, like DropBox or Google Drive, that keep a historical record of the files. Nevertheless they do not offer the possibilities of repositories of code like subversion, git, or mercurial. Coding objects allows to use the versioning and sharing tools that have been applied for years to software projects. No classical CAD program allows this kind of collaboration.

Why create a new tool, when OpenSCAD seems to successively accomplish these three requirements? In the next section we will motivate our decision to move from OpenSCAD to a new coding tool for designing 3D objects: the OOML. We believe, as it will be justified, that this is the natural evolution of OpenSCAD. The evolution consists of introducing the object oriented programming paradigm in the design of physical objects.

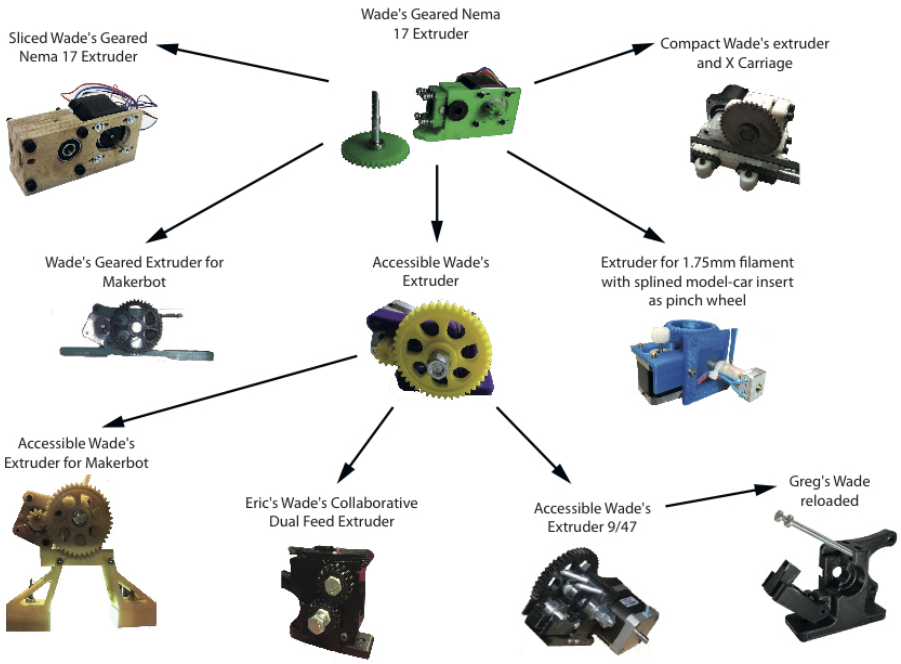


Fig. 7: Evolution of extruders developed in OpenSCAD. Single images courtesy of Thingiverse.com

2D Primitives	3D Primitives	Unary Operators	Composite Operators
Circle	Cylinder	Translate	Union
Square	Cube	Rotate	Difference
Polygon	Sphere	Scale	Intersection
Polyhedron		Mirror	Hull

Table 1: Primitive Components and Operations of OpenSCAD and OOML

4 Modelling Physical Things Through Code

As we stated previously, one of the major research directions of WYGIWYM editors is the development and study of abstract, human-oriented models for specifying the structure of objects and for manipulating them. These models permit the user to concentrate on a logical representation of data that resembles his or her vision of the reality modelled. Consequently, both OpenSCAD and OOML must define which are the primitives for modelling 3D things and the geometric operations that allow to combine these primitives to build more complex things.

Table 1 shows the primitive components in OpenSCAD and OOML and the existing operations to manipulate them (Fig. 8 and 9). These operations can be applied over a single component (unary operator), or over a set of components (composite operator), composing a more complex component from the simpler ones.

OOML and OpenSCAD build complex things as combinations or unary operations performed on other components. The structure is stored as a tree. Each node denotes a primitive,

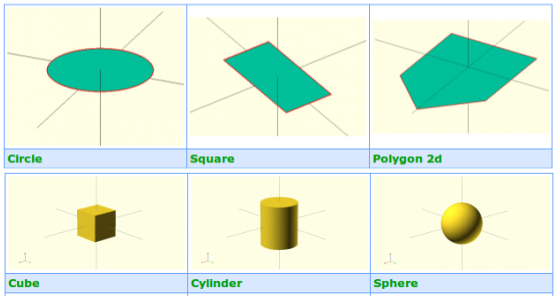


Fig. 8: Primitive Components in OOML and OpenSCAD

an unary operation, or a composition of primitives. Primitives can be located only at the end of branches. Unary operations can have only one child, and composite operations can have two or more children.

Fig. 10 shows how a tree is build step by step for the thing shown in Fig. 11. The corresponding OOML code is:

```
Component base = Cube(20,20,3); //Stage 1
base.translate(0,0,-10); //Stage 2
Component axis = Cube(5,5,20); //Stage 3
Component drill = Cylinder(1,20);
Component body = axis - drill; //Stage 4
Component support = base + body; //Stage 5
```

or more compact

```
Component support = Cube(20,20,3).translate(0,0,-10)
          + ( Cube(5,5,20) - Cylinder(1,20) );
```

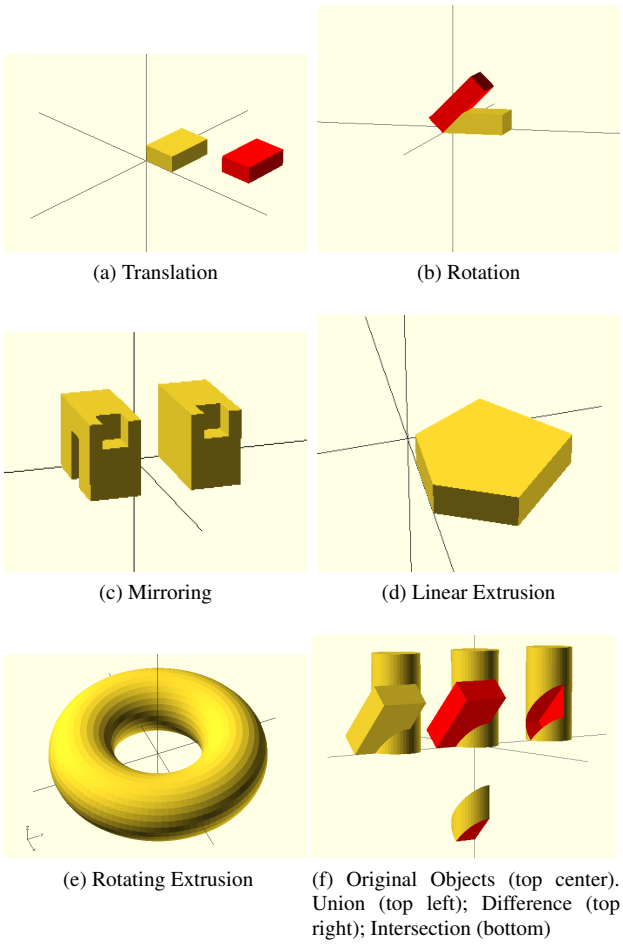
The same thing modelled in the OpenSCAD script is as follows

```
union(){
  translate([0,0,-10])
    cube([20,20,3], center=true);

  difference(){
    cube([5,5,20],center=true);
    cylinder(r=1,h=21, center=true);
  }
}
```

4.1 Introducing the Object Oriented Paradigm

Many years of software programming have driven to the object oriented paradigm (OOP) as an efficient methodology for implementing software. The benefits of the OOP lay in



(a) Translation

(b) Rotation

(c) Mirroring

(d) Linear Extrusion

(e) Rotating Extrusion

(f) Original Objects (top center). Union (top left); Difference (top right); Intersection (bottom)

Fig. 9: Operations

its modularity and reusability. OOP enhances community development and evolution of software through reutilization of code. To do so the OOP relies on following characteristics: 1) Encapsulation and modularity, 2) inheritance and composition, 3) abstraction, and 4) polymorphism. If the OOP could be applied to the design of physical things, it would be an advance in terms of collaboration and reutilization of the produced models.

Let's analyse each one of the OOP characteristics with respect to the design of physical objects.

Encapsulation and modularity. Physical objects have many quantitative properties, some of which can be considered independent, while others are dependent. For example, in a gear, the radius and the number of teeth, determine the teeth size and position. That means that a gear wheel can be defined by fixing only these two parameters and the rest will be calculated accordingly. Consequently, we are encapsulating some information that the designer does not need to set to define the gear. Using the proper interface the *consumer* designer can

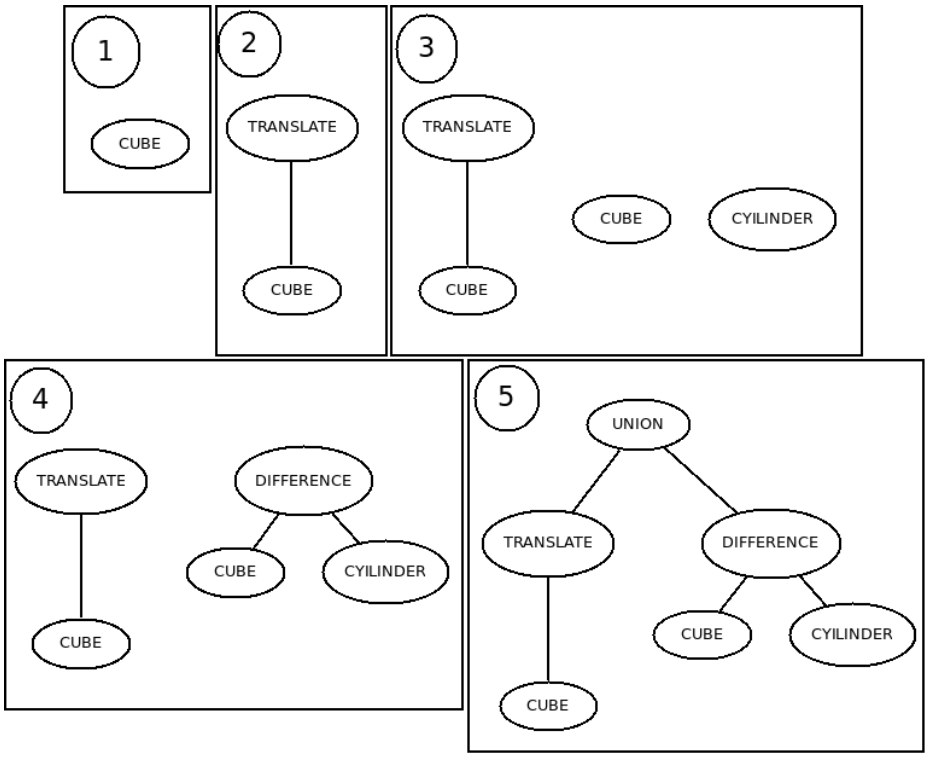


Fig. 10: Example of a Abstract Syntax Tree

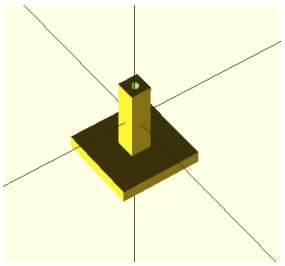


Fig. 11: Example of an OOML thing

define the physical object. It is the task of the *producer* designer of the thing to decide which interface the class must provide to the consumer of the object.

Inheritance and composition. The inheritance of classes is based on the IS-A relation. In this sense it is easy to see this relation in physical objects: a gear wheel IS-A wheel, and therefore two classes could be defined (*wheel* and *gear wheel*), one inheriting from the other. As for the composition, it is even clearer: a robot, for example, is composed of parts.

Abstraction. All wheels are conceptually the same, but each wheel can have different parameters. This illustrates that abstraction is an adequate methodology to define classes in physical objects.

Polymorphism. Polymorphism relates more to the implementation of the tool. Each primitive has all the information related to itself: how it is represented, how to generate the STL, which are their geometric properties, etc. Making use of polymorphism is a proper methodology to implement an object oriented library that models physical objects.

The success of the OOP together with its suitability to model physical objects motivates the creation of this new tool called the Object Oriented Mechanics Library. The chosen language has been C++, but the real contribution of this work is the idea of designing things through code following the object oriented programming paradigm. The particular implementation could vary while maintaining the same principles.

4.2 From OpenSCAD to the C++ OOML

Considering what has been said insofar, providing an open source tool that allows to design physical objects using the OOP paradigm is expected to boost the design of physical things, enhancing community collaboration and reusability.

According to the authors this is the major weakness of OpenSCAD: it is not object oriented. For this reason, we decided to explore the possibility of developing a library in C++ that, using object oriented programming, could be used to model solid objects. The expected contribution of this library is a general purpose modelling library, that allows collaboration and reutilisation of mechanical designs within a community of developers. There are other open source projects following the same idea. These are the pyOOML, developed Juan Gonzalez-Gomez co-author of the OOML, Cadmium, developed by Jayesh Salvi, or Pycado, by Julien Blanchard. All these tools implement a similar idea in python.

Another key to understanding the transition to the OOML is the concept of semantics. Geometric primitives are different from mechanical parts. Primitives are abstract concepts with some geometric properties. Mechanical parts are physical objects with a meaning in the real world. In OpenSCAD, parts are defined by combinations and operations of primitives, but they are not *declared* as a self standing object. For example, this piece of code could model two things in OpenSCAD.

```
//a top
union(){
  cylinder(r=30,h=10,center=true);
  cylinder(r=3,h=20,center=true);
}

//a wheel
union(){
  cylinder(r=30,h=10,center=true);
  cylinder(r=3,h=20,center=true);
}
```

A cylindrical top with a small cylindrical hanging and a wheel with a small joint axis are defined. Both have the same definition in OpenSCAD as they are geometrically equivalent. In OOML the same objects would be:

```
Component wheel = Cylinder(30,10) + Cylinder(3,20);
Component box_top = Cylinder(30,10) + Cylinder(3,20);
```

These pieces of code do not seem to offer any advantage one with respect to the other. Nevertheless, if someone will reuse our code, he will find easier to understand and work with the OOML variables *wheel* or *box_top* than with the definitions made in OpenSCAD. If we had to rotate the wheel in OpenSCAD we would have

```
rotate([45,0,0])
  //a wheel
  union(){
    cylinder(r=30,h=10,center=true);
    cylinder(r=3,h=20,center=true);
  }
```

in OOML the code would be

```
wheel.rotate(45,0,0);
```

The OOML provides human-readable coding, easier to follow and understand. In the following section we will describe how the OOML is implemented.

4.3 OOML Implementation

There are four core points of the OOML implementation:

- Physical objects are *programmed objects.*
- The parameters of the physical objects are the *attributes of the object.*
- Objects can be modified through their methods (unary operations).
- Objects can be combined resulting in other objects (composite operations)

The first version of OOML was released in July 2011. The second version of the OOML was released in February 2012. Together with the library a web page with all the required tutorials to start using it was created at http://iearobotics.com/oomlwiki. In October 2012 the OOML code was hosted at GitHub to allow others to contribute or fork the library.

The OOML applies the model of Object Oriented Programming to Mechanical Designs. Parts, designed as objects follow all the Object Oriented Programming properties:

- Encapsulation: Parts hide to the designer the attributes and methods that are not required for their definition, restricting access to some of the object's components. Doing so the OOML facilitates the bundling of data with the methods (or other functions) operating on that data.
- Dynamic dispatch – when a method is invoked on an part, the part itself determines what code gets executed by looking up the method at run time in a table associated with the object.
- Object inheritance (or delegation), parts can inherit from others following the *IS-A* relation. For example, a gear wheel *IS-A* wheel, inheriting its configuration methods and attributes, for example: set_radius(), set_thickness(), set_axis_radius().

– Reusability: Parts can be made by adding or composing other parts.

While in OpenSCAD all objects have the same semantic meaning (they are just combination or geometric operations with primitives), in OOML objects are divided in two categories:

– Primitive Components: They are geometrical entities, without a mechanical meaning, like a Cylinder, a Sphere or a Torus (Figs. 8 and 12).
– Parts: They have a mechanical meaning in a complex *thing*, like a wheel or a chain.(Fig. 13).

Both primitives and parts share the same interface as they inherit from a common abstract object (Fig. 14). Extending the OOML with new primitives or parts obliges the developer to maintain the interface which standardizes their utilization. An example of the extension are the set of primitives and parts developed by the community through their existing life. Some of these primitives, which do not exist in OpenSCAD, have already been added to the OOML (Fig. 12). The community has also developed a variety of parts. Some of these parts are designed to be printed. Others are models of existing things, created to aid in the design of of other things, in their majority robots (Fig. 13). As it can be seen in Fig. 14 Primitives inherit from Component, while Parts inherit from AbstractPart, which inherits from Component. All the unary operations and composite operations are performed over a Component.

A detailed description of the implementation alongside the class diagrams and code documentation can be found on the OOML webpage http://iearobotics.com/oomlwiki. Doxygen documentation is located at http://avalero.github.com/OOML

5 Impact

Using code to design physical objects has had a great impact since OpenSCAD was born. On Thingiverse there are around 4000 designs made with OpenSCAD (data taken on November 12th 2012).

In Figures 15 and 16 mobile robots designed using either OpenSCAD or OOML are shown, they are the proof of the viability of using these tools and low cost 3D printers to design mobile platforms. Figure 17 shows how they can be replicated in order to build a fleet. Figure 18 shows other robotic platforms, developed by our research group during the last year.

Figure 19 is a proof of the impact on the community of open source designed things. The SkyBot robot, mechanized in aluminium and plastic with traditional methods, has been available online for more than 5 years (electronics and production drawings). There have been several courses on universities and high schools using this robot to explain the basics of motion control and reactive control. During these years this robot has not evolved mechanically. In 2011, the Mini-SkyBot, designed in OpenSCAD, was first printed and released on Thingiverse [8]. In less than a year a great number of robots derived from this initial design, all over the world [19]. The first full robot developed in OOML was the ProtoBot[1] (February 2012). Since then people around the world have replicated it and modified it, designing different kinds of wheels and supports for sensors.

Since the launching of the OOML there has been a continuous flow of visits to the website from all around the world. The statistics of visits can be seen in Fig. 20. This data has encouraged us to go on developing the tool.

[1] http://www.thingiverse.com/thing:18264

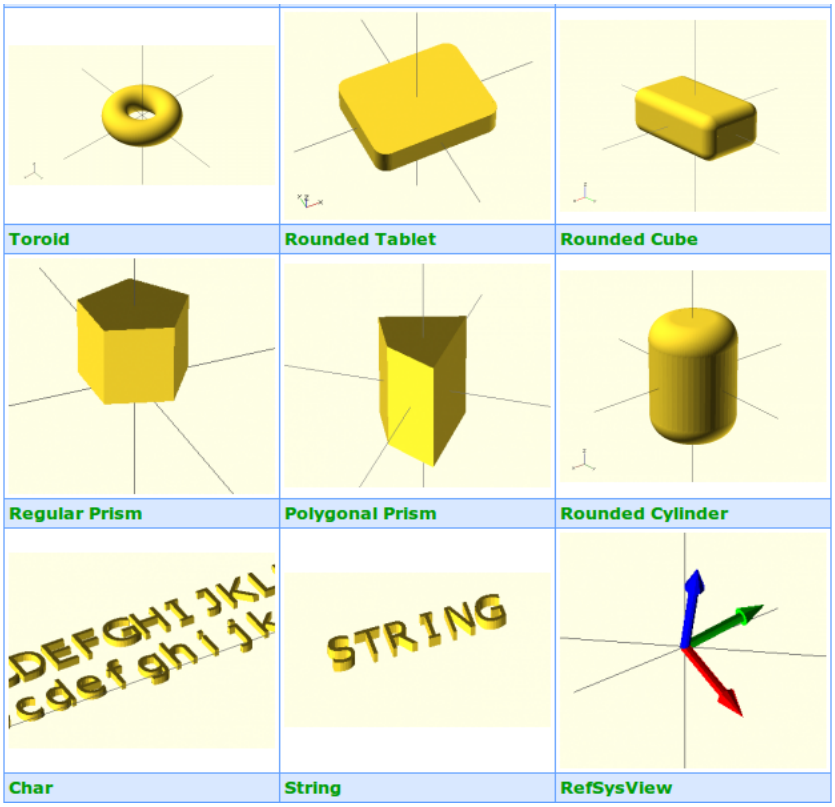


Fig. 12: OOML Extra Primitives

5.1 Impact on Education

OOML allows students to easily develop and share their own robotic platforms. Among the commercial educational platforms we can find a great variety of opportunities. These products are quite common in the educational environment, as they are affordable and easy to use. They usually come with associated software, which allows users to interface with the robot, having access to sensors and actuators, program them, and so forth.

The major disadvantage of these platforms is that they are closed. The users can hardly adapt them to their necessities. The reconfiguration of the platform may be a great advantage in order to be able to deploy the projects of the researchers, professors or students. The Lego MindStorm inherits the ”build-it-yourself” of the Lego traditional toys, but users are constrained to use the sensors provided by the manufacturer, as well as the development software.

Ad-hoc mini-robots have been built by research groups or university spin-offs mainly for educational purposes. These solutions overcome the limitations of commercial robots, providing cheaper and more adapted solutions.

This has also been our way of teaching robotics for many years, with our Skybot. In our courses, the students build the Skybot from scratch and then program it. Sometimes they are

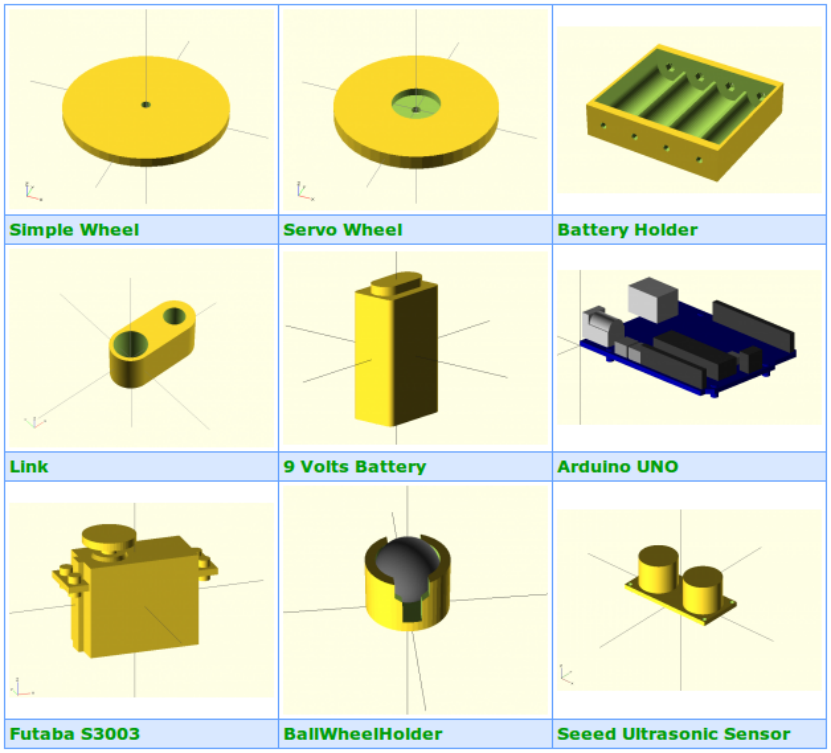


Fig. 13: OOML Parts

so motivated that they propose wonderful modifications to the robot design. Even though it is known beforehand that some modifications will not work well, we would like the students to discover it by themselves. In any case, it is not possible to implement these modifications during the course due to the time it takes for the manufacturer to build the parts. In the end we had to keep the platform without modifications, or in the best case, change it for the next course with new students.

To summarize, the classical way of teaching robotics must focus, by necessity, on the programming of the robotic agent given a particular platform. Even if this task alone can be quite challenging and inspiring, with our current proposal of open source printable robots, the teaching programme must not be focused *only* on the robotic agent, but it may also include its mechanical design. Beginning with a basic platform, like the Mini-Skybot, students can be guided through the design and programming process. In this way, they may discover the tight relation between hardware and software, and how each of them can and must adapt to the other's requirements in order to achieve a precise task. They may learn that a particular mechanical design suits better a precise task, test different alternatives, and so forth.

This methodology also includes something that is hardly considered in robotic programmes: students may learn that a change in the mechanical design could solve a problem better, faster, and more robustly than a software solution.

From an educational point of view the differences between both paradigms (WYSIWYG and WYGIWYM) do not only affect the way the designer has to work, but also the required

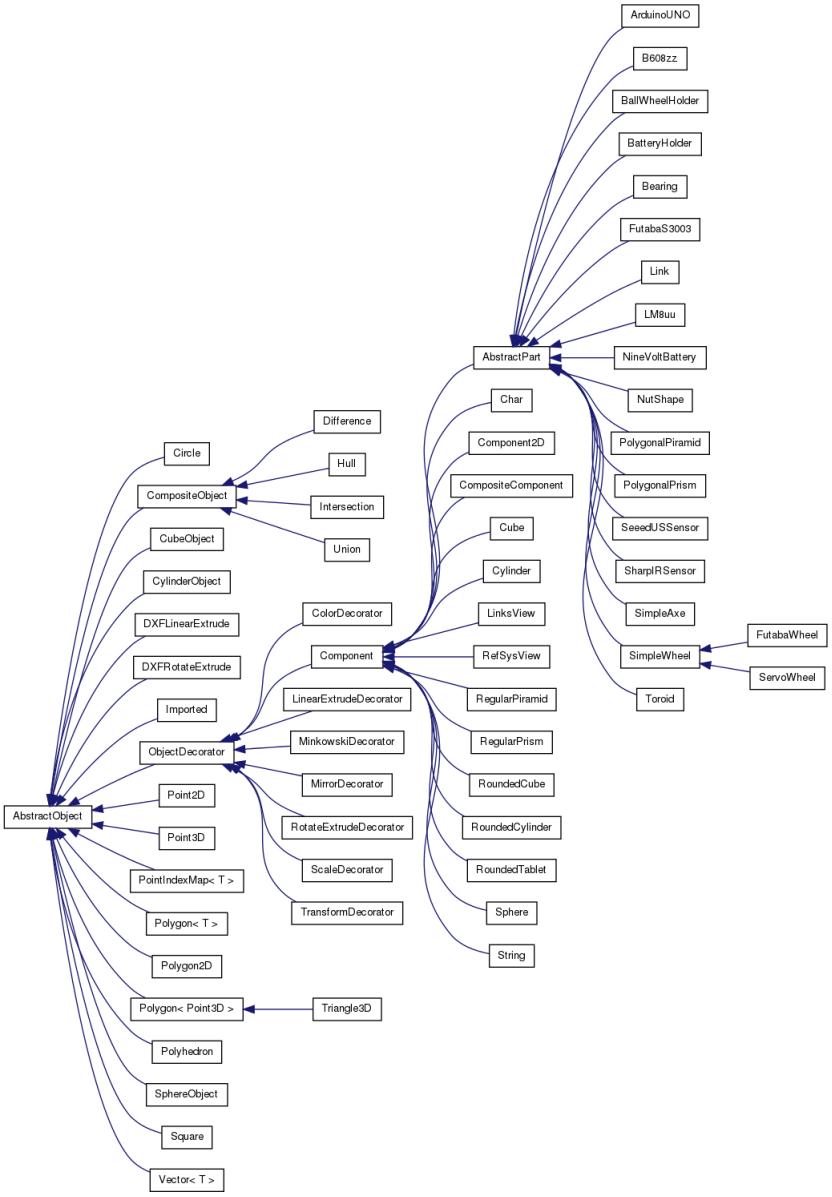


Fig. 14: OOML Class Diagram

skills, or, as it can be said from the education point of view, the skills a student will develop using it. Lizabeth Azrum has made interesting comparisons among some free CAD programs [1]. The following table summarizes her analysis. The comparison is made in terms of the skills that are developed using/learning them.

We can highlight two benefits that OOML may bring to the educational community:

Fig. 15: Indoor robots designed with code. Source Code is on http://www.thingiverse.com

– Designing through code forces the designer to understand the underlying geometry of the objects. The designer will understand the mathematical relations of geometry (intersection, union, difference, rotation, translation), and become mathematically confident by communicating and reasoning mathematically, by applying mathematics in real-world settings, and by solving problems through the integrated study of number systems, geometry, algebra, data analysis, and trigonometry.

Fig. 16: All terrain robots designed with code. Source Code is on http://www.thingiverse.com

Fig. 17: Fleets of robots printed with a 3D printer

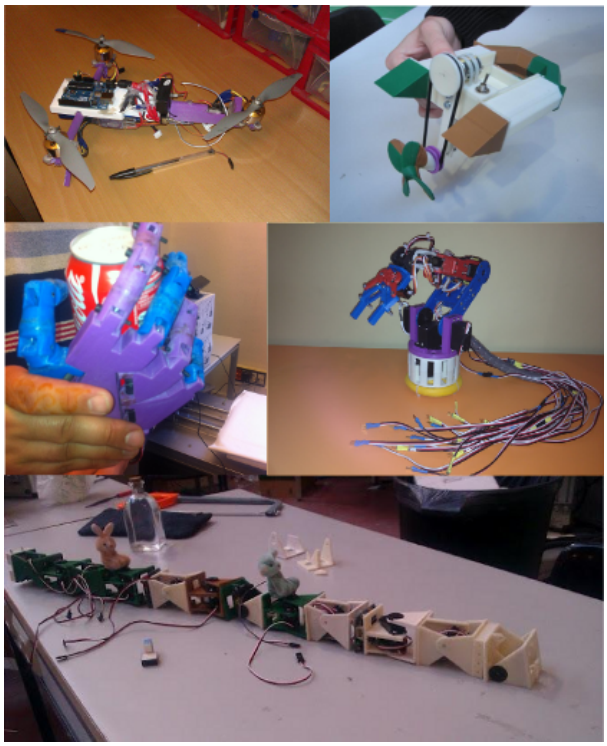


Fig. 18: Tricopter, boat, hand, manipulator, and modular snake robots, designed with code. Source Code is on http://www.thingiverse.com

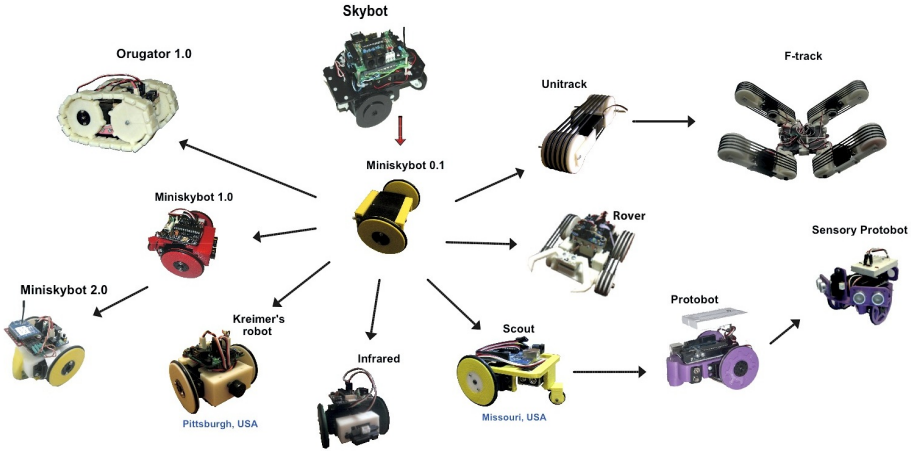


Fig. 19: Evolution and Diversification of the SkyBot Robot

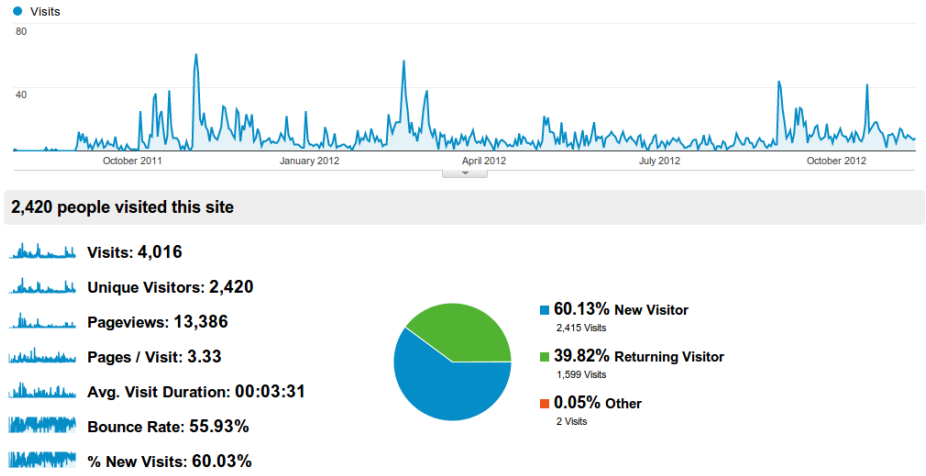


Fig. 20: Visitors of the OOML wiki page. Statistics generated by Google Analytics Online Tool

Paradigm	Std 1	Std 2	Std 3	Std 4	Std 5	Std 6	Std 7
WYSIWYG	✔	✔			✔	✔	✔
WYGIWYM	✔	✔	✔		✔	✔	✔

Table 2: Evaluation of CAD paradigms according to the NYS Learning Standards for Mathematics, Science, and Technology (http://www.p12.nysed.gov/nysatl/standards.html)

- Standard 1: Students will use mathematical analysis, scientific inquiry, and engineering design, as appropriate, to pose questions, seek answers, and develop solutions.
- Standard 2: Students will access, generate, process, and transfer information using appropriate technologies.
- Standard 3: Students will understand mathematics and become mathematically confident by communicating and reasoning mathematically, by applying mathematics in real-world settings, and by solving problems through the integrated study of number systems, geometry, algebra, data analysis, probability, and trigonometry.
- Standard 4: Students will understand and apply scientific concepts, principles, and theories pertaining to the physical setting and living environment and recognize the historical development of ideas in science.
- Standard 5: Students will apply technological knowledge and skills to design, construct, use, and evaluate products and systems to satisfy human and environmental needs.
- Standard 6: Students will understand the relationships and common themes that connect mathematics, science, and technology and apply the themes to these and other areas of learning.
- Standard 7: Students will apply the knowledge and thinking skills of mathematics, science, and technology to address real-life problems and make informed decisions.

- Designing through code allows to easily share the designs. The object code already encloses the geometry (which does not happen in a WYSIWYG design). Objects are easy to understand, modify, and re-share.

6 Conclusions

In this paper we have presented the OOML. The OOML is a C++ library that geometrically describes physical things using basic geometric primitives and geometrical operations. The 3D thing is modelled using a description tree. The nodes of the tree can be geometrical primitives or geometrical operations applied to the children nodes. These geometrical operations are object decorators or composite objects, and may have one or multiple child nodes.

OOML could be described in the following points:

- The OOML is Open Source, you can use it, modify it, and share it.
- The OOML is used to model physical things by describing it geometric properties.
- It allows designers to create mechanical parts using the C++ language and applying the Object Oriented Programming Paradigm.
- The OOML is semantics oriented. It applies a semantic categorization of things.
- It includes all the geometry operations required to manipulate objects in space.

There are other tools for modelling physical things, OpenSCAD is used for designing solid models that will be printed afterwards, OpenRAVE can model mechanisms and robots for simulation, etc. The key point of OOML is that it is not designed for a particular purpose. With the OOML the designer can model a thing and this model can be used (developing other tools that integrate the OOML) in simulations, for fabrication, etc.

The authors of this paper have used this tool for designing small open source robots and test on them new ideas and algorithms. This tool is expected to allow researchers around the world to create new robotic platforms or other physical designs facilitating the exchange of their models and their collaboration. Thanks to this, algorithms can be tested on others platforms, and consequently, objective benchmarking can be used. 3D printers allow for a fast and low-cost manufacturing of designed things. Examples of OOML code, documentation, and tutorials can be found in http://iearobotics.com/oomlwiki.

As it has been shown in Figures 15, 16 and 18, designing and printing robots is feasible. These robots are perfectly suitable for research as test-beds. Fleets of robots are easy and cheap to print, opening a new possibility in robotic swarm research, which up to now has been mostly done through simulation. Thanks to the Open Source and Object Oriented nature of the OOML the platforms can be modified, improved, combined, etc. opening a vast spectrum of possibilities to researchers.

The OOML is quite a recent library, and much work is to be done yet. Results after the first year let us think that it may have a great impact on the open research community. The current work to do is to increase the number of primitives and parts in the library. New functionalities are being added and already used by the community on its beta version. Besides generating OpenSCAD code we are working on directly generating STL files and adding an OpenGL viewer. In the near future the OOML is planned to be integrated into a robotics simulator. The OOML will be used to build the simulated robots and compute the collisions. It will be the community to enrich the library, and thus, a dissemination work is necessary in order to have an impact on the research community.

We will close this article summing up some of the key advantages of the OOML.

- OOML is fast and easy to learn allowing rapid design of things.
- 3D models can be shared using versioning tools.
- Models can be reused and evolved.
- Models can be parametric.
- Models can be used for a multiplicity of applications, not only for fabrication but also for simulation, structure analysis, visualization, etc.

– Thanks to its OOP implementation the OOML is easy to extend.

7 Acknowledgements

We would like to acknowledge all the students of Universidad Carlos III de Madrid that have participated in the last two years in the courses of "Open robots design, manufacturing, and programming". These courses were voluntary and did not belong to their official curriculum. They participated motivated by their enthusiasm and interest in open robotics. Their time and generosity is highly appreciated.

We also thank our former Department: Departamento de Ingenieria de Sistemas y Automatica of Universidad Carlos III de Madrid for supporting this initiative.

References

1. Lizabeth Azrum. Education standards, http://curriculum.makerbot.com/software.html. accessed 12/11/2012, 2011.
2. David Bak. Rapid prototyping or rapid production? 3D printing processes move industry towards the latter. *Assembly Automation*, 23(4):340–345, 2003.
3. Adrian Bowyer. The Self-replicating Rapid Prototyper, Manufacturing for the Masses. In *8th National Conference on Rapid Design, Prototyping & Manufacturing*, June 2007.
4. Simon Bradshaw, Adrian Bowyer, and Patrick Haufe. The Intellectual Property Implications of Low-Cost 3D Printing. *SCRIPTed 5*, 2010.
5. W. Cho, E.M. Sachs, N.M. Patrikalakis, and D.E. Troxel. A dithering algorithm for local composition control with three-dimensional printing. *Computer-aided design*, 35(9):851–867, 2003.
6. Erik 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.
7. R.A. Giordano, B.M. Wu, S.W. Borland, L.G. Cima, E.M. Sachs, and M.J. Cima. Mechanical properties of dense polylactic acid structures fabricated by three dimensional printing. *Journal of Biomaterials Science, Polymer Edition*, 8(1):63–75, 1997.
8. J. Gonzalez-Gomez, A. Valero-Gomez, A. Prieto-Moreno, and M. Abderrahim. A new open source 3D-printable mobile robotic platform for education. In Ulrich Rckert, Sitte Joaquin, and Werner Felix, editors, *Advances in Autonomous Mini Robots*, pages 49–62. Springer Berlin Heidelberg, 2012.
9. G.S. Hornby and J.B. Pollack. The advantages of generative grammatical encodings for physical design. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 600–607. IEEE, 2001.
10. Rhys Jones, Patrick Haufe, Edward Sells, Pejman Iravani, Vik Olliver, Chris Palmer, and Adrian Bowyer. RepRap-the replicating rapid prototyper. *Robotica*, 29(01):177–191, January 2011.
11. Anna Kochan. Rapid prototyping gains speed, volume and precision. *Assembly Automation*, 20(4):295–299, 2000.
12. C.X.F. Lam, XM Mo, S.H. Teoh, and DW Hutmacher. Scaffold development using 3D printing with a starch-based polymer. *Materials Science and Engineering: C*, 20(1):49–56, 2002.
13. Yeong-Bae Lee, Samuel Polio, Wonhye Lee, Guohao Dai, Lata Menon, Rona S. Carroll, and Seung-Schik Yoo. Bio-printing of collagen and vegf-releasing fibrin gel scaffolds for neural stem cell culture. *Experimental Neurology*, 223(2):645 – 652, 2010. ¡ce:title¿Beta-amyloid and tau protein abnormalities in Alzheimer's disease¡/ce:title¿.
14. B. Leukers, H. Gülkan, S.H. Irsen, S. Milz, C. Tille, M. Schieker, and H. Seitz. Hydroxyapatite scaffolds for bone tissue engineering made by 3D printing. *Journal of Materials Science: Materials in Medicine*, 16(12):1121–1124, 2005.
15. D. Marbach and A.J. Ijspeert. Co-evolution of configuration and control for homogenous modular robots. In *Proceedings of the Eighth Conference on Intelligent Autonomous Systems (IAS8)*, pages 712–719, 2004.
16. J.B. Pollack, H. Lipson, G. Hornby, and P. Funes. Three generations of automatically designed robots. *Artificial Life*, 7(3):215–223, 2001.
17. Christina Raasch, Cornelius Herstatt, and Kerstin Balka. On the open design of tangible goods. *RD Management*, 39(4):382–393, 2009.

18. Richard Stallman. Invited lecture at the 4th Congress of Free Software in Venezuela. 2008.
19. Alberto Valero-Gomez, Juan Gonzalez-Gomez, Victor Gonzalez-Pacheco, and Miguel Angel Salichs. Printable creativity in plastic valley uc3m. In *IEEE Educon 2012*. IEEE, April 2012.