Journal Title 1 (2016) 1–5
IOS Press

Ontology Based Design, Control and Programming of Modular Robots

Francisco Ramos [a,*],Andrés S. Vázquez [a],Raúl Fernández [a] and Alberto Olivares-Alarcos [a].
[a] *School of Industrial Engineering, University of Castilla-La Mancha, Avda. Camilo José Cela, 3, 13071 Ciudad Real, Spain*
E-mails: Francisco.Ramos@uclm.es, AndresS.Vazquez@uclm.es, Raul.Fernandez@uclm.es, Alberto.Olivares@uclm.es

Abstract. This work presents a proof of concept for an end-to-end process for the agile design, control and programming of robots. The user selects a set of abilities for the desired robot, and the system automatically generates the robot structure and the controllers needed for the high level programming of the robot. This process relies on two pivotal concepts for our approach: ontologies and modular robots. An ontology (ADROn), constructed upon the IEEE Standard Ontologies for Robotics and Automation (ORA), is used by an intelligent system to relate the robot types with the requested set of abilities. Then, a parameterized design process generates the passive components (e.g. links) and selects the active modules (e.g. actuators) from a collection of 3D printable, modular robotic components (ParMoR). Finally, the controllers, generated as ROS services, are interfaced with a visual programming language, Bitbloq, suitable for novice users. Examples of the generation process of different types of robots (wheeled, snake, humanoid, hexapods) are detailed. The process is presented as an educational platform for the teaching of robotics, while the applicability to other fields and the advantages of this methodology are also discussed in the paper.

Keywords: Modular robotics, Ontologies, Agile robotics, ROS, BitBloq

1. Introduction

Nowadays, robotic paradigms are changing and, with them, the industry. Until now, the predominant model for automated robotic processes has been a highly-specialized robotic workcell where robots are chosen depending on task specifications, such as manipulability, reachability or payload. End-effector changing and robot reprogramming allows companies to reuse robots giving some flexibility to workcells. However, this flexibility is limited mainly by two factors that are approached in this work: difficulty for software reprogramming and hardware reusability. These factors have made robots principally suitable for big companies with long term products which really compensate for the high cost of robot installation. We identify both software reprogramming and hardware reusability as key aspects for agile robotics [1, 2].

Robot reprogramming is evolving with the new paradigm of collaborative robots, wherein robots, thanks to new programming and control methods, can operate alongside the production personnel in dynamic or semi-structured human environments. Some representative research works can be found in [3, 4]. In the last few years, collaborative robots have jumped to real industry facilities. Some examples are Baxter and Sawyer of Rethink Robotics [5], LBR iiwa of KUKA [6], Yumi of ABB [7] and UR series of Universal Robots [8].

Robot reusability has also a hardware dimension. On many occasions the same robot configuration cannot be used for two different tasks (e.g. reachability, payload). The paradigm of Reconfigurable Modular

*Corresponding author. E-mail: Francisco.Ramos@uclm.es.
This work has been supported by Mundo Reader S.L. and the CDTI under expedient IDI-20150289: BOTBLOQ: An Integral Ecosystem for the design, manufacturing and programming of Do-It-Yourself robots.

0000-0000/16/$00.00 © 2016 – IOS Press and the authors. All rights reserved

Manipulators, *RMMs*, which have also rapidly moved from research [9, 10] to industry [11], is the exponent of robot reusability. Some of the challenges of *RMMs* are the automatic generation of kinematics, dynamics and control [12] and also the mapping between tasks and manipulator configurations [13].

This mapping must answer the following question: how can the knowledge that relates tasks to configurations be generated in a synthetic manner? To achieve this, we propose the use of ontologies, widely applied to expert systems for different applications such as medical advisory [14], and lately drawing some attention in robotics as well [15, 16]. They would allow the inference of this mapping by settling adequately the robotic domain knowledge.

In this work we deal with both reprogramming and hardware reusability within the same framework: our system, using task specifications, infers robot designs that can be built from reusable robot modules and, at the same time, it generates controllers and easy programming blocks that can be used by non specialized personnel. Two main aspects define our framework: a parametrized modular robot architecture and an ontology based agent.

Let us clarify that this work is a proof-of-concept. The scope of this work could be as extensive as desired: industrial robots plus household robots plus entertainment robots plus... However, we do not claim to be able to create a specific robot for any of the (arguably infinite) possible tasks achievable by any kind of robot. We aim to present the process and we focus on the educational aspects of robotics, in which we try to smartly bring robotics design closer to non-qualified users, while giving an overview of the applicability to other fields.

2. Why are these the right tools?

In this section we justify the selection of the two main tools of our setup, that is, an ontology and a parametrized modular architecture, explaining how they are beneficial to the co-design of robots and motion controllers in an automatic manner.

2.1. Why ontologies?

Ontologies, in a computational sense, are formal and explicit specifications of conceptualizations [17] and provide enough concepts and relations to articulate models of specific situations in a given domain. However, they do not simply represent but also can be used to generate knowledge: if an ontology is correctly designed, we do not need to specify all facts explicitly, but we can also use inference methods to extract implicit knowledge. For instance, if we define the class *HumanoidRobot* as a robot which includes two arms and two legs, when we instantiate that class we do not need to specify that our instance has two arms and two legs.

When conceiving any sort of design process, it is important to take into account how *purpose/functionality*, *possible implementations* and *resources* interact. In [18], Censi describes a theory to deal with co-design problems in a principled way. In that work a design problem is defined as a tuple of *function space*, *implementation space*, and a *resources space*, plus the two maps that relate implementations to functions and implementation to resources. It also shows the existing relationship between functionalities, structures and resources. Either if it comes from designers or from users, any design is created under a set of specifications that the final system must fulfill. At the same time, we usually have some restrictions imposed by the finite amount of available resources. Therefore, we could just consider a limited number of possible implementations which fit both, requisites and resources. When we apply this idea to the robotic domain, if user requirements are understood as a list of capabilities that a robot must possess, the possible implementations of the structure of a robot rely on, among others, this list of capabilities. However, [18] lacks the use of ontologies that could help to represent and exploit all knowledge related to the design.

In fact, given one of the sides of the relationship between robot capabilities and the physical structure of the robot, a knowledge system (knowledge base plus reasoner) could infer the existence of the other one and vice versa. For instance, if there exists a relationship of necessity between having a gripper (physical device) and the capability of grasping, information could be inferred bidirectionally: on the one hand if the robot possesses a gripper, it can grasp objects; and on the other hand if a robot can grasp, it must possess a gripper. In both cases, an ontology seems to be the perfect tool to use during the inference process.

Finally, the knowledge behind any design process must bring together the theoretical knowledge related to the specific domain and the common knowledge extracted from experience. If we want machines to be able to design robots as humans do, or even better, it is compulsory to comprehend the human design process

and then to represent that knowledge in a machine-friendly format. Our approach, generic as it is, relies on this issue. Ontologies [17] are the most well known method to represent knowledge in current literature, existing several successful examples [19, 20], and it is being standardized in the field of Robotics and Automation (R&A) [16] for this very reason. Based on these premises, an ontology proves an excellent choice to store our knowledge base.

2.2. Why modular robotics?

Robots are usually designed manually, which makes this process very expensive, time consuming and also quite difficult to adapt to different scenarios. For this reason many works deal with the automatic design of both robots and their controllers, often based on modular design approaches [21].

Modular robots are widely used in different domains such as education, research or industry (see [22–24] for complete reviews). There are several factors that determine whether a modular robot is suitable for a specific domain. In some cases the same modular platform is used for both education and research domains but rarely for the industry too. This is usually achieved by using different programming languages: simple ones for education (usually visual languages with blocks like Lego Mindstorms or Scratch, used for example with Cubelets and MOSS robots from ModRobotics) and more advanced for research applications (like Robotis Bioloid which have a graphical C++ based programming tool called *RobPlus Task*). In the industry, companies have their own languages like RAPID (ABB) or VAL 3 (Stäubli). For the specific case of collaborative robots, programming is usually done following a lead-through methodology, like ArtiMinds in Universal Robots, Intera in Rethink Robotics or Sunrise Workbench in KUKA robots.

It is important to indicate that just because a modular platform has an easy programming language (e.g. Blocky or Scratch) it does not mean that every robot created with that platform will be easy to program. Lead-through programming can also be a very slow technique and it is not appropriate in some cases. For example, using this method for the movement planners of complex robots (e.g. humanoids) can be tedious and sometimes is not powerful enough (e.g. the walking of a humanoid robot can be statically programmed using lead-through techniques but not dynamically, which means that it will fall down in then presence of external forces). This makes complex robot programming out of the reach of novice users [25]. That is why many modular platforms for education include previously designed robots with movement planners already programmed. Some examples are the Bioloid humanoid robot of Robotis, the quadruped robot of Fable [26] or the JD humanoid of EZ-Robot.

This impracticality of robot programming restricts the freedom of users in creating new robots or even modifying existing ones. This leads to the conclusion that it is necessary to find a way that helps novice users to program their complex robots, e.g. obtaining planners automatically. At this point the concept of *co-design* as the simultaneous design of mechanisms and their controllers becomes relevant. For instance, [27] describes a method to simultaneously optimize quadruped robot gaits and the mechanism design parameters. They start from biologically inspired good candidates which are later optimized using an evolutionary algorithm (EA). Another similar work is presented in [28], where the authors introduce a method for co-optimizing robot physical designs and their motion using a gradient-based optimization approach. Though the aim of our work is similar, we focus on the way that the ontology can extract useful information for the structure and the control designer rather than the optimization problem. In fact, we only use optimization solvers to minimize link material as explained in Section 4.2.2. For both the body and the controller design we use a recursive algorithm that, starting from a parametric kinematic configuration, finds the best link dimensions and actuator characteristics that meet user requirements.

A work, very similar to our approach, that deals with *design selection* is [29]. In this work the authors present an end-to-end system that integrates the low-level design generation, and a high-level mission planning with their own modular robot architecture. We also present an end-to-end approach but with the difference that the robot is automatically designed while in the referenced work they provide a library to help users with the manual design. Another end-to-end effort that goes from task specification to implementation of robots is developed in [30]. It presents a knowledge transfer framework, GTax, which aims to simplify robot programming and enables transfer of knowledge between. This framework is developed for manufacturing robots and, hence, it addresses practical problems for industrial applications such as performance requirements. Nevertheless, our approach tries to span a wider robotic domain and, to date, it does not account for those requirements.

Several of the most well-known approaches to automatic design of robots come from the field of evolutionary robotics. This happens because EAs can come up with solutions which humans could not imagine [31]. An advanced development of EA, Memetic Computing techniques, are hybrid EAs that employ deterministic local search in the evolutionary cycle to solve optimization problems, outperforming each technique separately. These techniques can be applied for example to the control wheeled robots [32]. However, EAs are still at the research stage, which makes them unsuitable for education or other domains like industry.

In our approach we propose a Parametric Modular Robotic (ParMoR) platform that can be used for education and research and can be considered as a demonstrator for other domains like industry. Our framework helps users to develop and program complex robots (see Section 4). Similar to other platforms, we propose an intuitive, visual programming language (Bitbloq [33]) for novice users, for example inexperienced personnel, and more advanced programming (Python/C++ with ROS) for researchers or experienced users.

The main difference with other modular platforms, shown in Table 1, is that ours does not limit users to several specific robots with previously programmed controllers. In our approach users have (theoretically) an infinite number of complex robots available, with their respective controllers helping with the reprogramming (i.e. low level details are transparent to the users). These robots and controllers are automatically generated from parametric configurations.

Other requirements we have imposed on our modular robot approach are:

- Open hardware/software modular platform.
- Easy module building with 3D printing and accesible electronics.
- Easy and robust robot assembly.
- Possibility to build any robot configuration.
- Different programming languages for different level users.
- *Possibility to program any complex robot, regardless of the user level.*

Table 1
Modular Robotics Platforms used in both education and research

	Lego EV3	Ez-Robot	MOSS	Bioloid	Fable	ParMoR
Open Hardware/Software architecture						✓
3D printed modules simple electronic		✓			✓	✓
Simple assembly robots robust enough		✓		✓	✓	✓
Allows build any robot configuration	✓	✓	✓	✓	✓	✓
Different programming depending of user level	✓	✓	✓	✓	✓	✓
Programming any robot configuration, regardless of user level						✓

3. Automatic Design of Robots Ontology

The Automatic Design of Robots Ontology (ADROn) defines concepts and relations that are to be used for the automatic conceptual design/selection of robots. ADROn has been constructed upon the IEEE Standard Ontologies for Robotics and Automation (ORA) [16]. This standard, written in SUO-KIF[1] language [34], provides a formalism that allows to represent and reason with the knowledge gathered by the experts in the field over years of research. It is constructed upon the Suggested Upper Merged Ontology (SUMO) [20] which provides high level classes, such as $Motion_S$[2] (a subclass of $Process_S$) or $Entity_S$ (universal class of individuals), and the relations between them, such as $AntisymmetricRelation_S$, which are, in turn, subclasses of abstract entities. A partial, very reduced taxonomy of SUMO will be displayed in Figure 3.

The potential of ORA has already been explored in some cases of study [35] and it is still under development. In fact, several Working Groups of the IEEE are developing a suite of ontologies for different fields of robotics [36–38]. ORA is divided into four ontologies, whose dependences are displayed in Figure 1. They are required to construct more specific concepts belonging to other ontologies and consist of:

- **CORA** (Core Ontology for Robotics and Automation) includes the fundamental concepts in

[1] Standard Upper Ontology Knowledge Interchange Format is a declarative semantics language designed for use in the authoring and interchange of knowledge.

[2] Each ontology concept used in the article that do not belong to our ontology will have a subscript to indicate the origin of the concept: S stands for SUMO, C for CORA, X for CORAX, R for RPARTS and P for POS.

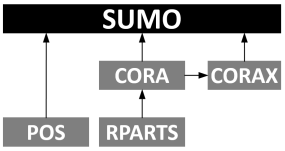


Fig. 1. Ontologies involved in the ORA Standard.

the R&A domain, such as $Robot_C$ or $robotPart_C$, as well as their definitions, attributes, constraints and relationships.

– **CORAX** defines concepts too general (could be used in many other domains) to be in the CORA ontology and needed for modeling but not covered by SUMO. For example, a $Design_X$ is a proposition that idealizes the structure of an $Artifact_S$.
– **RPARTS** provides an extensible set of roles that specializes a $robotPart_C$, such as $robotSensingPart_R$, but it does not describe the actual devices that could play that role (e.g. *ultrasoundSensor* or *IMUSensor*).

In its actual state, ORA does not completely match the needs of this work. While it states several key concepts (to our purpose) such as $RobotPart_C$ or $RobotMotion_X$, it lacks specific robot motions and robotic parts that could be interrelated to obtain structural dependencies. Therefore, an extension for the ORA ontologies is required for our automatic design process.

In order to extend ORA, we have considered two aspects related to our approach which show the necessity of that extension: the resources we use to build robots and the proposed methodology to do it automatically. On the one hand, we work with a modular architecture of robots and, therefore, it is necessary to define some physical concepts, which are yet to be included in ORA, to represent ParMoR elements in our knowledge system. On the other hand, our approach needs a mapping from actions to the structural parts that are needed to perform them. Finally, we need to define the different types of robots that we might find depending on the resulting design (see Section 3.3). In order to cover all needed knowledge, several new concepts are proposed in the following subsections, where we provide a definition in natural language, but also a formalization in First Order Logic. For a better understanding, we also include some code written in SUO-KIF when needed.

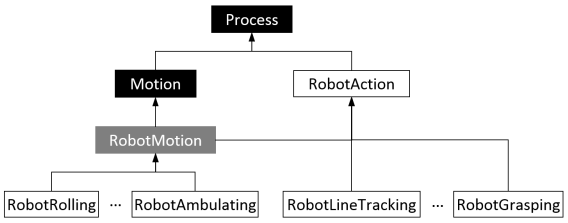


Fig. 2. *RobotAction* declaration and relation with SUMO and CORA concepts.

3.1. Robot actions

ADROn defines concepts regarding the actions that a robot can perform under the class *RobotAction*, which is a subclass in $Process_S$ and a superclass of $RobotMotion_X$, as shown in Figure 2. It indicates a process in which the agent is a $Robot_C$. Examples of *RobotAction* are *RobotAmbulating* or *RobotLineTracking*.

3.2. Structural robot parts & Structural requirements

In CORA, $robotPart_C$ is defined as a subrelation of $part_S$ which relates devices and robots. Intuitively, when we think about a part of a robot, our thoughts are more likely to consider them as physical objects than as relations between devices and robots. Nevertheless, as the standard states, devices are considered robot parts while attached to a robot, but they are not inherently a robot part, since they exist by themselves and, in most cases, they can be connected to other kinds of devices. Because of this, $robotPart_C$ is a relation and not a physical entity.

Even though we agree with the definitions given in the standard, we still need to represent two specific concepts related to physical components which are not yet covered. The main physical elements defined in ADROn are *StructuralRobotPart* and *Module*, which are subclasses of $Device_S$ and $Artifact_S$, respectively, as depicted in Figure 3. Aligned with them we also introduce the concept of *StructuralRequirement* which is a $BinaryPredicate_S$ that links each *RobotAction* to the *StructuralRobotParts* involved in the performance of those actions.

Therefore, a *StructuralRequirement* (*SR*) captures the existing relationship between a *RobotAction* (*RA*) and the *StructuralRobotPart* (*SRP*) which plays a necessary role in the action. This concept lets us infer the necessary structural parts that a robot should have to

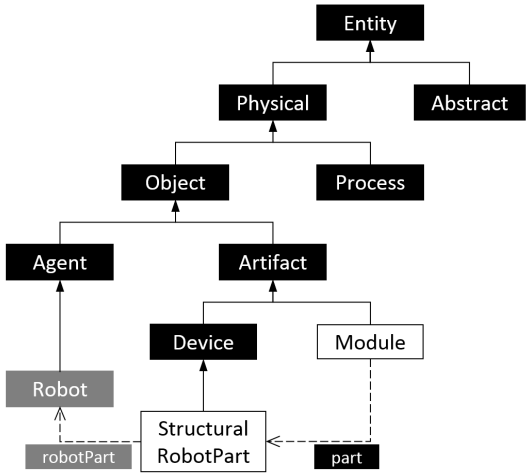


Fig. 3. Taxonomy of the main physical concepts in ADROn (white) and relation with SUMO (black) and CORA (gray).

fulfill the actions requested by the user. It is formalized as follows:

$$\forall x \exists y : RA(x) \Rightarrow SRP(y) \wedge SR(x,y)$$

The corresponding code written in SUO-KIF is the following:

```
(instance StructuralRequirement BinaryPredicate)
(instance StructuralRequirement InheritableRelation)
(domain StructuralRequirement 1 RobotAction)
(domain StructuralRequirement 2 StructuralRobotPart)
```

Hence, a *StructuralRequirement* determines, for example, that if a $Robot_C$ is going to grasp an object, it needs a specific *StructuralRobotPart* to do it (e.g. a robot gripper). In our ontology, *RobotGrasping*[3] is defined not only as a subclass of *RobotAction* but also of $Grabbing_S$ to be consistent with the standard, which is based on SUMO.

$$\forall x \exists y : RobotGrasping(x) \Rightarrow RA(x) \wedge RoboticGripper(y) \wedge SR(x,y)$$

```
(subclass RobotGrasping RobotAction)
(subclass RobotGrasping Grabbing)
(=>
  (instance ?GRASP RobotGrasping)
  (exists (?GRIPPER)
    (and
      (instance ?GRIPPER RoboticGripper)
      (StructuralRequirement ?GRASP ?GRIPPER))))
```

[3]We use "RobotGrasping" in our ontology instead of "RobotGrabbing", which would be consistent with SUMO class, because the first term is widely used in the domain, and both are synonyms.

Continuing with the other new concepts, on the one hand, an instance of *StructuralRobotPart* represents any *Module* or set of *Modules* that is a $robotPart_C$ (rP) of a $Robot_C$ and plays an essential role in a specific action of a robot. For example, *RobotLeg* is a part of the structure of a robot that is requested to ambulate (either to walk or to run). On the other hand, an instance of *Module* will be any artifact (passive or active) which can be included in a robot (e.g. IMU sensors, servomotors, links, etc.). Below, a formalization of those concepts is given:

$$\forall x \exists y \exists z : SRP(x) \Rightarrow Device(x) \wedge RA(y) \wedge SR(y,x) \wedge Robot(z) \wedge rP(x,z)$$

$$\forall x \exists y \exists z : Module(x) \Rightarrow Artifact(x) \wedge SRP(y) \wedge part(x,y) \wedge Robot(z) \wedge rP(y,z)$$

Again, the code implemented in SUO-KIF to define those concepts in our ontology is as follows:

```
(subclass StructuralRobotPart Device)
(=>
  (instance ?STRUCTURE StructuralRobotPart)
  (exists (?ROBOT ?ACTION)
    (and
      (instance ?ACTION RobotAction)
      (StructuralRequirement ?ACTION ?STRUCTURE)
      (instance ?ROBOT Robot)
      (robotPart ?STRUCTURE ?ROBOT))))

(subclass Module Artifact)
(=>
  (instance ?MODULE Module)
  (exists (?ROBOT ?STRUCTURE)
    (and
      (instance ?ROBOT Robot)
      (instance ?STRUCTURE StructuralRobotPart)
      (robotPart ?STRUCTURE ?ROBOT)
      (part ?MODULE ?STRUCTURE))))
```

Examples of subclasses of *StructuralRobotPart* are *EndEffector*, *RobotLimb* or *RobotTrunk*.

3.3. Robot types

ADROn also defines a number of robot types such as *HumanoidRobot* which are subclasses of $Robot_C$ according to the taxonomy outlined in Figure 4.

Each of these robots consists of one or more *StructuralRobotPart* depending on their definition. For example, the following code defines a *HumanoidRobot* subclass consisting of two *RobotArm* and a *RobotTrunk* which are, in turn, subclasses of *StructuralRobotPart*. In addition, as long as *HumanoidRobot* is a subclass of *BipedalRobot*, it also inherits two instances of *RobotLeg*. In this case we show just the code written in SUO-KIF since the formalization in First-Order Logic might be confusing.

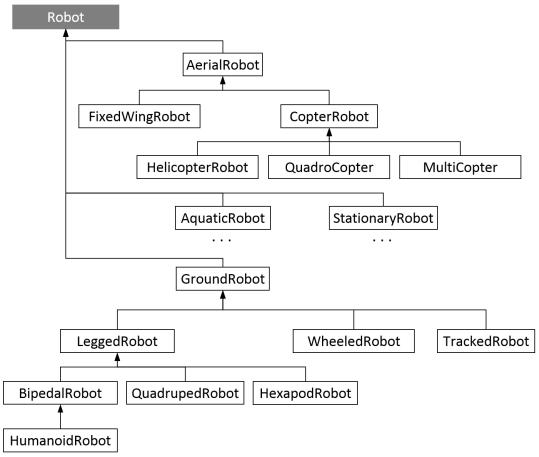


Fig. 4. Partial robot classification depending on the environment/locomotion.

```
(subclass GroundRobot Robot)
(subclass LeggedRobot GroundRobot)
(subclass BipedalRobot LeggedRobot)
(=>
  (instance ?rob BipedalRobot)
  (exists (?leg1 ?leg2 ?trunk)
    (and
      (instance ?leg1 RobotLeg)
      (instance ?leg2 RobotLeg)
      (instance ?trunk RobotTrunk)
      (robotPart ?leg1 ?rob)
      (robotPart ?leg2 ?rob)
      (connectedTo ?leg1 ?trunk)
      (connectedTo ?leg2 ?trunk)
      (robotPart ?trunk ?rob)
      (not (equal (?leg1 ?leg2))))))

(subclass HumanoidRobot BipedalRobot)
(=>
  (instance ?rob HumanoidRobot)
  (exists (?arm1 ?arm2 ?trunk)
    (and
      (instance ?arm1 RobotArm)
      (instance ?arm2 RobotArm)
      (instance ?trunk RobotTrunk)
      (robotPart ?arm1 ?rob)
      (robotPart ?arm2 ?rob)
      (connectedTo ?arm1 ?trunk)
      (connectedTo ?arm2 ?trunk)
      (robotPart ?trunk ?rob)
      (not (equal (?arm1 ?arm2))))))
```

4. Automatic robot design system

This section gives an overview of the complete procedure for generating completely functional robots from (very basic) user requirements. Figure 5 shows the process flow of the complete system. This process is summarized as follows:

1. The user, using a software application, describes the behavior(s) to be performed by the robot.

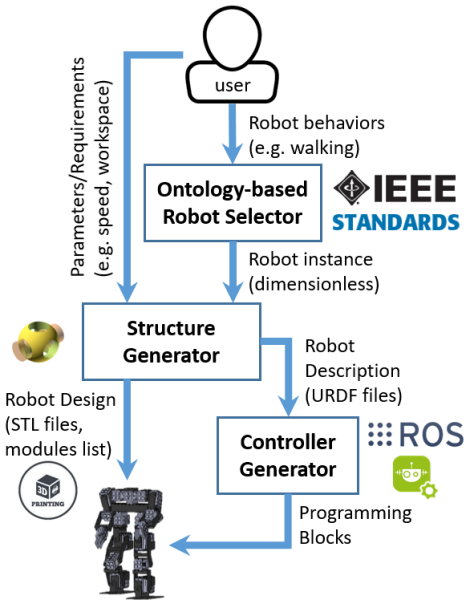


Fig. 5. Flow diagram of the automatic generation of robots.

2. A knowledge system, consisting of a reasoner (Sigma [39]) and a procedure to extract relevant information from ADROn, infers a robot base configuration.
3. Depending on the inferred configuration, a set of parameters/requirements are specified by the user in order to create the physical structure.
4. The base configuration together with the user requirements are passed on to the Structure generator which selects the concrete active modules and designs passive modules according to kinematic and dynamic considerations. According to these modules, robot description and 3D printing files are created.
5. Finally, the formal description of the robots is represented in ROS using Unified Robot Description Format (URDF) models and a parameterized controller is provided to simplify the control stage.

For example, given the behavior of grasping objects, the automatic designer asking the ontology would select a manipulator robot and after the optimization would obtain the link dimensions based upon a certain rigidity. This can be related to the maximum payload that the robot arm must be able to carry by allowing a maximum tip deflection when performing maneuvers.

The following subsections will illustrate the process in further detail.

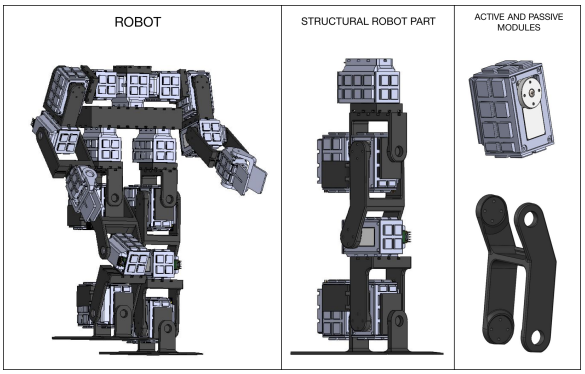


Fig. 6. Instances of the main concepts in ADROn related to the *Robot* concept from CORA.

4.1. Ontology-based robot selection

According to ADROn, a robot consists of one or more *StructuralRobotPart* and each of them consists of, at least, one *Module*. Figure 6 shows: an instance of a *HumanoidRobot* (subclass of $Robot_C$); an instance of one of its *RobotLegs* (subclass of *StructuralRobotPart*); and some instances of the *Modules* (active and passive) that constitute the *RobotLeg*.

ADROn includes the definition of every module of our modular architecture (e.g. *IRProximitySensor*, *Servomotor*, etc.) along with every action that a robot can perform and the relationships between *RobotActions* and *StructuralRobotParts*.

The conceptual generation of a robot is a three-step process. First, the robot instance generator receives a set of *RobotActions* that the robot is required to perform. Then the reasoner uses semantic queries to determine the *StructuralRequirements* implied by the set of actions. Subsequently, the instance generator matches these requirements with the hardware available in the base configurations defined in ADROn. If several matches are found, the generator asks the user some questions inferred from the ontology to disambiguate the solution. Finally, the system creates the conceptual (instance) design of a robot able to perform those actions and passes it to the structure generator. This process is schematized in Procedure 1.

As an example, we present now a very straightforward example of generation of a robot with the ability of *RobotWalking*.

1. First, the generator user demands a robot with a walking behavior.
2. A querying process in ADROn determines *RobotLeg* as a *StructuralRequirement*.

Procedure 1 Robot Instance Generation

Require: sRA ← set of *RobotActions*
  for each *RobotAction* in sRA **do**
    determine *StructuralRequirement*
  end for each
  solutions ← match all *StructuralRequirements*
  while solutions.length > 1 **do**
    generate disambiguation question
    update *StructuralRequirements*
    solutions ← match all *StructuralRequirements*
  end while
  return instance of solution

3. A search through the base configurations obtains all the matches: *HumanoidRobot*, *QuadrupedRobot* and *HexapodRobot*.
4. Disambiguation is done by querying the ontology about the additional capabilities that matched robots possess. In this case, we discover that only the *HumanoidRobot* has the capability of grasping so the generator asks the user "Does the robot need to grasp?".
5. The answer is positive, so the generator determines *RobotGripper* as a new *StructuralRequirement*.
6. It searches again through previous matches and determines that the appropriate robot is a *HumanoidRobot*.
7. The generator provides a dimensionless base configuration of a *HumanoidRobot*. The conceptual design is over and the parameterization process begins.

Actually, this methodology has its flaws, maybe the most important is what to do when several robots have exactly the same capabilities (e.g. *QuadrupedRobot* and *HexapodRobot*). In future versions we will face this problem and try to solve it with a more complex process, for instance, we could differentiate the two robots cited before by knowing that with six legs our robot could walk more easily over rough terrain, as its static stability is higher. However these concepts are yet to be implemented in ADROn.

4.2. Robot structure generation

Once the base configuration has been determined, the generation of the physical structure begins. The base configuration together with the user requirements (e.g. robot speed, payload, workspace) are passed on to the automatic designer which selects the concrete

active modules and designs passive modules for the requested robot. Figure 7 depicts the flow of the Structure Generator block of Figure 5.

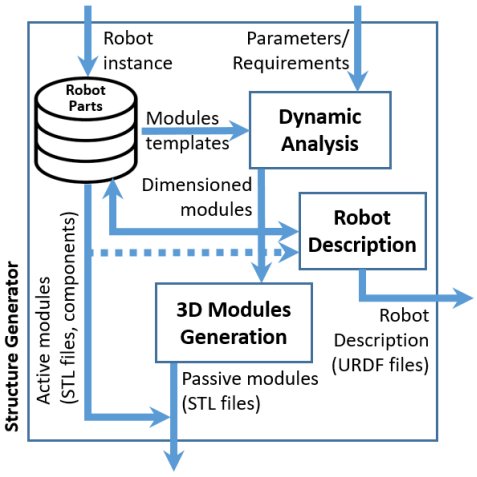


Fig. 7. Flow diagram of the structure generation of robots.

We can see three main components: the dynamic analysis, the robot description and the 3D modules generation. Also a database provides the definition of all active modules and the templates of the passive modules to the components. These will be detailed in subsequent sections.

4.2.1. Dynamic analysis

The user variables are combined with the pool of modules available in ParMoR to dimension the robot passive modules and select the adequate active modules.

Some of the user requirements might condition the physical structure of the robot due to the restrictions they impose. Therefore, a study of the robot dynamics, specially in a worst case scenario, is needed to guarantee proper functioning of the robot. This study determines passive modules dimensions and torque demands for the actuators of each joint. The process iterative and follows these steps:

1. A tentative value for links lengths is assumed depending on desired workspace and dimensions of the active modules that must fit in the structure.
2. The recursive Newton-Euler method [40] is applied to the kinematic chains of the different base configurations in order to calculate the torques that must be exerted at each joint to actuate the robot as specified by user requirements, and the forces and torques endured by each link. This evaluation will strongly depend on user requirements (e.g. payload) and ParMoR modules (weights, inertias, dimensions...).
3. Active modules fulfilling the calculated requirements are chosen from the ParMoR architecture.
4. Passive modules are designed following a material optimization process and a stress analysis detailed in following subsections.
5. The process is restarted if there are no active modules fulfilling the requirements or the passive modules design gives an unfeasible solution.

Figure 8 displays an example of different designs obtained with different user requirements. As can be seen, even though both robots show the same abstract structure, (3 dof manipulator), the Structure Generator determined a different geometry for passive modules (larger workspace) and different active modules (because of higher inertial forces).

Fig. 8. Robot A. (left) Two high-torque actuators and a middle–torque actuator. Robot B.(right) One high-torque module and two middle-torque actuators.

4.2.2. Passive modules optimization

Once the lengths of the links have been calculated, a material optimization process begins. The parametrization process must avoid bending of links over a given limit while reaching the workspace requested by user requirements. Additionally, because we are manufacturing our modules with a 3D printer, we intend to use the minimum amount of material that guarantees both aims.

Hence, an optimization problem arises, which aims to minimize solid volume of printable links given some values, such as the length, geometric restrictions depending on the shape of the cross-section, and a min-

imum requested rigidity to avoid bending, which can be related to maximum payload of manipulator.

Cylindrical hollow beams have been considered for the robot links. The optimization problem for them is stated as follows:

$$\begin{aligned} \underset{L,r_2,r_1}{\text{minimize}} \quad & f = L(r_2^2 - r_1^2) \\ \text{subject to} \quad & r_2 \leq r_{2max} \quad r_2 - r_1 \geq e \\ & L_{min} \leq L \leq L_{max} \quad (r_2^4 - r_1^4)L^{-3} \geq \tilde{K}_0, \end{aligned} \tag{1}$$

where L is the length of the link, r_1 and r_2 are inner and outer radii, e is the minimum thickness allowed, and $\tilde{K}_0$ is a constant related to rigidity and dependent on Young's modulus, cross-section inertia and link length. The solution to this nonlinear optimization problem, (obtained using the Karush-Kuhn-Tucker method [41]), tends to build up material towards the external radius r_2, increasing the cross-section inertia. The dimensions of the optimized passive links of the manipulator are given by

$$\begin{aligned} & r_2 = r_{2max},\ r_1 = \sqrt[4]{r_{2max}^4 - \tilde{K}_0},\ L = L_{min} \\ & f = \pi L \left(r_{2max}^2 - \sqrt{r_{2max}^4 - \tilde{K}_0} \right). \end{aligned} \tag{2}$$

This solution is only valid if $r_2 - r_1 \geq e$ (as outlined in Figure 9). Otherwise, we set $r_2 - r_1 = e$ and radii are obtained from solving the following equations system

$$\begin{aligned} & r_2 - r_1 = e \\ & r_2^4 - r_1^4 = \tilde{K}_0. \end{aligned} \tag{3}$$

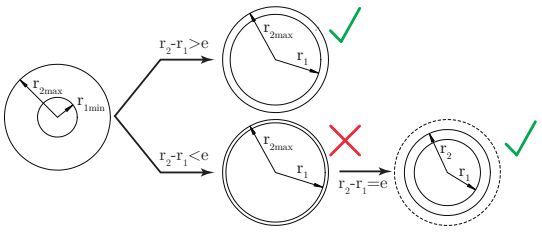


Fig. 9. Solutions depending on thickness restriction

4.2.3. Stress analysis

Each link of the robot has been analyzed as a beam clamped on one end with an external force and torque applied at the free end. These loads include inertial forces/torques due to the movement of the rest of the links. The effect of gravity has been assumed to be concentrated on both ends and equally divided. The following hypothesis have been assumed:

- Highest stress is presented at the clamped end. Hence, this section will be analyzed for having the highest risk of fracture.
- In the clamped end, slope and deflection are zero.
- Deflection in the free end is maximum and proportional to applied force.

Relation between clamped end stress and dynamic loads applied to the link has been solved from the stiffness equation given by:

$$\mathbf{f} = \mathbf{K} \times \mathbf{u} \tag{4}$$

where $\mathbf{f}$ represents the vector of characteristic forces and torques, $\mathbf{u}$ is the vector of displacements, and $\mathbf{K}$ is the stiffness matrix.

The beam has been discretized in a single beam element with two nodes (one in each end), each of them with 3 degrees of freedom.

With this equation, we can calculate the loads in the clamped end, which allows us to perform stress analysis and evaluate the possibility of a fracture in the link.

Additionally, as the geometric discontinuities cause a local increase in stress, the perpendicular union between the hollow cylinder and the dovetail faces generates a stress concentrator. This effect has been alleviated using fillets (see Figure 10).

4.2.4. ROS robot description

Each type of robot has a predefined script, written in Xacro language, that automatizes, once the parametrization is finished, the URDF file generation. Xacro is an XML macro language that simplifies the creation of URDF files (usually a tedious task) with a template and a set of values for its parameters. The use of these files allow the automation of the controller generation, as will be detailed in Section 4.3.

4.2.5. 3D printable files

A set of functions is available in OpenSCAD for creating the Stereo Litography (STL) files of the newly designed passive modules. In Figure 10 we can see different passive modules of beam type. They consist of two dovetail faces, a hollow cylinder and inner and outer fillets between faces and cylinder.

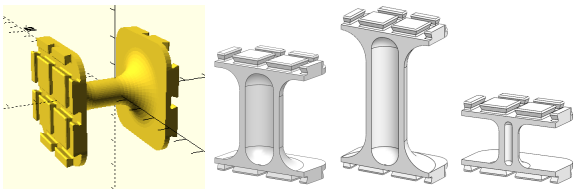


Fig. 10. Different links created from the same OpenSCAD function

4.3. *Co-design of motion controllers and robot structures from parametric kinematic models*

The last step in the automatic generation of functional robots is the creation of a controller for performing the behaviors requested by the user. This controller will also facilitate the robot programming, shielding the users from having to write low-level motion planners. The problem or controller design will be addressed as a part of a co-design problem of mechanisms and motion controllers. For this reason, the particular aspects of mechanism design for each robot will be also summarized in the following subsections, introducing the parametric kinematic modeling for each kind of robot and how it is used to generate motion controllers As it is seen in Figure 11 two parameters are sent to the controller generator: the robot design in URDF and the robot behaviors list described by the ontology (e.g. the user could only have specified the behavior *walking* but there are more behaviors necessary to control a humanoid robot like running or climbing). With this information, and as seen in Figure 11, the controller generator follows these steps:

1. Parametric planners for each behavior, based on analytical expressions, are selected from a database.
2. These planners are numerically parametrized using the information of the URDF.
3. A ROS service is created with each planner. An experienced user could use at this time the ROS API to program the robot.
4. Bitbloq blocks (i.e. planner functions) are generated for each service of ROS. These blocks can be used by novice users like k-12 students.

The outcome of this process is a set of ROS services that can be used by experienced users or that can be converted to blocks for visual programming.

4.3.1. *Parametric controller for wheeled robots*

Our automatic designer includes two configurations of wheeled robots : differential and skid-steer. The radius r of the wheels and the distance L_1, L_2 between them are the customizable parameters (see Figure 12). Depending on user specification (i.e. robot speed, payload and load area) our automatic designer will choose one of the configurations: differential for minimum payload and load area, and skid-steer for bigger payloads and load areas. L_1 and L_2 are automatically adjusted to meet load area specifications. Wheel radius r together with the type of actuators (active modules) are calculated through a dynamic analysis for velocities and torques (see Figure 12 for some examples).

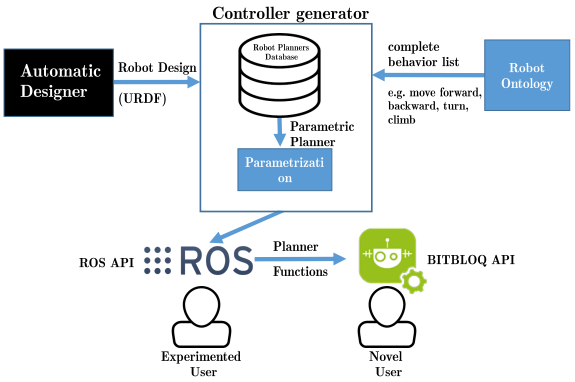


Fig. 11. Automatic Control Generator.

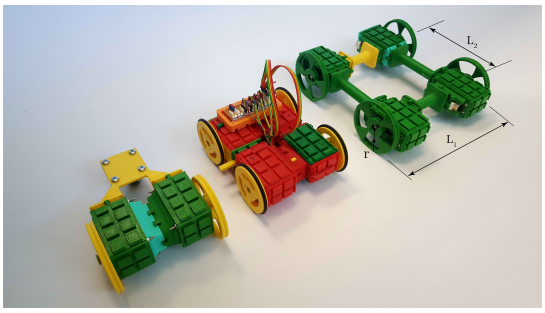


Fig. 12. Real robots generated according to different user specifications.

The motion controllers for these robots are based on the well known analytical solution for differential drive and skid-steer mobile robot kinematics [40]. For example, the movement of a differential mobile robot of Figure 13 can be defined with Eq. 5 and Eq. 6. As is seen, this movement will directly be conditioned by parameters l_1 (distance between wheels) and r (wheel radius).

$$\omega = \frac{r\dot{q}_r - r\dot{q}_l}{l} \tag{5}$$

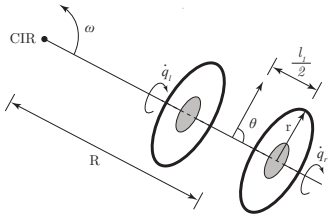


Fig. 13. Kinematic analysis of a differential robot.

$$R = \frac{l\,(r\dot{q}_r + r\dot{q}_l)}{2\,(r\dot{q}_r - r\dot{q}_l)} \tag{6}$$

We have used the *diff_drive_controller* ROS package, which includes the aforementioned differential drive and skid-steer kinematics to automatically generate wheeled robot motion controllers. Our robot parametrization (i.e. number of wheels, robot dimensions L_1, L_2 and wheel radius r) is included in the URDF files using Xacro, which are used by the package to generate effective low level controllers that can be used by other high level existing controllers in ROS like the *Navigation* stack.

4.3.2. Parametric controller for snake robots

Our automatic designer includes a base configuration of snake robots in which the modules are chained in two different orientations (vertical and horizontal) shifted by 90º. The number M of modules is the parameter that the automatic designer will adjust depending on user specifications (i.e. type of movement required to the snake robot). In order to generate statically stable gaits [42], the designer, as shown in Figure 14, will generate snakes with $M = 3$ for rolling, $M = 5$ for rectilinear locomotion, $M = 6$ for sidewinding, $M = 8$ for rotating and $M = 14$ for circular paths. Figure 15 shows a real snake robot with $M = 6$ (i.e. it can roll, move forward and in sideways).

We have written a ROS motion controller which uses waves generators [43] for vertical $\varphi_{v_i}(\phi)$ and horizontal modules (see Eq. 7) to produce the different movements . As is seen in Eq. 8,9, the design parameter M is included. The rest of parameters are tuned according to the desired gait in such a way that the conditions of stability are satisfied [44]. Same as with other robots, the kinematic model (which includes parameter M) is represented as a URDF file using Xacro.

$$\begin{aligned} \varphi_{v_i}(\phi) &= A_v sin\,(\phi + (i-1)\Delta\phi_v) \\ \varphi_{h_i}(\phi) &= A_h sin\,(\phi + (i-1)\Delta\phi_h + \Delta\phi_{vh}) \end{aligned} \tag{7}$$

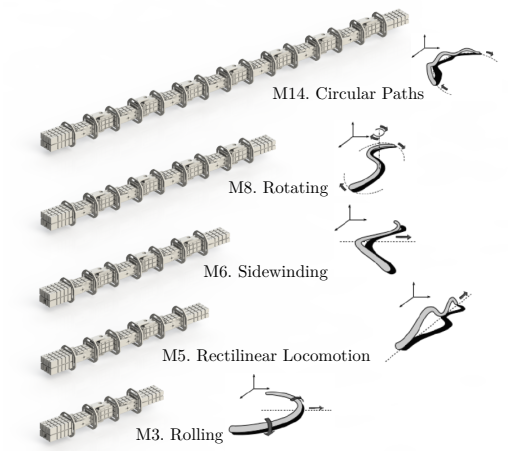


Fig. 14. Different length snake robots generated with our designer.

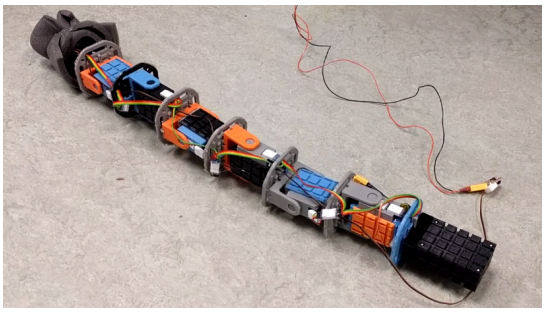


Fig. 15. Snake robot with $M = 6$.

where $i \in \{1, ..., \frac{M}{2}\}$ and

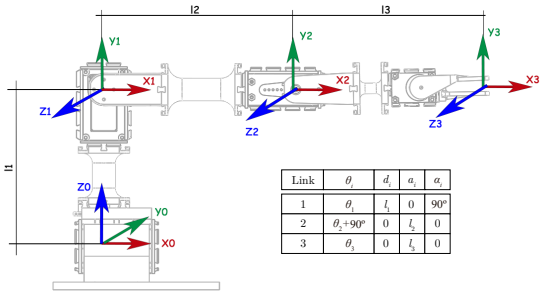
$$A_v = 2\alpha_v sin\left(\frac{2\pi k_v}{M}\right), \quad |\Delta\phi_v| = \frac{4\pi k_v}{M} \tag{8}$$

$$A_h = 2\alpha_h sin\left(\frac{2\pi k_h}{M}\right), \quad |\Delta\phi_h| = \frac{4\pi k_h}{M} \tag{9}$$

Discussion It is proven that snakes with fewer modules can perform similar gaits [44]. However, the motion of these snakes is less smooth than statically stable snakes. In newer versions the automatic designer will determine M using more parameters. For example, the minimum M could be calculated so that a snake can move inside a pipe with a certain diameter.

4.3.3. Parametric controller for robot manipulators

Parameterized robot manipulators generated with our system consist of robots with same kinematic configuration but with different length links and different actuators that fulfill user specifications of workspace



Link	θ_i	d_i	a_i	α_i
1	θ_1	l_1	0	90º
2	θ_2+90º	0	l_2	0
3	θ_3	0	l_3	0

Fig. 16. Kinematic Model for the *3 dof* parametric manipulator

and payload. Figure 16 shows the parametric kinematic model of the *3 dof* base manipulator where the L_1, L_2 and L_3 are numerically evaluated by the automatic designer in order to fulfill user workspace requirement as depicted in Section 4.2. This kinematic model is included in a URDF using Xacro and them converted into a KDL (*Kinematics and Dynamics Library*) model which is finally used by *MoveIt!* ROS tool in order to obtain a robust motion controller of the robot.

The low level control of the robot is carried out through ActionLib [45] which allows the controller to have continuous feedback of sent trajectories.

Discussion The parametric controller for this kind of robots have been successfully tested in the Gazebo simulator [46] and also with three real robots of different size (see Figure 17). These robots and their controllers fulfill user specifications of workspace and payload but other important specifications in industry like accuracy, dynamic response, etc. have not been dealt with in this work.

4.3.4. Parametric controller for hexapod robots

Our automatic designer uses a parametric rectangular hexapod as a base configuration (see Figure 18). Legs links L_1, L_2 and L_3, (see Figure 19), are automatically defined so that the hexapod satisfies user requirements (i.e. a hexapod which can avoid obstacles of height h). As a consequence of the variability of leg size, actuators with different specifications (i.e. torque) will be selected by the designer. Body links L_{a1} and L_{a2} will also change in order to adjust the reachable area for each leg (area free of self-collisions as shown in blue in Figure 18) to the maximum stride length for a stable tripod gait [47]. We have developed a motion controller in ROS that generates tripod, ripple, and wave locomotion for any hexapod obtained from the base configuration. Basically, the kinematic model for each leg (see Figure 19) is modeled in Xacro, con-

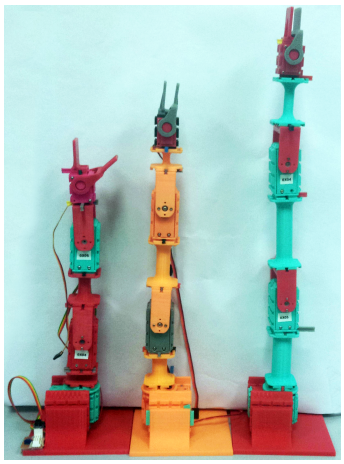


Fig. 17. Real manipulator obtained from three different user specifications.

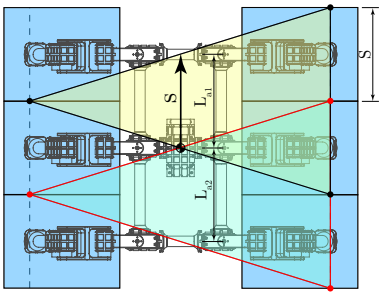
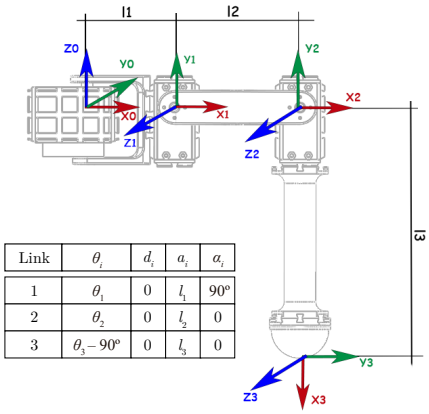


Fig. 18. Hexapod body parametrization according to a stable tripod gait.



Link	θ_i	d_i	a_i	α_i
1	θ_1	0	l_1	90º
2	θ_2	0	l_2	0
3	$\theta_3 - 90º$	0	l_3	0

Fig. 19. Kinematic Model for a *3 dof* hexapod leg.

verted to URDF and used with the KDL ROS package in order to obtain, via inverse kinematics, the joint trajectories for each locomotion type.

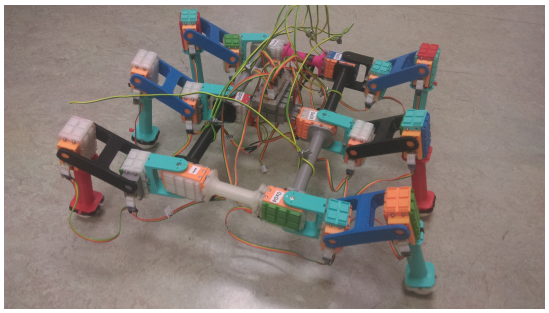


Fig. 20. Real hexapod robot build from the parametric kinematic model of Figure 18.

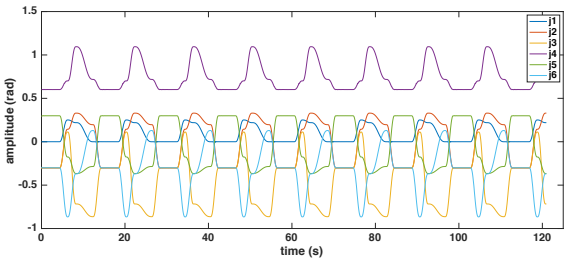
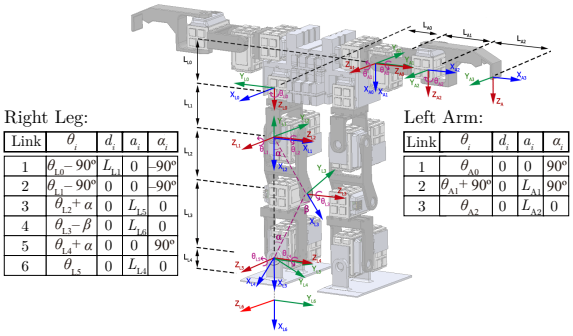


Fig. 22. Joint trajectories of right leg of humanoid during several steps of a stable gait.



Right Leg:

Link	θ_i	d_i	a_i	α_i
1	$\theta_{L0} - 90º$	L_{L1}	0	-90º
2	$\theta_{L1} - 90º$	0	0	-90º
3	$\theta_{L2} + \alpha$	0	L_{L5}	0
4	$\theta_{L3} - \beta$	0	L_{L6}	0
5	$\theta_{L4} + \alpha$	0	0	90º
6	θ_{L5}	0	L_{L4}	0

Left Arm:

Link	θ_i	d_i	a_i	α_i
1	θ_{A0}	0	0	90º
2	$\theta_{A1} + 90º$	0	L_{A1}	90º
3	θ_{A2}	0	L_{A2}	0

Fig. 21. Kinematic model of the base humanoid robot.

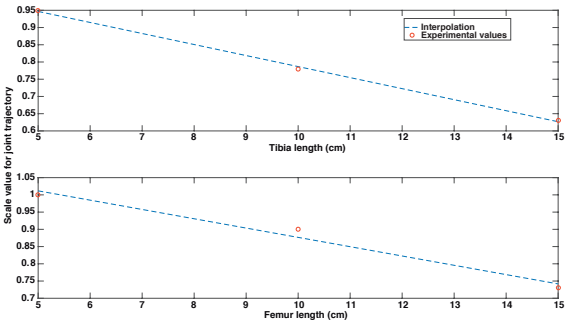


Fig. 23. Conversion maps for humanoid trajectories.

Discussion The motion planner generates statically stable gaits over flat surfaces. This has been proven with different size hexapods in simulation (using the simulator Gazebo) and with the real hexapod of Figure 20. However, this controller is not reactive, i.e. the hexapod will not react to obstacles or rough terrain. This feature will be developed in future works.

4.3.5. Parametric controller for humanoid robots

For a humanoid robot, length of passive modules (e.g. femur and tibia) can be parameterized . The kinematic chain of these robots is the most complex as is presented in Figure 21, but as well as previous robots, this kinematic model is translated, using URDF via Xacro into a KDL model for generating the joint references (i.e. inverse kinematics) from the desired movement path.

The process for the controller generation is as follows:

1. A stable gait already defined for a specific robot (i.e. Darwin-OP [48]), has been used to obtain a set of joint trajectories that serve as a basis for our humanoid. Figure 22 shows these trajectories for the right leg.
2. Then, FFT has been applied to these trajectories to obtain main frequency components for each joint maneuver. These have been used to create a new set of parameterized joint trajectories, which we will call the synthetic locomotion.
3. Conversion maps that determine the relation between size of leg's passive modules and synthetic locomotion parameters have been empirically calculated. For instance, it has been stated that, if joints 2 and 5 are scaled by a factor, body tilt can be changed to maintain stability, while the step size can be modified scaling trajectories of joints 3, 4 and 6. Figure 23 shows these maps, which happen to be almost linear.
4. Finally, joint trajectories for the concrete robot are calculated scaling the synthetic locomotion of Step 2 by factors given by conversion maps of Step 3. As an example, Figure 24 shows joint 2

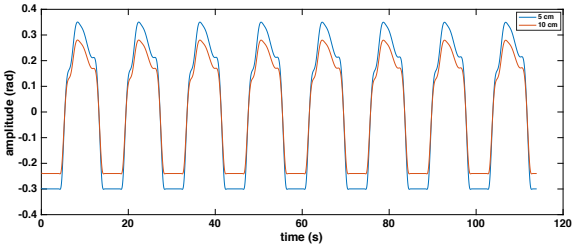


Fig. 24. Trajectories for leg joint 2 of humanoid robot of Figure 21.

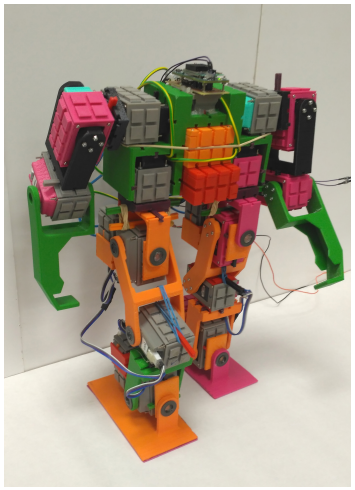


Fig. 25. Real Humanoid based on a parametric design.

trajectories for the robots in Figure 21, which are obtained from:

$$A_j = (-0.0320 \cdot x + 1.1067) \sum_{i=0}^{N} a_i \sin(\omega_i t + \phi_i)$$

where N is the number of frequency components of the synthetic locomotion; a_i, ω_i and ϕ_i are the FFT values obtained in step 2, and x is the length of the passive module (tibia) in centimeters.

Discussion As well as for the hexapods, motion controllers for different sized humanoids have been tested in the Gazebo simulator and with the real robot of Figure 25 obtaining stable walks over flat surfaces. However, these motion controllers do not use dynamics to keep equilibrium (i.e. if the robot is pushed it will fall down). We are currently working with Inertial Measurement Units (IMUs) to develop this kind of reactive controllers.

4.4. Interfacing ROS controllers with Bitbloq

Bitbloq is an open source tool for online visual programming of electronics and robots developed by BQ. It has been designed to be used mainly in education. Bitbloq is similar to Scratch (i.e. they both use blocks) with the difference that the programs generated in Bitbloq run directly in the controllers while in Scratch programs run in the computer.

We have enhanced Bitbloq adding an interface that incorporates robots generated with ParMoR. Basically, we have added the robots included in the base configuration database of the automatic generator to Bibloq. Programming blocks, corresponding to the parametric controls of the robot planners database, for each robot have been also created and made available for Bitbloq users.

If some user wants to use a behavior of a robot (e.g. walking), he or she only has to drag and drop the corresponding blocks. Bitbloq then generates automatically code in Python that subscribes to the ROS controller. It is important to note that robot parameterization does not affect Bitbloq as this is dealt within the ROS controller (e.g. two different sized humanoids auto generated with ParMoR will have the same blocks in Bitbloq).

The following summarizes an example of programming a snake modular robot with Bitbloq.

- The process starts with the automatic generation of the snake and its ROS controllers using ParMoR's methodology explained in this work. The ROS controllers are loaded in the Intel Edison of the Main Module (defined in Section 5.1).
- Then, the user can select in Bitbloq a snake from the list of available robots (see Figure 26).
- Programming blocks for the snake are automatically shown to the user as seen in Figure 27. The user can drag and drop them to *write* a program.
- Bitbloq program is translated to Python and executed as a ROS node.
- This program uses services of the ROS controllers to move the snake as the user commanded.

5. Parametric Modular Robotics platform (ParMoR)

We have developed ParMoR to be used with the automatic designer agent presented in this work. Its prin-

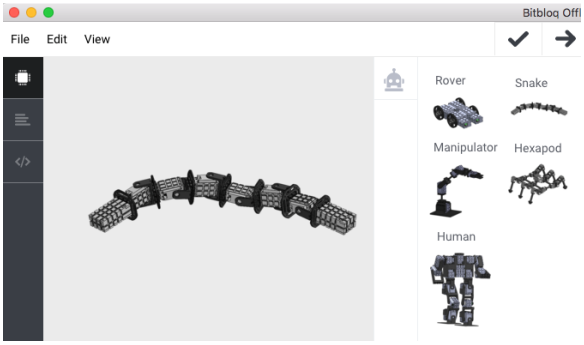


Fig. 26. Snake Robot and list of available parametric ParMoR robots in Bitbloq.

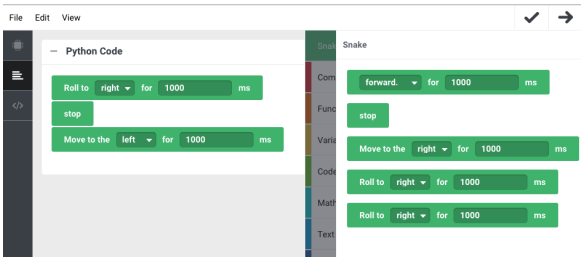


Fig. 27. Example of programming a snake robot with blocks in Bitbloq.

cipal characteristic is its parametric modules which allows the agent to parametrize and optimize designs.

In the following we present the main components of our robotic platform (i.e. active and passive modules)

5.1. Active modules

Active modules (modules with electronic parts) have been designed to be easily encapsulated using 3D printed faces. They include rails for dovetails pins which allow users a simple but robust mechanical joining of modules. Figure 28 shows an example of the design of an active module with the aforementioned faces.

Active modules, with the exception of the main module, include an ATtiny85 microcontroller to control the electronic parts (e.g. actuator, sensor) and to communicate, via I^2C, with the main module. In the following, we summarize the active modules developed for ParMoR to date:

- Main module (1 model) based on a Intel Edison microcomputer with ROS integrated.

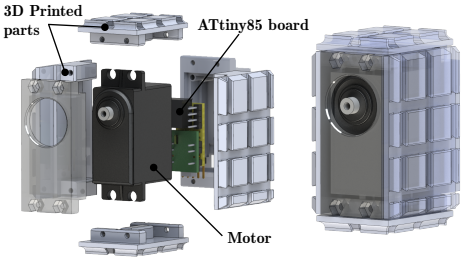


Fig. 28. Assembly of an active module.

- Battery module (1 model) based on a 1500mAh 11.1V Li-Po battery.
- Sensor modules (4 models) for object detection (contact, ultrasonic, infrared) and robot localization (IMU).
- Actuator modules (6 models) for continuous rotation or position control with three different torque models per type (i.e low, medium and high torque).

Figure 28 also shows the electronics (i.e. servo and ATtiny85 board) of the active module used as an example.

As active modules communicate via I^2C, the maximum number of modules of a robot is limited by the address space (2^{10}), and also by the total bus capacitance of 400pF. In fact, we have verified that the maximum number of active modules that can be connected in our bus without exceeding bus capacitance is 24. However this number is big enough as the maximum number of active modules (degrees of freedom) needed for any robot configuration in our system is 19 (i.e. humanoid robot) .

5.2. Parametric passive modules

The passive modules (i.e. mechanical parts with no electronics) of our modular architecture have been designed parametrically. This allows the automatic designer to adjust their geometry to meet user requirements. Figure 10 showed an example of two passive link modules with different geometry obtained from the same base design.

An example of the parametrization of passive modules is shown in Figure 29 where the links of a robotic arm have a different design, after the optimization process explained in Section 4.2. This guarantees adjusting the reach of the arm to a certain length minimizing the amount of material. Other examples are the parametrization of feet which is used to increase stability in a walking robot by increasing the contact surface

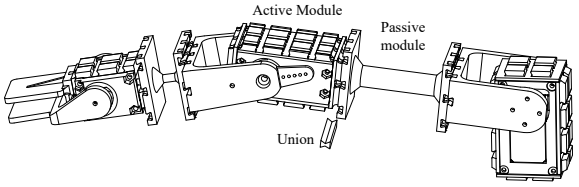


Fig. 29. Modular robot architecture.

with the floor, or wheels, on whose radius relies the maximum speed a wheeled robot can achieve. Modules can also be easily joined using dovetails pins as is depicted in Figure 29.

6. Discussion

In this work we have presented our approach for automatic design, controller generation and easy block programming for complex robots. The modular architecture presented in this work is manufactured using low-cost, fast prototyping techniques (3D printers) using PLA (PolyLactic Acid) as the printing material. This makes modular robots presented in this paper not really suitable for industry applications as the *3D printed* robots have the following issues:

- PLA is not durable enough for repetitive operations.
- PLA is not stiff which introduces backlash and non desirable deformations.
- Servo motors are not reliable for 24/7 operation.

Changing *3D printing* fast prototyping with other classic prototyping like *CNC metal machining* and servomotors with industrial brushless motors should be the first step to introduce our approach to the industry domain. However, our work is not presented as a final product for industry but as a demonstrator of how automatic modular robot design and automatic controller generation can be used to obtain an agile robotic system.

In fact, not all the robots presented in this work have a direct use in industry (i.e. humanoids, snakes or hexapods) but we do believe that they are a good testbed of the capabilities of our approach. The straightforward example of robot configurations that are used in industry are the manipulation robots that our system can generate. Imagine a small canning company that one day decides to automatize the process of palletizing their cylindrical cans. As cylindrical cans are

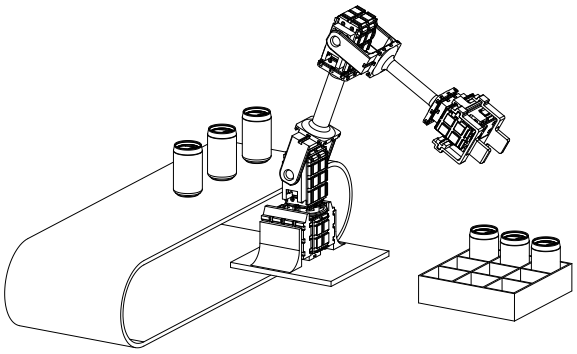


Fig. 30. 3 DOF robot generated by our system for pick&place of cylindrical cans.

symmetric on their longitudinal axis a robot capable for pick and place with just positioning is suitable (as depicted in Figure 30)

The inferred robot corresponds to the following definition in SUO-KIF of *manipulator3DOF* which is defined as a subclass of industrial robot (subclass of stationary robot).

```
(subclass stationaryRobot Robot)
(subclass industrialRobot stationaryRobot)
(subclass manipulator3DOF industrialRobot)
(=>
  (instance ?rob manipulator3DOF)
  (exists (?serv1 ?serv2 ?serv3 ?grip)
    (and
      (instance ?serv1 Servomotor)
      (instance ?serv2 Servomotor)
      (instance ?serv3 Servomotor)
      (instance ?grip roboticGripper)

      (robotPart ?serv1 ?rob)
      (robotPart ?serv2 ?rob)
      (robotPart ?serv3 ?rob)
      (robotPart ?grip ?rob)
      (connectedTo ?serv1 ?serv2)
      (connectedTo ?serv2 ?serv3)
      (connectedTo ?serv3 ?grip))))
```

After some time, the company prospers and decides to produce sardine cans which are oval. As ovals are not symmetric the robot needs now the capability of orienting the gripper. Our approach will generate a new robot as is seen in Figure 31.

The inferred robot in this case corresponds to the following definition in SUO-KIF of *manipulator6DOF* which is defined as a subclass of industrial robot (subclass of stationary robot).

```
(subclass stationaryRobot Robot)
(subclass industrialRobot stationaryRobot)
(subclass manipulator6DOF industrialRobot)
(=>
  (instance ?rob manipulator6DOF)
```

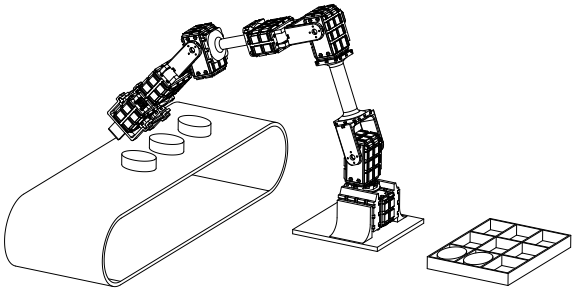


Fig. 31. 6 DOF robot generated by our system for pick&place of oval cans.

```
(exists (?serv1 ?serv2 ?serv3
                ?serv4 ?serv5 ?serv6 ?grip)
  (and
    (instance ?serv1 Servomotor)
    (instance ?serv2 Servomotor)
    (instance ?serv3 Servomotor)
    (instance ?serv4 Servomotor)
    (instance ?serv5 Servomotor)
    (instance ?serv6 Servomotor)
    (instance ?grip roboticGripper)

    (robotPart ?serv1 ?rob)
    (robotPart ?serv2 ?rob)
    (robotPart ?serv3 ?rob)
    (robotPart ?serv4 ?rob)
    (robotPart ?serv5 ?rob)
    (robotPart ?serv6 ?rob)
    (robotPart ?grip ?rob)
    (connectedTo ?serv1 ?serv2)
    (connectedTo ?serv2 ?serv3)
    (connectedTo ?serv3 ?serv4)
    (connectedTo ?serv4 ?serv5)
    (connectedTo ?serv5 ?serv6)
    (connectedTo ?serv6 ?grip))))
```

The company just have to upgrade its robot with the new modules that our system generates, while the new controller and the blocks for an easy programming are as well provided.

Currently, our application lets users configure a manipulator modifying four different parameters: (a) shape of the objects to work with, (b) payload, (c) rough radius of the workspace and (d) degrees of freedom (DOF). With that information we are able to infer the gripper we need (depending on the shape of the object) and the configuration of the final robot. The modular robotic architecture developed by Schunk [11] is able to create a large number of different configurations for industrial manipulators. The differences among the robot instances depend on, in a first classification, two values that are part of the parameters used in our robot design generation: DOF and payload. Therefore, it can be deducted that applying our methodology to an industrial modular platform is possible and, in fact, it is one of the main targets for new developments. Obviously, this extension would require new concepts in the ontology regarding performance measurements, such as repeatability or dexterity, new industry-oriented tasks and new physical concepts such as a variety of end-effectors for different purposes.

Although the previous example is very clarifying, there are other situations where companies, using different robots, could benefit from our approach. For example, there are companies that use snake robots for pipe inspection, search or rescue. Depending on the application, snake robots should have at least a specific configuration and minimum number of *DOF* which our system would determine.

6.1. Limitations

Our approach is far from being ready for an industrial application and even if we just used it for education or research, we would find some limitations. It is the aim of this section to identify some of those limitations and propose how we could manage them in the future.

In the previous sections we have shown how our approach is able to decide the design of a robot automatically from the desires of a user. We have carefully selected the capabilities that a user can choose for a robot to have, so that they can be performed with the resources which are at our disposal (modular architecture and ontology). When writing our knowledge base, we made sure that every possible capability was connected to at least one structural part (e.g. walking and robot leg) and that we could build a feasible robot which was able to perform the expected action. All of this is possible because in our knowledge base we have included instances of robots (including their parts) which can be built with our resources. Note that, though implicitly, all the capabilities of those robots are also defined. Thus, our system will never find a situation in which the proposed design does not fulfill the user desires and nor a design which is not feasible, since our system is totally deterministic. In future versions we will explore how to introduce some randomness through heuristics.

Due to the lack of randomness in our process, we have not implemented a way of measuring how good our approach is yet, however, we have started searching for some potential metrics. Modularity, Regularity and Hierarchy (MR&H) [49] have been identified by researchers as useful principles for designing complex systems. Note that these characteristics can also

be seen in nature. Even though in [49] those concepts are considered from an evolutionary point of view, they are general and can be applied to any design process. The evaluation of those characteristics can be seen as a metrics of how good a design is. Considering the general definition we can understand better how they fit within our work.

- **Modularity:** The modularity value of a design is a count of the number of structural modules in it.
- **Regularity:** Amount of reuse of the modules used in the design process.
- **Hierarchy:** Number of nested layers of modules.

These metrics, while not yet assessed in our platform, are clearly aligned with our work, as we are using a modular architecture with reusable modules and several hierarchy levels ($Robot_C \rightarrow robotPart_C \rightarrow StructuralRobotPart \rightarrow Module$).

7. Conclusions

This work presents our comprehensive approach for agile design, control and programming of a novel modular robotic platform designed to be used in different domains (i.e. educational and research) and as a proof of concept for the industry domain.

An extension to the ORA Standard, ADROn, has been developed, providing new concepts that relate robot capabilities and structural robot parts or robot types. A knowledge system, consisting of ADROn and an existent reasoner, processes the user requirements (desired robot capabilities) to select the most adequate robot configuration. ParMoR, a collection of 3D printable, modular components, serves as a testbed to the structure and controller designers for the robots selected by the knowledge system. This is a co-design problem that has been solved with parameterized designs, both for structure and controllers, for the robots available in ADROn. The controllers have been encapsulated with the visual programming language Bitbloq, abstracting the low-level control execution, performed in ROS, from inexperienced users.

This end-to-end process conceals the complexity of designing robots and their controllers from novice users, such as non-engineering personnel or even high-school students, allowing them to develop and program a large number of complex robots of a certain number of base configurations.

As discussed, there still exists a considerable gap for its application in industrial environments, but the design process described in this article is generic and the tools provided are scalable. Therefore, it could be extended to other domains of robotics with the appropriate extensions of the ontology and the adequate modular platforms.

References

[1] Bauml B, Hirzinger G. Agile Robot Development (aRD): A Pragmatic Approach to Robotic Software. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems; 2006. p. 3741–3748.

[2] Chen IM. Rapid response manufacturing through a rapidly reconfigurable robotic workcell. Robotics and Computer-Integrated Manufacturing. 2001;17(3):199 – 213. Available from: http://www.sciencedirect.com/science/article/pii/S0736584500000284.

[3] Kruse D, Radke RJ, Wen JT. Collaborative human-robot manipulation of highly deformable materials. In: 2015 IEEE International Conference on Robotics and Automation (ICRA); 2015. p. 3782–3787.

[4] Cherubini A, Passama R, Crosnier A, Lasnier A, Fraisse P. Collaborative manufacturing with physical human–robot interaction. Robotics and Computer-Integrated Manufacturing. 2016;40:1 – 13. Available from: http://www.sciencedirect.com/science/article/pii/S0736584515301769.

[5] Rethink Robotics;. Accessed: 2017-4-15. http://www.rethinkrobotics.com/.

[6] KUKA LBR iiwa;. Accessed: 2017-4-15. https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa.

[7] ABB YuMi;. Accessed: 2017-4-15. http://new.abb.com/products/robotics/es/robots-industriales/yumi.

[8] Universal Robots UR Series;. Accessed: 2017-4-15. https://www.universal-robots.com/.

[9] Chung WK, Han J, Youm Y, Kim S. Task based design of modular robot manipulator using efficient genetic algorithm. In: Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on. vol. 1. IEEE; 1997. p. 507–512.

[10] Valente A. Reconfigurable industrial robots: A stochastic programming approach for designing and assembling robotic arms. Robotics and Computer-Integrated Manufacturing. 2016;41:115 – 126. Available from: http://www.sciencedirect.com/science/article/pii/S0736584516300928.

[11] Schunk Modular Robotic System;. Accessed: 2016-09-1. https://de.schunk.com/de_en/gripping-systems/category/gripping-systems/handling/modular-and-mobile-gripping-systems/schunk-lightweight-arms/.

[12] Kelmar L, Khosla PK. Automatic generation of kinematics for a reconfigurable modular manipulator system. In: Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on. IEEE; 1988. p. 663–668.

[13] Paredis CJ, Khosla PK. Kinematic design of serial link manipulators from task specifications. The International Journal of Robotics Research. 1993;12(3):274–287.

[14] Jonquet C, Musen MA, Shah NH. Building a biomedical ontology recommender web service. Journal of biomedical semantics. 2010;1(1):S1.

[15] Tenorth M, Beetz M. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. International Journal of Robotics Research. 2013 April;32(5):566 – 590. Available from: http://ijr.sagepub.com/content/32/5/566.short.

[16] IEEE Standard Ontologies for Robotics and Automation (IEEE Std 1872-2015); 2015. Available from: http://ieeexplore.ieee.org/document/7084073/.

[17] Guarino N, Oberle D, Staab S. In: Staab S, Studer R, editors. What Is an Ontology? Berlin, Heidelberg: Springer Berlin Heidelberg; 2009. p. 1—17.

[18] Censi A. Monotone Co-Design Problems; or, everything is the same. In: American Control Conference (ACC), 2016. IEEE; 2016. p. 1227–1234.

[19] Berners-Lee T, Hendler J, Lassila O, et al. The semantic web. Scientific american. 2001;284(5):28–37.

[20] Niles I, Pease A. Towards a standard upper ontology. In: Proceedings of the international conference on Formal Ontology in Information Systems. ACM; 2001. p. 2–9.

[21] Cellucci D, MacCurdy R, Lipson H, Risi S. 1D Printing of Recyclable Robots. IEEE Robotics and Automation Letters. 2017 Oct;2(4):1964–1971. Available from: http://dx.doi.org/10.1109/LRA.2017.2716418.

[22] Takacs A, Eigner G, Kovacs L, Rudas IJ, Haidegger T. Teacher's Kit: Development, Usability, and Communities of Modular Robotic Kits for Classroom Education. IEEE Robotics Automation Magazine. 2016 June;23(2):30–39.

[23] Sprowitz A, Moeckel R, Vespignani M, Bonardi S, Ijspeert AJ. Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot. Robotics and Autonomous Systems. 2014;62:1016–1033.

[24] Ahmadzadeh H, Masehian E, Asadpour M. Modular Robotic Systems: Characteristics and Applications. Journal of Intelligent & Robotic Systems. 2016;81(3):317–357. Available from: http://dx.doi.org/10.1007/s10846-015-0237-8.

[25] Schweikardt E, Gross MD. Learning About Complexity with Modular Robots. In: Proceedings of the 2008 Second IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning. DIGITEL '08. Washington, DC, USA: IEEE Computer Society; 2008. p. 116–123. Available from: http://dx.doi.org/10.1109/DIGITEL.2008.49.

[26] Pacheco M, Fogh R, Lund H, Christensen D. Fable: A Modular Robot for Students, Makers and Researchers. In: Proceedings of IROS 2014 Workshop on Modular and Swarm Systems; 2014. .

[27] Digumarti KM, Gehring C, Coros S, Hwangbo J, Siegwart R. Concurrent Optimization of Mechanical Design and Locomotion Control of a Legged Robot. In: Climbing and Walking Robots (CLAWAR); 2014. .

[28] Spielberg A, Araki B, Sung CR, Tedrake R, Rus D. Functional co-optimization of articulated robots. In: 2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017; 2017. p. 5035–5042. Available from: https://doi.org/10.1109/ICRA.2017.7989587.

[29] Jing G, Tosun T, Yim M, Kress-Gazit H. An End-To-End System for Accomplishing Tasks with Modular Robots. In: Proceedings of Robotics: Science and Systems. AnnArbor, Michigan; 2016. .

[30] Huckaby JO. Knowledge transfer in robot manipulation tasks [PhD]. Georgia Institute of Technology. Georgia; 2014.

[31] Lipson H, Pollack JB. Automatic design and manufacture of robotic lifeforms. Nature. 2000;406(6799):974–978.

[32] Zhu Z, Xiao J, Li J, Wang F, Zhang Q. Global path planning of wheeled robots using multi-objective memetic algorithms. Integrated Computer-Aided Engineering. 2015;22(4):387–404. Available from: https://doi.org/10.3233/ICA-150498.

[33] BitBloq;. Accessed: 2017-2-28. http://bitbloq.bq.com/#/.

[34] Pease A. Standard Upper Ontology Knowledge Interchange Format; 2009. Web document, http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/sigma/suo-kif.pdf.

[35] Jorge VA, Rey VF, Maffei R, Fiorini SR, Carbonera JL, Branchi F, et al. Exploring the IEEE ontology for robotics and automation for heterogeneous agent interaction. Robotics and Computer-Integrated Manufacturing. 2015;33:12–20.

[36] Fiorini SR, Carbonera JL, Gonçalves P, Jorge VAM, Rey VF, Haidegger T, et al. Extensions to the core ontology for robotics and automation. Robotics and Computer-Integrated Manufacturing. 2015;33:3–11. Special Issue on Knowledge Driven Robotics and Manufacturing.

[37] Bayat B, Bermejo-Alonso J, Carbonera JL, Facchinetti T, Fiorini S, Goncalves P, et al. Requirements for building an ontology for autonomous robots. Industrial Robot: An International Journal. 2016;43(5):469–480.

[38] Fiorini SR, Bermejo-Alonso J, Gonçalves PJS, de Freitas EP, Olivares-Alarcos A, Olszewska JI, et al. A Suite of Ontologies for Robotics and Automation [Industrial Activities]. IEEE Robot Automat Mag. 2017;24(1):8–11. Available from: https://doi.org/10.1109/MRA.2016.2645444.

[39] Pease A. The sigma ontology development environment. In: Working Notes of the IJCAI-2003 Workshop on Ontology and Distributed Systems. vol. 71; 2003. .

[40] Siciliano B, Khatib O. Springer handbook of robotics. Springer; 2016.

[41] Pedregal P. Introduction to Optimization. Springer-Verlag New York; 2004.

[42] González-Gómez J, Zhang H, Boemo E. Locomotion principles of 1D topology pitch and pitch-yaw-connecting modular robots. In: Bioinspiration and Robotics Walking and Climbing Robots. InTech; 2007. .

[43] González Gómez J, Zhang H, Boemo EI, Zhang J. Locomotion capabilities of a modular robot with eight pitch-yaw-connecting modules. In: 9th International Conference on Climbing and Walking Robots, CLAWAR; 2006. .

[44] Gómez JG. Modular Robotics and Locomotion: Application to Limbless Robot [phdthesis]. Universidad Autonoma de Madrid; 2008.

[45] Robot Operating System;. Accessed: 2016-09-1. http://www.ros.org/.

[46] Koenig N, Howard A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. vol. 3. IEEE; 2004. p. 2149–2154.

[47] Lee TT, Liao CM, Chen TK. On the stability properties of hexapod tripod gait. IEEE Journal on Robotics and Automation. 1988;4(4):427–434.

[48] Ha I, Tamura Y, Asama H. Development of open platform humanoid robot DARwIn-OP. Advanced Robotics. 2013;27(3):223–232. Available from: http://dx.doi.org/10.1080/01691864.2012.754079.

[49] Hornby G. Toward the Computer-Automated Design of Sophisticated Systems by Enabling Structural Organization. In: Symposium on Complex System Engineering'07. Citeseer; 2007. .