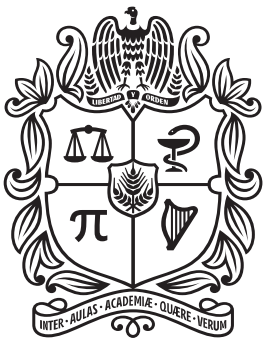
A Modular Robot Architecture Capable of Learning to Move and Be Automatically Reconfigured

Rodrigo Moreno Garcia
M.Sc. in Industrial Automation



Universidad Nacional de Colombia
School of Engineering
Systems and Industrial Engineering
Bogota, D.C.
March, 2019

A Modular Robot Architecture Capable of Learning to Move and Be Automatically Reconfigured

Rodrigo Moreno Garcia
M.Sc. in Industrial Automation

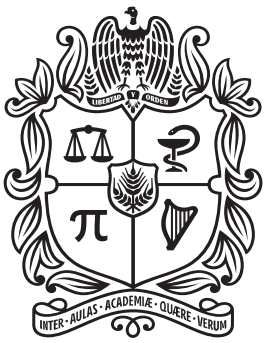
A dissertation submitted in partial fulfillment of the requirements for the degree of
Ph.D. in Computer and Systems Engineering

Advisor
Jonatan Gomez Perdomo
Ph.D. in Computer Science

Coadvisor
Andres Faina Rodriguez-Vila
Ph.D. in Industrial Engineering

Research line
Modular Robots

Research group
Research Group on Artificial Life



Universidad Nacional de Colombia
School of Engineering
Systems and Industrial Engineering
Bogota, D.C.
March, 2019

Title in English

A Modular Robot Architecture Capable of Learning to Move and Be Automatically Reconfigured

Título en español

Arquitectura de Robot Modular Capaz de Aprender a Moverse y Ser Automáticamente Reconfigurada

Abstract: Tackling the problem of making a modular robot automatically learn the movements necessary to locomote in different environments is not an easy task. The ability of modular robots to have an arbitrary morphology provides an advantage over usual monolithic robots when moving in different environments. However, being able to reconfigure also has its problems. Movement control for reconfigurable robots is difficult to design and implement. Morphology can also influence the sensing capabilities of a modular robot. Only a few studies include sensor information when adjusting or optimizing controllers for modular robots. The main contribution of this work is the development of an architecture that includes a locomotion training framework that enables a modular robot to move in different environments taking into account sensor information. The framework is composed of four main parts: a control strategy, a configurable environment approach, an adaptation mechanism and a new modular robot platform: the EMERGE modular robot. The EMERGE modular robot platform is designed to be easy to be assembled and can be quickly reconfigured thanks to the magnetic connectors present in its modules. This in turn enables an external agent, like a robot manipulator to reconfigure the robot. Results show that well coordinated movements turn out to be very important for controllers using sensors to improve when being adapted. The mechanisms inside the controller, for example, decision structures, also play a major part in allowing a robot to adapt to move in different environments and be improved. Evaluating robots in reality is a very expensive task and differences between simulation and reality also make robots behave very differently. The magnetic connector makes the assembly of an EMERGE morphology easier but hinders the disassembly process.

Resumen: Resolver el problema de hacer, de forma automática, que un robot modular se mueva en diferentes ambientes no es tarea fácil. La habilidad de los robots modulares de tener morfología arbitraria provee una ventaja sobre robots monolíticos normales al moverse en diferentes ambientes. Sin embargo, ser capaz de auto reconfigurarse tiene sus propios problemas. El control de movimiento para robots modulares es difícil de diseñar e implementar. La morfología de los robots también influencia la capacidad de percibir de los robots modulares. Solo contados estudios incluyen información sensorial al ajustar u optimizar controladores para este tipo de robots. La mayor contribución de este trabajo es el desarrollo de una arquitectura de robot modular que hace que este

pueda moverse en diferentes ambientes teniendo en cuenta información sensorial. Esta arquitectura está compuesta por cuatro partes principales: una estrategia de control, un modelo de ambiente configurable, un mecanismo de adaptación y una plataforma de robot modular nueva: el robot EMERGE. El robot modular EMERGE, es diseñado para ser fácil de construir y de reconfigurar gracias a sus conectores magnéticos. Esto también posibilita a un agente externo, como un manipulador robótico, a reconfigurar el robot. Los resultados de los experimentos muestran que la buena coordinación del robot es muy importante para que los controles que usan sensores puedan mejorar. Los mecanismos internos del controlador, por ejemplo, las estructuras de decisión también tienen un rol importante al adaptar el robot a diferentes ambientes. Evaluar robots en la realidad es una tarea muy costosa y las diferencias entre la simulación y la realidad hacen que los robots se comporten muy diferente. Los conectores magnéticos hacen que armar las morfologías de módulos de EMERGE sean fáciles de armar, mas no de desarmar.

Keywords: Modular Robots, Coordination, Configurable environments, Sensors, Locomotion, EMERGE, Automatic Reconfiguration

Palabras clave: Robots modulares, Coordinación, Ambientes Configurables, Sensores, Locomoción, EMERGE, Reconfiguración Automática

Dedicated to

All the people that strive to make their dreams work in reality.

Thanks to

- My advisor Jonatan Gomez and my coadvisor Andres Faiña, who helped me and gave me the courage to not give up through this very difficult process.
- Luis Fernando Niño, Jesus Alberto Delgado, Richard Duro and Leonardo Bobadilla, who served as jury and evaluated this thesis.
- My family, my parents and my brother, who never stopped giving me their support, in many different ways, even in the most difficult of challenges.
- Kasper Stoy, the Real Group and the IT University of Copenhagen, who gladly received me and showed me another perspective of the world of research.
- Ernesto Cordoba and the Labfabex (Laboratorio Fábrica Experimental) and students David Silvera, Oscar Gracia and Julian Franco who allowed me to use their robotic manipulator and helped me control it and perform the reconfiguration experiments.
- Frank Veenstra, Ceyue Liu, Camilo Alaguna, Arles Rodriguez, Jorge Contreras, Cristian Martinez, Camilo Cubides and all the Alife Group, and in general to all the friends that have collaborated with me and without which this thesis would not have been possible.
- This project has been possible thanks to the financial support of the DIB Bogota (Dirección de Investigación Bogota) under the program *Convocatoria del Programa Nacional de Proyectos para el Fortalecimiento de la Investigación, la Creación y la Innovación en Posgrados de la Universidad Nacional de Colombia 2013-2015 - Modalidad 3. Proyectos de investigación, creación o innovación en desarrollo*, code 23418.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Moving through different environments is a task that seems trivial when performed by many living beings. In general, the majority of living beings we interact with have bodies that do not change shape, at least in the time scale where locomotion takes place and depending on the species, and are almost always aware of what tho do when they perceive an obstacle. Internally, however, livings beings coordinate different body parts when locomoting. Sensor information travels throughout the body and adjusts this coordination accordingly. Imagine now that instead of a living being, a robot is trying to move through different environments, but this robot can have any shape and any sensor distribution. The robot still has to coordinate all its parts and integrate information from all its sensors.

This is the case of modular robots. A modular robot is a multi-robot system that encapsulates part of its functionality in basic units called modules. By joining modules together, in different ways, different robot morphologies can be built. The ability of modular robots to have an arbitrary morphology provides an advantage over usual monolithic robots when moving in different environments [113].

However, being able to reconfigure also has its problems. The control of movement for reconfigurable robots is difficult to design and implement. For example, a robot in a quadruped configuration moves its limbs in a way that is not very useful when the robot reconfigures into a snake. Past studies in modular robot locomotion have concentrated in designing control strategies that cope with the reconfiguration ability of modular robots [113]. Yet, only few of them include sensor information when adjusting or optimizing controllers. Sensing obstacles plays a very important role in generating and adapting movements for traveling through different environments. This can be observed in the plethora of sensor arrangements and systems that are found in living beings.

Yoneda et al [135] use real valued genetic algorithms to evolve controller parameters for simulated modules attached trough actuated springs to each other. The modules are in a ring configuration and must travel to a light source, that each module can sense

with its own light sensor. Zahadat et al [140] evolve controllers based on Fractal Gene Regulatory Networks (FGRN) that can react to changes in the environment. Fitness is computed based on distance to two individual target goals where the simulated robotic system must arrive to, using locomotion movements. Even if these works consider modular robots with sensors, they do not examine the influence of sensor signals in module coordination and only use one morphology for testing, which is important as different modular robot morphologies can have different sensing capabilities. Rossi et al [94] make this influence explicit by including a term in the control of inter-module joints and also tests in different morphologies. In [94] they evolve sinusoidal controller parameters for different configurations of a modular robot with sensors on one of its modules. However, only one module is fitted with two proximity sensors, called "head" module. Evolution tunes the influence of the "head" module sensors readings in the output of all modules sinusoidal controllers. Furthermore, only Zahadat et al work with realistic modules, and none consider communication among modules.

Besides, several works that involve modular robots learning to locomote, including the ones described, most often than not use simple environments [18, 19, 24, 52, 66, 67, 85, 103, 111, 119, 120, 121, 136, 137, 138, 140], like flat surfaces and isolated obstacles. Few studies use rough terrains, made by randomly adjusting the height of parts of the terrain [28] and only some employ environments with multiple features[123].

This work proposes a locomotion training framework that enables a modular robot to move in different environments taking into account sensor information. The framework is composed of four main parts: First a control strategy, which includes a coordination mechanism that uses communication to coordinate neighboring modules, a sensor information handling mechanism, that aggregates and filters sensor signals coming from all modules, and a decision mechanism which explicitly defines the behavior of the coordination mechanism based on the outputs of the sensor information handling mechanism. Second, a configurable environment approach, which allows the controller to be tested in different environments with different features. Next, an adaptation mechanism to adapt the controller to the different features of the environment. And finally a new modular robot platform: the EMERGE modular robot. The EMERGE modular robot platform is designed to be easily assembled and can be quickly reconfigured thanks to its magnetic connectors.

The locomotion training framework is part of the main purpose of this work: define a modular robot architecture capable of learning to move according to the environment and be automatically reconfigured. In this respect, this dissertation concentrates on the following objectives:

To characterize the different ways modular robots can learn by doing a literature review. The basic concepts of modular robots, their control and learning strategies used are reviewed (Chapter 2). Movement control for reconfigurable robots is difficult to design and implement. Deciding when to use specific movements is not trivial when changing configurations. Most existing automatic controller generation strategies

have only made robot morphologies move in flat surfaces with simple obstacles. Sensor information has also been greatly ignored and only few works study the influence of sensor information in the automatic generation of controllers, in very basic settings.

To define a learning model to enable a modular robot to learn to move by itself in different configurations by using a learning technique. A coordination mechanism using CPG, a way to aggregate and filter sensor information being communicated among modules, as well as an ANN decision mechanism, are defined and used as a basic control strategy for the model. A configurable environment model is introduced as a mechanism to train in different features of structured environments. Evolutionary algorithms are used as an adaptation technique to train the robot controller to move in the different environment features. All this parts comprise a locomotion training framework that enable a modular robot to learn to move. Modular robots using the defined locomotion training framework (Chapter 3) are expected to generalize in environments that possess the same features in different arrangements.

To define an automatic reconfiguration strategy for a specific type of modular robot. Taking advantage of the fast connection feature of the passive magnetic connectors of EMERGE modules, a method for assembling and disassembling modules using an external robot manipulator is proposed as a practical alternative to self-reconfigurable robots and manual reconfiguration systems (Chapter 4). This is specially useful in chain type modular robots like EMERGE, where self-reconfiguration is limited due to kinematic restrictions. For this purpose a force analysis of the magnetic connectors of EMERGE in regard to the reconfiguration process is performed and a proof of concept test is carried out using real industrial manipulator arms.

To test the movement learning model proposed and the automatic reconfiguration strategy in a modular robot hardware with a limited number of DOF. The locomotion training framework is tested in simulation and in reality (Chapter 5). Controllers are evolved in simulation using different approaches that include the use of previously generated seeds with well coordinated movements. Different morphologies built using EMERGE modules are used for these tests. The configurable environment approach is used to measure the generalization ability of the robots, first for controllers without sensors and later for controllers with sensors. The best controllers obtained in simulation are transferred to reality and a small evolution experiment is also performed directly in the real EMERGE modules.

The main contribution of this work is the development and testing of an architecture including a locomotion training framework to enable a modular robot to learn to move in different environments. This framework is used to train controllers capable of integrating sensor information in a distributed way and for which the control model is defined. A configurable model of the environment that can show different features to an adaptation process in a controllable way is also proposed and tested [75]. A new modular robot prototype designed to be easily built and open for anyone to use and modify, the EMERGE modular robot, is also proposed and described in this work [77]. A reconfiguration strategy

for EMERGE modules using an external robotic manipulator is also defined [73]. The remainder of this work is organised as follows:

Chapter 2 provides a look into the literature of modular robots, the most representative prototypes to date, existing control strategies and their problems when used for learning to move in different environments. Chapter 3 introduces the locomotion training framework that enables a modular robot to move in different environments. Each part is explained using a simple modular robot with sensors. Chapter 4 describes the new modular robot platform and the automatic reconfiguration strategy designed. Chapter 5 describes experiments performed to test the capabilities of the locomotion training framework in simulation and in reality. Finally some conclusions and future work are outlined.

CHAPTER 2

Basic Concepts

2.1 Introduction

Search and rescue missions, space exploration, and similar tasks sometimes are performed in situations and environments that are harsh, or even completely inaccessible, to humans. Robots are designed as tools for helping humans to explore such environments. While exploring these kinds of environments, robots encounter different kinds of obstacles. One possible solution to enable robots to avoid obstacles is to use remote control. However, different factors like distance or control signal interference demand robots to have some level of autonomy, ideally working in a completely autonomous way. Different sponsored competitions and challenges have been devised with the idea of developing a completely autonomous robot [47].

The morphology of a robot determines its capacity to explore different environments (to move in different conditions). In other words, a robot with wheels can move in a specific set of environmental conditions, different from those where robot with tracks or limbs can move. Most of the time, a robot is designed without the possibility of changing its form, but including different ways of moving that can tackle different kinds of terrains [12]. A modular robot is a multi-robot system that encapsulates part of its functionality in basic units called modules. Different robot morphologies can be built by joining together a predefined number of modules. The ability of modular robots to have an arbitrary morphology provides an advantage over usual monolithic robots when moving in different environments [113].

For example, a modular robot system can adopt a wheel configuration to move very quickly in flat surfaces [96] and become a quadruped when facing environments containing small obstacles and uneven terrains. Changing the morphology of the system can be achieved by the modules themselves or by an agent external to the modules. However, being able to reconfigure has its problems. Movement control for reconfigurable robots is

difficult to design and implement. For example, a robot in a quadruped configuration uses movements to move its limbs, which are not very useful when the robot rearranges itself into a snake configuration. Control mechanisms that can cope with the reconfiguration ability of modular robots have been designed and tested in several prototypes [113].

Morphology can also influence the sensing capabilities of a modular robot. Sensing obstacles plays a very important role in generating and adapting movements for traveling through different environments. This can be observed in the plethora of sensor arrangements and systems that are found in living beings. In monolithic robots, sensor information usually travels from the sensors to the controllers in order to be used. Since the robot is not designed to change its morphology, the receiving controller can interpret the sensor position without trouble. In reconfigurable modular robots, any sensor information generated by a module can be difficult to handle as its position and orientation can vary from morphology to morphology, thus the spatial meaning of a sensor can change. A mechanism for handling sensor information that is capable of coping with this spatial meaning issue is also required in order to let a robot move through different environmental conditions.

In this chapter, the basic concepts of modular robots are introduced. First, the main features of modular robots are presented, including an analysis of some modular robot prototypes that have been used in locomotion tasks. Second, a short review is presented about the strategies that have been developed for controlling modular robots, which are the base for the control strategy proposed in this work. Finally, some control techniques for solving the problem of automatically generating controllers are also presented.

2.2 Modular Robots

Modular robots have an advantage over monolithic robots when traveling through different environments, since they are able to change their morphology to avoid obstacles, move in uneven terrains or simply enter previously inaccessible places. Modular robots are mechanically connected compositions of autonomous devices, called modules, which encapsulate part of their functionality [113]. Identical modules are easy to produce and can be assembled in various configurations leading to different robot morphologies (Figure 2.1). These robot morphologies are bigger and can achieve more complex tasks than individual modules [49]. Depending on the specific system, modules are likely to have their own processor units, sensors, actuators and means of communication with other modules.

Initial modular robots research was concentrated on developing the mechanical components of its modules. Fukada and Kawauchi, for example, developed a cellular distributed robot (CEBOT [29]) inspired on rapid CNC tool interchangers in the eighties. In the nineties, Chirikjian [15], Murata [83] and Yim [128] developed modular robot systems based on lattice structures and modules with simple actuators.

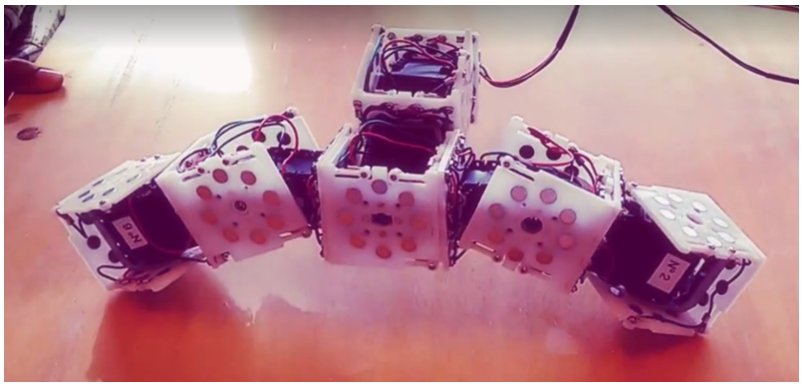


FIGURE 2.1. Modular Robot

Later, research has focused more on developing distributed algorithms for control and dynamic self-discovery of module topologies [69, 100]. Other works concentrated on stochastic assembly and simulation of thousands or even millions of modules. More recently, research about module designs, comprising modules with mobile capabilities[25] have appeared as well as modular robot systems that can perform high level tasks [50].

Modular robots are often designed to have one or more of the following features:

- Re-usability and reconfigurability: Re-usability is achieved by means of modularity [70]. It is cheaper to use the same materials by reconfiguring similar parts, rather than making a completely new solution to the same task.
- Self-reconfiguration and self-assembly: While manual assembly and disassembly have been used extensively [68, 28] to reconfigure modules, it requires an operator, which reduces autonomy. A self-reconfigurable modular robot can change its modules assembly configuration by itself [81].
- Scalability: Scalability is often difficult in mechanical systems but can be easily achieved by using modules [81]. Modular robot systems are, in theory, capable of adding up functionality by increasing the number of modules.
- Robustness and Reliability: Modular self-reconfigurable robots could react to failure in one or more of its modules by rearranging the whole structure or simply by disconnecting the malfunctioning module from the others [98]. Robots could also replace the malfunctioning module with a new one [27]. Modular robots that are reliable and robust are, however, still far from being implemented pratically in real hardware.

2.2.1 Types of Modular Robots

Modular robots can be either homogeneous (all their modules share the same design and features) or heterogeneous (there are different types of modules with different actuation

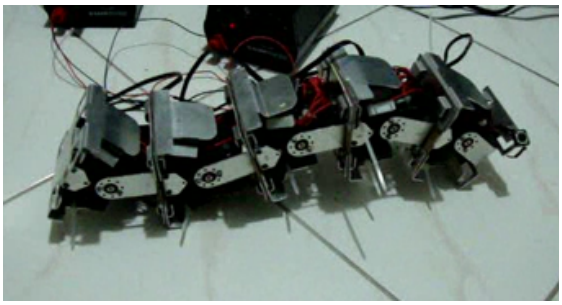
and sensing capabilities) [81]. A more meaningful way of classifying modular robots is based on their reconfiguration capabilities. There are three types of reconfiguration in modular robot systems: Lattice-based and chain-based, as established by Yim et al. [131], and mobile based (Figure 2.2).

- Chain-type: Chain-type modular robots can form, as the name implies, chain like structures when attached to each other. Chains of modules can attach to other chains in relatively arbitrary positions depending on the connector type. Limbed configurations, like, arms, legs, or tentacles, formed by modules, can be used for locomotion and manipulation tasks. Self-reconfiguration is difficult in chain based modular robots due to the arbitrary position of the connections that can be formed [81], and the many DOFs (Degrees of Freedom) that the robot system can potentially have. Chain-type modular robots have a good power-weight ratio with few actuators. Some chain type modular robots have only one degree of freedom per module, which does not stop them from reaching any point in space [33], for example, by bending themselves (Figure 2.2a).

- Lattice-type: Latticed-based modular robots are capable of aligning themselves into periodic, and almost always symmetric configurations. Lattice based modular robots are usually self-reconfigurable due to the periodic nature of the positions that modules can take, i.e. every step used to go from a given configuration to another is already well defined [131]. Each module is concerned only with the nearest position along the lattice rather than with all the arbitrary positions a chain-based module can achieve, thus better abstractions can be used than with chain-based modules [131]. Lattice restricted movements are also a disadvantage, because modules cannot reach anything outside of the lattice structure (Figure 2.2b).

- Mobile-type: Mobile-based modular robots reconfigure by freely moving their modules out of a given configuration to another in contrast with chain and lattice-based robots, which never disconnect from the main structure to perform a movement. Modules can move by means of threads or wheels [50] (Figure 2.2c).

Some prototypes, like SMORES [25], explore the possibilities of mechanical designs that merge the features of the three types of reconfiguration: lattice, chain and mobile (Section 2.2.2.6). Different types of modular robots can move in different ways. Pure lattice-type robots can move by reconfiguring their modules, often called *flow*, but they must solve first the self-reconfiguration problem. Self-reconfiguration in chain-type modular robots comes with the problem of connecting and disconnecting modules from the structure and aligning them. However, traveling through different terrains is easier in chain-type robots than in lattice-type. The former ones do not have to self-reconfigure in order to move, i.e. chain-type modular robots form limbed configurations with high power-weight ratio, that move by using their main actuators.



(A)

(B)

(C)

FIGURE 2.2. Types of modular robots: (a) Chain Type,(b) Lattice Type, (c) Mobile type

2.2.2 Modular Robot Prototypes

Numerous modular robotic systems have been developed over the last decades. Table 2.1 shows some of the modular robot prototypes that have been developed around the world. As mentioned before, the morphology of a module directly influences its ability to locomote while avoiding obstacles. The achievable morphologies for a specific modular robot system are determined by the module (or module types in the case of heterogeneous robots) and connector designs. The shape and complexity of a module also limits the kind and number of sensors that can be placed in it. Different types of connectors enable different types of reconfiguration and how fast they can be performed. The module and connector designs can be as elaborated as required, for example to be able to achieve a high number of morphologies or to self-reconfigure, but this can also make their assembly more difficult. This subsection summarizes some of the most representative ones. For a more detailed overview check [113].

2.2.2.1 Polybot

Polybot modules are built as 1DOF hinges with a rotational actuator at their center. Connectors are placed in opposing faces parallel to the hinge actuator and in some versions on the faces supporting the rotational actuator (Figure 2.3 Polybot G1 and G3). Chain morphologies with only one chain or chains and perpendicular bifurcations are possible with this type of module, depending on the specific version of the design currently, five versions of the module design exist (Figure 2.3) [27]. Additionally, wheel, worm, and legged robots have been built using Polybot modules. Due to its hinge shape, sensors are placed inside the module, in the case of orientation and force sensors, and on the connector faces that are parallel to the rotational actuator (infrared docking aid). Versions G2 and G3 have connectors that let them self-reconfigure based on pin-hole mechanisms and Shape Memory Alloy (SMA) wires. Module construction varies from version to version, being the self-reconfigurable ones the most complex to build since they possess small actuated mechanisms and specialized parts (Figure 2.3 Polybot G3).

2.2.2.2 M-TRAN

M-TRAN is a modular self-reconfigurable robot developed by the National Institute of Advanced Industrial Science and Technology (AIST) of Japan and the Technological Institute of Tokyo (Tokyo-Tech). Each module is made using two semi cylindrical, semi cubical parts with connectors on 5 of their flat faces. One of the semi-cubical parts has male connectors while the other has female connectors. Male connectors contain hooks that can be automatically extended or retracted providing a mechanical link with female modules that enables self-reconfiguration. A stick connects the two semi-cubical parts extending what was previously a simple hinge into two (Figure 2.4b) [84]. Each end of the central stick contains a rotational actuator that makes the semi cubical parts rotate

TABLE 2.1. Modular robot Prototypes around the world. (Authors inside table)

Name	**Class**	**DOF**	**Author**	**Affiliation**	**Year**
CEBOT	Mobile	Various	Fukuda et al.	Nagoya	1988
Polypod	Chain	2-3D	Yim	Stamford	1993
Metamorphic	Lattice	3-2D	Chirikjian	JHU	1993
Fracta	Lattice	3-2D	Murata	MEL	1994
Tetrobot	Chain	1-3D	Hamlin et al.	RPI	1996
3D Fracta	Lattice	6-3D	Murata et al.	MEL	1998
Molecule	Lattice	4-3D	Kotay & Rus	Dartmouth	1998
CONRO	Chain	2-3D	Will & Shen	USC/ISI	1998
PolyBot	Chain	1-3D	Yim et al.	PARC	1998
TeleCube	Lattice	6-3D	Suh et al.	PARC	1998
Vertical	Lattice	1-2D	Hosakawa et al.	Riken	1998
Cristal	Lattice	4-2D	Vona & Rus	Dartmouth	1999
I-Cube	Lattice	1-3D	Unsal	CMU	1999
Pneumatic	Lattice	1-2D	Inoue et al.	TiTech	2002
UniRover	Mobile	2-2D	Hirose et al.	TiTech	2002
M-TRAN	Hybrid	2-3D	Murata et al.	AIST	2002
ATRON	Lattice	1-3D	Stoy et al.	U.S Denmark	2003
Swarm-Bot	Mobile	3-2D	Mondada et al.	EPFL	2003
Stochastic 2D	Mobile	0-2D	White et al.	Cornell U.	2004
SuperBot	Hybrid	3-3D	Shen et al.	USC/ISI	2005
Stochastic 3D	Mobile	0-3D	White et al.	Cornell U.	2005
Catom	Lattice	0-2D	Goldstein et al.	CMU	2005
Prog. Parts	Mobile	0-2D	Klavins	U. Washington	2005
Molecube	Chain	1-3D	Zykov et al.	Cornell U.	2005
YaMoR	Chain	1-2D	Ijspeert et al.	EPFL	2005
Miche	Lattice	0-3D	Rus et al.	MIT	2006
JL-I	Mobile	3-2D	Zhang et al	U. Hamburg	2006
Shady3D	Chain	3-3D	Yoon et al	MIT	2006
EM-Cube	Mobile	0-2D	Byoung	Dran Lab.	2007
Evolve	Chain	2-3D	Chang et al.	NUS	2008
Morpho	Lattice	1-3D	Chin-Han et al.	Harvard/MIT	2008
ODIN	Lattice	1-3D	Lyder et al.	Moller Inst.	2008
Roombots	Lattice	1-3D	Sproewitz	U. Lausane	2008
Beanbag Robotics	Mobile	1-2D	Kriestel et al.	Cornell U.	2008
Locokit	Hybrid	1-3D	Larsen et al	Moller Inst	2010
L-shaped	Hybrid	3-3D	Kutzer et al	Johns Hopkins U.	2010
Cross-Ball	Hybrid	2-3D	Meng et al	Stevens IT	2011
U-Bot	Chain	2-3D	Zhao et al	Science Garden of HIT	2011
SMORES	Hybrid	4-3D	Davey et al	U. New South Wales	2012
MICROTUB	Chain	1-3D	Brunete et al	U. Carlos III	2012
EDHMOR	Chain	Hetero	Faina et al	Universidade da Coruña	2013
REPLICATOR	Chain	Hetero	Liedke et al	EU Project	2013
Fable	Chain	Hetero	Pacheco et al	DTU	2013

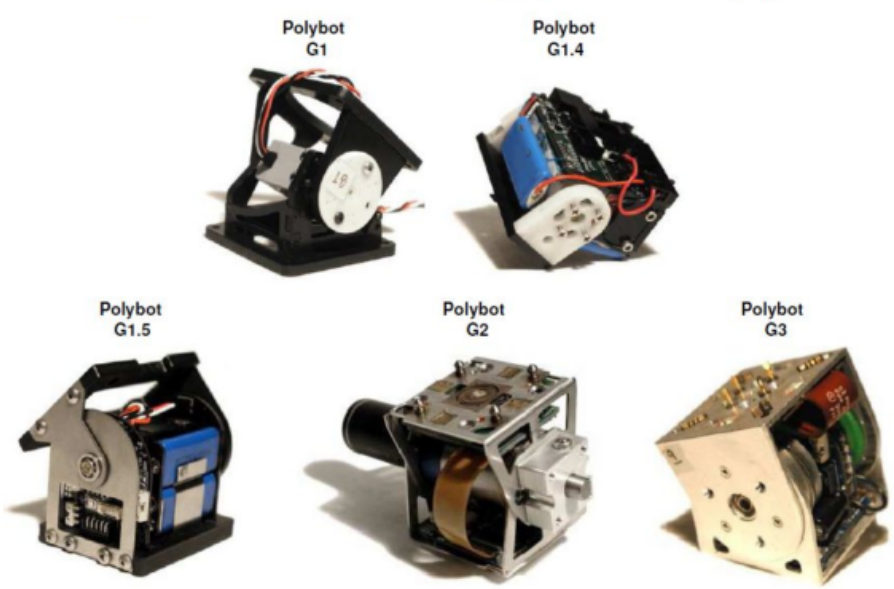
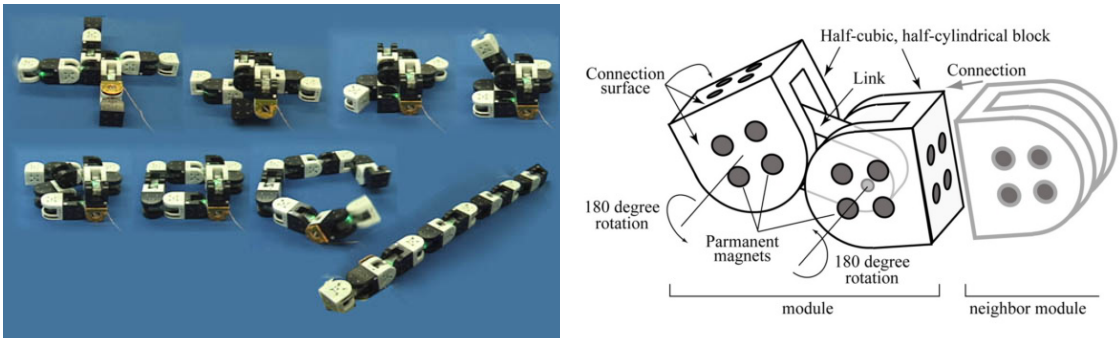


FIGURE 2.3. Different versions of Polybot (Taken from [88])



(A) (B)

FIGURE 2.4. M-TRAN: reconfiguring (a) and module (b) (Taken from [1])

about the stick. The module shape lets M-TRAN modules form chain type and lattice type structures, because each semi cubical part can fold itself by 90 degrees around the central stick. Limbed and snake configurations, as well as wheels, that can turn into each other by self-reconfiguration, have been tested (Figure 2.4a) [54]. Cameras and other types of sensors are placed in special modules that are attached to a robot configuration. The modules themselves only include orientation and joint position sensors. Although the advantages that the module design offers its shape requires specialized DC motors and gears in order to move joints and connectors that make assembly more difficult. Mechanical hooks also make the reconfiguration process slower and more energy inefficient than with other types of connectors.

FIGURE 2.5. ATRON modules (Taken from [20])

2.2.2.3 ATRON

Developed by the University of Southern Denmark, the ATRON modular robot is a self-reconfigurable robot with semi-spherical shape [10]. The module is composed of two hemispheres connected by a rotational joint (Figure 2.5). The hemispheres can rotate relative to each other and can act as wheels thanks to a slip ring located in the center of the module. Modules have automatic mechanical male connectors based on hooks that can latch to female connectors in other modules allowing the connection of up to 8 modules to the same base. ATRON modules semi-spherical shape enables tightly packed lattice connections that can move by self-reconfiguration. The main rotational actuator also allows ATRON modules to move like chains. Snakes, cars and other similar configurations have been achieved. Each module houses a tilt sensor as well as infrared proximity sensors at the connectors to help align modules during self-reconfiguration. The slip ring enables a different kind of main actuator and mechanical design for a modular robot, compared to other prototypes like Polybot, but it also complicates the construction of the module as very specialized parts should be used to maintain electrical connection between hemispheres. Although mechanical connectors are strong [86], they can make the self-reconfiguration process slow compared to other kinds of connector mechanisms.

2.2.2.4 Superbot

Superbot is a robot built in the Polyphormic Robotics Laboratory of the Information Science Institute of the University of South California by Salemi et al [96]. The basic structure of a Superbot module has two semi cubical parts joined together by a central link, in the same fashion as M-TRAN modules. The main difference is an extra actuated rotational joint located in the center of the link, that lets the two semi cubical parts rotate in the same axis of the central link (Figure 2.6) [101]. This extra DOF enables Superbot modules

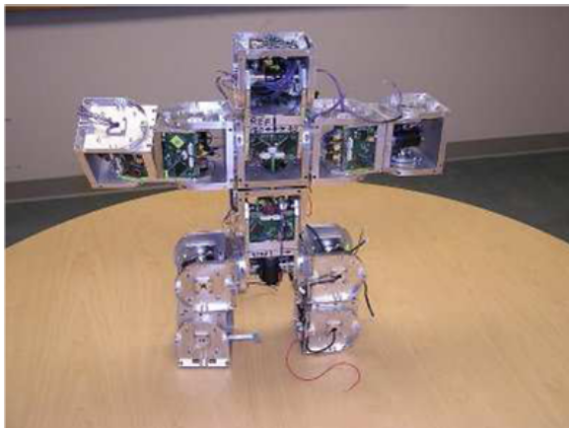


FIGURE 2.6. SuperBot in biped configuration (Taken from [58])

to form configurations similar to those formed by M-TRAN modules and CONRO modules [14]. Superbot modules have mechanical connectors in faces corresponding to those of the M-TRAN module. Reported sensors include an onboard 3D accelerometer/inclinometer sensor [96] and an externally mounted camera [91]. The extra DOF has the disadvantage of adding more parts (actuators, gearboxes) to the overall design. Superbot also has the disadvantage that its connector faces need to be secured with screws to other modules making the reconfiguration process slow.

2.2.2.5 REPLICATOR

The REPLICATOR project is an EU funded project with the aim of developing a super-large-scale swarm of autonomous mobile micro-robots that can self-assemble into large artificial organisms. REPLICATOR modules are heterogeneous and three different types exist (Figure 2.7). All of them are capable of moving by themselves: A backbone module and a scout module, both with morphology similar to Polybot modules. The backbone module has a powerful rotational actuator that can lift several other modules, has actuated connectors and can move sideways in the plane thanks to additional actuators. The scout module specializes in fast locomotion using tracks and houses proximity sensors to scout its surroundings [62]. An active wheel module is also available to carry electronics and battery packs [90]. Chain-type morphologies (worms, limbed) are possible with the first two types of modules, and can be extended by using the active wheel type of module. Backbone modules and scout modules pack a lot of functionality in their body, which of course complicates the module assembly.

2.2.2.6 SMORES

Self Assembling Modular Robot for Extreme Shape Shifting is a modular robot system created in the University of Pennsylvania. Modules possess four DOF, three of them in the same plane, and one perpendicular to the others. It is designed to be able to recon-

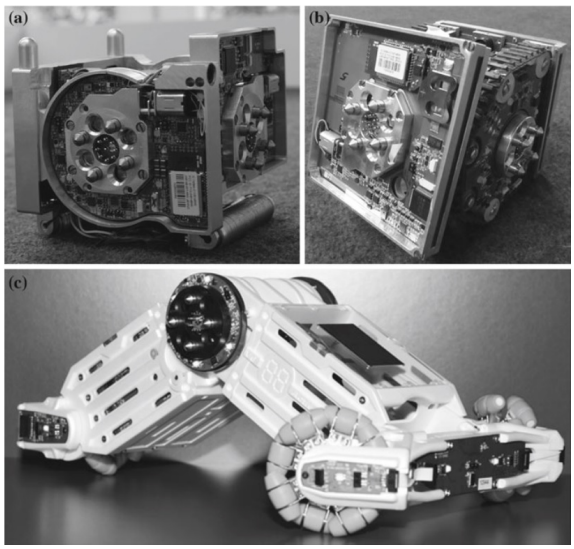


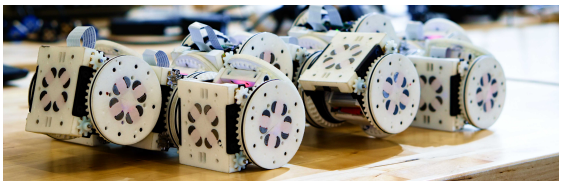
FIGURE 2.7. REPLICATOR modules (Taken from [90])

figure as a lattice, chain or mobile type modular robot, as a universal modular system [25]. The overall module shape resembles a Polybot module with circular rotating connector faces attached (Figure 2.8a). SMORES modules can achieve similar configurations to those of other modular robots like M-TRAN or Superbot with the addition of being mobile. Genderless connectors are placed on the four available faces, being three of them active and one passive. The combination of active and passive connectors enables self-reconfiguration and, since the connectors are magnetic, this process is faster than in other self-reconfigurable modular robots. However, the process still requires the use of a mechanical key and the movement of the whole connector face, which makes the design complex to build. A later version called SMORES-EP (Figure 2.8b) uses Electro Permanent (EP) magnets instead of normal magnets [50] in its connectors. Electro permanent magnets can de-polarize and re-polarize when short pulses of current through a coil generate a magnetic field around them. Thanks to the use of EP magnets, mechanical separating mechanisms are not needed anymore, but more electrical energy is consumed. No sensors have been reported for this system.

2.2.2.7 Fable

This modular robot system is designed to explore modular playware [87]. This heterogeneous chain type modular robot system is composed of three different types of modules following a simple to build design (Figure 2.9): Joint modules, with actuated joints; branching modules that are used to connect several modules together in tree-like configurations; and termination modules that close off open connectors on a robot and provide extra sensing and actuating modes to the robot. The overall shape of the module resembles one half of an American football with cut ends. All different types of modules have a magnetic genderless connector, which can be connected to other similar connectors of different sizes. Only manual reconfiguration is possible given the shape of the module and the specific

(A)



(B)

FIGURE 2.8. SMORES: Original SMORES Modules (a), SMORES-EP (b) (Taken from [72])

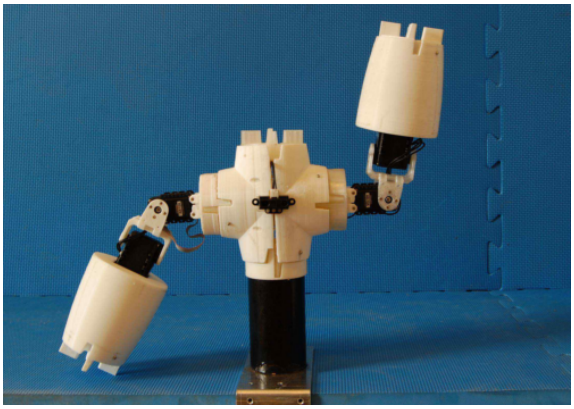


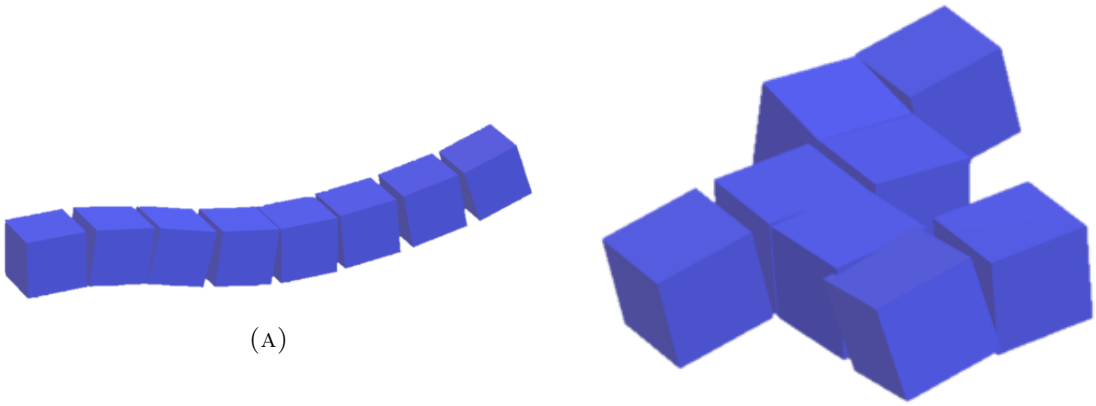
FIGURE 2.9. Fable modules (Taken from [87])

connector design, but this also enables robots to be reconfigured very quickly compared to other platforms. Limbed robots, snakes and vehicles have been achieved. Modules contain accelerometers and gyroscopes, as well as a small speaker and termination modules with proximity sensors.

2.2.2.8 Printable modular robot

With a similar shape to that of Polybot modules, this robot structure is designed to be completely 3D printed using standard FDM printers [55]. Modules use magnets to attach to each other and electrical connections and communications are routed through the magnets themselves. Magnetic connectors provide a very simple and quick way to reconfigure modules manually. However, only two faces of the module are used as connectors, hence only simple morphologies like snakes can be achieved. The hinge rotational joint is actuated by an off-the-shelf hobby servo and other electronic components are also off-the-shelf parts (Figure 2.10). This modular robot is an example of a very simple system and lacks other features, like the capabilities of forming more complex configurations and read-

FIGURE 2.10. Printable Modular Robot (Taken from [55])



(A)

(B)

FIGURE 2.11. Modular robot in two different morphologies: A snake morphology (a), a random morphology (b)

ing sensors, that would make it more useful for implementing and testing the techniques described in this work.

2.3 Control

Locomotion movement control for modular reconfigurable robots is difficult to design and implement. As morphology can be arbitrary, movements or mechanisms designed to correctly move one morphology could not be useful for others. Control strategies used for modular robots have to cope with their distributed nature and high number of actuated DOF and sensors. Centralized and decentralized control methods have been developed [79].

On one hand, centralized strategies use a main controller, usually an external computer or a specific module of the structure, which sends commands to every module at a given time. Centralized controllers have the advantage of being very simple to devise

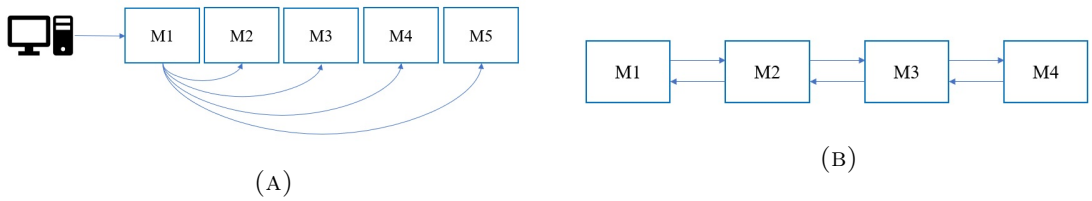


FIGURE 2.12. Centralized control (a) vs Distributed control (b) in modular robots. Centralized controllers must communicate with all modules in a morphology.

and implement. On the other hand, decentralized controllers work distributing the tasks that were done by the central controller among the modules in the robotic structure. One way of achieving this is defining a task coordinator module [111]. Any module inside the structure can act as a coordinator, and the coordinator can be changed in case of failure.

Due to several problems, centralized controllers are not the first option to effectively control modular robots [44]. First, trying to communicate with all the modules of the structure may create bottlenecks and hinder the scalability of the system. Second, the controller robustness is diminished, because if the central controller fails, all the system fails. Thus, this work will concentrate in the use of decentralized control strategies.

2.3.1 Decentralized Control Strategies

Several decentralized techniques have been proposed for controlling modular robots [79]. According to Stoy et al. [111], there are two main types of decentralized control:

- Synchronous Decentralized Control: Each module performs synchronized tasks using internal clocks or sequential algorithms, "Movement tables" are examples of this kind of control. Global synchronization of the system can produce its own problems and decrease performance.
- Asynchronous Decentralized Control: Asynchronous decentralized control strategies consider each module as an independent and autonomous entity, like in multi agent systems. In this case, global behaviors "emerge" from local interactions between modules [139]. Examples of this type of control are Central Pattern Generators.

The most basic example of a synchronous decentralized control strategy for modular robots involves creating sequences of movements for each module, which is also called "movement tables". This form of control was proposed by Yim [129] in 1994 and is based on a table that represents, for a given robot morphology, the state of every module at every step of a time driven sequence of movements. The table contains module identifiers in its columns vs. time steps in its rows. Once the last time-step is reached, the sequence starts anew from the first time step. This strategy enables the implementation of cyclic movements or "gaits", which are common in living beings.

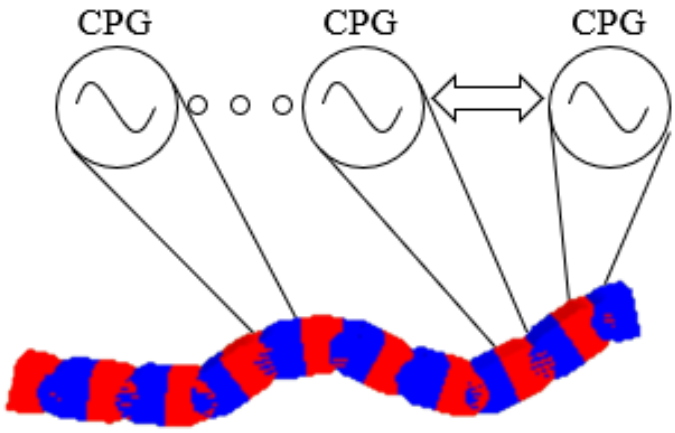


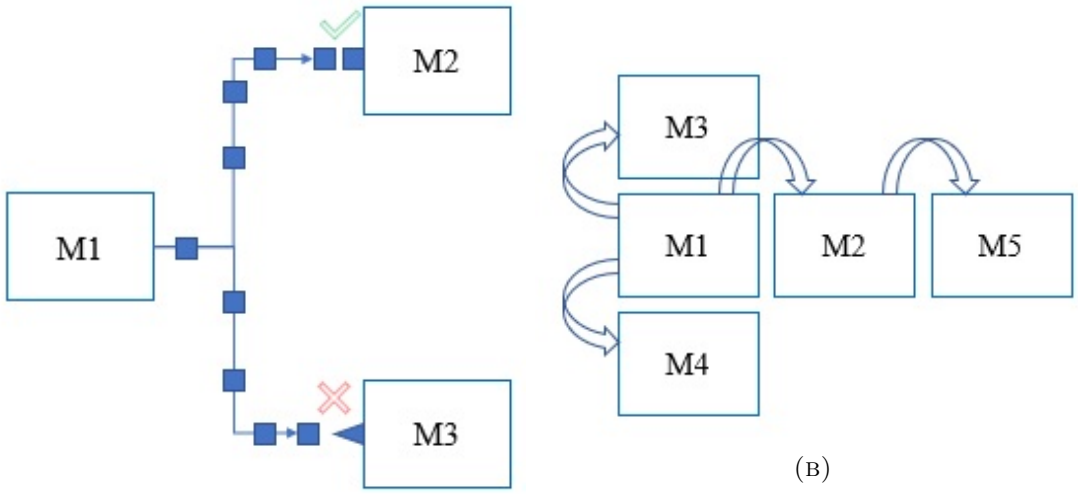
FIGURE 2.13. Example of CPG generating coordinated movement on a snake configuration of a modular robot

The table can be hard coded or created using an algorithm that assigns movements to modules at each time step. Module i executes only its corresponding column (the ith column) of the table, which is stored inside the module. The main problem with this control strategy is that a movement table must be created for every configuration the robotic system can take. Another problem is that it does not take into account sensor information, rendering it to be an open loop approach.

Sinusoidal movement generators have also been used to control the movement of modular robots synchronously [94, 28]. In this case, a sinusoidal wave is generated for each actuator in the robotic structure. Each wave is configured with a different phase, thus different modules are coordinated to achieve a gait. The main disadvantage of this control technique is that there should be an strict synchronization among all modules in order to move the robot correctly.

In the case of asynchronous decentralized control strategies, there is a technique that draws inspiration from biological control structures found mainly on vertebrates. These structures are capable of generating complex oscillatory movement from a relatively simple input and are usually in charge of repetitive tasks like chewing, swimming, walking or breathing [35]. The generated movement can be synchronized with the movement produced by other similar structures [115, 127]. Structures of this type are called Central Pattern Generators (CPGs) and have the advantage of being a decentralized control method, which can work with local information to achieve coordinated movements [46]. CPG structures can be used to generate locomotion movements in a simple fashion by using their local coordination features, thus providing a good low-level locomotion control and coordination mechanism [92].

With these properties, different CPG implementations have been used to control a number of modular robot prototypes obtaining robust locomotion patterns that can support different terrain conditions for a given morphology (Figure 2.13). CPGs can thus be used to provide an stable coordination strategy that is able to generate basic locomotion



(A) (B)

FIGURE 2.14. Hormone inspired messages. aMessages are processed differently in each arriving module, b Example of message propagation.

movements for different robot morphologies. Moreover, movements generated by CPGs for a specific robot morphology can be tuned, so that the robot can locomote in different environmental conditions [84]. Feedback can be introduced to adjust the output to sensor information as in [84, 94]. However it has been used in very specific ways. Depending on the type of implementation, CPGs can be very unstable at the beginning of a robot movement, while control signals converge to stable values.

Hormone inspired messages is another technique used to asynchronously control modular robots (Figure 2.14). In previous work [102] hormone inspired messages have been defined as messages that can be interpreted in different ways by different receiving modules and that can be delayed or modified before being propagated [42]. This kind of control strategy is decentralized, robust to changes in configuration [97, 110] and has been used to achieve synchronization among sets of modules [16, 43]. Hormone inspired messages differ from CPGs in that they are somewhat separated from the module themselves, that is, modules only react to hormones arriving from their neighbors, while CPGs controlled modules can act in complete isolation from other CPGs.

Similar to some variations of hormone inspired messages that model diffusion reactions [99, 37], Gene Regulatory Networks (GRN) model the control mechanisms of a number of important cellular behaviors [71, 7]. Since a parallel between cells in a body and modules in a modular robot configuration can be made, GRN based techniques have been used to control modular robots. In [140] Zahadat et al. uses GRN as reactive controllers inside individual modules of a configuration that can execute a light following task. Meng et al. have also used a hierarchical controller with a GRN to control self-reconfiguration of a modular robot in simulated environments.

Although some of these bio-inspired control techniques are capable of including sensor information in their calculations, usually no more than one or two sensors are included in the control model for a whole robot morphology [94, 82]. Modular robots can potentially have sensors in each of its modules. This feature can be exploited to study how several simple sources of sensor information throughout the body (as a first step towards proprioception or sensor-actuator fusion) can affect the motion of the robot and how motion can be adapted taking into account sensors when moving in different environments.

2.3.2 Automatic controller generation

Implementing the control techniques described in subsection 2.3.1 to actual modular robots can be a tedious task, specially when controllers have several tunable parameters and different controllers can be used for different modules. Additionally, typical modular robot configurations have several modules and each of them has its own control parameters. For this reason several works have concentrated in the automatic generation of controllers for robot morphologies or individual modules. Methods like reinforcement learning [24, 19], evolutionary algorithms [137] and motion planning [136, 114], have been used for this purpose.

Reinforcement learning is a machine learning method in which a reward, or punishment, is given to an agent when it follows a policy (an action or a set of actions that are executed by the agent) [2]. Reinforcement learning involves the use of a critic. A critic is someone or something that gives a reward or a punishment to the learning agents based on their actions and the state of the environment. An internal representation of the utility of each action given a certain external state or observation is necessary and it is updated over time using predefined formulas. The main objective is to choose the best action in terms of reward or the best state in terms of utility.

Works that make use of reinforcement learning in modular robots often test in simulated and rather abstract models, as some issues, like grow in state variables and actions, can lead to higher complexity. Varshavskaya et al [120], Shiba et al [103] and Karigiannis and Tzafestas [53], follow this approach. Varshavskaya, Pack and Rus apply a Gradient Ascent in Policy Space (GAPS) [119] algorithm and a Distributed Gradient Ascent in Policy Space (DGAPS) [120] algorithm to square shaped modules which can slide alongside each other in a simulated 2D environment with basic physical laws. Shiba et al [103], applies Q-Learning to a 4-armed planar robot to make it reconfigure from an initial configuration to a goal configuration. Karigiannis and Tzafestas [53] use reinforcement learning in a nested-hierarchical multi-agent environment where all agents are links in a 2D manipulator.

Communication has been seldom explored in works using reinforcement learning approaches and only few works have studied the role of communicating sensor information between modules when learning. The DGAPS algorithm proposed by Varshavskaya et al [121] will converge to a local optimum given that the distributed agents get the same

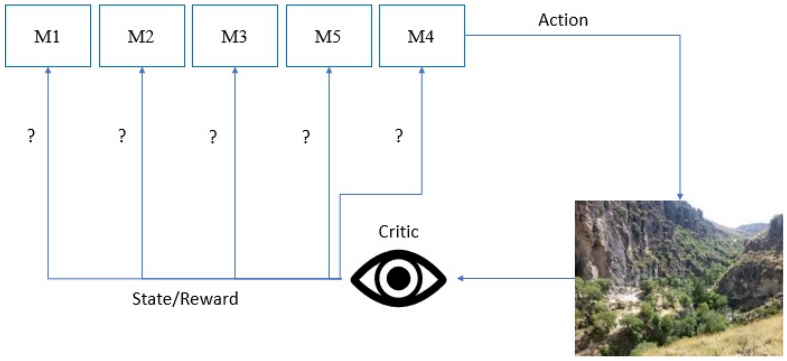


FIGURE 2.15. Assigning reward and state information to individual modules is an issue when using reinforcement learning in modular robots

experience and reward as a centralized learner, something that is accomplished by means of agreement algorithms. Karigiannis and Tzafestas [53] make each agent "see" which actions are performing other agents below in the hierarchy and calculate the probability of choosing an action.

Another issue is how to assign the reward so that the group of agents cooperate with each other towards the same goal (Figure 2.15). In [119], Christensen et al. [18], and D'Angelo et al [24] all modules share a reward signal which is how far the complete robot reaches in a certain direction in a given time. Christensen et al. [18] use reinforcement learning in actual modular robots, they use reinforcement learning to learn movements in different configurations of the ATRON robot using gait tables. D'Angelo et al [24] also use reinforcement learning to generate controllers for different configurations of YAMOR modules in flat surfaces.

Evolutionary algorithms are also used to develop and optimize movement behaviors for modular robots. Evolutionary algorithms [26] are inspired by biological evolution and are population based optimization algorithms. A population of solutions for a given problem is generated and tested against a performance measure. Then, selected individual solutions are modified in different ways to produce new solutions that hopefully will have a better performance measure than the originals. Mechanisms such as mutation and crossover are used to obtain candidate solutions from the last generation. The processes of selecting and modifying solutions in the population are usually inspired in the biological processes that take part in natural evolution (Figure 2.16).

Evolutionary algorithms take large numbers of generations and fitness measurements which make it a very slow process therefore it is often done in simulation [66, 5, 135, 117, 85, 20, 118, 140, 94]. Simulations allow abstract representations of modular robots to be used. Bennett and Rieffel [5] evolve function based programs as controllers for 2D square shaped modules in simulation for a set of different problems involving module failures and navigating narrow spaces using sliding movements. Torres and Zagal [118] also use a genetic algorithm to evolve the locomotion control of a group of simulated 2D robots in different environments in simulation.

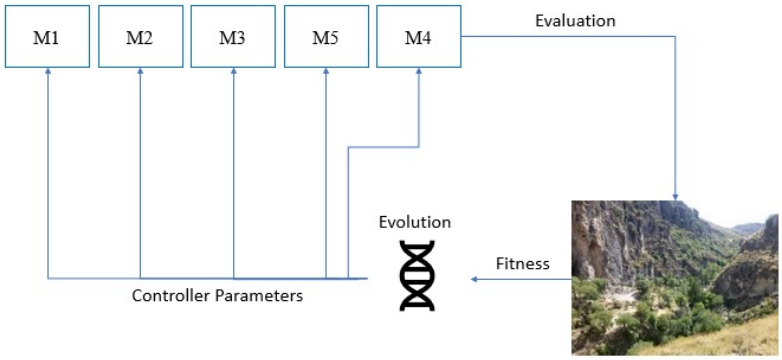


FIGURE 2.16. Evolution is usually used to optimize controller parameters

More often than not, evolution is used to produce locomotion control of different configurations of modular robots. Sometimes evolution of controllers is accompanied by the evolution of the morphology, taking advantage of the simulated approach. Marbach et al [66] and Toley et al [117] use evolutionary algorithms to evolve different configurations of simulated modular robots for locomotion and structural tasks respectively. Faina et al [28] also evolve the morphology of modular robots for a locomotion and a painting task. Ostergard and Lund [85] apply coevolution with genetic algorithms to generate movement by reconfiguration from different initial configurations, which are also evolved, of a simulated ATRON robot.

Only a few studies include sensor information when adjusting or optimizing controllers for modular robots. This limits the movements that can be generated by controllers only to cases when there are no sensors present in the system. Yoneda et al [135] use real valued genetic algorithms to evolve controller parameters for simulated modules attached trough actuated springs to each other. The modules are in a ring configuration and must travel to a light source which each module can sense with its own light sensor. Zahadat et al [140] evolve controllers based on Fractal Gene Regulatory Networks (FGRN) that can react to changes in the environment. Fitness is computed based on distance to two individual target goals where the simulated robotic system must arrive to using locomotion movements. Rossi et al [94] evolve sinusoidal controller parameters for different configurations of a modular robot with sensors on one of its modules. Only one module is fitted with two proximity sensors, called "head" module. Evolution tunes the influence of the "head" module sensors readings in the output of all modules sinusoidal controllers.

Simulation results obtained from evolutionary processes have also been tested in real robots. Murata and Kurokawa [82], Yoshida et al [137] and Kamimura et al [52, 51] describe the use of genetic algorithms to optimize the parameters and network connections between CPG controlling the robot M-TRAN in simulation and then use the resulting parameters to move a real robotic structure. Yoshida et al [138] also describe the use of evolutionary algorithms with movement tables in M-TRAN. However, transferred results are affected by differences between the simulated robots and their real counterparts, a problem called the reality gap.

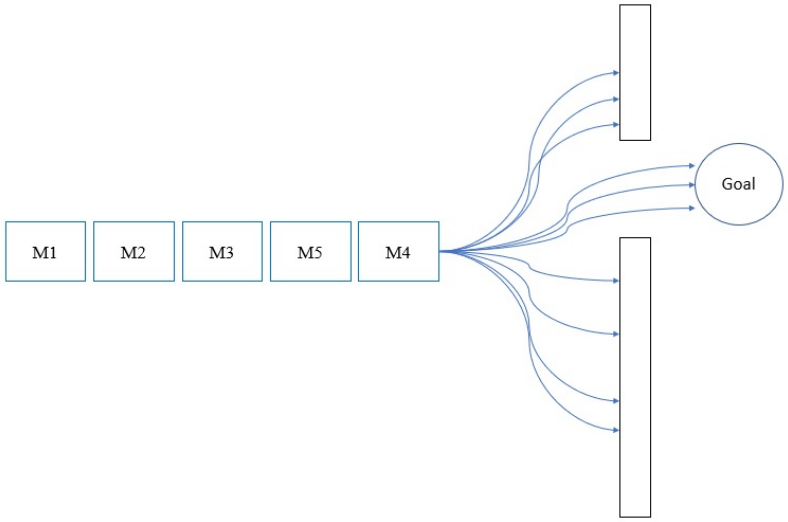


FIGURE 2.17. Motion Planning

Another technique, motion planing, has been used to automatically generate modular robot controllers for moving in an environment. The motion planning problem involves calculating a precise description of how a robot must move in order to get from an initial point in space to a goal while evading obstacles [59, 104] (Figure 2.17). This problem has been well studied for decades in robotics and different techniques have been proposed to solve it, including sampling based techniques like Rapidly Exploring Random Tress (RRTs) and combinatorial based techniques. Motion planning has been applied to generate locomotion movements in modular robots like M-TRAN [136] and REPLICATOR [123, 124]. It has also been used to generate dynamic reconfiguration movements in a chain type modular robot [114]. One of the main limitations of using motion planning is that the whole environment in which the robot will move has to be known beforehand.

Techniques different from evolution, reinforcement learning and motion planning to the locomotion problem include Marbach and Ijspeert [67] which use Powells method, a gradient free optimization algorithm, to adjust the parameters of CPG controllers and the configuration of a modular robot in simulation. Ranasinghe and Shen [91] propose a surprise based learning algorithm which creates a model of the environment and updates it according to discrepancies in observations when executing different actions on the SuperBot robot. Jing et al [50] propose a high level control system that relies in a previously created set of motions and configurations to perform a given task. A set of conditions and rules are specified and an automata is created that solves the specified task.

2.4 Conclusion

In this chapter, the basic concepts of modular robots, controllers, and their automatic generation, that have been used for the locomotion task, were presented. The literature review includes the different types of modular robots that have been developed to date

and analyzed some modular robot prototypes, how their module morphology affects their ability to move and reconfigure, the sensors they can carry and how complex is to build them. The feature of modular robots to have their morphology changed provides an advantage over usual monolithic robots when moving in different environments. However, movement control for reconfigurable robots is difficult to design and implement. Deciding when to use specific movements is not trivial when changing configurations. Control mechanisms that can cope with the reconfiguration ability of modular robots have been designed and tested in several prototypes.

Even after implementing specific controller strategies, tunning them for modular robots can be a difficult task. That is why several studies have concentrated on the automatic generation of control strategies. Still, most of them have only made robot morphologies move in flat surfaces with simple obstacles. Sensor information has also been greatly ignored and only few works study the influence of sensor information in the automatic generation of controllers, in very basic settings. The next chapter will define a framework for training modular robot morphologies to move from one point to another while addressing these limitations. This framework includes the design of a control strategy that can handle sensor information coming from an arbitrary set of sensors in order to adjust the locomotion movements. The framework also includes the development of the EMERGE modular robot platform, which can be built using simple off-the-shelf components and uses simple magnetic connectors to enable rapid deployment of different morphologies.

CHAPTER 3

Locomotion Training Framework

Modular robots can potentially have an advantage over monolithic robots when moving through different environmental conditions. This advantage comes from their ability to reconfigure, be it by themselves or by an external agent. However, the morphology of a modular robot is not known before hand and the robot can have an arbitrary set of sensors inside its modules. Generating the movements necessary to make a modular robot move in different environments is a difficult task.

To tackle the problem of generating locomotion movements in modular robots, that is, to make them able to move in different environments, this work proposes a locomotion training framework that has four main parts (Figure 3.1): A modular robotic platform from which morphologies are built, an adaptable controller that allows the robot to change its behavior depending on incoming sensor information, a configurable environment which allows the controller to be trained and tested in different environments and a way to train the adaptable controller to move in different environments. Next chapter describes the modular robotic platform. This chapter describes the adaptable bio-inspired controller, which is based on central pattern generators, hormone inspired messages, and a decision mechanism; this chapter also explains how the configurable environment and the method for training the adaptable controller work. A simple chain type modular robot is used for explaining the way the adaptable bio-inspired controller works.

3.1 Control Strategy

Two previous works are the base of the approach proposed in this work. In a previous work (Moreno and Gomez [76]) a similar control system for a chain type modular robot is designed and implemented. The previous work is also based on CPGs and Hormone inspired messages and allows an arbitrary configuration of a 1 DOF chain type modular robot to traverse an obstacle filled environment. Modules sum a small value to get the

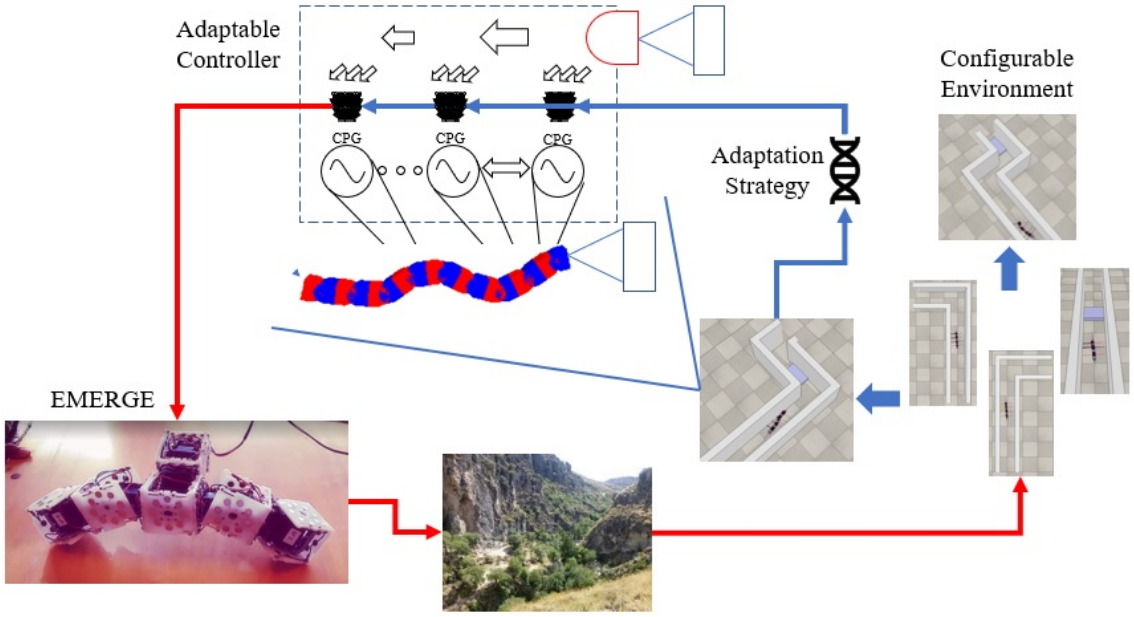


FIGURE 3.1. Locomotion training framework that includes and adaptable controller, a configurable environment, an adaptation strategy and a modular robot platform. Blue arrows indicate processes that can be done in simulation, red arrows indicate processes that can be done in reality.

CPG parameters towards predefined set-points, that represent different movements, for every incoming hormone message. Hand designed movements take into account sensor information coming from proximity sensors placed in all of the robot modules. The work in [76] lacks however the way to automatically generate controllers for a modular robot configuration. This is addressed in Rossi et. al. [94] (See figure 3.3), in which a module is fitted with two proximity sensors, as a sort of "head" module, and this module is attached to different planar configurations. Every actuator of every module is controlled using an individual sinusoidal generator. Theses sinusoidal generators contain a term that is affected by the values read in the "head" module sensors. Evolution is used to adjust how much proximity sensors in the "head" module affect controllers in all actuators, and the overall movement of the robot. A wall is used as a simple obstacle between the robot starting position and a goal to test whether the robot can evade it using the "head" sensors. To continue expanding both works, and help a modular robot learn to locomote through different environments, the use of a control strategy composed of three parts is proposed: a coordination mechanism, a sensor information propagation mechanism and a decision mechanism. The control strategy contains parts specifically aimed to coordinate modules in a decentralized fashion while handling and communicating sensor information obtained from an arbitrary set of sensors. The current strategy is defined for its usage in homogeneous modular robots that meet the following criteria:

- Modules have a way of communicating information to other modules, that is, between any two modules.

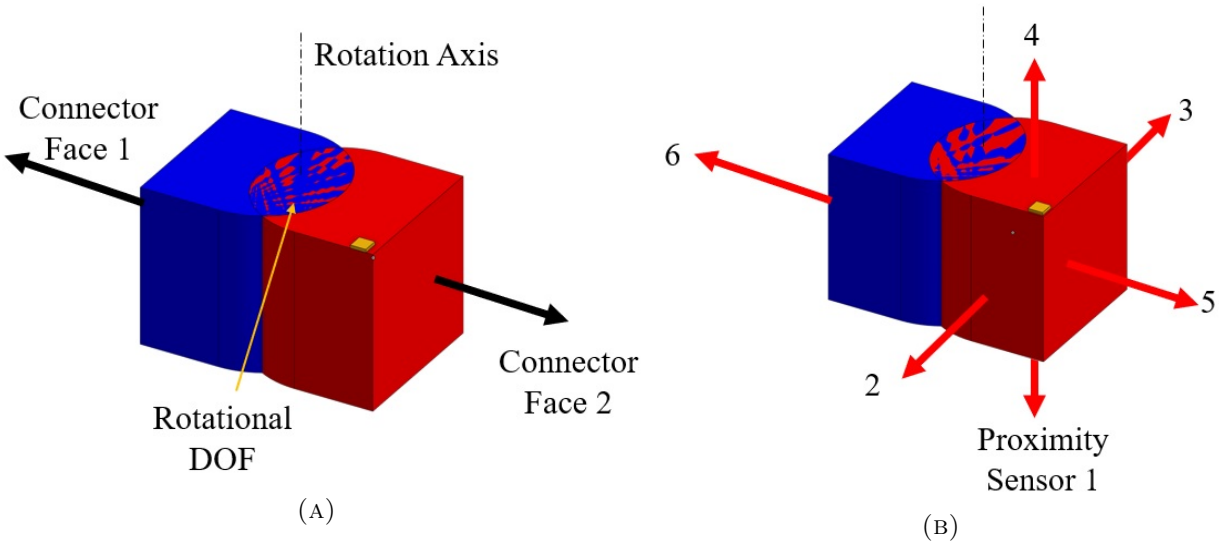


FIGURE 3.2. Simple Ex modular robot with sensors. The orientation sensor is internal to the module.

- Communication is only necessary among nearest neighbors. Further reaching communication mechanisms are also allowed.
- Modules are able to sense their environments through the use of one or more well characterized sensors.
- Only simple sensors are required (e.g. simple infrared proximity sensors, orientation sensors, etc, ...)

For the purpose of explaining the control strategy, a simple homogeneous modular robot with one rotational DOF is used (See figure 3.2), called Ex module. The module is fitted with proximity sensors on each of its sides and one orientation sensor (Figure 3.2b). Modules have only three connector faces and communication is only allowed among nearest neighbors. The control strategy can be, however extended to other types of modular robots with higher numbers of actuators and sensors. Time is assumed to be discrete, that is, time advances following a fixed time step Δt.

3.1.1 Coordination

As established in chapter 2, CPGs provide coordination among different modules in a robot structure. In the same way as in [76], a CPG based coordination mechanism is used in this work to generate coordinated locomotion movements for different robot morphologies. The output of an oscillator CPG is used to control the movement of a module actuator. A module has as many CPGs as actuators and each CPG controlled actuator can be coupled to other CPGs of the module itself or neighboring modules. The output of a module is coordinated with other modules output so that coordinated movements or

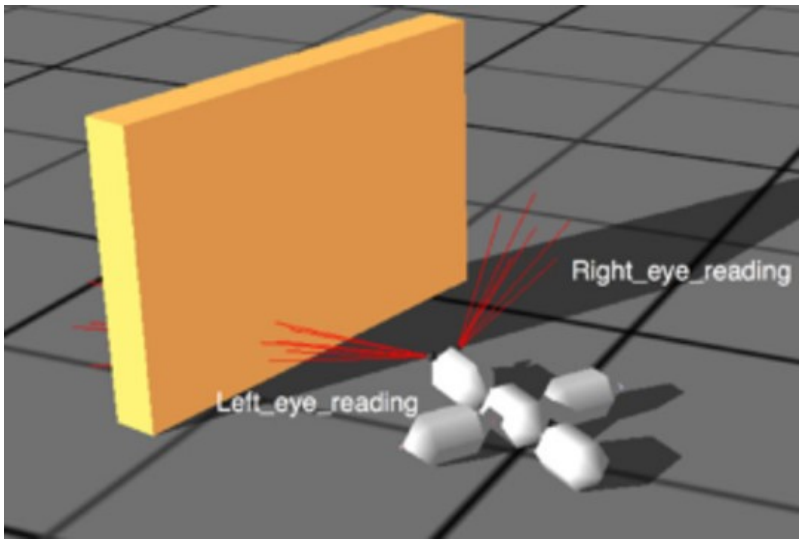


FIGURE 3.3. A modular robot with sensors (Taken from [94])

gaits [1] are generated in different robot morphologies. In the case of having a one DOF module, with one actuator, a module's movement can be controlled by tunning CPG individual parameters (Figure 3.7). A set of 1 DOF modules fitted with CPGs coordinate the morphology movement based on local interactions (Figure 3.8c).

The specific implementation of CPG used in this work is based on phase coupled oscillators described in equations 3.3 to 3.6 and come from [23, 22]. Phase coupled non linear oscillators provide an abstraction of the main features of CPGs. A range of parameters can be used without affecting the stability of the output [79]. Let R be a modular robot with n modules, where M_i represents the ith module of the robot, and which is configured following topology T (equation 3.1).

$$T = \{\langle i, j, l, k, o\rangle | i, j \in \{1, 2, \ldots, n\} \wedge l, k \in \{1, \ldots, f\} \wedge o \in \{1, \ldots, or\}\} \tag{3.1}$$

Where the tuple $\langle i, j, l, k, o\rangle$ represents a connection between modules M_i and M_j, l and k represent the corresponding connector faces of modules M_i and M_j used in the connection (f is the number of connector faces that each module has), and o is the specific orientation between connector faces in the connection (or is the number of possible orientations between connectors). Figure 3.4 shows the possible orientations between connector faces of module Ex and figure 3.5 shows a simple connection example, figure 3.6 shows an example topology with more modules.

Module M_i is defined by the tuple in equation 3.2.

$$M_i = (S_i, Th_i, V_i, R_i, X_i, \Delta\phi_l^{(i)}) \tag{3.2}$$

[1] gait refers to specific cyclic movements that let animals locomote as defined in [46]

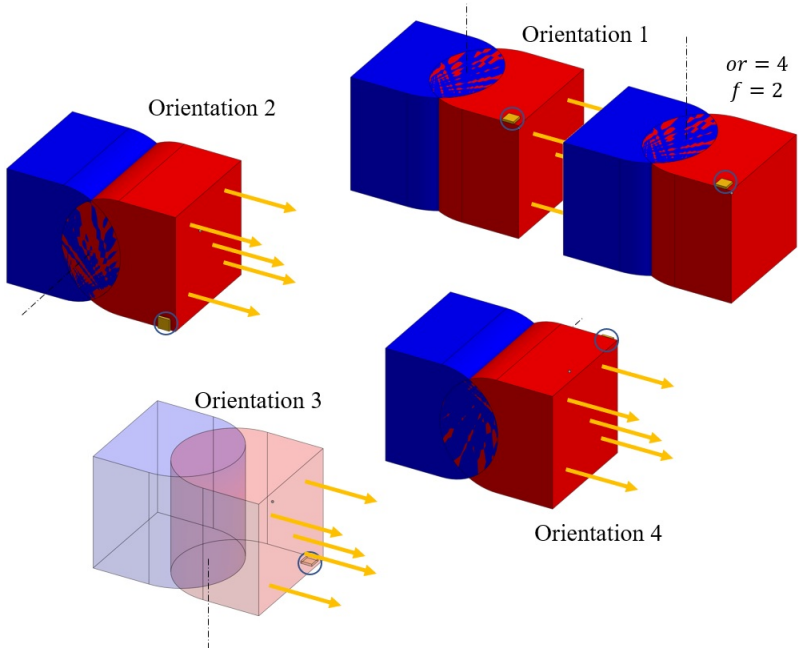


FIGURE 3.4. Possible connection orientations of module Ex, f represents the number of connection faces of the module and or is the number of possible orientations between connector faces.

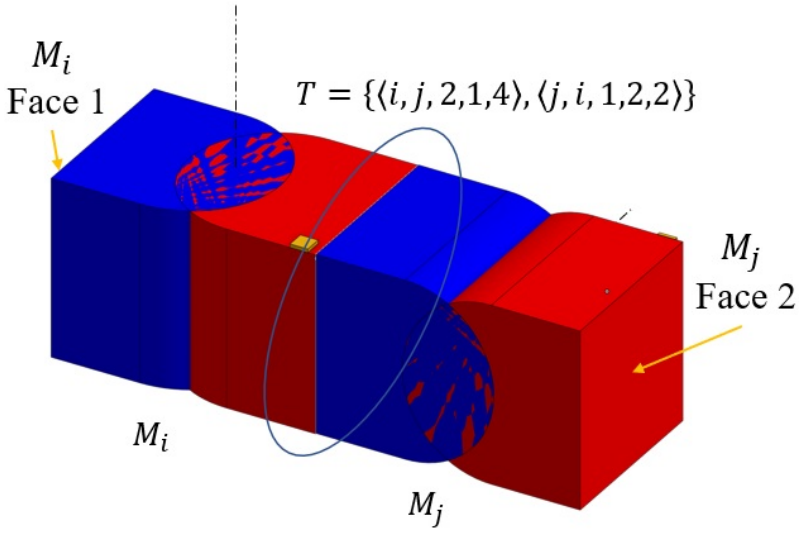


FIGURE 3.5. Example connection with the Ex modules, T represents the topology of the robot.

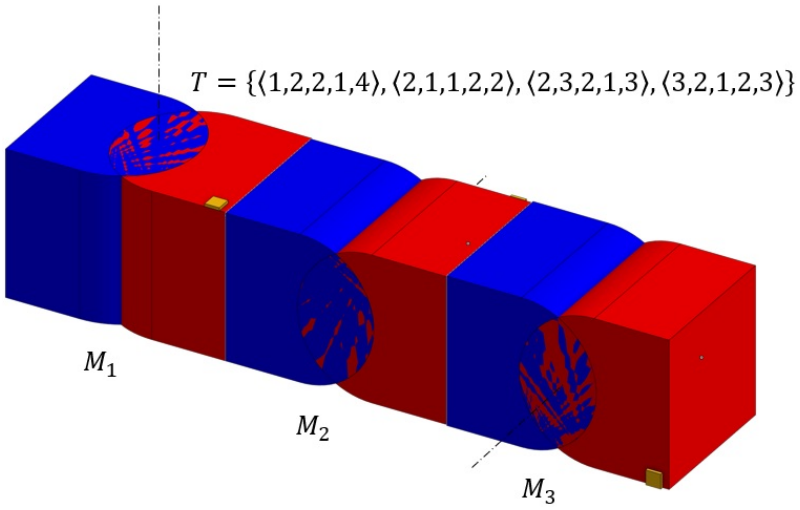


FIGURE 3.6. Example robot topology using Ex modules, T represents the topology of the robot.

Where S_i is the ordered set of all sensors that trigger a message when activated ($S = \{s_1, s_2, \ldots, s_m\}$, m is the number of sensors included in the message, see section 3.1.2), $Th_i = \{th_1, th_2, \ldots, th_m\}$ contains the threshold values at which the corresponding sensors in S_i are activated, and $V_i = \{v_1, v_2, \ldots, v_m\}$ stores the values read by each sensor. The order of S_i, Th_i and V_i is related to the sensors location around the module (See figure 3.11 for an example). As the modules considered are homogeneous, the order of S_i, Th_i and V_i is the same for all modules in a robot R. The rest of module M_i parameters (R_i, X_i and $\Delta\phi_l^{(i)}$) are explained in equations 3.3 to 3.6.

$$\dot{\phi_i} = \omega_i + \sum_{l=1}^{f} g_{i,T}(l) wSin(\phi_l - \phi_i - \Delta\phi_l^{(i)}) \tag{3.3}$$

$$\ddot{r_i} = a_r(\frac{a_r}{4}(R_i - r_i) - \dot{r_i}) \tag{3.4}$$

$$\ddot{x_i} = a_x(\frac{a_x}{4}(X_i - x_i) - \dot{x_i}) \tag{3.5}$$

$$\theta_i = x_i + r_i Cos(\phi_i) \tag{3.6}$$

Equation 3.3 shows the coupling between the CPG phase of module M_i (ϕ_i) and a neighbor module connected on face l. Each module has its own phase difference parameter ($\Delta\phi_l^{(i)}$) for each of its connected neighbors. For ease of notation the function $g_{i,T}$ is introduced (Equation 3.7), which returns whether module M_i is connected to another module through face l.

$$g_{i,T}(l) = \begin{cases} 1, & \exists j, k, o \mid \langle i, j, l, k, o \rangle \in T \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

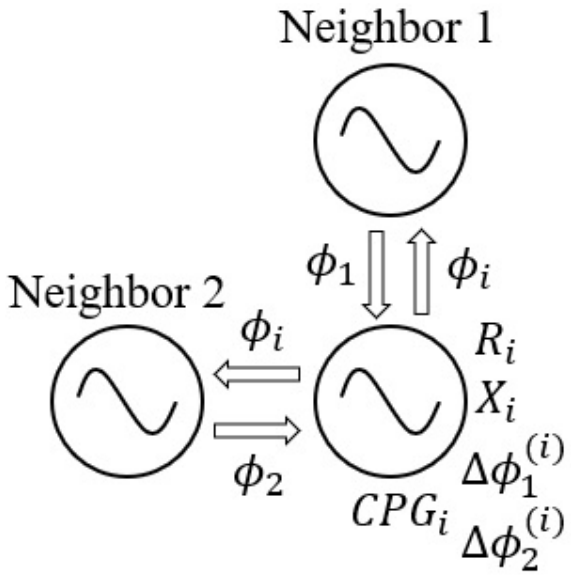


FIGURE 3.7. Parameters that adjust the output of a module CPG: Amplitude (R_i), Offset (X_i) and phase differences with neighbors ($\Delta\phi_l^{(i)}$). A module's CPG sends its phase (ϕ_i) information to its neighbors and receives their phases (ϕ_l)

Also in equation 3.3, ω_i represents the intrinsic frequency of the CPG oscillator; w is the coupling strength between the oscillator on module M_i and its neighbors, and will remain fixed through all this work. If the phase difference of module M_i with the module connected in face l ($\Delta\phi_l^{(i)}$) matches the opposite sign value of the same parameter in the neighboring module, the modules are guaranteed to converge to the phase difference specified by $\Delta\phi_l^{(i)}$ [22]. However, if $\Delta\phi_l^{(i)}$ and its neighbor's counterpart don't match, the modules phases will converge towards an intermediate phase difference value, which is represented in figure 3.8 as $\Delta\phi_{ij}$.

Equations (3.4) and (3.5) describe control laws that make the amplitude r_i and offset x_i in the output (equation 3.6) converge to the set points R_i and X_i. The parameters a_x, a_r are weights used to control the speed of convergence of the amplitude and offset to their respective set points and will also be fixed through all this work. Thanks to these control laws the CPG implementation also presents a smooth transition on the output even if the input is changed abruptly [23]. This smooth transition feature can help protect the robot actuators in the case of a rapidly changing control system. Equation (3.6) describes the oscillator's output (θ_i) as a cosine function of the phase, amplitude and offset. The output is used to set the position of a module actuator, in the case of module Ex this position is an angular position (See figure 3.8). Using this model a CPG output in module M_i is adjusted by the following parameters: Amplitude R_i, offset X_i and all phase differences with neighbors $\Delta\phi_l^{(i)}$.

Algorithm 1 describes the behavior of the CPG controller inside a module for every time step. The CPG equations (Line 5) can be updated by using any differential equation approximation method. As described by equations 3.3 to 3.6, this implementation of

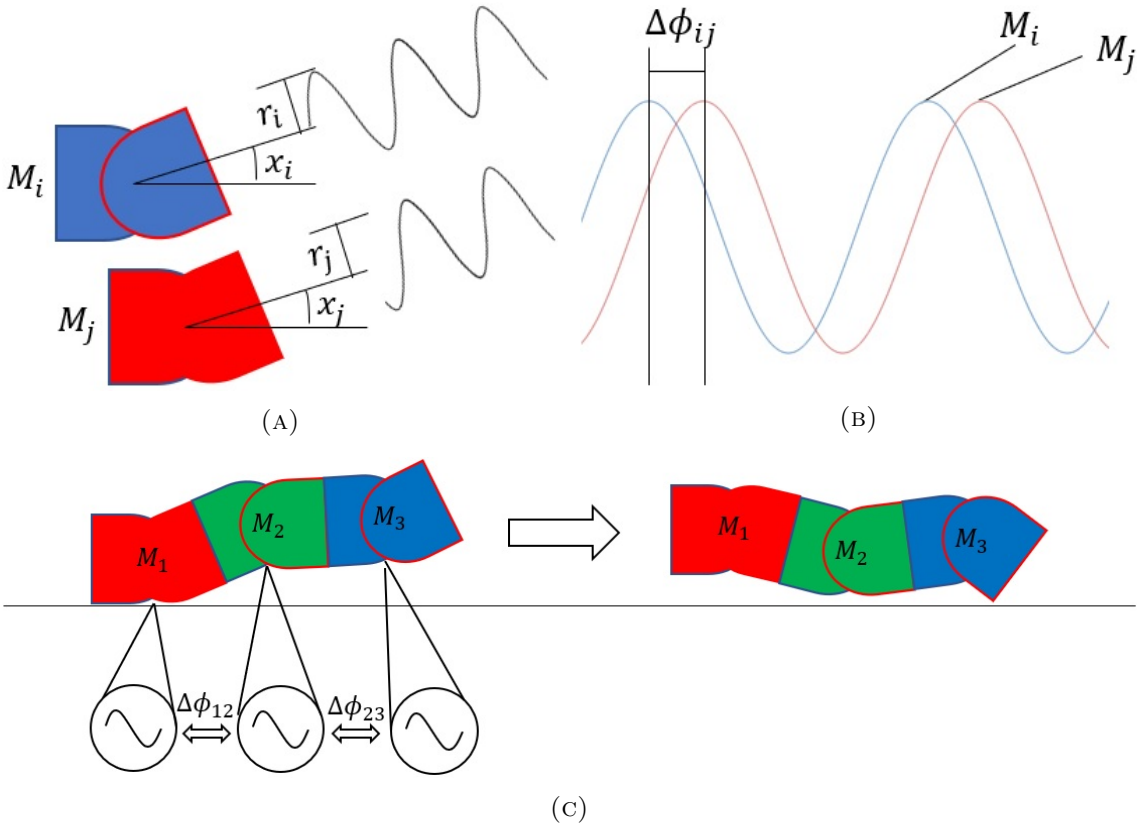


FIGURE 3.8. CPG oscillator output: (a) Two Ex modules rotational actuators controlled by coupled CPG oscillators.(b) Phase difference between the two modules. (c) A set of three modules CPGs ($\Delta\phi_{12}$ and $\Delta\phi_{23}$ represent the phase difference between modules 1 and 2, and modules 2 and 3 after convergence).

CPG as a phased coupled non linear oscillator works as an open loop controller. Although feedback can be introduced in equations 3.3 to 3.6 to give them the ability to adjust the output to sensor information as in [84, 94], the specific way in which sensor information coming from the module's (or other modules) sensors should influence a CPG is difficult to define given the reconfigurable nature of modular robots. That is, a module can be used in different positions and orientations in different robot morphologies (See figure 3.9. For this reason we choose to handle sensor information across different robot morphologies, and how sensor information influences the parameter adjustment of the CPG controlled modules, with two other different mechanisms that will be described in the following sections.

Algorithm 1 CPG controller cycle in module M_i for every time step.

Require: $\Delta t, T, \phi_t, \dot{r}_t, \dot{x}_t, r_t, x_t$

loop ▷ forever
  for all $g_{i,T}(l) = 1$ **do**
    Receive ϕ_l
  end for
  Update CPG equations (3.3-3.6)
  $\phi_{t+\Delta t} = \dot{\phi}_t \Delta t + \phi_t$
  $\dot{r}_{t+\Delta t} = \ddot{r}_t \Delta t + \dot{r}_t$
  $\dot{x}_{t+\Delta t} = \ddot{x}_t \Delta t + \dot{x}_t$
  $r_{t+\Delta t} = \dot{r}_t \Delta t + r_t$
  $x_{t+\Delta t} = \dot{x}_t \Delta t + x_t$
  for all $g_{i,T}(l) = 1$ **do**
    Send ϕ_i to neighbor connected in face l
  end for
end loop

3.1.2 Sensor information handling

Sensor information plays a very important role in the adaptability of robots to different and possibly changing environments. Living organisms show this in the plethora of sensory systems that they use to do everyday tasks. Living beings show very specialized sensory structures that can detect from light to sound and other types of physical phenomena [89]. In the case of modular robots, sensors can be placed around the module's body so that it can detect distance to an obstacle or its own orientation to the ground (Figure 3.2b). Each module can have its own set of sensors and different modules can also have different sets (Figure 3.9).

A variation of Hormone inspired messages is used in this work to handle how sensor information is routed across a modular robot morphology without regard for the specific morphology of the robot or the type of sensors used. Hormone inspired messages have been used to achieve synchronization of modules [16, 43], or to directly control the movement of modular robots [97, 110]. In this work coordination is already achieved locally by the interconnected CPGs and, in contrast with previous works, the use of Hormone inspired

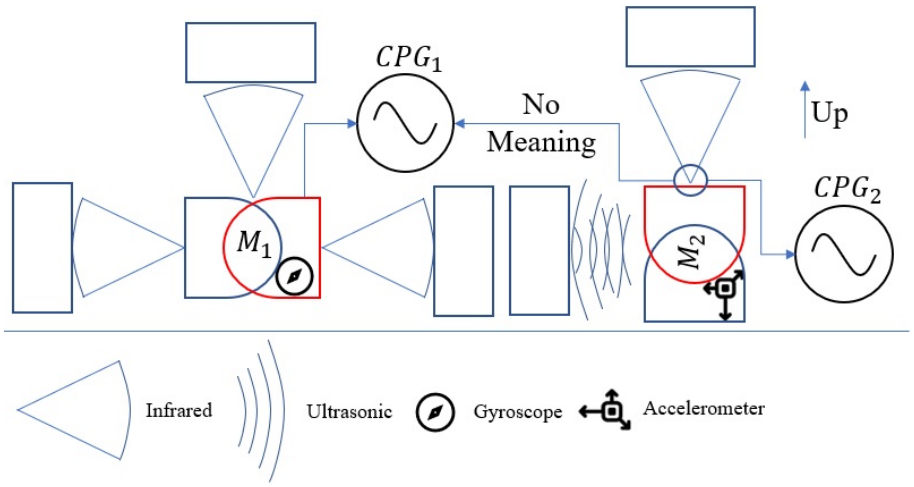


FIGURE 3.9. Modules can have different sensor sets. For example, M1 and M2 have different sets of proximity sensors, in unique positions and orientations, as well as inertial sensors. The position and orientation of each proximity sensor determines its main sensing space. The spatial orientation of the module also changes the meaning of the sensor information.

messages as a mechanism that only propagates sensor information throughout a modular robot is proposed. Algorithm 2 describes the general hormone mechanism.

The general hormone mechanism, as used in previous studies, is divided into three main parts: Generation, Reception and Propagation.

- Generation: The generation part depends directly on the sensors state. In the case of having proximity sensors, the position and orientation of a sensor determines its main sensing area (Figure 3.9). When sensor s_p $(p = 1, 2, \ldots, m)$ is activated, i.e. its values goes over a threshold $th_p \in Th$ (For example, when a proximity sensor detects an object in its sensing range) the module stores the state of the sensor in $v_p \in V_i$ (Line 5). Where v_p can take either a default value, that signals that sensor s_p has not been activated, or the value read by s_p, if s_p has been activated. All sensor values are normalized in the range $(0, 1)$ This sensor information set V_i is then sent to the nearest connected neighbors of the module as a generated hormone message on every cycle of the process. If no sensor was activated during one cycle, a message is neither generated nor sent to other models. Sensors that are not included in the messages, e.g. orientation sensors, indirectly influence the way in which messages are generated. For instance, the orientation sensor of the example module determines the orientation of the module relative to the ground (Figure 3.10), the proximity sensor directly facing the ground can be thus filtered (its value never stored) even if it activates.

- Reception: In the reception part, as the name implies, messages containing sensor information, that come from other modules, are received and processed. In modular robot configurations, modules are attached to other modules in different faces and orientations (Figure 3.9). Receiving a message from a module connected in a

Algorithm 2 General hormone messages mechanism inside module M_i.

Require: $0 \leq \alpha < 1, \epsilon, T, \Delta t, \phi_0, \dot{r}_0, \dot{x}_0, r_0, x_0, tw$

$t = 0$
loop ▷ forever
  Generation:
  for Each $s_p \in S, p \in \{1, 2, \ldots, m\}$ **do**
    $v_p = \begin{cases} s_p, & s_p \geq th_p \\ \text{default}, & \text{otherwise} \end{cases}$
  end for
  Sense orientation (ori) ▷ Sense environment
  if $\exists p, v_p \neq$ default **then**
    for all $g_{i,T}(l) = 1$ **do**
      Send V_i to neighbor connected in face l
    end for
  end if
  Reception:
  for all received V_l **do**
    $V_l' = \rho_{i,T}(l) V_l$
  end for
  $\langle R_i, X_i, \Delta\phi_l^{(i)} \rangle = \text{DecisionMechanism}(V_i, V_l', \text{ori}, tw, t)$ ▷ Algorithm 5
  $\langle \phi_{t+\Delta t}, \dot{r}_{t+\Delta t}, \dot{x}_{t+\Delta t}, r_{t+\Delta t}, x_{t+\Delta t} \rangle = \text{UpdateCPG}(T, \Delta t, \phi_t, \dot{r}_t, \dot{x}_t, r_t, x_t)$ ▷ Algorithm 1
  Propagation:
  for all received V_l' **do**
    $V_l'' = \alpha V_l'$
    $V_l''' = \begin{cases} \emptyset, & max(V_l'') \leq \epsilon \\ V_l'', & \text{otherwise} \end{cases}$
    for all $V_l''' \neq \emptyset$ **do**
      Send V_l''' to connected neighbors different from l
    end for
  end for
  $t = t + \Delta t$
end loop

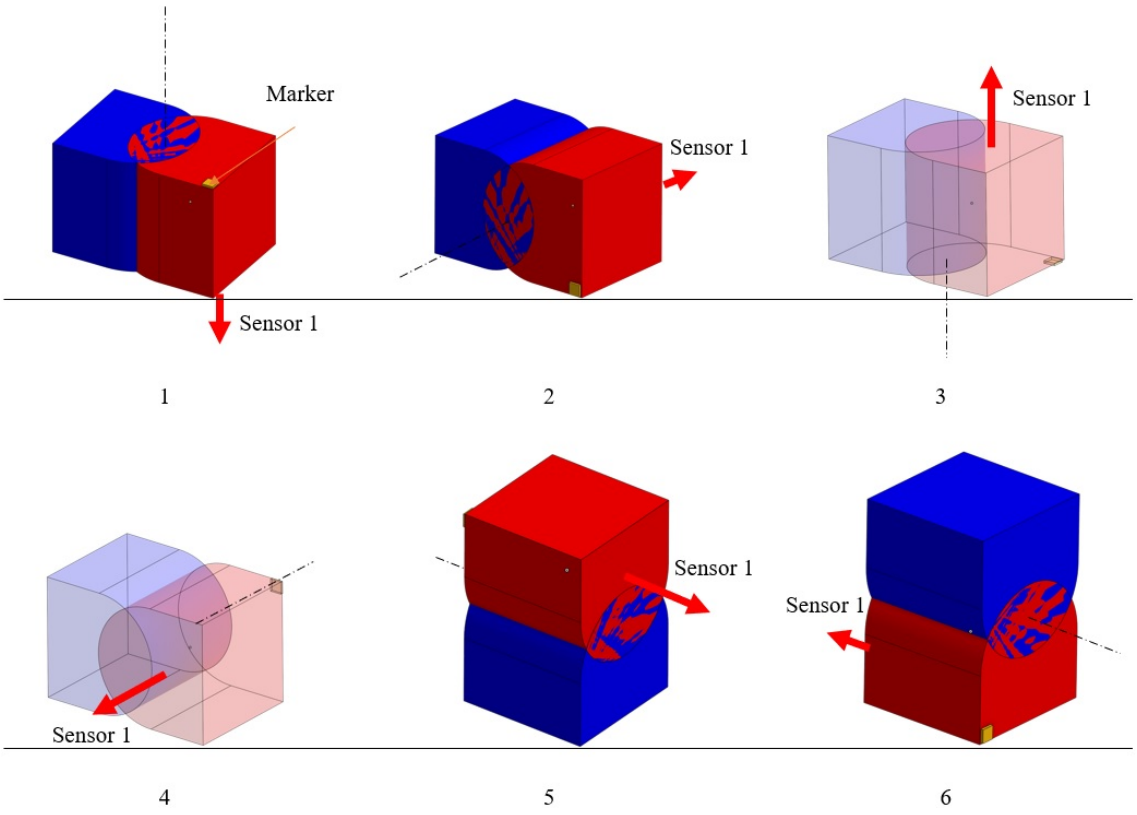


FIGURE 3.10. The Ex module can take six different orientations relative to the ground (horizontal plane). The orientation sensor identifies the module orientation.

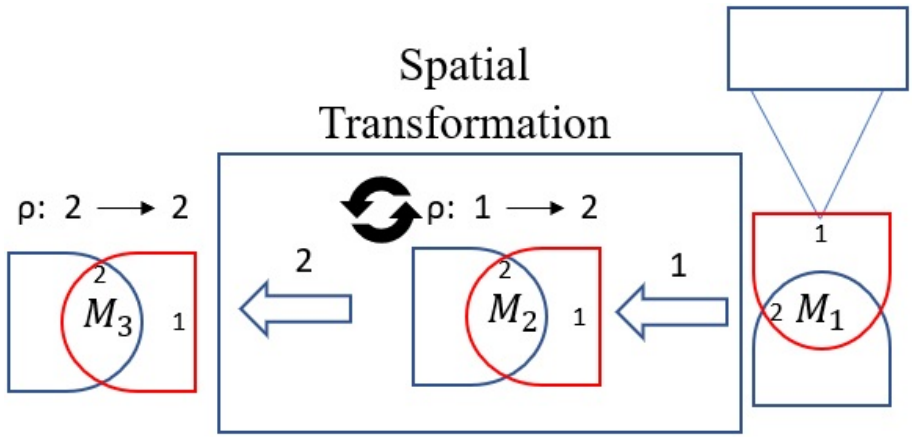


FIGURE 3.11. Messages are transformed given the spatial configuration of the modules in order to maintain spatial meaning. Numbers represent sensors around the module and also determine the position of the sensors in the vector S_i, Th_i and V_i.

different spatial orientation renders the incoming sensor information meaningless to the receiving module [112] (Figure 3.9). Therefore, the incoming message should be transformed using a spatial transformation function as in line 15. The transformation function $\rho_{i,T}(l)$ defines the transformation in terms of the module connections. For module Ex, the transformation function is a permutation matrix $\rho_{i,T} : \mathbb{N} \to \mathbb{N}\times\mathbb{N}$ that changes the order of values in incoming messages V_l (Figure 3.11).

Incoming messages (V_l) and the output of the module own sensors (V_i), like the orientation sensor, are then fed to the decision mechanism in order to change the CPG controller (Section 3.1.3).

- Propagation: In the propagation part, hormones are attenuated by using a proportional factor α which is based on the number of hops a given message can make before disappearing (Line 21 and figure 3.12a). If the highest sensor value in a given message is less than a threshold (ϵ) the message is eliminated from the module (Line 22).

  Only non empty messages V_l''' are propagated to the nearest neighbors. A message is only propagated to the modules it didn't come from, that is they are forwarded (Figure 3.12b and line 24). The two previous mechanisms (attenuation and forward propagation) prevent messages from hopping from module to module indefinitely.

In this way, a set of modules can use sensor information to adjust their behavior for their own specific needs. The combination of all the modules movements creates a coordinated movement in reaction to the sensor triggering (or all sensors that trigger at that moment). Due to their basic role in the movement of the robot, these message associated movements can be related to motor primitives [79].

Different robot configurations can have different sensor capabilities as sensors are arranged in different positions. For example, a one level morphology, that is a morphology

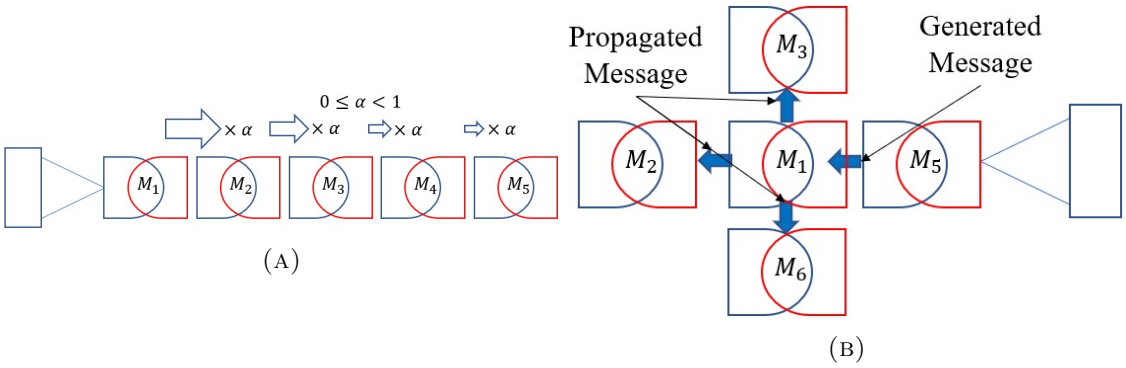


FIGURE 3.12. Hormone elimination mechanisms. (a) Hormones are attenuated as they hop from module to module by a factor α, and (b) are only propagated to the modules they don't come from.

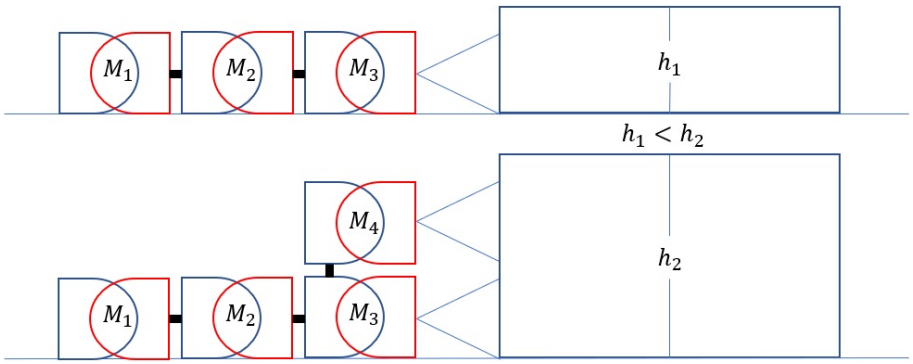


FIGURE 3.13. Different robot morphologies have different sensor arrangements that limit what can be sensed.

that has only modules touching the ground, can only see obstacles at its same level. This morphologies can not distinguish between objects of different heights (Figure 3.13).

3.1.3 Decision Mechanism

As mentioned in section 3.1.2, when a message is propagated through a robot morphology each module can interpret the message information and use that information in its own way. In this work, sensor information messages are used to change the module internal CPG parameters. A decision mechanism is then necessary to interpret the messages coming from different modules and to change the CPG controller parameters accordingly. There are three main issues that must be overcome by a decision mechanism:

The first one is related to the speed at which incoming messages arrive and the speed at which the decision mechanism reacts, or adapts, to arriving messages. As messages are generated in all modules of a module morphology, several messages can potentially be generated and propagated almost at the same time, arriving very close to one another. The actual speed depends on the specific type of implementation (communications latency, time step duration, processing unit frequency, etc.). Several messages arriving rapidly from

different sources means a very rapidly changing input signal. A very rapidly changing signal on the input of the decision mechanism creates jittery behavior on the output, if the decision mechanism changes the CPG parameters at the same rate. This jitter is observable even with the damping behavior of the CPGs, and can potentially damage the robot actuators.

To further prevent the jitter on the movement of the robot a filter should be implemented, either as part of the decision mechanism or as a pre-processing mechanism. As the filter prevents rapid changes in the incoming information, it also helps sort out which information is maintained in time, and is actually relevant to the robot movements, and which information disappears quickly, and can be considered noise. The input filter acts then as a first stage in sorting out the incoming information to set the controller parameters in a meaningful way.

The second issue is also related to sorting out relevant information for the movement of the robot. Each module in a modular robot morphology has its own spatial configuration. In this work the spatial configuration of the module is defined by the orientation of the module relative to the ground (Figure 3.10). Using this orientation as a way to differentiate among modules, the decision mechanism decides which incoming information influences the CPG parameters and in which way. Differentiation using spatial information helps each module specialize in a way that contributes to the overall robot locomotion. The use of spatial information to differentiate robots has also been used in similar works by specifying a measure related to the current morphology of the robot [36], e.g. the distance of the module to the center of the morphology. However this metric is difficult to define for unknown morphologies. In this work spatial information comes solely from what the module detects.

For example, a simple decision mechanism that solves the first two issues adds a small number to the current parameters for every message that arrives. The sum associated to a message tries to make the current CPG parameters go towards previously defined values. Over time the current CPG parameter values become the sum of all the contributions of all messages that have arrived. Summing small amounts to the control set-points also lets the contribution of each message be tuned by other variables like the distance measured by a sensor, i.e. by multiplying the sum by the measure. Using this approach the robot aggregates all the information coming in the form of messages to set the CPG parameters.

The sum decision mechanism has been used in previous work to manually design controllers that use both CPG and hormone inspired messages [76]. However, an automatic adaptation technique is used in this work instead of a manual process. Although the sum decision mechanism is capable of representing different movements that depend on sensor input, it presents some limitations when being automatically adjusted.

Consider the situation in which the robot tumbles and drastically changes its orientation. When a robot turns over, not only the meaning of the incoming sensor information can change, but also the role of each of the modules in the movement of the morphol-

ogy. The values toward which the small sums are working their way must be changed completely. This asks for conditional structures to be added to the decision mechanism to cope with changes in orientation of the robot. Thus, in the end the sum mechanism would look more like a computer program, with if statements for each different orientation a module can be in. An automatic adaptation technique has to cope with this and add the conditional structures from scratch, making the adaptation process more complex in turn. This illustrates the third issue for decision mechanisms: a representation that is simple enough to adapt automatically.

A different kind of decision mechanism simplifies the adaptation process by using a simpler way of representing different situations. This is the case with Artificial Neural Networks(ANN). Artificial Neural Networks can represent different function combinations by means of multiplication and sum operations, and have been used to control robots for different tasks [39]. Moreover, ANNs are able to represent the non-linearity present in most biological systems when combining movements, something that the sum decision mechanism lacks due to the exclusive use of linear combinations of set points. ANNs are represented by a directed graph in which nodes are connected by weighted edges ($w_{u,v} \in \mathbb{R}$ representing the weighted edge going from node u to node v). Each node receives inputs from other nodes and computes an unique output using a predefined activation function. Some nodes receive inputs from the outside world and are therefore called input nodes, conversely, some relay signals to the outside and are called output nodes. The way in which nodes are connected to each other is called the network topology and is usually defined in layers containing different numbers of nodes, e.g. a network has an input layer, an output layer, and intermediate layers, often called hidden layers. Let W be the set of all edges weights in a network topology, if the network topology is fixed an automatic adaptation technique only has to adjust the values in W to change the output of the network for a given input, this makes ANNs very simple to automatically adapt.

This work uses an ANN as the decision mechanism inside each module. The number of input nodes depends on the number of incoming messages and signals coming from other sensors. For example, in the Ex case, each module has an ANN that receives the sensor message information coming from other modules and from the same module, and the orientation of the module as inputs. As mentioned in the last section the sensor messages have to be adjusted to the spatial configuration of the module with respect to its neighbors. In the case of orientation sensors an input is defined for every distinct orientation relative to the ground (Figure 3.10). On the output side of the ANN are the parameters controlling the individual module CPG. As stated in section 3.1.1, the CPG output of module M_i is controlled using three set points: Amplitude R_i, offset X_i and phase differences $\Delta\phi_l^{(i)}$. Phase difference parameters match the number of connections to other modules, and thus other CPGs, that a module has in an specific configuration. In the case of having two connector faces a module has up to two independent phase difference parameters to deal with. In the Ex module, which has two connectors, its ANN has four outputs (Figure 3.14 for an example ANN). Neurons can have different types of activation functions but for

the sake of simplicity this work uses a sigmoid function centered in 0 (Equation 3.8), the output of any individual neuron goes from 0 to 1.

$$y = \frac{1}{1 + e^{-x}} \tag{3.8}$$

An explicit filter for incoming sensor messages is implemented in order to solve the first issue of decision mechanisms. i.e. the high rate at which sensor information messages arrive. Every incoming message is accumulated over a period of time, this is achieved by summing sensor values of the same type, that is, sensor values in the same position in the module (i.e. in the same position in the V array), and normalizing the sum (Algorithm 3). The resulting summed an normalized values are then fed to the ANN as inputs (Algortihm 4).

Algorithm 3 Hormone filter per time step.

Require: V_i, V_l, Buf

1: $C = 0$ ▷ C:Accumulation array
2: $c_p = c_p + v_p, \forall v_p \in V_i$
3: $r_c = 1$ ▷ r_c: Normalization counter
4: **while** $\exists V_l$ **do**
5: $\quad c_p = c_p + v_p, \forall v_p \in V_l$
6: $\quad r_c = r_c + 1$
7: **end while**
8: $C = C/r_c$
9: Buf = Buf $\cup\, C$

Algorithm 3 is repeated in every time step of the controller. An accumulation array C is used to store the summed sensor values (lines 2 and 5). After all sensor values have been summed the resulting value in C is divided by the number of summed messages r_c and stored in a buffer (Buf). The buffer is maintained over a predefined time window tw. When the time window ends, a similar normalized sum is used to accumulate all summed messages in the time window (Algorithm 4). This time a different array A is used to store the accumulated values (Line 8) in each time step in Buf (C), and is also divided by the number of arrays in the buffer r_a, A is then fed to the ANN inputs over the next time window.

Algorithm 5 defines the main decision mechanism procedure for every time step. The ANN receives the module orientation relative to the ground (ori) and the accumulated sensor information in A (after filtering incoming messages over a time window tw) and returns the CPG parameters.

3.2 Configurable Environment

Modular robots applications envision robots to locomote in different environments with vastly different features. For example, in space exploration robots have to tackle dirt,

Algorithm 4 Hormone accumulation algorithm per time window tw.

Require: tw, t
Buf $= \emptyset$
loop ▷ forever
  Buf $=$ Accumulate(Buf, V_i, V_l) ▷ Algorithm 3
  if $t\%tw = 0$ **then**
    $A = 0$ ▷ A: Accumulation array
    $r_a = 0$ ▷ r_a: Normalization counter
    for all $C \in$ Buf **do**
      $a_p = a_p + c_p$
      $r_a = r_a + 1$
    end for
    $A = A/r_a$
    Buf $= \emptyset$
  end if
end loop

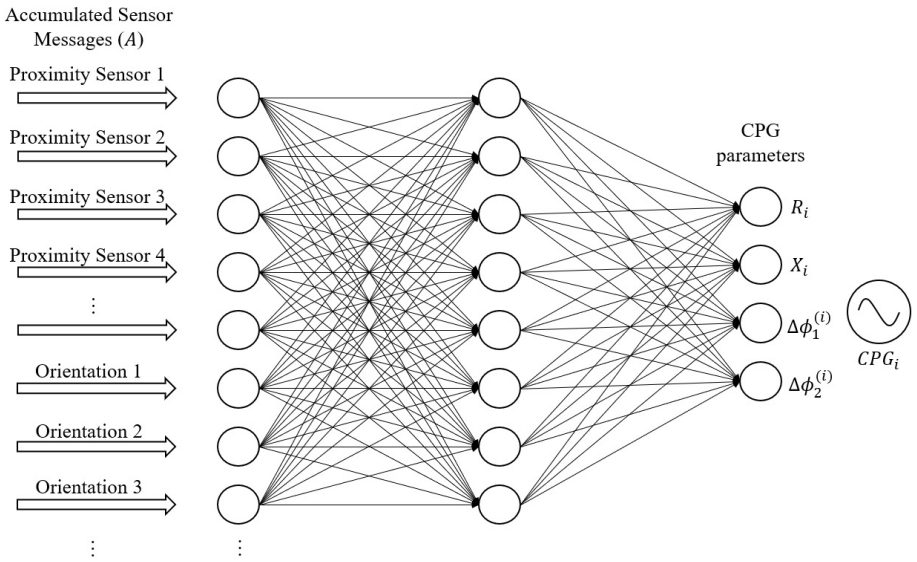


FIGURE 3.14. An ANN is used as decision mechanism taking incoming messages as inputs and CPG parameters as outputs. This example ANN matches the Ex module model.

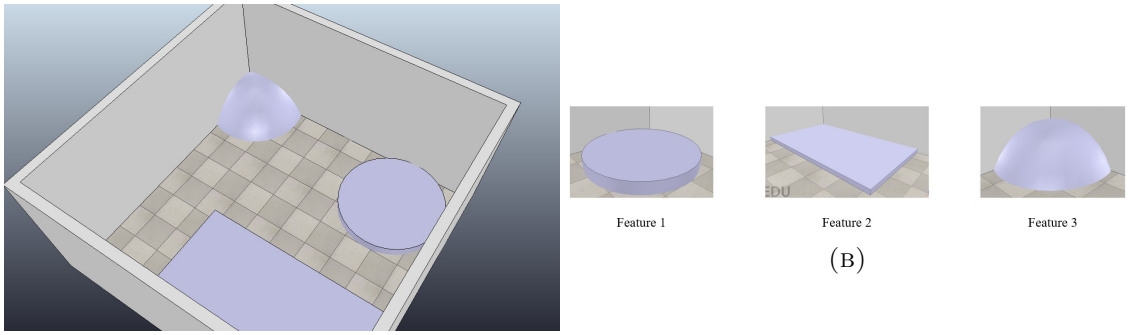
Algorithm 5 Decision Mechanism.

Require: tw, t, V_i, V_l,ori
loop ▷ forever
  $A =$ AccumulationFilter(V_i, V_l, tw, t) ▷ Algorithm 4
  $\langle R_i, X_i, \Delta\phi_l^{(i)} \rangle = ANN(A, \text{ori})$
end loop

slopes, rocks and other obstacles. Using the control strategy defined in section 3.1, a modular robot with sensors in all of its modules can detect and react differently to different parts, or features of an environment. However, it is not clear how the environment, or parts of it, should be presented to an adaptation process for the resulting robot to be able to locomote in all the environment. Several research works that involve robot systems adapting and learning to locomote use, most often than not, only a single part of the environment [5, 51, 82, 91, 94, 124, 135], the vast majority being flat surfaces [18, 19, 24, 52, 66, 67, 85, 103, 111, 119, 120, 121, 136, 137, 138, 140]. Few studies use rough terrains, made by randomly adjusting the height of parts of the terrain [28], and only some employ environments with multiple features [123].

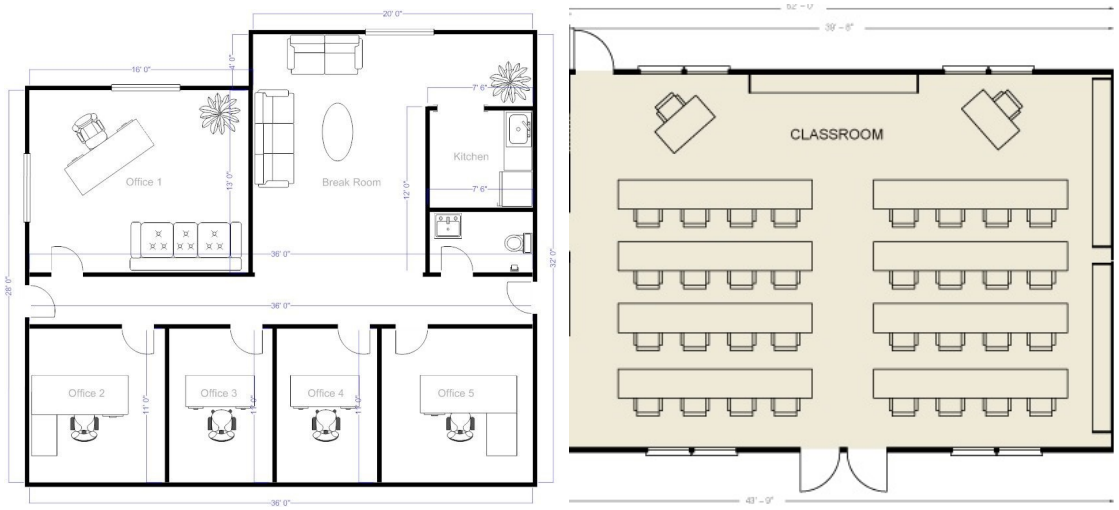
In order to give a robot trained with an adaptation process the ability to travel through different parts of the environment, this work proposes a configurable environment model. To simplify the discussion and application of the configurable environment model only structured environments are considered. In this work structured environments are defined as environments in which basic features can be separated from one another without ambiguity (Figure 3.15). Examples of structured environments include office or class rooms (Figure 3.16). This rooms can be separated into corners, straight corridors, stairs, doors, etc. In contrast, in an unstructured environment features like corners and slopes are usually merged together so much that they are difficult to separate from one another (Figure 3.18). Structured environments can be separated in several ways, however, in this work only 2D separation is considered, that is, environments are only separated on the ground plane, like separating parts of a map. The parts of the environment obtained are considered sub-environments which can present a robot with different kinds of scenarios. Separation of the environment by height is not considered in this work. Moreover extracted sub-environments are expected to posses the following characteristics (Figure 3.17):

- Each sub-environment must contain only one feature of the structured environment.
- Sub-environments should have explicit start and exit points. This points are used to connect different sub-environments to one another.
- Sub-environments should specify the direction of travel. i.e a robot can move over a step from bottom to top or vice versa. One of these two directions must be enforced to prevent ambiguities when analyzing results.
- To further specify the direction of travel sub-environments must have explicit boundaries (e.g. walls, pitfalls, etc)
- When connecting one sub-environment to another, using the exit point of one and the start point of the other, dimensions should match, i.e. the connecting sides must have matching dimensions.
- The general dimensions of the sub-environment should be adjusted for the size of the robot to be tested.



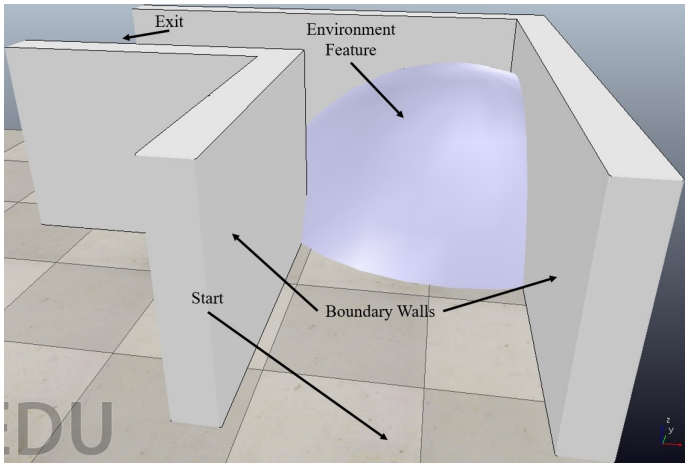
(A) (B)

FIGURE 3.15. A simple structured environment. (a) The full environment, (b) extracted features.

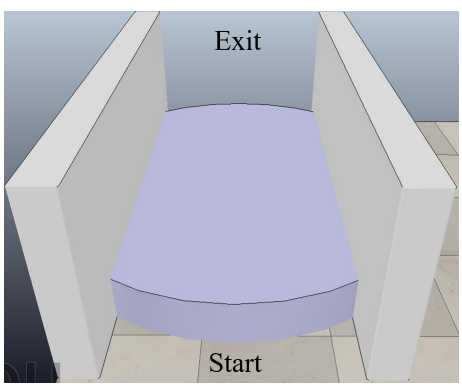


(A) (B)

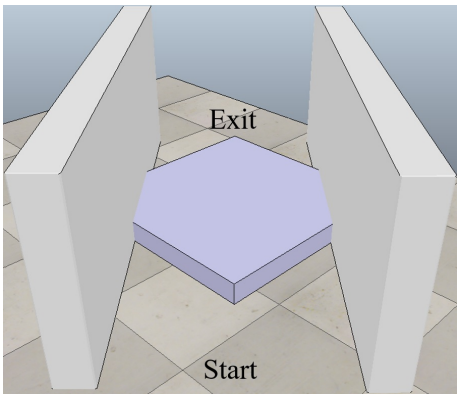
FIGURE 3.16. Examples of real world structured environments. (a) Office environment, (b) Classroom environment.



(A)



(B)



(C)

FIGURE 3.17. Sub-environments examples.

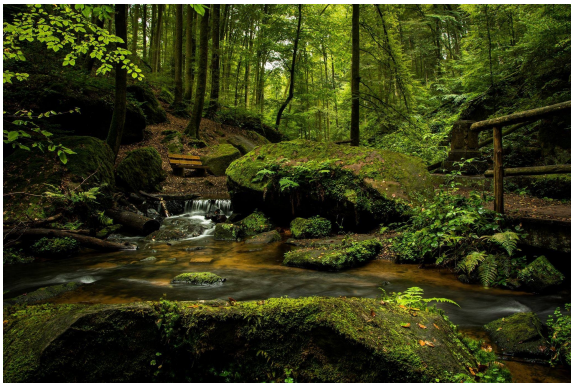


FIGURE 3.18. Example of unstructured environment (taken from pexels.com)

Sub-environments can be combined in different ways, depending on how they were separated, and additional sub-environments can be added, extending an already existing environment. For example, if an office is divided into corners and corridors these can be put together in different ways to form different configurations of an office space. The same doors, corridors and turns can be used to compose different office buildings. Another example includes piecing together different kinds of turns to form a race track. By combining different sub-environments more complex alternatives to flat surfaces can be presented to a robot and how the combination affects the robot movements. Another advantage of using sub-environments is that robot reactions and movement in each specific sub-environment can be studied independently.

Previous decomposition approaches focus on generating behaviors for specific tasks that are combined to solve a main task [60, 126, 61, 94, 108, 30]. In [60] Lee et al. use task decomposition to evolve controllers for pushing a box to a goal with a Khepera robot by evolving controllers for the sub tasks of getting to the box, circling the box and pushing the box towards a certain direction. Controllers are evolved independently for each task and a decision mechanism is needed for all the different evolved controllers to work as one [126, 61]. Sub-environments can be seen as an alternative to task decomposition.

Let E be an structured environment and $Se = \{se_1, se_2, \ldots, se_z\}$ the set of sub-environments that can be extracted from E (z being the number of sub-environments extracted). Environment E can be defined in terms of sub-environments as in equation 3.9, in which $x \oplus y$ implies that sub-environment y start point is connected to sub-environment x exit point, that means that a robot must start traveling environment E from the leftmost sub-environment all the way to the rightmost one.

$$E = se_1 \oplus se_2 \oplus se_3 \oplus \cdots \oplus se_z \tag{3.9}$$

A new environment E' can be built by changing the order of sub-environments or by adding more of the same sub-environments as needed (Equation 3.10).

$$E' = se_2 \oplus se_3 \oplus se_1 \oplus se_2 \tag{3.10}$$

It is expected that a robot trained in an environment formed by an specific order of sub-environments should be able to travel through any other environment formed by a different permutation of the same sub-environments (Figure 3.19). It is possible, however, for the specific order of sub-environments to affect the training process as different sub-environments could have different training difficulties, i.e. it is not the same to learn to go over an obstacle before learning to move straight than the other way around. This implies the possibility of there being an order that is best to train in, which is tested in later chapters.

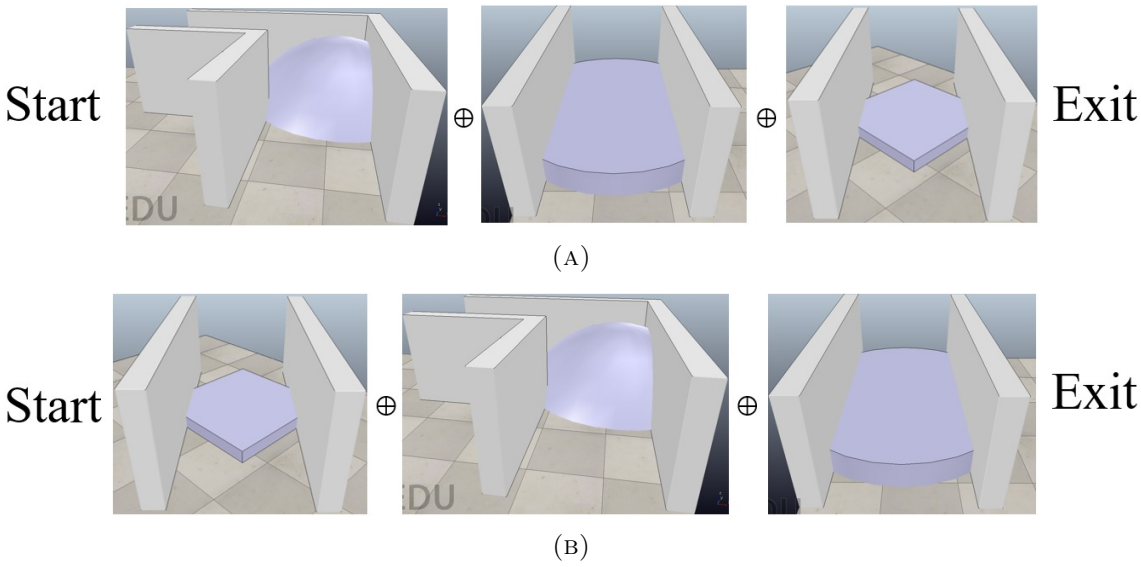


FIGURE 3.19. Environments formed by different permutations of the same sub-environments.

3.3 Adaptation

Manually designed controllers that make use of the coordination, sensor handling and decision mechanisms of the control strategy proposed in section 3.1, can be generated for each robot morphology that needs to move. In a manual design ANN weights are adjusted by a process of back propagation in which different examples of how a robot should react to incoming sensor information, according to a human designer, are used. However, this is not a simple process for two reasons: First, since the robot can have an arbitrary morphology, the designer could not be completely aware of what moves/examples to present to the back propagation process. Second, the designer needs to provide examples to each module in the morphology, which can be many. Thus, setting the controller manually is a daunting task for non-trivial robot morphologies. For this reasons, an automatic adaptation technique that allows the robot to produce movements without regard to its morphology (shape and number of modules) is preferred. The main objective of the adaptation technique is to obtain a set of ANNs that configure the parameters of the individual modules CPG controllers given the sensor inputs arriving from other modules and the module itself, in order to let the robot locomote from the starting point to the exit point of an environment.

The main adaptation process can be external to the robot, as in optimization algorithms, or internal, as in non supervised and on-line approaches. In this work a process external to the robot, evolutionary algorithms, is chosen as the automatic adaptation technique. The decision to choose evolutionary algorithms is motivated by the simplicity of the implementation on one side, as evolutionary algorithms use only a global performance measure of the task to be optimized. In other techniques, like reinforcement learning, assigning a significant reward signal to each module is a difficult problem in itself [18]. On the other side, evolutionary algorithms are chosen because of the curse of dimensionality:

In the worst case scenario the adaptation technique has to adapt independent ANNs for each module in a morphology, thus the number of parameters to optimize escalates proportional with the number of modules. A population based technique like evolutionary algorithms can better tackle this kind of high dimensionality problems.

Evolutionary algorithms are population based optimization methods that follow a performance measure or fitness [26]. A population of different solutions for a given problem is generated and tested against the performance measure. Then, selected individual solutions are modified in different ways to produce new solutions that will hopefully have a better performance measure than the originals. The processes of selecting and modifying solutions in the population is usually inspired in the biological processes that take part in natural evolution. These processes include mutations, crossovers and competition.

Algorithm 6 Evolutionary Algorithm

Require: E, R

1: Generate an initial population of $\hat{W}$s ▷ Randomly or from a seed
2: Evaluate the fitness of all $\hat{W}$s using E and R
3: **repeat**
4: Select $\hat{W}$s from the population ▷ Using fitness
5: Generate new $\hat{W}$s ▷ Applying genetic operators to old $\hat{W}$s
6: Evaluate the fitness of the new $\hat{W}$s
7: Replace $\hat{W}$s in the population by the best new $\hat{W}$s
8: **until** Some stopping condition
9: Return the best $\hat{W}$ in the population

Algorithm 6 describes the main parts of the evolutionary algorithm used. The weights of all module's ANNs in the robot morphology are stored as an array of real numbers $\hat{W}$, encoding a solution. The fitness of each solution $\hat{W}$ is evaluated using a measure of the distance traveled by robot R in environment E, that will be fully defined on chapter 5. In the end, the evolutionary process returns the solution $\hat{W}$ with the best fitness when the stopping condition is met.

Using evolutionary algorithms and the sub-environment approach to train the control strategy proposed in section 3.1, a robot that travels through different environments can be obtained. This approach also allows the robot to adapt to the obstacles it finds in the environment through the use of its sensors. The task to be adapted does not amount to navigation as the robot still has to steer to go in one of the two directions and avoid crashing into the different walls and obstacles, but it does not have to decide among different paths.

The generalization capacity of a trained robot controller, that is the capacity for performing well in tasks and environments in which it was not trained, is measured with the help of the sub-environment approach. A robot controller trained in one or various configurations of the environment is tested in other configurations that it has not seen before. If the controller does well in this new, unseen configurations it is said to generalize well (Figure 3.20). A difficult part of measuring generalization in this way is to determine how

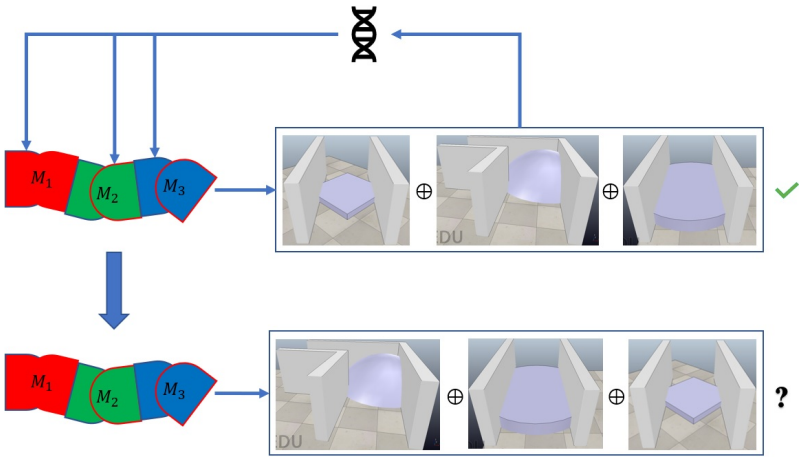


FIGURE 3.20. A robot controller that can locomote in configurations of the environment that it has not been evolved in, is said to generalize well.

many features/configurations of the environment should be used in the training phase for a trained robot controller to generalize better, this can be solved by incrementally adding features to the environment. However this is out of the scope of this thesis.

Algorithm 7 describes the general training process. Environments (E) in this work are considered permutations of sub-environments Se. Robots are then tested in permutations different from the E that was used for training.

Algorithm 7 Main adaptation process.

Require: Structured Environment,Robot$R$

1: Extract sub-environments $Se = \{se_1, se_2, \ldots, se_z\}$
2: Select sub-environments for training ($E = se_1 \oplus se_3 \oplus \ldots$)
3: $\hat{W}$ = evolve(E, R) ▷ Algorithm 6
4: Test a robot using $\hat{W}$ in environments with different permutations of Se
5: Return $\hat{W}$ that performed best

3.4 Conclusion

In this chapter a locomotion training framework for training controllers for modular robots that lets them travel from one point to another in different environments is presented. First a coordination mechanism using CPGs and a way to respond to sensor information, using a variation of Hormone Inspired Messages, as well as an ANN decision mechanism, are defined and used as a basic control strategy. Structured environments and sub-environments are introduced as a mechanism to show different features of an environment, which can trigger different sensor responses in a robot. Evolutionary algorithms are used as an adaptation technique to train the robot controller to move in the obtained sub-environments. Modular robots using the defined locomotion training framework are expected to generalize in environments that possess the same features but in a different arrangement. Next

chapter will describe the modular robot platform used to test the control strategy and the training methods.

CHAPTER 4

The EMERGE Modular Robot

4.1 Introduction

Testing robots with different morphologies in reality involves a lot of effort and resources. Experiments can take advantage of modular robot systems to quickly do tests in real life. The advantage is that simulated morphologies can be very quickly assembled using real modules, that can be reused. However building the modules of an existing modular robot system can be a complex and expensive process. For a start, every modular robot platform uses a different design for its modules (see chapter 2), which are not usually immediately available. Even when having the designs, building modules is complex and time consuming, specially if self-reconfiguration capabilities are present, which often make use of specialized mechanisms and materials. The introduction of rapid prototyping technologies such as 3D printing has helped reduce the costs associated with building specialized parts, but it can still take a lot of time to print every part of a robot and assemble sensors and actuators into it [3, 64]. Additionally, automatic connection mechanisms not only increase the weight and energy consumption of the module but also take a lot of time to switch between connected and disconnected states, making self-reconfiguration a cumbersome process. Simpler manual connectors, like in [28], still make use of moving parts and screws that increase the time to complete an assembly. Apart from the complexities of the building process not all existing modular robots possess sensors to detect their environment or, if present, sensors are used almost exclusively to provide a feedback loop for reconfiguration processes. This chapter describes a new modular robot platform, the EMERGE (Easy Modular embodied Robot Generation) module, which aims to simplify the module building process while at the same time increasing the system reconfigurability in a practical way, and which includes sensors with the purpose of the detecting its surroundings and studying how sensor information must be used in a modular robot to enable different morphologies to travel through different environments.

EMERGE modules are designed to be easily built using relatively cheap, commercially available components and their hardware design files are open for anyone to use and modify [1]. Modules are easy to assemble to other modules, using passive magnetic connectors, so that different morphologies can be assembled, tested, and reconfigured quickly in reality. Magnetic connectors are present in four faces of the module which allow the EMERGE platform to build more complex morphologies with more capabilities than similar open modular robot platforms [55]. Proximity sensors are embedded in the module faces and each module can communicate with the others, giving EMERGE the capability to implement and run the model described in chapter 3.

By taking advantage of the fast connection feature and other physical properties of the EMERGE module, external agents, like a robotic manipulator, can be used to automate not only the assembly but also the disassembly of morphologies. This in turn enables the possibility of completely automating experiments, increasing the rate at which experiments can be made in reality, and making EMERGE an ideal platform for fields like evolutionary robotics, in which a great number of tests are necessary. The EMERGE platform has been designed and implemented in collaboration with Ceyue Liu from the China University of Mining & Technology, Beijing, Andres Faina and Frank Veenstra from IT University of Copenhagen, and Henry Hernandez from Universidad Nacional de Colombia.

4.2 EMERGE

The EMERGE module design adheres to the following design objectives:

- The module parts are simple and off-the-shelf.
- The mechanical and electronics designs are easy to assemble.
- The modules are easy and quick to reconfigure.
- The module's design is open for anyone to use and modify.

EMERGE is a chain type modular robot with a semi-cubic shape in which two halves of the cube are joined together by a central hinge, resulting in one rotational DOF. Among the different types of modular robot systems, chain type modular robots exceed in their ability of generating high torque movements, relative to their mass, and can mimic limbs and other structures useful for locomotion. EMERGE modules are designed around a commercial servo motor, the Dynamixel AX-12A (Figure 4.2), this servo motor has a generic chassis used by similar models which makes easy to swap the motor for a more powerful one if needed. The position of the motor in the module also makes the design easy to be modified in order to use other kinds of servo motors. Often, modular robots include elaborated mechanical connectors to enable modules to connect to each other, however this

[1] https://sites.google.com/view/emergemodular

TABLE 4.1. Magnet properties

Quantity	**Value**	**Units**
Diameter	12.7	mm
Thickness	3.175	mm
Magnetization	N42	–
Br(Residual Induction)	1.32	Tesla
Force on contact(Steel Plate)	28.64(2.92)	N(Kg)
Material	Neodymium	–

can increase the module's weight and make the module building process more complex (Chapter 2 and section 4.4). To enable the easy assembly and reconfiguration of modules by an external agent, EMERGE uses permanent magnets to mechanically bond connectors. Four cylindrical neodymium magnets of 12mm of diameter (Table 4.1 for magnet properties) are arranged in a cross configuration in each connector. Magnetic bonds have the advantage of being strong enough to maintain the robotic structure integrity, but can also be easily disconnected by using the right movements or tools.

As a consequence of using magnets, connector faces in the module are divided into male and female faces (Figure 4.4). Magnets on the male face are arranged to attract magnets on the female face. Taking advantage of this gender differentiation, and in order to strengthen the connectors, magnets are housed in a 3D printed layer with special features. In female faces, the 3D printed layer has four holes positioned along the face diagonals in an X pattern. Male faces have protrusions that match the holes of the female faces. Protrusions and holes in the mating connector faces prevent magnets from sliding in the face plane, which is one of the main weaknesses of magnetic bonds. With this design, joint faces can endure up to 1.3 Nm of torque and 88.29 N of pulling force before disconnection.

The number and positions of connectors in a module determine the type of morphologies that can be built. To enable the assembly of 3D morphologies, and not only planar ones, EMERGE has four connecting faces, three of them contiguous and perpendicular to each other, fixed to the motor output shaft, and the remaining one fixed to the bottom of the motor (Figure 4.1). All faces are attached to the servo motor by using brackets and all the faces that would complete the cube are purposefully left empty to avoid collisions between parts of the module. Limbed and multiple layer morphologies as well as snake like morphologies are possible with this platform (Figure 4.3a). The mating connectors restriction and their organization results in assemblies that are organized as trees, that is, each module has a parent, or root, module and a root module has up to three children, or leaf modules (Figure 4.3b). When connecting to a parent module, a leaf module can connect in four different orientations relative to the center of the connector faces (Figure 4.5). Each orientation is separated by a 90 degrees rotation from the last one.

A PCB (Printed Circuit Board) layer is designed to be placed between the 3D printed connection faces and the Dynamixel brackets on each connection face. The PCB layer and

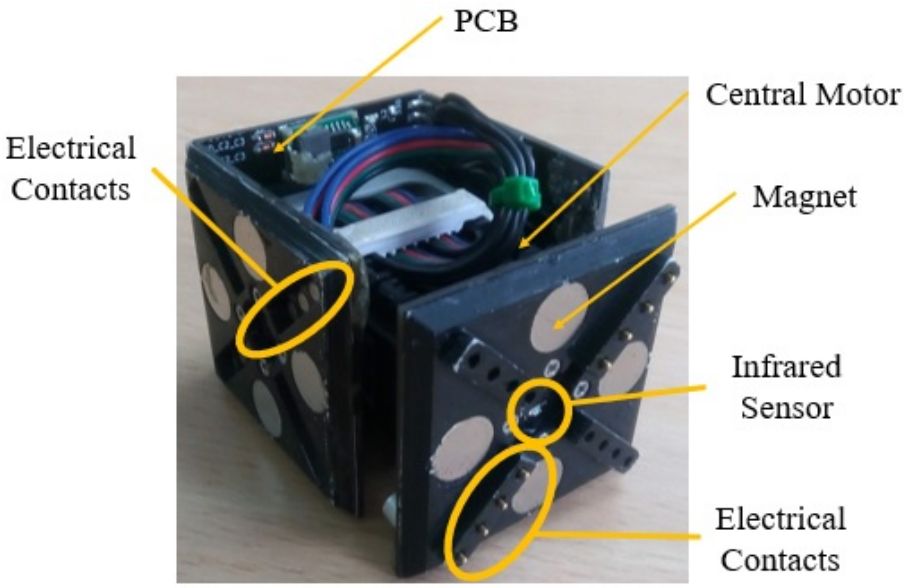


FIGURE 4.1. The EMERGE robotic module.

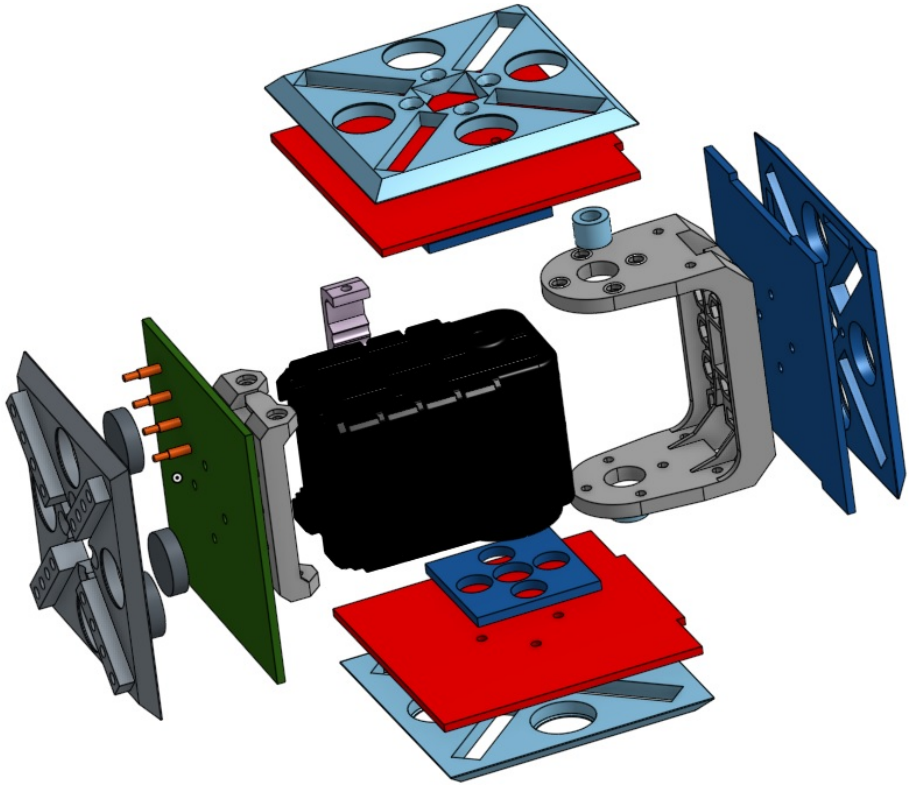
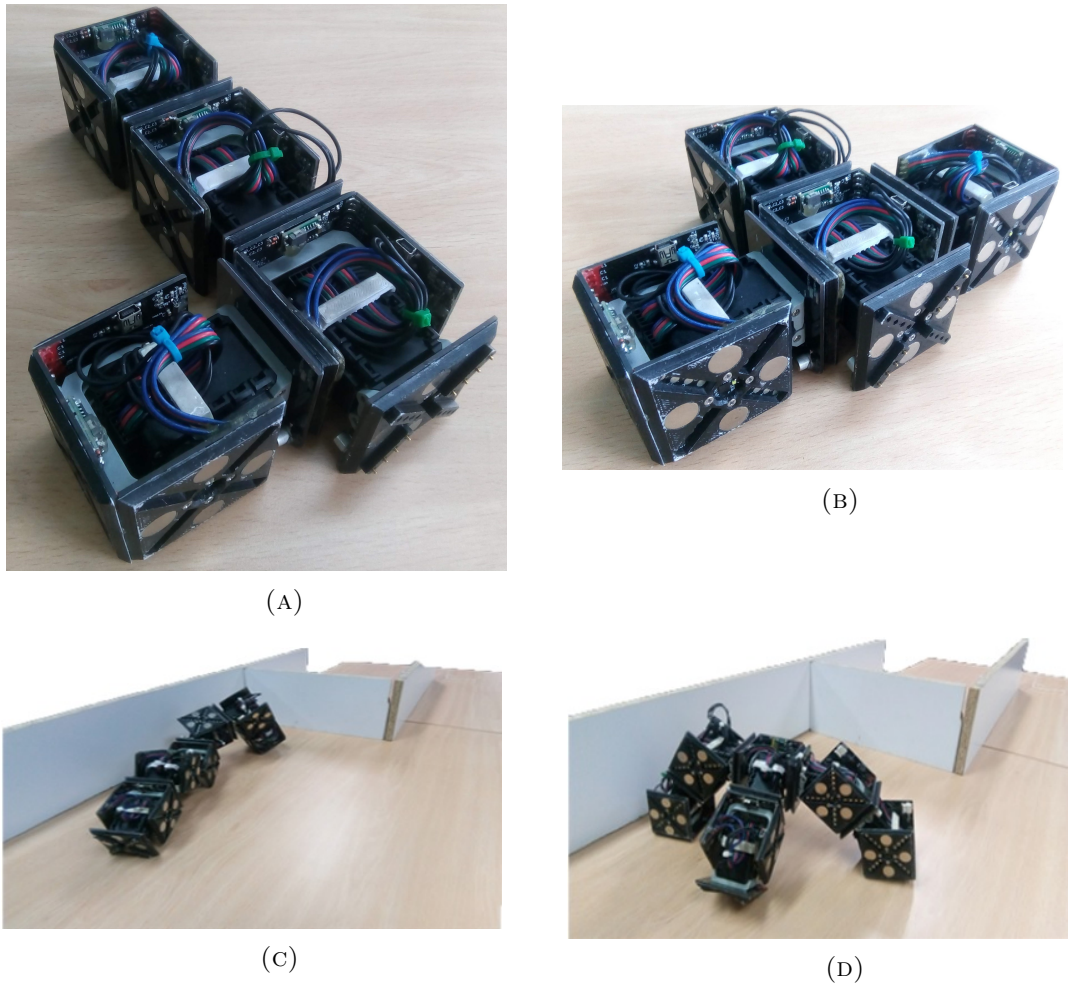


FIGURE 4.2. Exploded view of the module.



(A) (B) (C) (D)

FIGURE 4.3. EMERGE morphologies connect like tree structures: (a) L shaped morphology. (b) T shaped morphology. (c) Snake morphology. (d) Tripod morphology.

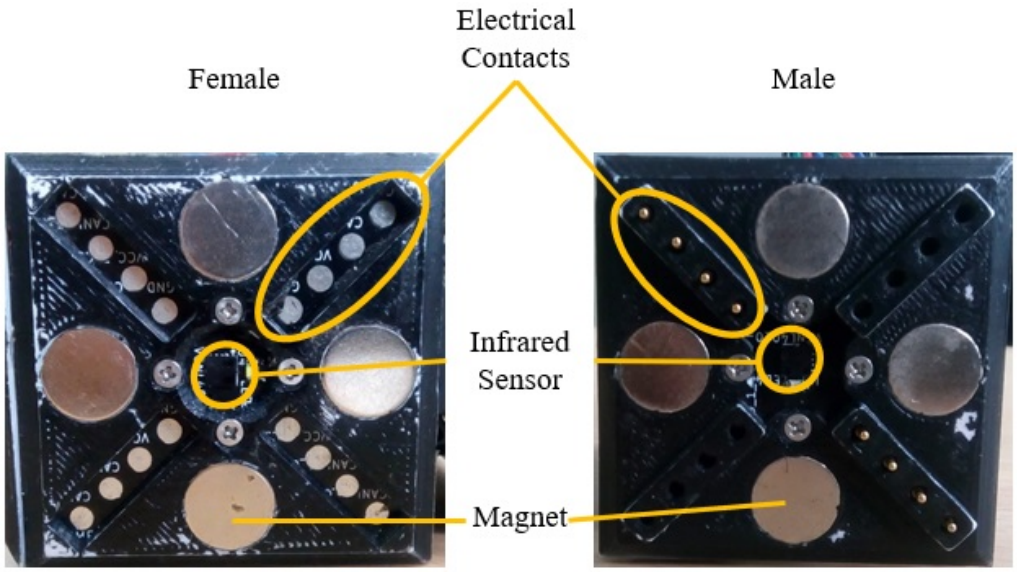


FIGURE 4.4. EMERGE module connector faces: (Left) Female face, (Right) Male face.

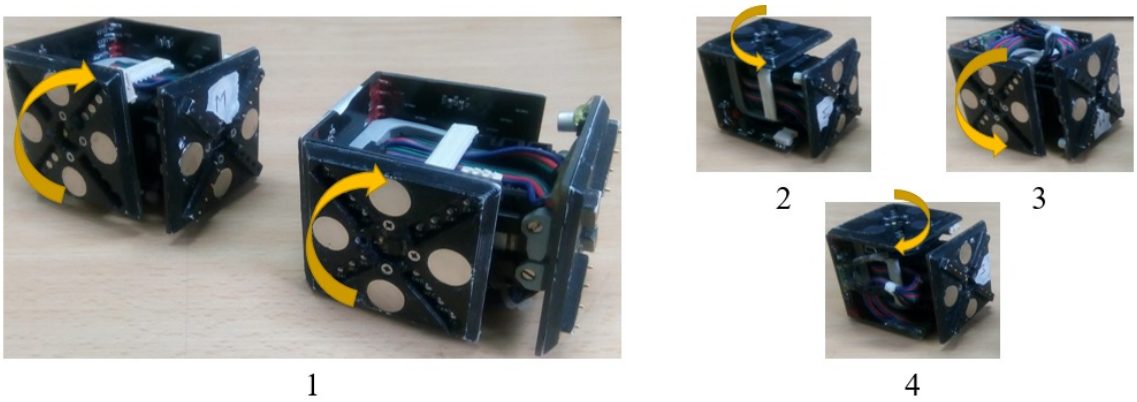


FIGURE 4.5. A leaf module (Left) can be connected in four different orientations relative to its root connector (1-4). These orientations can also be differentiated by the central actuator's rotation.

other electronic parts were designed by Henry Hernandez from Universidad Nacional de Colombia. The PCB layers provide a way of organizing the electrical connections inside the module and, at the same time, a place where electronic components, like microprocessors and sensors, can be soldered to. For example, spring loaded pins are soldered to the male PCB layer and are housed by the 3D printed layer, while corresponding pads are placed on the female PCB in the matching holes. Infrared sensors (Vishay VCNL4010) are soldered to the outer face of the PCB layer and measure distance to external obstacles through a special window at the center of the 3D printed layer (Figure 4.4). To minimize cables even more, PCBs below the three contiguous female faces are connected to each other through contacts that run along their touching borders (Figure 4.6). A PSoC (Programmable System on Chip) microcontroller (32 bit ARM Cortex M PSoC from Cypress), housed on the inside of the module, reads the sensors through an I2C bus. The microcontroller also controls the servo motor through a half duplex serial interface and communicates with the other modules via CAN bus. Electrical power and CAN signals are routed through the contiguous PCB layers, through cables to the remaining face, and onto the four spring loaded pins and pads in each of the connector faces. Modules share electrical power and communicate through these pins and pads on the connectors so only one module should be connected to an external power source.

A simpler version of the electronic design is also implemented in some modules. In this version, the PCB layer only contains tracks to route the half duplex serial connector of the Dynamixel motor (only 3 spring loaded pins are needed). Neither sensors nor microcontroller are present. Module movements are controlled externally by communicating with the servo motor internal microcontroller. Table shows the most important physical features of the EMERGE module as well as the most relevant electrical and mechanical ratings.

Some limitations of the EMERGE design include the lack of internal power supply or batteries, however a special kind of battery module has been developed by a student at the IT University of Copenhagen for the EMERGE platform. Thus module morphologies

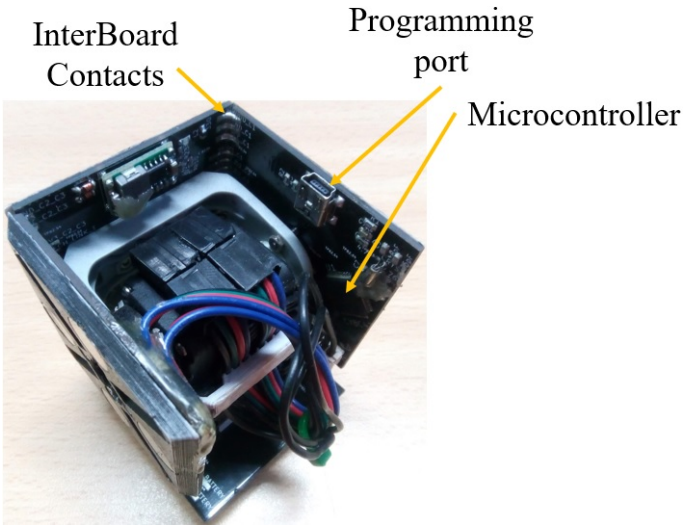


FIGURE 4.6. PCB below the three contiguous female connectors.

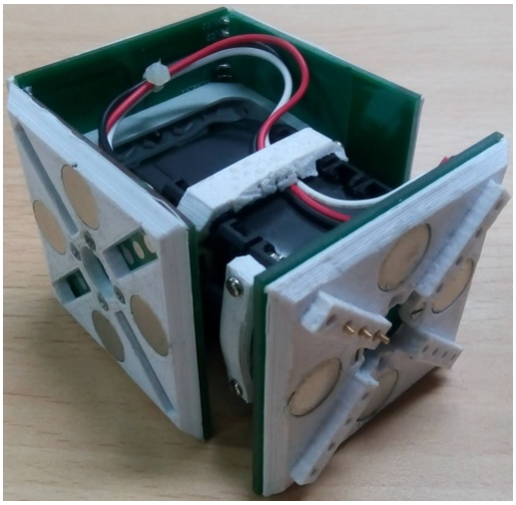


FIGURE 4.7. Simple version of the EMERGE module.

TABLE 4.2. EMERGE module features

Parameter	**Value**
Module weight	0.206(Kg)
Max Motor torque	1.5(Nm)
Dimensions	6.1x5.5x8(cm)
Infrared max detection distance	20(cm)
Idle current consumption	80(mA)
Whole module operating voltage	11.7(V)
Intermodule Communications	CAN, Serial half duplex
Max connector pull force	88.29(N)
Max connector torque	1.3(N.m)

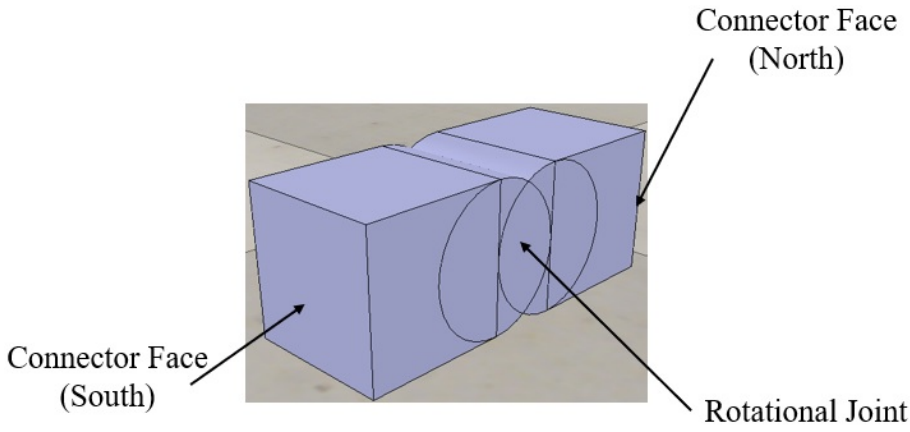


FIGURE 4.8. The abstract module is made of two semi-cubic parts linked by a rotational joint. Each part has only one connector face.

must be connected to an external power source. An orientation sensor is also missing but can be made by attaching a small helper module to one of the connecting faces and onto the CAN bus. Infrared sensors in male faces also present a problem in which the 3D printed protrusions reflect light coming from the sensors and trigger ghost readings.

4.3 Simulation

Two versions of the EMERGE module are used to implement and test the locomotion framework proposed in simulation (Chapter 3). The first more abstract version has been implemented to test a module comprised of two halves and a central rotating joint and was defined before the EMERGE design was complete. The second version implements the EMERGE design final features and provides a more realistic version of the modules. Both are implemented using the V-REP simulator [93].

4.3.1 Abstract Module

The first simulated version abstracts the main features of the EMERGE module. The simulated module itself is only composed of two semi-cubic halves, one rotational joint and proximity and orientation sensors. Only two faces, of all module faces, are used as connectors, one per each semi-cubic half. The two connection faces are located directly across each other (Figure 4.8) and allow modules to build simple chains. Six infrared proximity sensors are placed on six distinct faces of the module (Figure 4.9) facing outwards. Each sensor with a maximum measuring distance of 20cm and a minimum of 1cm. The central rotational joint is controlled by a proportional controller. The detailed properties of the simulated module can be seen in table 4.3.

When forming chains, as in the real EMERGE platform, modules are only allowed to connect in an specific order. Since there are only two connector faces in the simulated

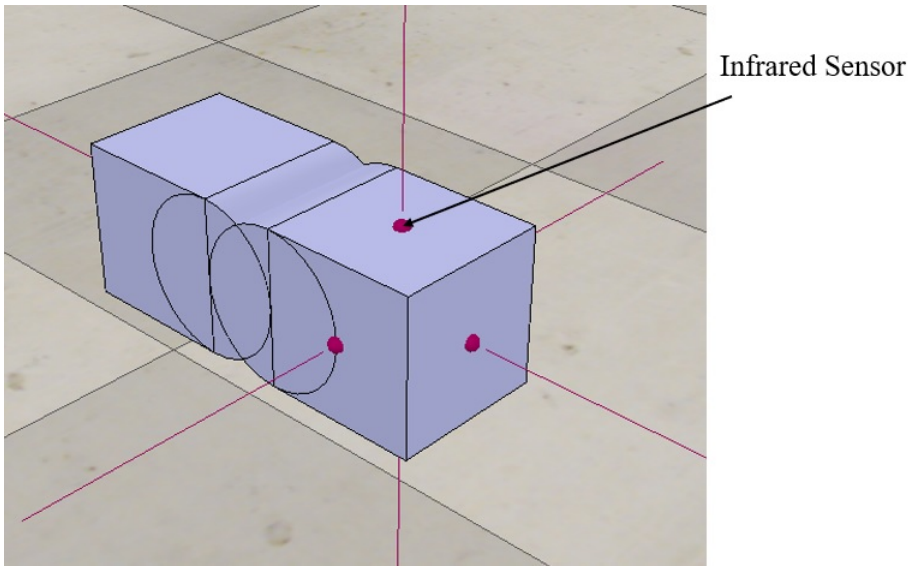


FIGURE 4.9. Six infrared sensors are located in six distinct faces of the cubic module.

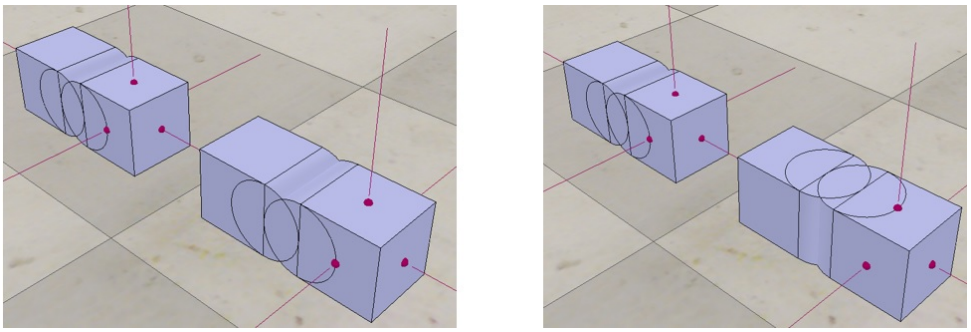


FIGURE 4.10. The abstract module can be connected to other modules only in two different orientations: (Left) with its rotational axis parallel to the ground, (Right) with its rotational axis perpendicular to the ground.

module one of them is labeled north and the other south, as in magnetic connectors (Figure 4.8). Only north faces are allowed to connect to south faces and vice versa. Connections between modules are performed using force sensors inside V-REP, this special kind of joint can be set to break when forces and moments affecting it go over a certain threshold. Modules can connect to each other in two orientations with the module's actuator rotational axis parallel to the ground, or with the module's actuator rotational axis perpendicular to the ground (Figure 4.10). Communication is only allowed between connected modules (nearest neighbors).

Abstract simulated modules leave out many features of mechanically realistic modules. These features include the fitting of the motor, where sensors would be attached to, and how different connector faces are placed around. Still, it provides a very simple first approximation to a one DOF rotational cubic module that can interact with different obstacles in the environment. Another distinctive feature of the abstract module is that it has six sensors as opposed to the real EMERGE module. This is also a consequence

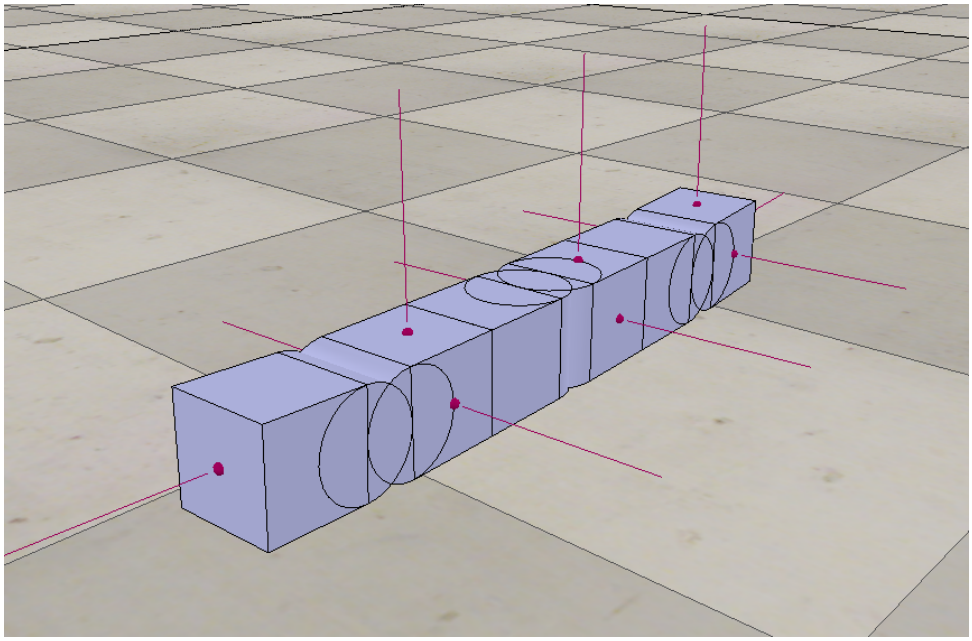


FIGURE 4.11. Snake like configurations made by connecting several abstract modules.

TABLE 4.3. Abstract simulated module parameters

Parameter	**Value**
Module mass	0.14(Kg)
Max. Joint Torque	0.726(Nm)
Central Joint P parameter	0.1
Dimensions	10x10x25(cm)
Dimensions (half)	10x10x15(cm)
Infrared max detection distance	20(cm)
Infrared min detection distance	1(cm)

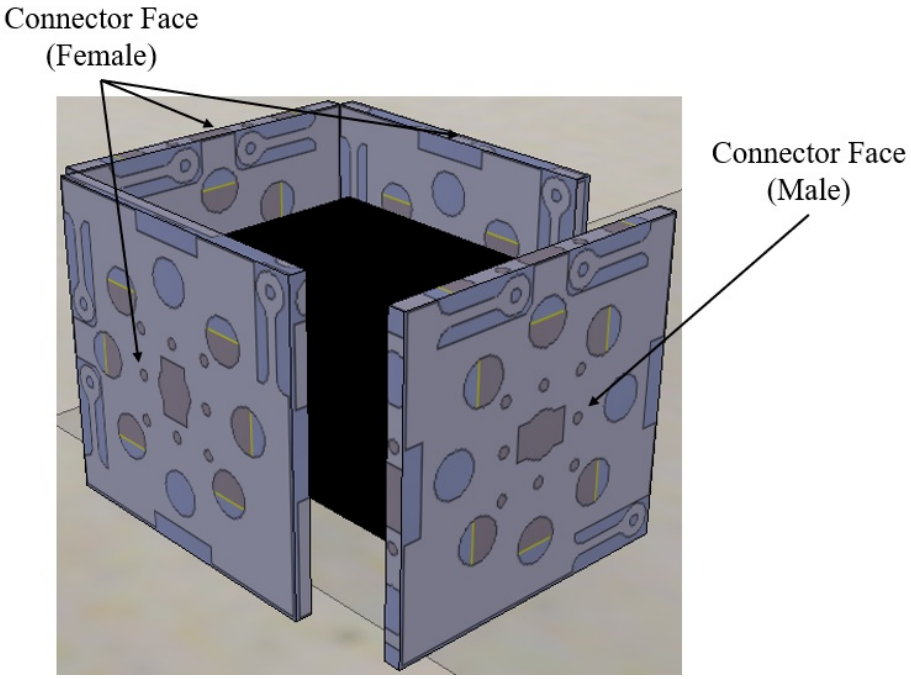


FIGURE 4.12. The realistic module has four connection faces, three of them perpendicular and contiguous to one another as in the real EMERGE module.

of feature abstraction, which allows sensors to be attached to any place in disregard of realistic mechanical restrictions.

4.3.2 Realistic module

The second simulated module provides a more realistic depiction of the EMERGE module. Realistic modules have four connection faces and their shape includes the servo motor enclosure as well as empty faces to let the central actuator rotate without parts colliding with one another (Figure 4.12). As with the real module, this simulated module is capable of building more complex morphologies than with the abstract module (Figure 4.13). Table 4.4 shows the realistic simulated module main physical properties which mimic the real EMERGE module features. The central rotational joint is controlled by a proportional, integral controller.

Connector faces are organized as in the real EMERGE module: three female faces are attached to the motor output shaft and the male face is fixed to the bottom of the servo motor body. The connection order enforced by mating connectors (only male-female connections are possible), coupled with the different orientations that modules can take relative to one another when connecting (Figure 4.5) enable the same kind of tree like morphologies possible with the real module (Figure 4.14). Connections are also carried out by using force sensors in V-REP.

The realistic simulated module uses the same kind of infrared proximity sensors as the abstract module, but this time only four sensors are placed only at the center of connection

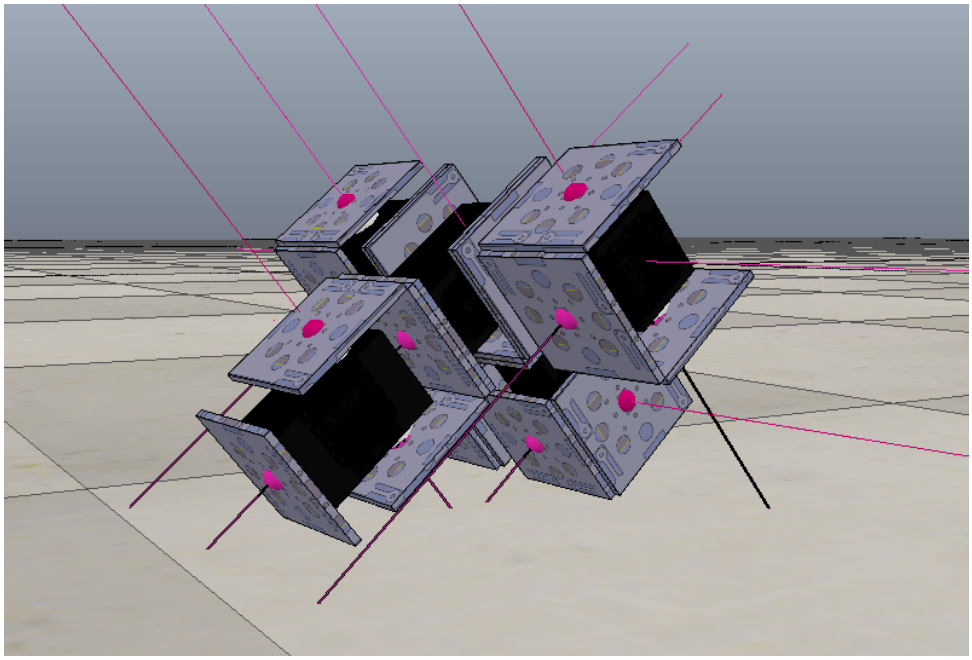


FIGURE 4.13. Limbed and multiple layered configurations made by connecting several realistic modules.

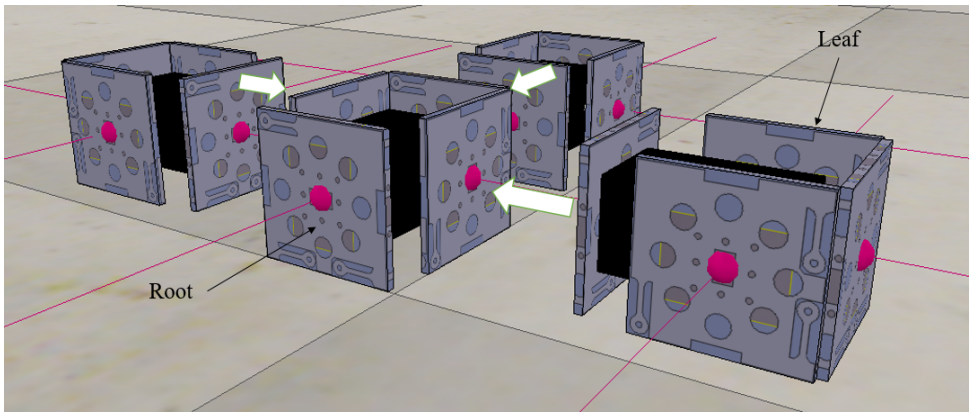


FIGURE 4.14. The gender restriction results in assemblies organized as trees. There is a root module to where other modules connect.

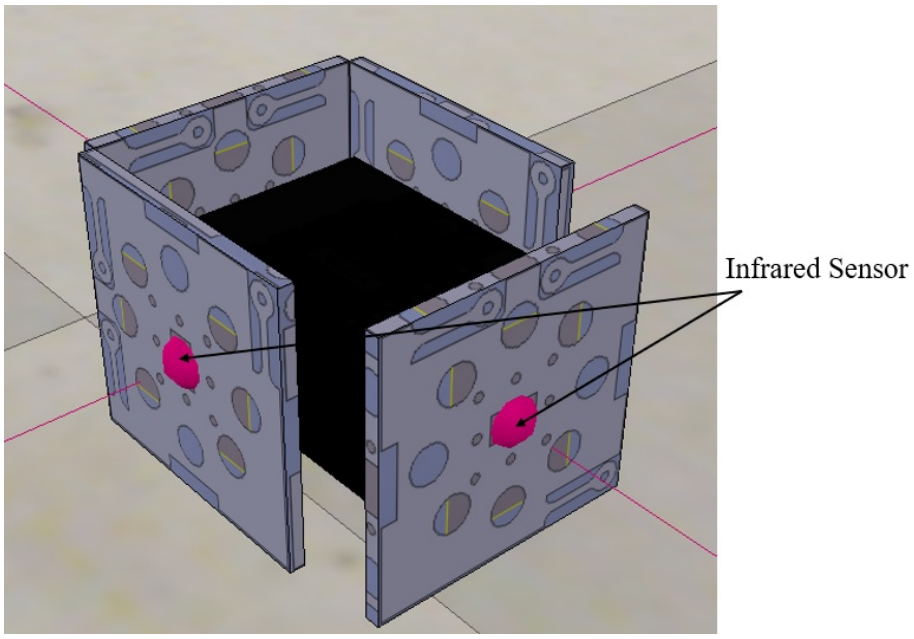


FIGURE 4.15. The realistic module uses four infrared proximity sensors, as in the real EMERGE module.

TABLE 4.4. Realistic module parameters

Parameter	**Value**
Module mass	0.165(Kg)
Max. Joint Torque	1.5(Nm)
Central Joint P parameter	0.1
Central Joint I parameter	0.01
Dimensions	6.1x5.5x7.7(cm)
Infrared max detection distance	20(cm)
Infrared min detection distance	1(cm)

faces, as in the real EMERGE module (Figure 4.15). Communication is also performed only between adjacent modules, that is, only between nearest neighbors.

4.4 Reconfiguration

This section was done with the help of Andres Faina, Frank Veenstra, David Silvera, Julian Franco, Oscar Gracia, Ernesto Cordoba and Jonatan Gomez, the paper can be found in [73]. Ideally modular robots are able to reconfigure its modules to readjust its morphology and adapt to a variety of tasks. Currently there are two alternatives for reconfiguration: Manual reconfiguration and self reconfiguration. Manual reconfiguration is by far the most used method to reconfigure modules [68, 28], however it requires an operator, which reduces the autonomy of the robot. Self-reconfiguration enables the modules to autonomously disconnect and reconnect from the main robotic structure [130, 141, 113]. However, self-

reconfigurable robots still have several deficiencies and have only been investigated in mock-up experiments in laboratory conditions [133, 38].

Deficiencies include weak connection mechanisms and heavy modules, which are a consequence of active parts in the connection mechanism, additionally self-reconfiguration involves complex algorithms and movements of individual modules that are difficult to generate and that require specialized sensors to check whether two modules are really connected [132]. This is specially complex in chain type modular robots, like EMERGE, where a solution should check that the formed chains are feasible[134, 114]. This work proposes an automatic reconfiguration method for the EMERGE modules as an alternative to self-reconfiguration. Specifically, the reconfiguration process is addressed through a dedicated robot arm responsible for the automated assembly and disassembly of the modules. Through externalizing the reconfiguration mechanism, the design of the EMERGE modules can be kept free of active connection mechanisms and the complexity of the reconfiguration algorithm can also be reduced to moving modules around with the manipulator

The process of reconfiguring modular robots with a manipulator is different from the automated assembly of parts in manufacturing [17, 9, 56] in the lack of fixtures, and other ways to hold parts, on one side, and in that the system is not only required to pick and place the modules, but also requires a mean of separating them, on the other.

Manipulators have been used to build or change the shape of modular structures [116]. Other external reconfiguration devices, also used with modular structures include mobile robots [125] and drones [63]. From the modular robots perspective, Saldana et al. [95] have designed decentralized algorithms for assembling different kinds of structures using mobile modular robots. Furthermore, Brodbeck et al. [11] describes robots composed of two different types of modules (passive and active) that are joined by an industrial manipulator with hot glue adhesives, which can automatically test robot morphologies and controllers in an arena.

This work's approach comprises two main advantages over Brodbeck et al approach: using magnets to connect the EMERGE modules is faster compared to using hot glue fixtures, and, more importantly, this work's system is able to automatically disassemble the modules. To fulfill this last advantage, the forces of the EMERGE magnetic connectors when approaching one another, that is, the forces involved in the attachment and detachment of the modules are investigated. Moreover, this section also describes tests done with an active gripper approach, a passive gripper approach is tested in [73].

4.5 Magnetic connector force analysis

To study the behavior of the EMERGE module's magnetic connector in regard to the automatic assembly and disassembly with the active and passive approaches, forces between magnets are modeled by using a dipole field model [34]. In this model, each magnet is

TABLE 4.5. Magnet properties

Quantity	Value	Units
D	12.7	mm
t	3.175	mm
Br	1.32	Tesla
cf	1/10.618	-
μ_0	$4\pi \times 10^{-7}$	N/A^2

represented by a moment $\vec{m}$ (a vector in 3d space) that can be approximated by equation 4.1.

$$\vec{m} = \frac{V \cdot Br}{\mu_0} cf\hat{u} \tag{4.1}$$

V is the volume of the magnet calculated as a cylinder with diameter D and thickness t, Br is the residual induction of the magnet, μ_0 is the vacuum permeability, and $\hat{u}$ is the unit vector. The magnet properties used in this work can be seen in table 4.5. A correction factor cf is introduced to adjust the model due to non modeled phenomena. The field generated by a moment m can be calculated as:

$$\vec{B} = \frac{\mu_0}{4\pi} \frac{(3\vec{m} \cdot \vec{r})\vec{r}}{\|\vec{r}\|^5} - \frac{\vec{m}}{\|\vec{r}\|^3} \tag{4.2}$$

where $\vec{r}$ is a vector going from the magnet's position to the point of interest. The field generated by multiple magnets is calculated independently and then summed at the point of interest. To calculate the force that a magnet $\vec{m}_0$ exerts on another magnet $\vec{m}_1$ first the field generated by $\vec{m}_0$ ($\vec{B}_0$) is calculated in the position of $\vec{m}_1$. The force $\vec{F}$ is then:

$$\vec{F} = \nabla(\vec{m}_1 \cdot \vec{B}_0) \tag{4.3}$$

To find the force that one EMERGE connector exerts on another, one connector is placed at the origin of a Cartesian coordinate space facing in the positive X direction (Figure 4.16). Another connector is then placed at the positions and orientations of interest and the force exerted is calculated using the dipole field model.

The simplicity of this model limits its applicability to cases where magnets are away from each other, however, it can still produce a good estimate of the forces involved. Using this setup, three cases related to the reconfiguration process are considered:

- *Force between two separating aligned connectors:* The force between two connectors aligned at the center while being separated along the x axis can be seen in figure 4.17. Both connector magnets moments are placed so that they attract each other. The minimum separation distance for one module to be held by friction (F_f, wood table in contact with 3D printed ABS) is measured experimentally using the setup in

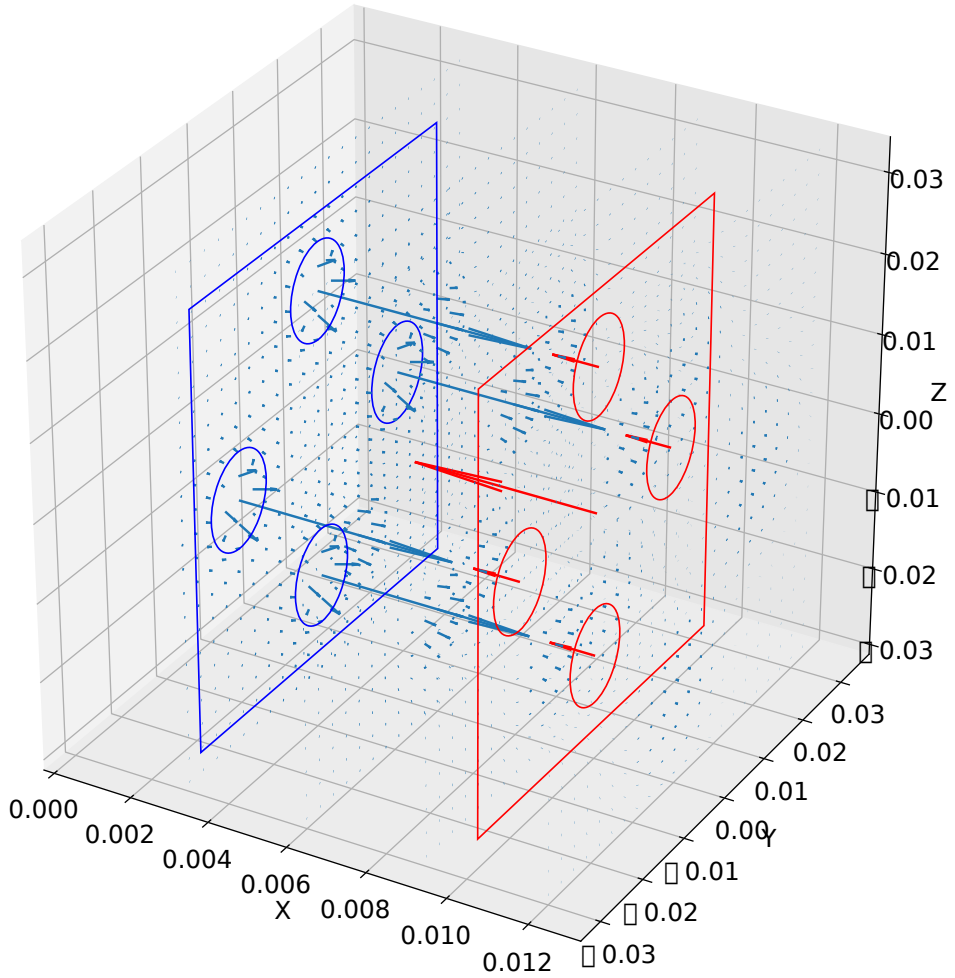


FIGURE 4.16. Connectors magnetic model: The field (blue arrows) due to one connector magnets (left-blue) is calculated. Force at a second connector (right-red) magnets is found using the dipole field model (red arrows), the arrow at the center of the second connector is the sum of all four magnet forces.

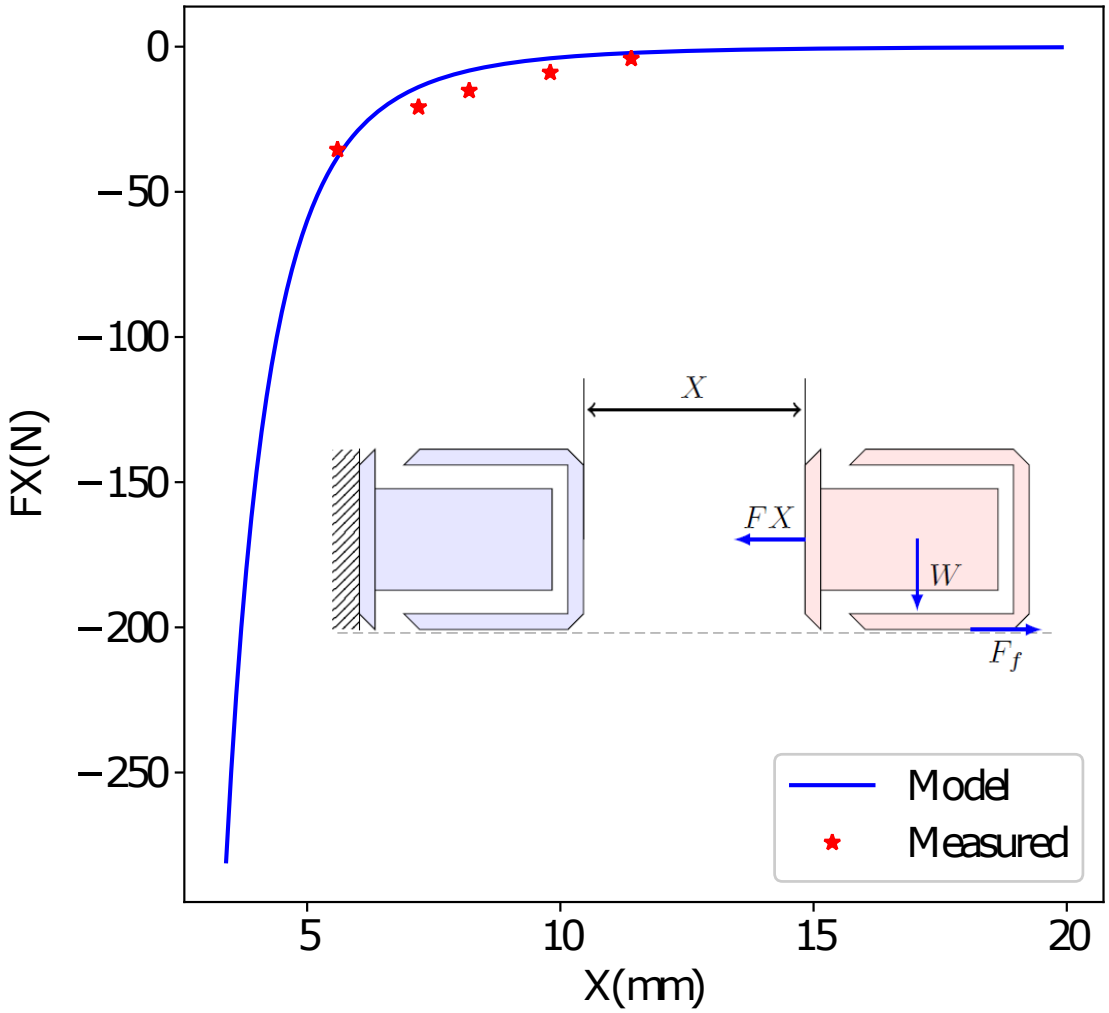


FIGURE 4.17. Force between two connectors aligned at the center (Figure 4.16) when their separation distance in x is varied (inside diagram). Friction forces are denoted as F_f. Red dots show the average measured force of the real connector at distances of: 5.6, 7.2, 8.2, 9.8 and 11.4mm

figure 4.17. One module is fixed and the other released from different positions with their connectors aligned. After 20 measures the minimum distance was found to be 20 mm ± 1mm. The force of the connector is also measured at specific distances to validate the model and tune the correction factor in table 4.5.

- *Forces between misaligned connectors:* The force between two misaligned connectors can be found by initially placing the second connector at a fixed distance from the origin one in X and varying the distance in Y (Z remains fixed at 0). The force sampled as Y is varied can be seen in figure 4.18. The resulting FY force helps correct small misalignments in the assembly process, but can provoke the same misalignments in the disassembly process.

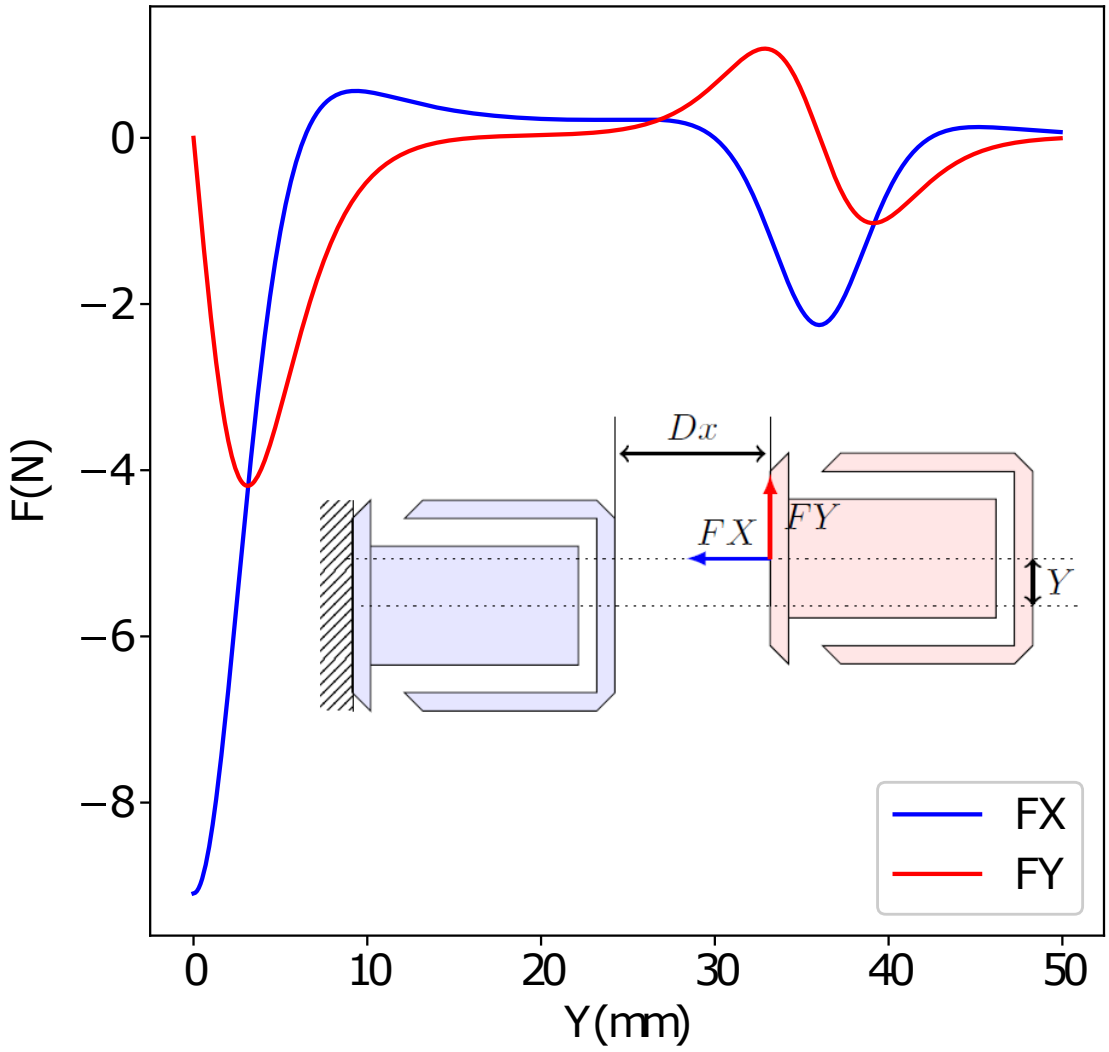


FIGURE 4.18. Force between two connectors separated by a fixed distance (Dx) of 8mm in x from each other, when y is varied (inside diagram)

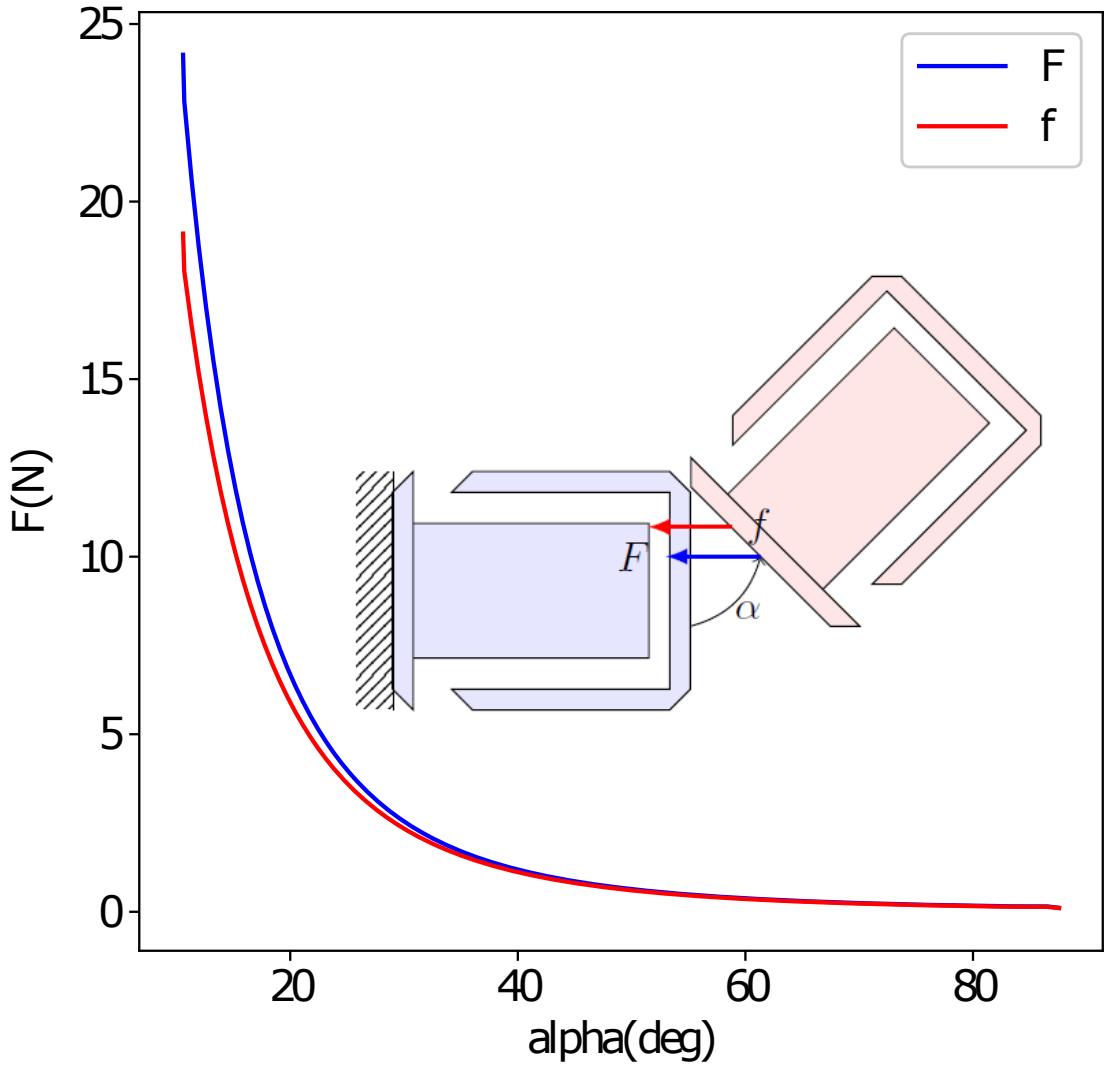
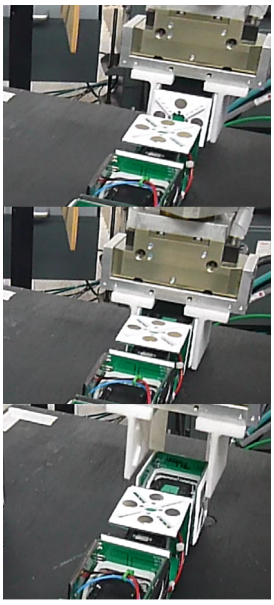


FIGURE 4.19. Forces between two connectors being separated by a circular detach movement. Total force (F) on the connector compared with the magnitude of the force on the magnet closer to the center of the movement (f).

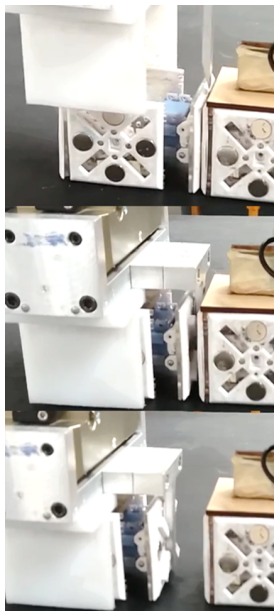
- *Forces between connectors separating at an angle:* The forces between two connectors being separated by a circular detach movement can be seen in figure 4.19. Figure 4.19 shows that the force on the magnet closer to the center of the movement decreases slightly slower than the overall force on the connector. This magnetic force prevents the connectors from separating, which is solved by continuing the movement until the two connectors are perpendicular to each other.

4.5.1 Active gripper approach

The first gripper is composed of active parts attached to a Yaskawa MH6 Motoman robot manipulator (Figure 4.20). The gripper uses two moving fingers that close around and



(A)



(B)

FIGURE 4.20. Attachment and detachment mechanisms using an active gripper approach. The gripper uses two moving fingers to close around and hold one module and a knife to separate two connectors. (a) depicts the attachment of module to a morphology. (b) shows how a module is detached from another by using the knife to separate the connectors.

hold one EMERGE module (Figure 4.20a). A secured module can then be positioned and oriented in order for it to be connected to another module. To disconnect modules, a knife part (5mm thick) is introduced between connectors to separate them, after that the free module is held by the active fingers and put in a place where it could not connect anymore to the morphology (Figure 4.20b). Unfortunately, the application programming interface (API) of the Motoman is not available at the time of implementing this work; therefore, a visual feedback system to track the positions of the modules cannot be used. Instead, the teach pendant is used to record the movements of the robot and place the modules where we want them to be picked up.

The procedure is: (1) position the gripper above a module ensuring the alignment of the active fingers with the module's shape, (2) move the gripper down until the module is covered, (3) close the active fingers around the module, (4) lift the module up to a safe distance above the floor, (5) move the end-effector to the side of another module, (6) move the end-effector down, (7) move the end-effector toward the other module's attaching face, and (8) release the module by opening the active fingers. Similarly, to detach a module from a 2D morphology: (1) move the gripper above the desired module, (2) align the knife with the module's connection with the other modules, (3) move the gripper down to make the knife separate the connection, (4) close the fingers around the module and (5) move the module away from the other module.

(A) (B)

FIGURE 4.21. Assembly and disassembly process carried out in the repeatability test with the active gripper. (a) shows three frames of an assembly process with an 8 module configuration. (b) shows three frames of a disassembly process with another 8 module configuration.

Using the attachment and detachment movements described, two tests are performed using the active gripper approach. In a repeatability test, two planar robot configurations with 8 modules each are repeatedly assembled and disassembled (10 times) to check for any kind of problem that could arise in the process. Figure 4.21 shows the assembly and disassembly process carried out with the two configurations. For the assembly process, the knife is detached from the gripper. As the reconfiguration system lacks a visual positioning system, individual modules are placed in predetermined positions on the table, then the robot arm travels to each module, secures it with the gripper, and moves it to its destination.

The experiments determined that misalignments are less likely to affect the assembly process due to the connector's self centering forces, analyzed in section 4.5, and also because modules are separated enough for the manipulator to correctly align one connector face to the other. As a consequence, all 10 trials were successful. During the disassembly process, movements of the whole structure due to a module being separated were greater than expected, thus the structure's position had to be manually corrected. This problem increases as fewer modules remain in the structure, that is, friction forces are not enough to oppose the magnetic connector forces and modules can be moved further distances (Section 4.5 and video [78]), this problem shows that a positioning system is necessary for the automatic reconfiguration to work properly.

The force that the knife needs to apply in the downward direction (G in figure 4.22) to separate one module from a robot morphology is measured using the robotic manipulator

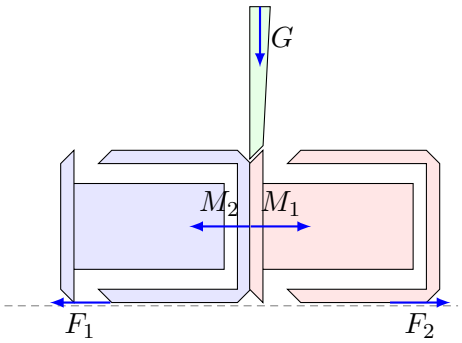


FIGURE 4.22. Force (G) used by the active gripper's knife when separating a module from a planar robot morphology. F_1 represents the friction force of the rest of the modules attached, F_2 is the friction force of one individual module with the ground

equipped sensors. For this purpose, the torques in each of the robot's motors are registered as the knife's tip moves down. The force is then calculated based on the total torque and the position of the knife relative to the robot arm. We performed 10 measurements and the average maximum force was found to be 5.4 N $\pm$ 0.1N. This low force ensures that the disassembly process using the active gripper can be done with less powerful robot manipulators.

4.5.2 Limitations of External Manipulator Approach

The main limitation of using an stationary external device for reconfiguration is that the process is only possible if all the modules are near the manipulator's workspace. Also, only 2D modular robot morphologies are considered. This is because there is no disconnection mechanism embedded in the connectors of the modules. Movements of the robotic structure in the disassembly process show that a visual feedback system, or another positioning system, is a must for the reconfiguration to work properly.

4.6 Summary

In this chapter a new modular robot platform, the EMERGE module was presented. EMERGE is a chain type modular robot designed to be easy to be built and which increases reconfigurability in a practical way. Sensors are included in the module faces with the explicit purpose of studying how sensor information must be used in a modular robot to enable different morphologies to travel through different environments. Modules are built using relatively cheap, commercially available components and their hardware design files are open for anyone to use and modify.

The EMERGE module has mating passive magnetic connectors which, thanks to their spatial location, result in tree like organized assemblies. Thanks to the magnetic connectors, modules can be quickly and easily assembled and disassembled in order to test

different robot morphologies in reality. An arrange of spring-loaded pins and pads in the connectors ensure the distribution of electrical power and communications among modules in a structure. These electrical contacts are embedded in specially designed 3D printed faces and are attached to a PCB layer that routes connections inside the module. Electronic components like a microcontroller, which manages communication, servo motor control and sensor reading, and the sensors themselves are placed on the PCB, although a microcontroller-less version has also been implemented. Either version of the electronics allows the movements of the robot to be controlled in evolutionary or learning locomotion experiments, with the advantage of being able to run full embedded controllers in the microcontroller version. Simulated modules have also been implemented for the same purposes as the real platform.

Taking advantage of the fast connection feature of the passive magnetic connectors, this chapter showed that it is possible to automatically assemble and disassemble EMERGE modules using a robot manipulator as a practical alternative to self-reconfigurable robots and manual reconfiguration systems. This is specially useful in chain type modular robots like EMERGE, where self-reconfiguration is limited due to kinematic restrictions. Although the analysis of the connector shows that it can apply a self-centering force and tests showed that this force simplifies the assembly of the structures, it also makes them more difficult to disassemble. Therefore, a positioning system, i.e. visual feedback, is necessary for the robot manipulator to keep track of modules that move due to disassembling forces. Automatic reconfiguration can be specially beneficial in fields that optimize the morphology and control of robots [105], for example, evolutionary robotics.

The simple design of the EMERGE module poses limitations for testing controllers in different environments, including the lack of an internal power supply, the lack of an internal orientation sensor and possible infrared ghost readings due to the design of the male connector face. Nevertheless, the first two limitations can be overcome by using helper modules and an external power supply. The third limitation will be addressed in a future iteration of the design. On the external reconfiguration side, some challenges, like being able to reconfigure 3D morphologies, and the lack of a visual positioning system for the manipulator, remain to be addressed in order to improve the robotic arm reconfiguration system to a state where continuous experiments can be done. A visual positioning system was implemented with the passive gripper approach described in [73], although with some limitations.

The next chapter will describe the experimental setup used to test the implementation of the locomotion training framework from chapter 3 on the EMERGE module, as well as the results obtained.

CHAPTER 5

Experiments

5.1 Introduction

A controller capable of using sensor information to modify its behavior, a configurable environment approach, a way to adapt controllers and a modular robot platform all set the scene to enable a modular robot to move in different environments. However, different tests are necessary to establish the ability of the defined components to actually generate controllers capable of moving and generalizing to unseen conditions.

Having defined all these parts of the locomotion training framework in previous chapters, this chapter describes the experiments performed to test whether evolution can adapt a controller for a modular robot to travel in different environments.The chapter starts by testing controllers that use only the CPG coordination mechanism in environments defined using the configurable environment approach in simulation. Next, also in simulation, a generalization measure is proposed and tested using combinations of sub-environments housing primitive features of the environment. Resulting controllers are compared to controllers generated using a gradient based optimization algorithm.

Controllers with all sensor parts are then evolved in different robot morphologies. Coordination is found to have a high impact on the ability of evolved controllers to generalize, this is tested by using different kinds of initial populations with different coordination capabilities. An incremental short challenge approach is also done to confirm the coordination assumption. Finally, controllers are transferred and tested in the real EMERGE modules and a test of the full locomotion training framework is done in reality.

TABLE 5.1. Parameter values of the CPG coordination mechanism

Parameter	Value	Range
a_r	50	-
a_x	30	-
ω_i	$2\pi \times 0.65$	-
w	7	-
R_i	-	$[-1; 1]$
X_i	-	$[-1; 1]$
$\Delta\phi_l^{(i)}$	-	$[-\pi/2; \pi/2]$

5.2 Configurable environments and Evolution

Using the configurable environment approach described in chapter 3 a robot controller can be evolved to generate the movements necessary to travel through a structured environment. As a structured environment is separated into sub-environments, these sub-environments are used to train the robot. Each sub-environment shows different features of the given structured environment to the adaptation process, making the training more controllable. Moreover, if a robot is trained in all sub-environments then it should be able to move through the original environment and through environments built from the same parts. In this section, a controller for a modular robot with an specific topology is evolved using the configurable environment approach. The controller is evolved in all sub-environments extracted from the structured environment and it is later tested whether it can travel through a different environment built using the same sub-environments.

There are, of course, different ways of using environments in the training process. Even if all available environments are used, using them in different ways implies different fitness measures and thus can possibly produce different behaviors (with their own advantages and disadvantages) and results of the evolutionary algorithm. For this reason, two different ways of using environments are implemented and compared in this section.

A modular robot topology without sensors is used in this experiment in order to simplify the training process and show the effect of the configurable environment approach. Since the robot used in this case has no sensors only the coordination mechanism is used to control the modules and the decision mechanism is ruled out in favor of a monolithic approach, i.e. the same controller parameters are used to generate an unique behavior that lets the robot travel trough all environments.

The robot is assembled in a snake topology (Equation 5.1) in simulation using the basic simulated V-REP modules (Chapter 4). The parameter values of the CPG coordination mechanism can be seen on table 5.1, R_i, X_i and $\Delta\phi_l^i$ will be changed by the training process and thus intervals are shown instead. To enable the relatively simple chain of modules to move in different directions in a 3D space, modules are assembled in alternating orientations like in figure 5.1.

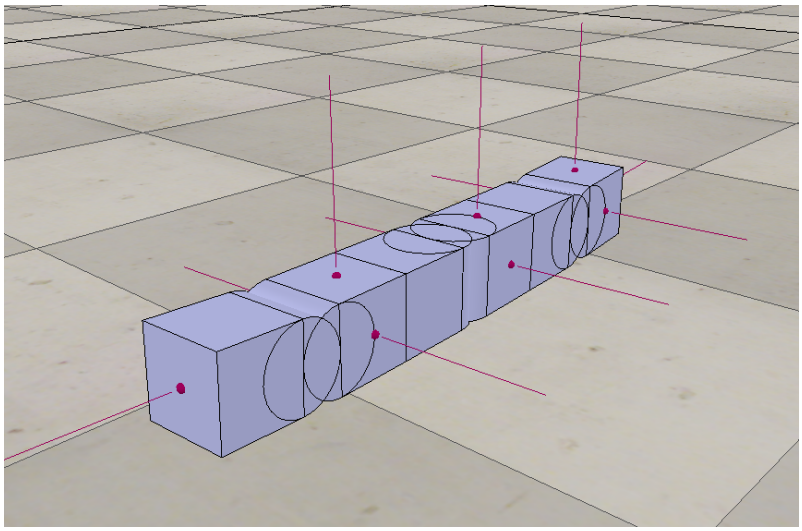


FIGURE 5.1. Basic simulated modules in alternating configuration.

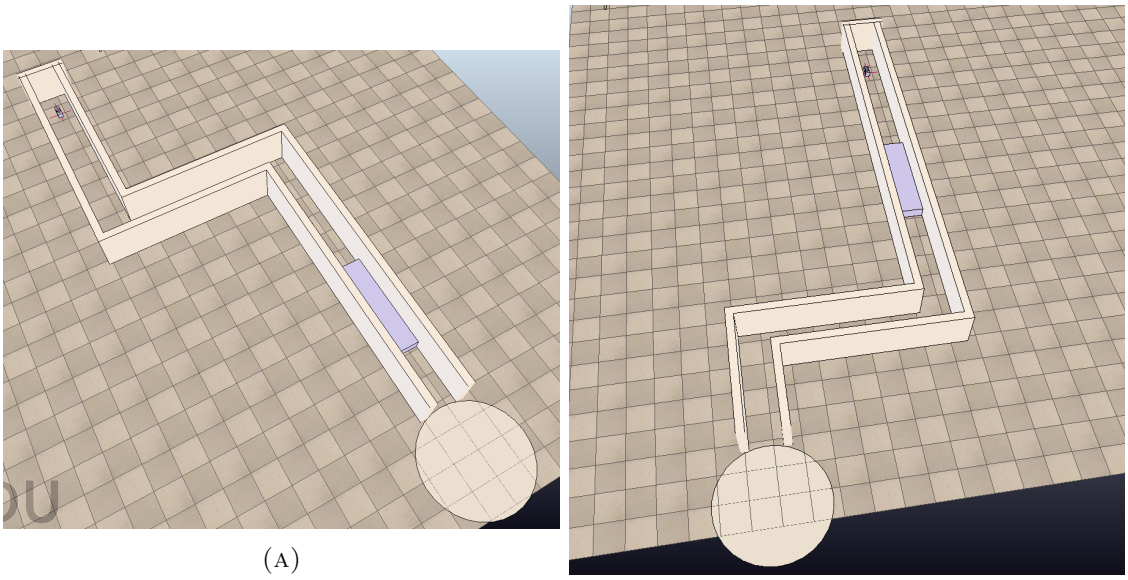
TABLE 5.2. Structured corridor environment: Abstract EMERGE module

Parameter	**Value**	**Description**
Width	0.6m	
Height	0.2m	Bump height
Wall height	0.8m	

$$T = \{\langle 1,2,2,1,2\rangle, \langle 2,1,1,2,2\rangle, \langle 2,3,2,1,4\rangle, \langle 3,2,1,2,4\rangle, \langle 3,4,2,1,2\rangle, \langle 4,3,1,2,2\rangle, \langle 4,5,2,1,4\rangle, \langle 5,4,1,2,4\rangle, \langle 5,6,2,1,2\rangle, \langle 6,5,1,2,2\rangle, \langle 6,7,2,1,4\rangle, \langle 7,6,1,2,4\rangle, \langle 7,8,2,1,2\rangle, \langle 8,7,1,2,2\rangle\} \quad (5.1)$$

A structured corridor environment, depicted in figure 5.2a and in which the robot finds turns and obstacles as well as long empty segments, is decomposed into four sub-environments *Straight*, *Turnleft*, *TurnRight* and *Bump*). All sub-environments have a similar distance from the start position to the exit position. For the sake of clarity, environments composed of only one sub-environment ($E = se$), like in the current case, will be called primitive environments, or *primenv*, from now on. Table 5.2 shows some of the most important features of the environment.

The starting position of the robot in each *primenv* is defined by the position of the last module (M_8) of the chain and is shown in figure 5.3d. *Primenv Straight* (Figure 5.3a) is a straight strip in front of the starting position, *TurnLeft* (Figure 5.3c) its a left turn after a shorter straight strip, *Bump* (Figure 5.3b) has a step that doubles the robot's height after some distance from the start of the same straight strip as in *Straight*, and *TurnRight* (Figure 5.3d) is a turn in the opposite direction of *TurnLeft*.



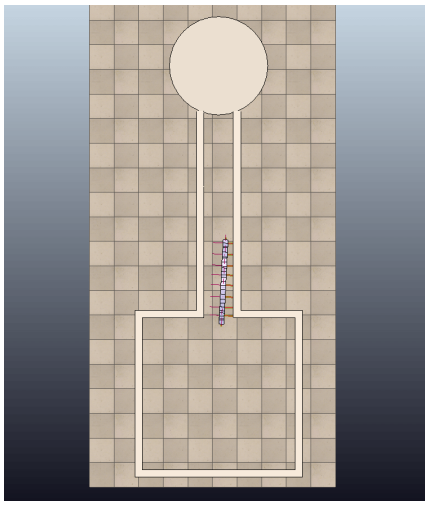
(A) (B)

FIGURE 5.2. Two environments with turns and obstacles. The light blue rectangle is a rectangular obstacle. The circle represents the exit area.

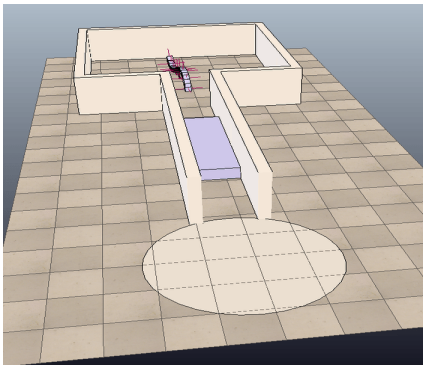
Individual controller solutions to be evolved contain only three parameters of the coordination mechanism (CPG): R_i, offset X_i and only one value of phase difference with neighbors $\Delta\phi_l^i$, which is used for all connections between modules. Although each module controller works individually from the rest, all controllers share the exact same three parameters.

Since all three parameters of the individual to be evolved are real numbers (max and min values are shown on table 5.1) the Differential Evolution (DE) Algorithm [109] is used. This evolutionary algorithm uses a special kind of mutation and crossover that relies on the difference between real valued parameters of different individuals to produce new ones and was implemented using the JEAF [13] framework. The evolutionary algorithm settings can be seen on table 5.3.

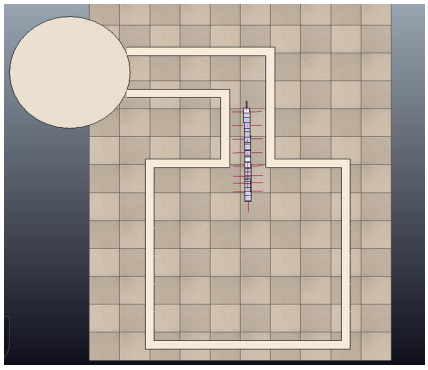
Regardless of the way *primenv*s are organized in the evolutionary process the same measure of robot performance is used individually in each sub-environment. This measure is divided into two stages.(Equation 5.2): first, if the robot is not able to get out of the environment under the maximum time allowed the fitness will be the distance from the exit to the first module (M_1) of the robot (D) plus the maximum time allowed for the test (T) in simulation time. Once the robot gets to the exit (Figure 5.3) its fitness will be the time it takes to complete the environment t. In this way the evolution improves the controller's performance by minimizing the fitness function, that is, by making the robot travel through the whole length of the environment faster. The shape of the fitness function allows other things to be inferred from the fitness value, for example, if an individual's fitness is less than T in an specific *primenv* then it was able to travel through its whole length.



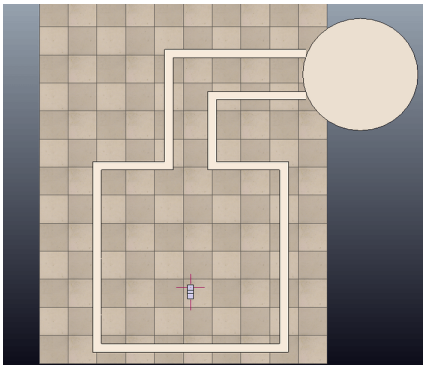
(A) *Straight*



(B) *Bump*



(C) *TurnLeft*



(D) *TurnRight*

FIGURE 5.3. The four primitive environments in which the robot is trained. The circle represents the exit point. The initial position of the robot is shown in (d).

TABLE 5.3. Differential Evolution Parameters: Using sub-environments

Parameter	Value
Population Size	32
Number of Generations	300
F	0.9
CR	0.9
Max. Evaluation time T	40(s)

$$F = \begin{cases} D + T & \text{if goal not reached} \\ t & \text{if goal reached} \end{cases} \tag{5.2}$$

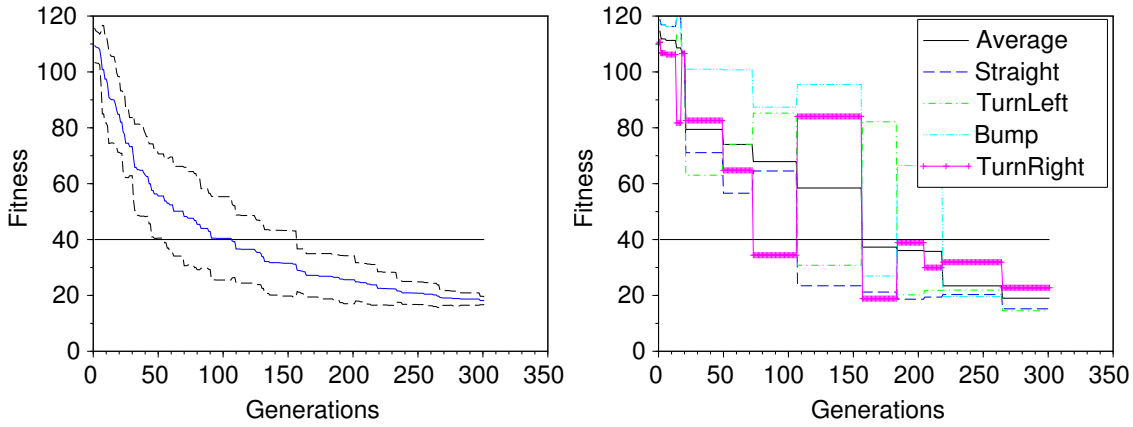
5.2.1 Using an aggregating measure of fitness

One way to train a robot controller in all *primenv*s and provide a unique measure of fitness that can be used by the evolutionary algorithm, is to use an aggregating measure of the performance of the controller in each *primenv*. In this case two different aggregating measures are used and compared: the average performance of the controller in all *primenv* and the worst performance in all *primenv*. Each measure is used for 10 runs of the evolutionary algorithm (DE).

Figure 5.4a shows the average of the best individual fitness per generation for 10 runs of the evolutionary process, using the average *primenv* fitness. Figure 5.4b shows the fitness for each *primenv* in one run. It should be noted from the figure, that using the average *primenv* fitness is deceiving in that a solution can perform really well in some *primenv* while performing poorly in the others. Whats more, using the average *primenv* fitness doesn't ensure that a controller is good in all *primenv*s at the same time, however in this case all evolutionary executions generate controllers that get to the exit in all individual *primenv*s.

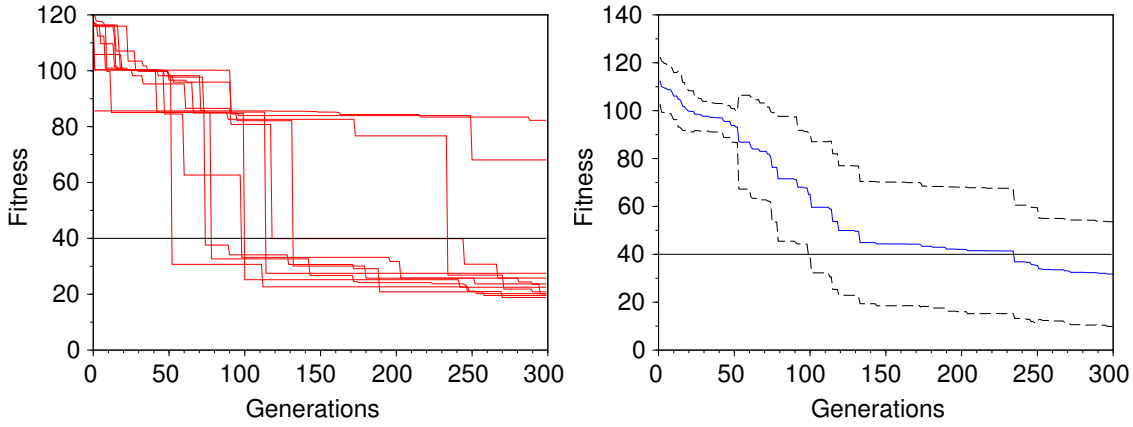
Figure 5.5 shows the best individual fitness per generation and the average of the best individual fitness per generation for 11 evolutionary runs using the worst *primenv* fitness. It can be seen from the figure that when using the worst *primenv* fitness measure the evolutionary process is not able to produce an individual that reaches the exit of all *primenv*s in 2 of the 11 runs for the allowed number of generations.

When presented with the two environments from figure 5.2 controllers obtained using the average *primenv* fitness are able to reach the exit of both in 14 out of 20 tries and using the worst *primenv* fitness in 17 out of 20 tries. This demonstrates that locomotion movements, enabling a robot to travel through different configurations of an structured environment, can be generated in a modular robot by using an evolutionary algorithm and the configurable environment approach with an aggregating fitness measure. Unsuccessful



(A) (B)

FIGURE 5.4. Performance of using the average *primenv* fitness: Average and standard deviation of the best individual fitness per generation (a) for 10 runs using the average *primenv* fitness. Also, fitness in each *primenv* for 1 run (b). A fitness under 40 (*bottom line*) means an individual reached the exit of the *primenv*.



(A) (B)

FIGURE 5.5. Performance of using the worst *primenv* fitness: Best individual fitness per generation a and average of best individual fitness per generation with standard deviation b for 11 runs of evolution using the worst *primenv* fitness. A fitness under 40 (*bottom line*) means an individual reached the exit in all *primenv*s.

controllers in the environments depicted in figure 5.2 can be attributed to the controllers exploiting features of the simulation to get a good fitness [48].

5.2.2 Using sequences of *primenv*s

In task decomposition, it has been shown that introducing a sequence when learning multiple tasks improves the speed and reliability of the evolutionary algorithm [94, 6, 4, 8]. Incremental evolution introduces the idea of a sequence in how robot controllers are evolved for different tasks. In incremental evolution [30] the same task is presented to an evolutionary algorithm with various levels of difficulty starting by the easiest one. As the robot is able to solve the task the difficulty is risen gradually until the robot learns the behaviors needed to solve a desired level of complexity in the given task. The same controller is expected to include the new found behaviors without using extra parts or modules. Although incremental evolution may involve changing the environment, variations of only one feature are usually used [80, 57, 106].

Similar to incremental evolution, in this case the robot is evaluated in all four *primenv* in a sequential fashion. Only if the individual being evaluated is able to reach the exit of one *primenv* under the maximum time allowed, it is evaluated in the next *primenv*, until an individual is capable of reaching the exit of all four *primenv*s. Each individual receives a bonus fitness each time it exits a *primenv*. These bonuses (Equation 5.3) are designed to be greater than the maximum observed fitness a robot can obtain in any individual *primenv* (this parameter is based in the observed fitness of several runs of the evolutionary process). f_e is the fitness obtained on *primenv* e using (5.2). A total fitness greater than 325 ($1000/T+300$) signals that the robot has reached the exit in all *primenv*s in the sequence.

$$F = \begin{cases} 1000/f_1 & \text{if goal not reached in } primenv_1 \\ 1000/f_2 + 100 & \text{if goal reached in } primenv_1 \\ 1000/f_3 + 200 & \text{if goal reached in } primenv_1 \text{ and } primenv_2 \\ 1000/f_4 + 300 & \text{if goal reached in } primenv_1 \text{ and } primenv_2 \text{ and } primenv_3 \end{cases} \tag{5.3}$$

Which sequence to use is a problem in itself. Bongard et al [6, 4, 8] show that the order in which different behaviors are incrementally learned is important for the success rate of the evolutionary process. In their work a quadruped robot with grasping capabilities is more successful in learning how to manipulate an object first and then move towards it than the other way around.

Thus *primenv*s are presented in three different sequences: *Straight - TurnLeft - Bump - TurnRight* (S1), *Straight - Bump - TurnLeft - TurnRight* (S2) and *Straight - TurnRight - TurnLeft - Bump* (S3). These first three sequences cover all permutations of the *primenv*s

TABLE 5.4. Average performance of the evolutionary algorithm in number of generations, with standard deviation, when using *primenv*s in different ways. The number indicates how many generations are necessary to find a controller that exits all *primenv*s in all cases.

Aggregated Fitness		Sequence			
Worst	**Average**	**S1**	**S2**	**S3**	**S4**
111.1 ± 52.26	108.4 ± 51.89	99.8 ± 22.35	94.7 ± 48.71	103.9 ± 52.02	175.8 ± 59.65

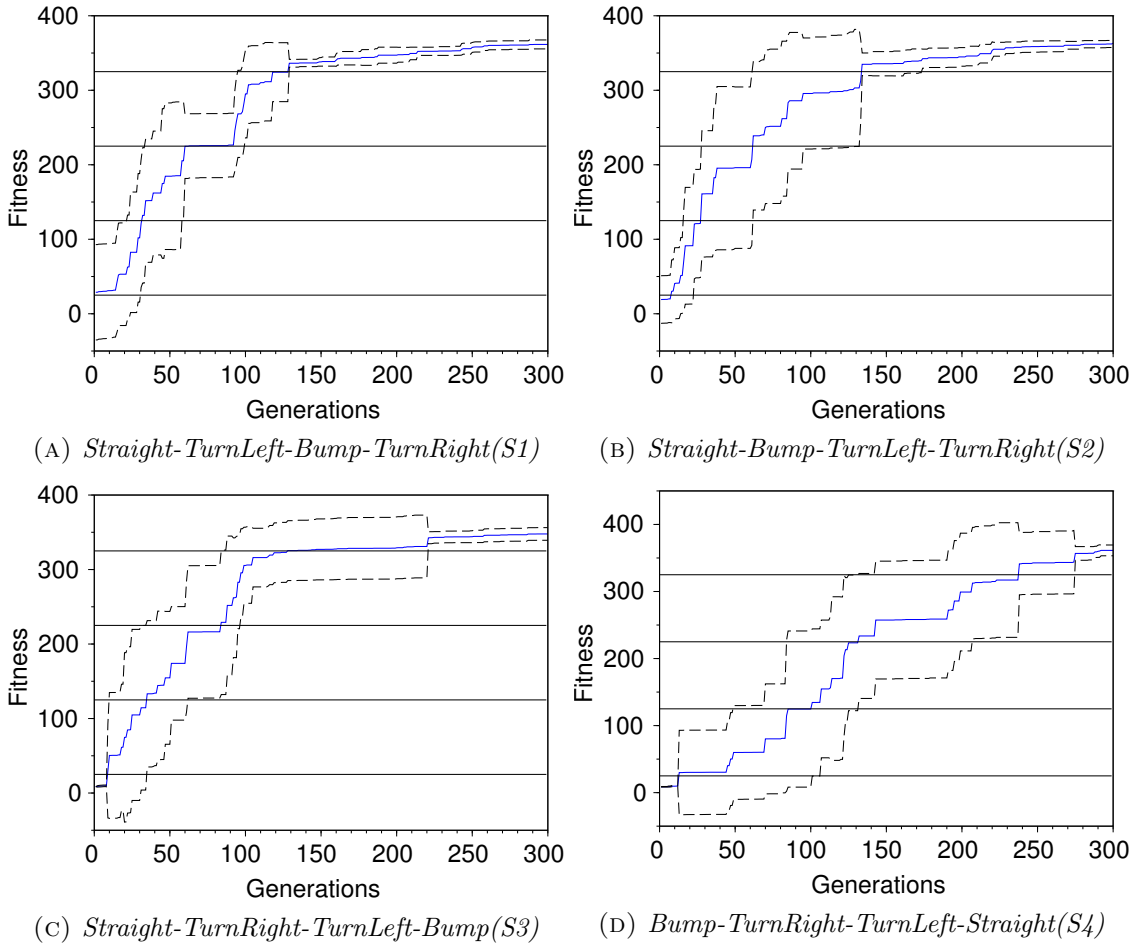
TurnLeft, *TurnRight* and *Bump* that are no mirrors of each other. That is, turning left and then turning right is considered to be the same as turning right and then turning left. The *Straight primenv* is always presented first as it is the simplest and all the others include a straight element. The last sequence considered (*Bump* - *TurnRight* - *TurnLeft* - *Straight*, S4) puts the *Straight primenv* at the end.

In figure 5.6 the average best fitness per generation for all the *primenv* sequences used is shown. It can be seen that in all the cases where a sequence is introduced the evolutionary process is able to produce a controller that makes the robot reach the exit of all four *primenv*s every time. An analysis of variance (ANOVA) test comparing the number of generations it takes for an evolution using *primenv* sequences to generate a controller that reaches the exit of all *primenv*s showed a significant difference, $F(5,53) = 3.73$, $p = 0.0057$ among sequences. The means and standard deviations are presented in table 5.4, which also shows the number of generations when using aggregating measures. Post-hoc comparisons using a Tukey HSD test showed that using sequences S1 and S2 is statistically significantly faster than when using S4 in generating controllers.

When presented with the two environments from figure 5.2, using sequence S1 produced controllers that are able to exit both under 300 seconds in 18 out of 20 tries. In contrast, using sequence S4 produced controllers that exit only in 11 out of 20 tries.

When the *primenv*s are used in sequences, the evolutionary algorithm generates controllers that exit of all *primenv*s in all runs, in contrast with the worst measure case. Also, using a sequence ensures that the evolutionary algorithm produces controllers that are able to reach the exit of all *primenv*s as opposed to using the average measure. The evolutionary process that uses the S1 sequence (Figure 5.6a) makes the evolutionary algorithm perform specially well as it is not only able to generate controllers that reach the exit in a low number of generations in all four *primenv*s but also in a consistent way when compared to the other strategies.

The statistically significant difference between using sequences S1,S2 and S4 to produce controllers indicates that the order of *primenv*s influences the performance of the evolutionary algorithm. Changing the place of the *Straight primenv* in the sequence (S4) makes the overall process take on average more generations to find a solution that reaches



(A) *Straight-TurnLeft-Bump-TurnRight(S1)*

(B) *Straight-Bump-TurnLeft-TurnRight(S2)*

(C) *Straight-TurnRight-TurnLeft-Bump(S3)*

(D) *Bump-TurnRight-TurnLeft-Straight(S4)*

FIGURE 5.6. Average best individual fitness per generation, with standard deviation, for 10 runs of evolution using sequences of *primenv*s. (*Black lines*) indicate environment transitions when a controller has successfully exited each *primenv*.

the exit of all four *primenv*s (Table 5.4). This hints to the idea that some environments are more or less complex to learn than others.

5.3 Generalization and Evolution

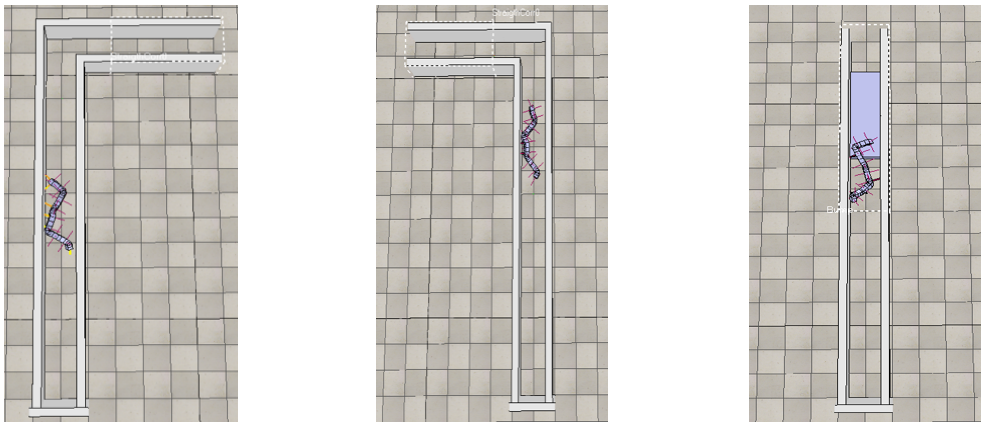
Last section shows that an evolutionary process is able to produce controllers that travel from the start to the exit of *primenv*s. Using a coordination mechanism and the configurable environment approach, controllers are able to generalize and move in other environments built using the same *primenv*s.

However, to what degree are the produced controllers able to generalize to unseen conditions? and how to measure this "degree of generalization" with the configurable environment approach? This section introduces the distance traveled by a controller in an special set of environments built by combining *primenvs* as a measure of generalization. These special environments are built by arranging *primenv*s in all their possible combinations, representing all possible conditions that a robot controller would have to face. In order to increase the effect of the distance traveled by the robot when measuring the fitness of a controller in an environment the fitness measure from last section is redefined. Similar to the previously used fitness measure a maximum time (T) constrains the robot evaluation (Equation5.4). This maximum time constrain divides the fitness measure in two, depending on whether the robot is able to exit the environment under the maximum time allowed. Distance (D)is measured between the first robot of the topology and the exit point of the environment, in manhattan distance, and normalized. The starting and exit points are shown in figure 5.8. Controllers are still able to improve their fitness by using less time (t) once they reach the end of the environment. Time t is also normalized using the maximum time T.

$$F = \begin{cases} 0.7D + 0.3T & \text{if goal not reached} \\ 0.3t & \text{if goal reached} \end{cases} \tag{5.4}$$

To determine the effect that showing different environments, built using *primenv*s, has on the generalization measure, controllers are evolved using three different fitness approaches. The *primenv*s have been simplified by eliminating the straight strip from the beginning (Figure 5.7) of last section *primenv*s. The resulting controllers are then tested in each of the 6 combination environments and compared.

- **All Combinations:** Controllers are evolved using all combinations of *primenv*s. The individual fitness is the sum of all fitness obtained in each combination environment.



(A) Right turn (B) Left turn (C) Bump

FIGURE 5.7. Modified sub-environments.

TABLE 5.5. Hill Climbing (HC) parameters: generalization test

Parameter	**One Combination Random Combination**	**All Combinations**
Number of Iterations	9000	1000
Mutation Rate	1/5th rule	-
Test Iterations	10	-
Step size	0.09 (m)	-

- **One Combination:** Controllers are evolved using only one combination of *primenv*s.The specific combination is chosen to be the one that yielded the best results in the experiments of last section ($left \oplus right \oplus bump$).
- **Random Combination:** Controllers are evolved using only one combination of *primenv*s but this combination randomly changes in each generation. The same combination is used to evaluate the whole population of the current generation.

Differential Evolution (DE) is once again used for this tests (Table 5.6). The DE algorithm is run 10 times using each of the defined fitness approaches. The same tests are also repeated using a Hill Climbing (HC) algorithm. Hill Climbing (HC) optimization algorithms are much simpler heuristics, compared to evolutionary algorithms. These algorithms modify a single individual using only a simple variation, in most cases a mutation. The current individual is replaced only if a better or equal (neutral) one is found. Controllers generated by DE and HC are compared to test the effect of the adaptation mechanism in the training process.

The HC algorithm is implemented using the *unalcol* library ([32]). Table 5.5 shows the algorithm parameters, the number of evaluations is set to be consistent with the DE case.

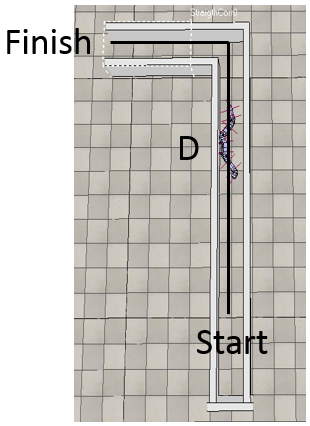


FIGURE 5.8. The distance traveled by the robot (D) used in the fitness function is measured from the current position of the first module of the topology to the exit point.

TABLE 5.6. Differential Evolution (DE) parameters: generalization test

Parameter	**One Combination Random Combination**	**All Combinations**
Population Size	30	20
Number of iterations	300	50
F	0.9	-
CR	0.9	-

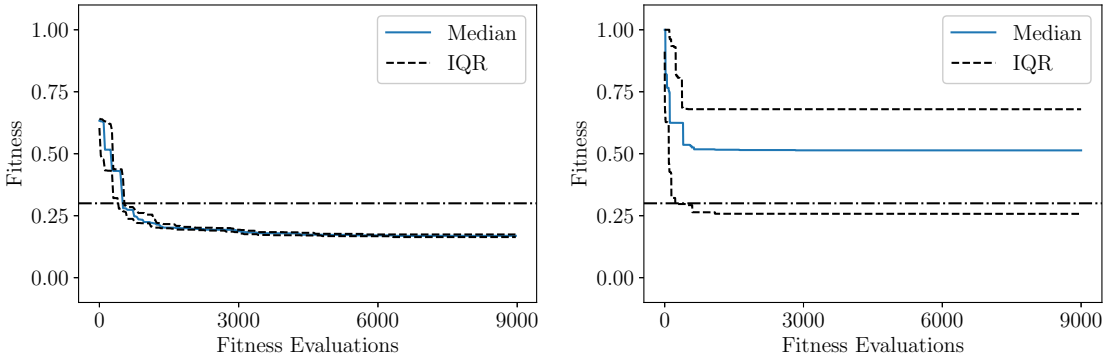
5.3.1 Differential Evolution vs Hill Climbing

Figures 5.9, 5.10 and 5.11 show the performance of the best controller fitness when using the one combination fitness approach, the random combination fitness approach and the all combination fitness approach respectively, for 10 different executions of DE and HC.

Figure 5.12 shows the overall distance traveled from the start of the environment, by the best controllers produced by DE and HC when tested in each 6 combination environments. Distance is normalized (1: full distance of the environment traveled. 0: No distance traveled). Results indicate that controllers produced by DE cover a greater distance than those generated by HC. A Kruskal-Wallis test shows a statistically significant difference between the median distance traveled by controllers generated using DE and HC ($p = 1.28701e^{-7}$ and figure 5.13). Figures 5.9 and 5.10 also show that DE always generates controllers that reach the exit of the environments presented, which is not the case when using HC.

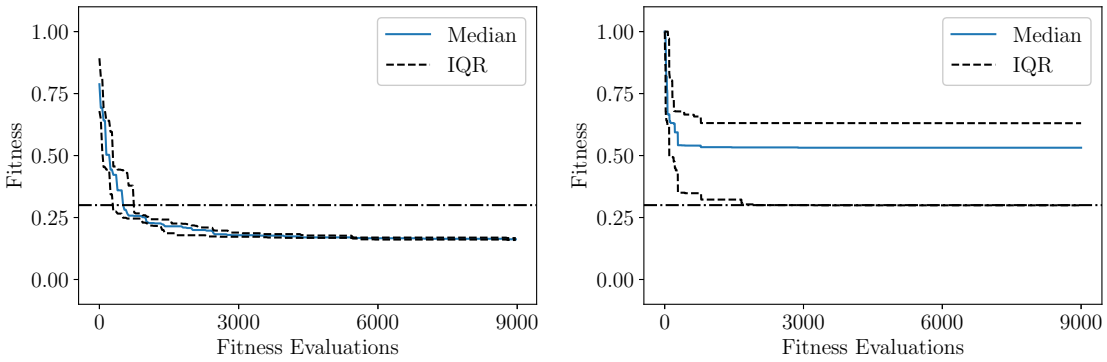
5.3.2 Comparison of fitness approaches

When comparing controllers among different fitness approaches, a Kruskal-Wallis test shows an statistically significant difference on the median distance traveled for all controllers, regardless of using DE or HC to produce them, when using the different fitness approaches ($p = 0$). Figure 5.14 shows that controllers generated using the all combi-



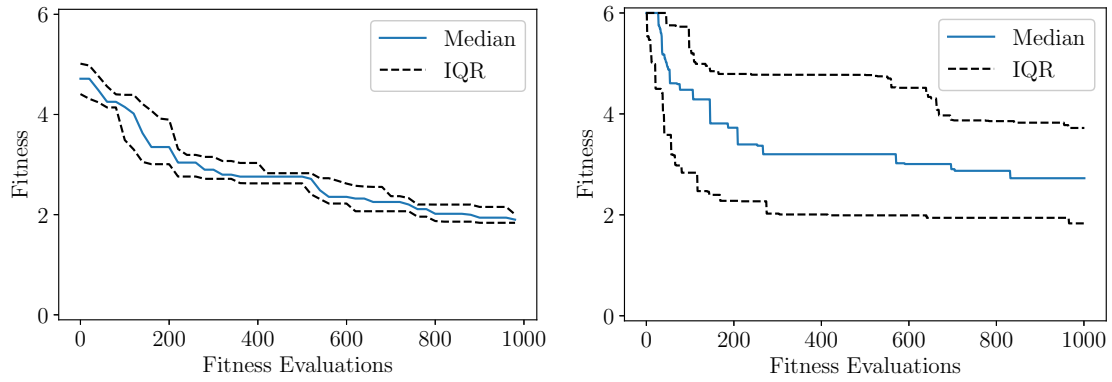
(A) (B)

FIGURE 5.9. Evolution of performance of the best controller fitness when using the one combination fitness approach for 10 different executions of (a) DE and (b) HC. Bars show the inter-quartile range. A fitness below 0.3 (dashed black line) indicates the individual exited the environment.



(A) (B)

FIGURE 5.10. Evolution of performance of the best controller fitness when using the random combination fitness approach for 10 different executions of (a) DE and (b) HC. Bars show the inter-quartile range. A fitness below 0.3 (dashed black line) indicates the individual exited the environment.



(A) (B)

FIGURE 5.11. Evolution of performance of the best controller fitness when using the all combination fitness approach for 10 different executions of (a) DE and (b) HC. Bars show the inter-quartile range.

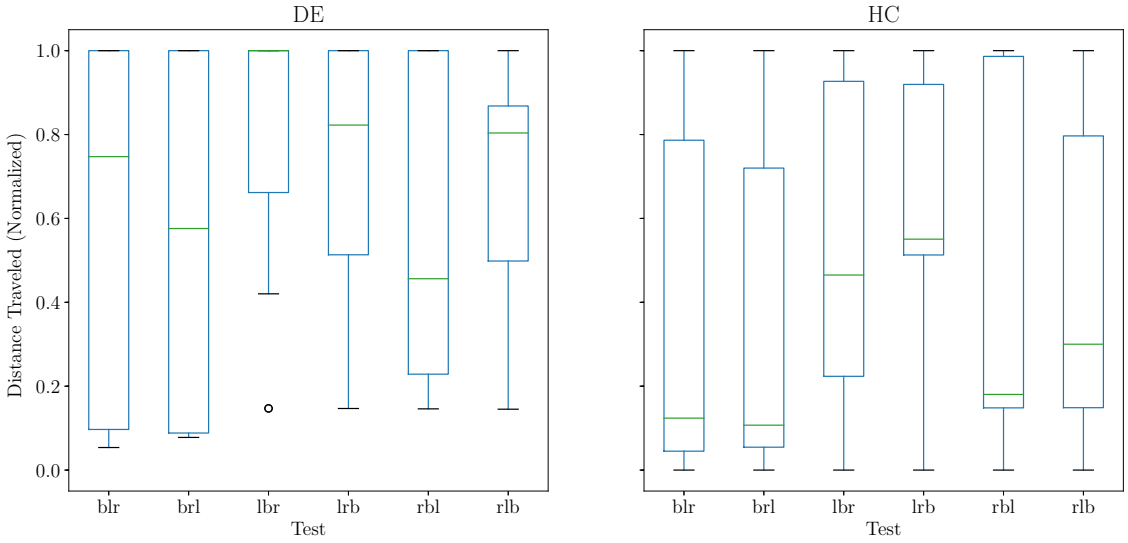


FIGURE 5.12. Overall distance traveled, by environment, of the best controllers produced using DE and HC when tested in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

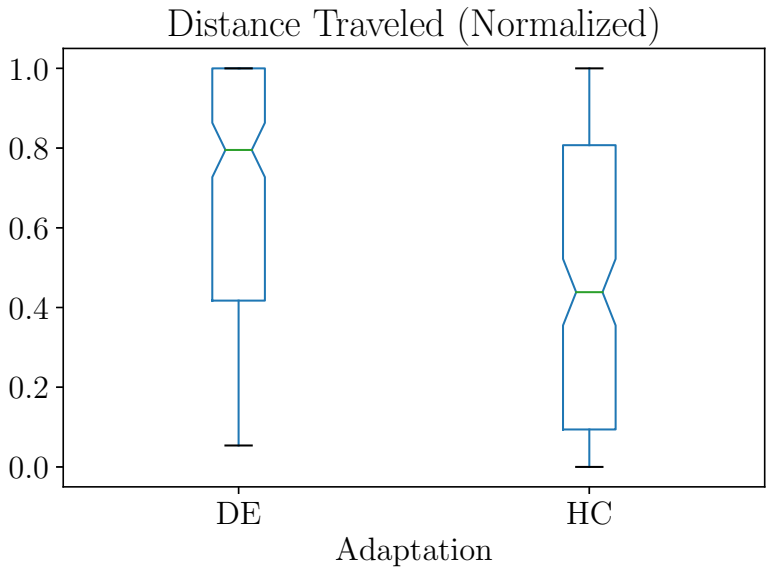


FIGURE 5.13. Overall distance traveled by the best controllers generated using DE and HC when tested in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

nations fitness approach travel more distance than the other two approaches and that the one combination approach travel slightly more distance than the random combination approach.

Although controllers generated by using the one combination or random combination fitness approaches do not travel as much distance as the ones generated by using the all combinations approach, the former are still able to travel some distance, that is, to generalize, to unseen environments. This suggests that evolved controllers may be able to improve their generalization ability as more combinations of the environment are used in the training process, however this topic is out of the scope of this work.

5.4 Sensors, Hill Climbing and Evolution

Last section showed how an evolutionary algorithm is used to adapt CPG coordination mechanisms in order to enable a robot to travel from the start to the exit of different environments. However, by only using the CPG coordination mechanism robots are unable to react to the environment, and can only advance blindly through different obstacles. To also enable a robot morphology to use sensor information, what is left of the control strategy described in section 3.1 is tested in this section, including the sensor information handling and decision mechanisms.

With all parts in the controller, messages are generated in each module as a sensor is activated. Messages contain information about each sensor location in the module and the sensor reading. In simulation, messages are generated once every time step and a single message combines the state of all activated sensors at that moment.

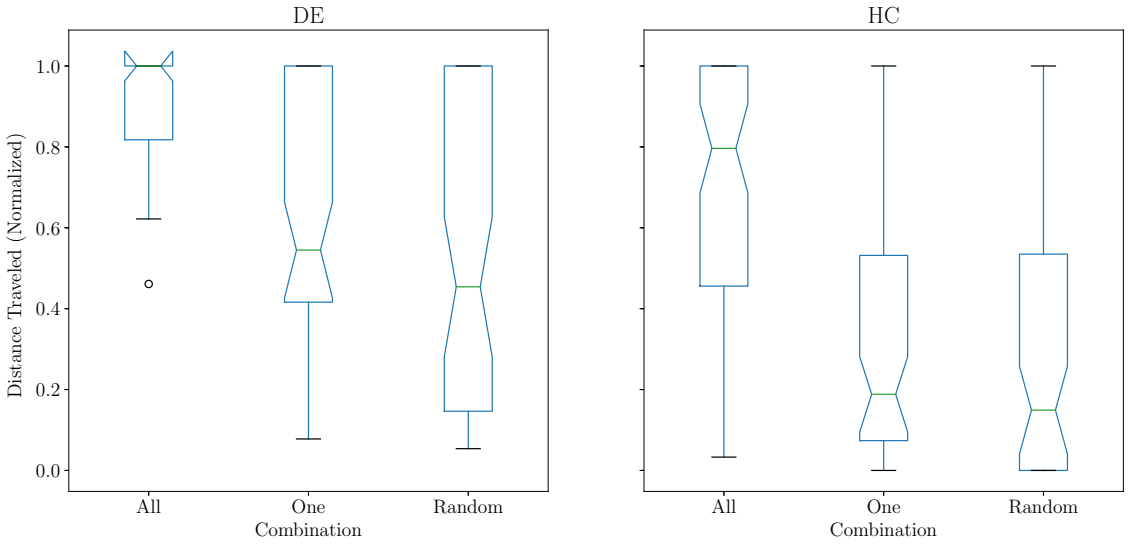


FIGURE 5.14. Overall distance traveled by the best controllers generated using DE and HC when tested in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

Starting from this section the realistic EMERGE module of section 4.2 is used. As presented there, this module has four proximity sensors, located in its four connecting faces, and one orientation sensor. The topology used is the same as in previous sections (Equation 5.1). Sensor messages for this type of module include readings of each of the four proximity sensors at first. Messages contain a vector of sensor reading values in which each position represents the location of a proximity sensor in the module. The spatial transformation function is then a permutation function that only changes the order of the message values when they arrive to a module connected in a different orientation than the originating module, in particular, the spatial transformation function matches the sensor positions of the incoming message with the locations of the sensors in the receiving module, given a topology.

Additionally, modules in different orientations can have sensors in faces that match sensorless faces of neighbor modules. Therefore, sensor messages contain two extra values representing the empty faces of the module, at the end of the vector. Figure 5.15 shows the position of the sensors in the realistic EMERGE module and their respective positions in the message. Figure 5.16 shows an spatial transformation example between modules in different orientations; the right module sends a message to the left one which changes the order of the message to match the positions of its sensors (including sensorless faces).

Sensor readings are normalized and attenuation and forward propagation are used to prevent messages from looping indefinitely in the robot morphology. Furthermore, sensors facing directly into the ground and sensors of connected faces are ignored. Table 5.7 summarizes the parameters of the sensor information handling mechanism. The elimination threshold parameter refers to the value under which a message is eliminated, in other words, if all sensor readings inside a message are below the elimination threshold

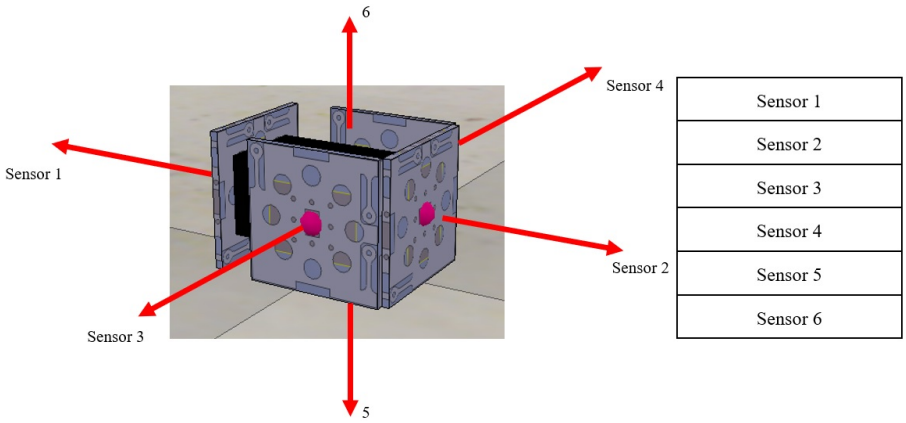


FIGURE 5.15. Sensor message example in the realistic EMERGE module

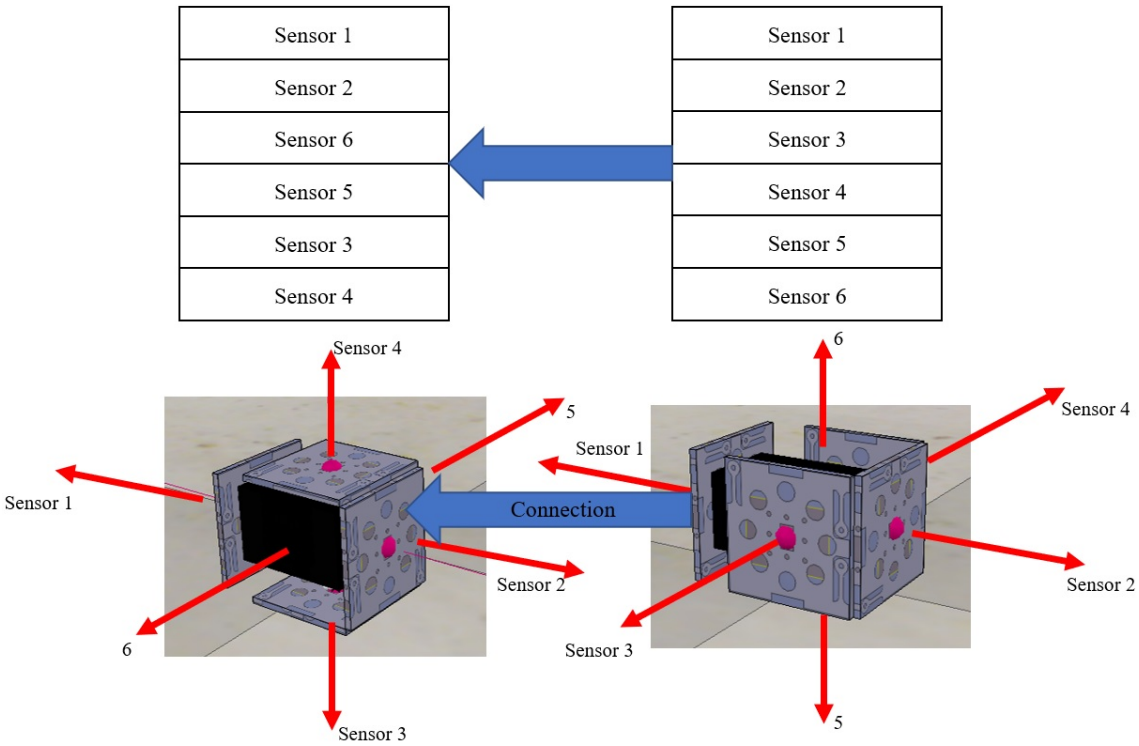


FIGURE 5.16. Sensor message spatial transformation example between modules in different orientations, using the Realistic EMERGE module.

TABLE 5.7. Parameters of the sensor information handling mechanism in simulation

Parameter	**Value**	**Range**
Sensor Reading	Normalized	$[0, 1]$
α	0.5	$[0, 1]$
Elimination Threshold	0.001	–

TABLE 5.8. Decision mechanism parameters in simulation

	Parameter	**Value**	**Detail**
Filter	Proximity sensor window	7	time steps
	Orientation sensor window	12	time steps
ANN	Input Layer	12	–
	Hidden Layer	12	–
	Output Layer	6	–
	Bias neuron	1	Input and Hidden Layers
	Activation function	Sigmoid	$y = \frac{1}{1+e^{-x}}$
	Total Number of Weights	234	
	Min weight value	-10	
	Max weight value	10	

the message is not propagated any more. These parameters were chosen experimentally to prevent a message from hopping through the full length of the topologies used.

When receiving sensor messages, modules use their decision mechanism to determine the behavior of the CPG coordination mechanism. An Artificial Neural Network (ANN) is used for this purpose. Before being fed to the neural network, incoming sensor messages are filtered in each module. Input filtering is divided into two stages: In the first stage, all incoming sensor message values in the current time step are averaged and stored, only messages indicating active sensors are averaged. In the second stage a moving average filter [45] is applied to the stored values, and the process starts anew. The first stage is necessary to aggregate incoming information from several sources. Both stages are used to reduce noise and rapid variability of the received information, which could make the output of the ANN also change rapidly. Orientation sensor values are filtered in a different way from proximity sensor values since simulated orientation sensor readings are discrete and represent the current orientation of the module relative to the ground (Section 3.1.2). Therefore, values are stored in each time step and after a predefined window of time, the most prevalent orientation is fed to the ANN. Orientations for the EMERGE module are defined in the same way as the orientations in figure 3.10. Table 5.8 shows the parameters of the ANN and the filter.

Following the sensor message defined above, the ANN has 6 inputs for filtered proximity sensors. Additionally, the current filtered orientation sensor value is split into 6 different inputs to increase the effect of the orientation of the module in the ANN calculations, leading to a way of differentiating modules from one another in a morphology. As a result, the decision mechanism ANN has a total of 12 inputs. A hidden layer is used

TABLE 5.9. Structured corridor environment: Realistic EMERGE module

Parameter	Value	Description
Width	0.4m	
Height	0.088m	Bump height
Wall height	0.8m	

as the output is not expected to be a linear combination of the inputs [39]. Both the input and hidden layers have one bias neuron as the ANN is expected to produce outputs with no or few inputs. The number of neurons in the hidden layer is set experimentally: simple controllers that set the CPG coordination mechanism parameters to one of the bests controllers generated in the no sensor experiments (experiments were repeated using the EMERGE realistic simulated module, with no sensors) were back-propagated and the network that allowed the robot to move forward was chosen. For a detailed description of the output layer see section 3.1.2. Summing up, the total number of weights in the ANN amounts to 234.

Compared to the individuals evolved in previous sections, which only have 3 parameters representing the parameters of the CPG coordination mechanism inside all modules, here, the individual to be evolved is an array of 234 real values representing the weights of the ANN (All modules use the same ANN as decision mechanism). The new individual is tested using the same one combination fitness approach as last section: Controllers are evolved using only one combination of *primenv*s ($l \oplus r \oplus b$) with a straight strip at the beginning of the combination. However, as a result of the realistic EMERGE module being smaller than its abstract counterpart, the *primenv*s dimensions are reduced to fit the robot dimensions. Table 5.9 shows the main parameters of the new environment.

Also starting from this section, the Hybrid Adaptive Evolutionary Algorithm (HAEA) is used instead of Differential Evolution [31]. The HAEA algorithm (Algorithm 8) has two main features that make it attractive for this problem: (1) it accepts an arbitrary number of operators, so different kinds of crossovers and mutations, even different from the classical mutation and crossover operators, can be used in the same implementation, and (2) individuals evolve independently from one another by producing offspring using operators selected based on probabilities unique to each individual. In this respect, operator probabilities are adjusted by a reward/punishment process, effectively making the adjustment of operator rates automatic, with individuals being only replaced if the fitness of an offspring is better than or equal to the fitness of the parent.

The parameters of the HAEA evolutionary algorithm can be seen in table 5.10. Three different operators are used in these and subsequent tests: A mutation, a linear crossover, and a simple crossover. The mutation operator adds small values to random positions of the individual chromosome, up to 10% of the total positions can be changed at the same time. In particular, the small added values are generated based on a power law like distribution. The linear crossover operator creates two offspring by doing two linear combinations of the individual and another parent selected using tournament selection

Algorithm 8 Hybrid Adaptive Evolutionary Algorithm (HAEA).

```
1: function HAEA(popsize, terminationCondition)
2:     t = 0
3:     P_0 = initPopulation(popsize)
4:     while terminationCondition(t, P_t)== false do
5:         P_{t+1} = ∅
6:         for each individual in P_t do
7:             rates = extractRates(individual)
8:             δ = random(0,1)
9:             oper = selectOperator(operators, rates)
10:            parents = selectParents(P_t, individual)
11:            offspring = apply(oper, parents)
12:            child = best(offspring, individual)
13:            if fitness(child) better than fitness(individual) then
14:                rates[oper] = (1.0 + δ)*rates[oper]                ▷ reward
15:            else
16:                rates[oper] = (1.0 - δ)*rates[oper]                ▷ punish
17:            end if
18:            normalizeRates(rates)
19:            setRates(child, rates)
20:            add(P_{t+1}, child)
21:        end for
22:        t = t + 1
23:    end while
24: end function
```

TABLE 5.10. HAEA Parameters: Evolution with sensors

Parameter	Value
Population Size	30
Number of iterations	100
Mutation Rate	0.2
Selection Tournament size	4
Operators	PowerLawMutation LinearXOver SimpleXOver

TABLE 5.11. Hill Climbing Parameters: Sensors

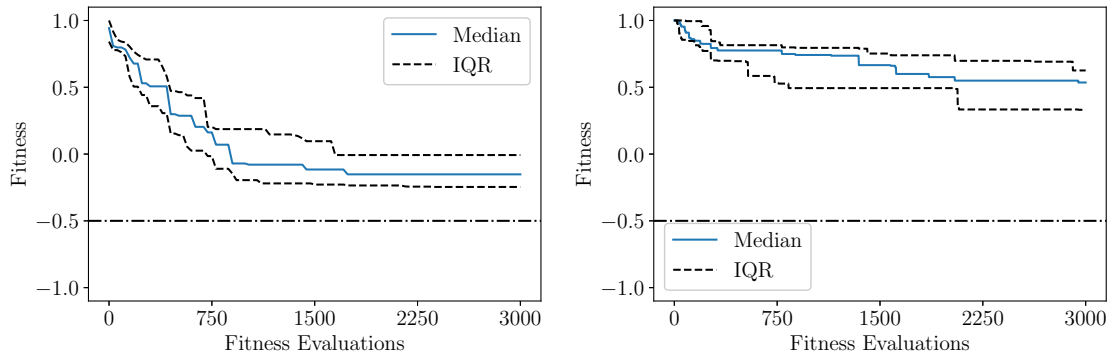
Parameter	Value
Number of iterations	3000
Mutation Rate	0.2
Mutation	PowerLawMutation

[31]. Linear combinations are performed in an element wise fashion and the coefficients are randomly generated using a Gaussian distribution. Finally, the simple crossover operator copies the first part of the individual into the offspring up to a certain random index, the remainder of the offspring is copied from another parent; a second child is created by doing the complementary operation. As mentioned before, replacement is performed on an individual basis. The same test is repeated using Hill Climbing (HC), with the same mutation operator as HAEA.

The fitness function is updated one last time to use less parameters keeping the emphasis on the distance traveled (Equation 5.5). Distance (D) is now measured from the start of the environment to the first module in the topology using normalized Manhattan distance [21]. Time (t) is also normalized using the maximum time available for the test (T). The fitness can be used either in a maximizing or minimizing setting.

$$F = \max D - \frac{t}{1+D} \Leftrightarrow \min -(D - \frac{t}{1+D}) \tag{5.5}$$

Figure 5.17 shows the performance of controllers per fitness evaluation generated using HAEA and HC for 10 runs of each algorithm. Results show that, in general, controllers generated using HAEA are able to move forward in the environment, below the 0.25 mark, which in the $l \oplus r \oplus b$ environment is about the end of the left turn. Furthermore, the best controllers reach the -0.25 mark, which indicates the end of the right turn, and the start of the bump obstacle. In contrast, the best HC generated controllers only reach the 0 mark while the median controller does not even reach the 0.5 mark. This could be a consequence of HC not being able to obtain good solutions by only modifying small parts of the initial controller each time. Figure 5.18 shows a robot running a controller generated using HAEA, it can be seen that the robot moves by encroaching into itself erratically.



(A) (B)

FIGURE 5.17. Performance of controllers generated using (a) HAEA and (b) HC for 10 different executions of each algorithm. Bars show the inter-quartile range.

In both cases (HAEA and HC) controllers are not able to go over the bump obstacle. The overall distance traveled in all combinations of *primenv*s used can be seen in figure 5.19. A Kruskal-Wallistest finds a statistically significant difference between the median distance traveled by controllers generated using HAEA and HC ($p = 0.000111597$ and figure 5.20). Figures 5.19 and 5.20 also show the overall distance traveled in all combination environments by controllers with no sensors (CPG-only) generated using HAEA for the same $l \oplus r \oplus b$ environment. These two figures indicate that controllers using sensors that have been evolved without an initial seed do not travel as much distance as controllers with no sensors (CPG-only). A Kruskal-Wallis test confirms the difference between the median distance traveled by HAEA generated controllers and CPG-only controllers($p = 0.0000236595$). Regardless of the type of controller or the adaptation mechanism used, all controllers seem to specialize on the environment used for training and have problems in the environment that start with a bump.

This bad performance of controllers with sensors compared to controllers without sensors could be explained by the explosion in the number of parameters (from 3 to 234) that makes finding good solutions more difficult in general. On one hand, controllers with sensors movements appear to be more erratic than those of controllers with no sensors (CPG-only) [74]. On the other hand, movements that allow the robot to move forward in the l and r *primenv*s could not be effectively modified by the system to go over the bump *primenv*, hinting a local optimum and a deceptive fitness landscape. This, of course, also affects the performance of the controllers in other combinations of the *primenv*s used, as can be evidenced in figure 5.20.

To show that the locomotion training framework is robust against changes in morphology, the test is repeated using a second robot morphology with topology as in equation 5.6 (Figure 5.21). The HAEA algorithm is again used as adaptation mechanism. A seed controller, trained with back propagation to output specific CPG values that only give some coordinated movements to the robot (obtained by experimentation), is used to generate the initial population in this test. Using this seed the initial population of the evolution-

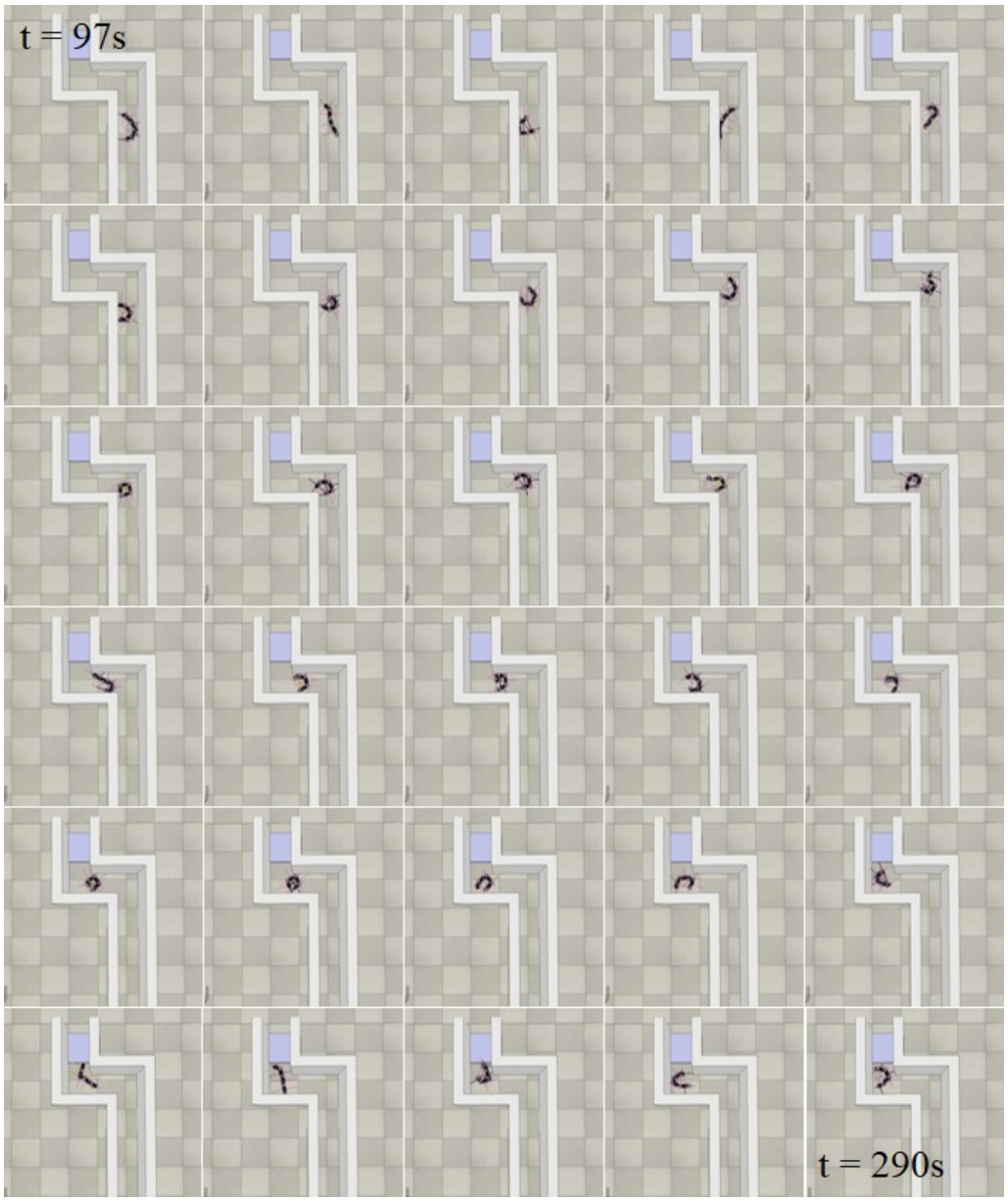


FIGURE 5.18. Simulated robot running a controller with sensors generated using HAEA (HAEA-S) in the $l \oplus r \oplus b$ environment.

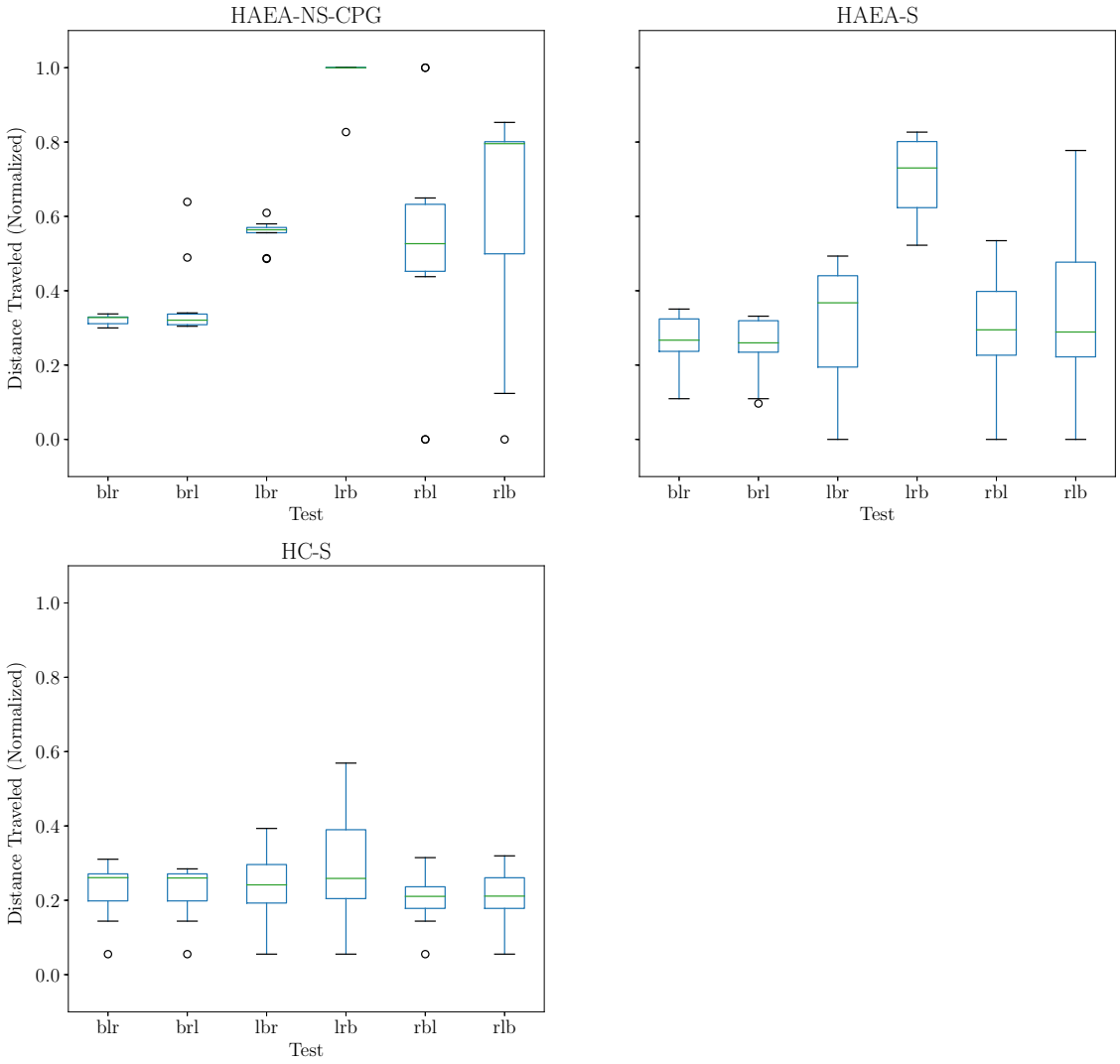


FIGURE 5.19. Overall distance traveled, by test environment, by the best controllers with sensors generated using HAEA (HAEA-S) and HC (HC-S) and HAEA evolving only the CPG coordination mechanism with no sensors (HAEA-NS-CPG) in the realistic EMERGE module, when tested in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

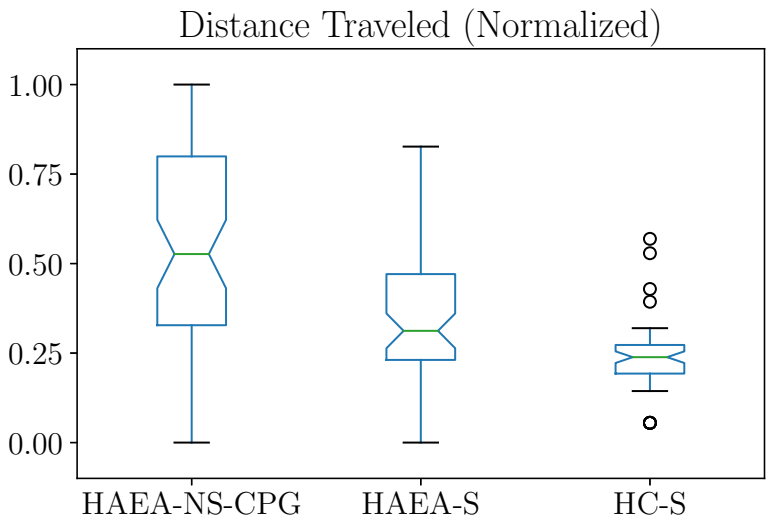


FIGURE 5.20. Overall distance traveled by the best controllers with sensors generated using HAEA (HAEA-S), HC (HC-S) and HAEA evolving only the CPG coordination mechanism with no sensors (HAEA-NS-CPG) in the realistic EMERGE module, when tested in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

ary algorithm is built using a mutation like method. Random values are added to each position in the chromosome around a pre-specified setpoint (0.2). Figure 5.22 shows the performance of the best controllers generated in 10 different executions of HAEA. Figure 5.24 shows the overall distance traveled by the best controllers generated for the T shape morphology in all combination environments, compared to those generated for the Snake morphology. A Kruskal-Wallis test shows an statistically significant difference of the median distance traveled for each morphology ($p = 0.000723859$). T-shaped controllers travel more distance than their snake counterparts (Figure 5.25). Figure 5.23 shows a robot in the T-Shaped morphology moving in the $l \oplus r \oplus b$ environment.

$$T = \{\langle 1,2,2,1,2\rangle, \langle 2,1,1,2,2\rangle, \langle 1,3,3,1,2\rangle, \langle 3,1,1,3,2\rangle, \langle 1,4,4,1,2\rangle, \\ \langle 4,1,1,4,2\rangle, \langle 2,5,2,1,2\rangle, \langle 5,2,1,2,2\rangle, \langle 5,9,2,1,2\rangle, \langle 9,5,1,2,2\rangle, \\ \langle 3,6,2,1,2\rangle, \langle 6,3,1,2,2\rangle, \langle 3,7,3,1,2\rangle, \langle 7,3,1,3,2\rangle, \langle 4,8,2,1,2\rangle, \\ \langle 8,4,1,2,2\rangle\} \quad (5.6)$$

Controllers for the T-shaped morphology tend to get stuck the left and right *primenv*s mainly due to its shape and dimensions (Video in [74]). Despite this disadvantage, controllers for the T-shaped morphology are able to move further in all combinations of *primenv*s than their Snake counterparts. The improved generalization ability of these controllers is explained by the use of the initial seed, which introduces better coordinated movements. A better coordination also makes modifying movements for tackling other environments easier, however, T-shaped morphology controllers are still not able to go

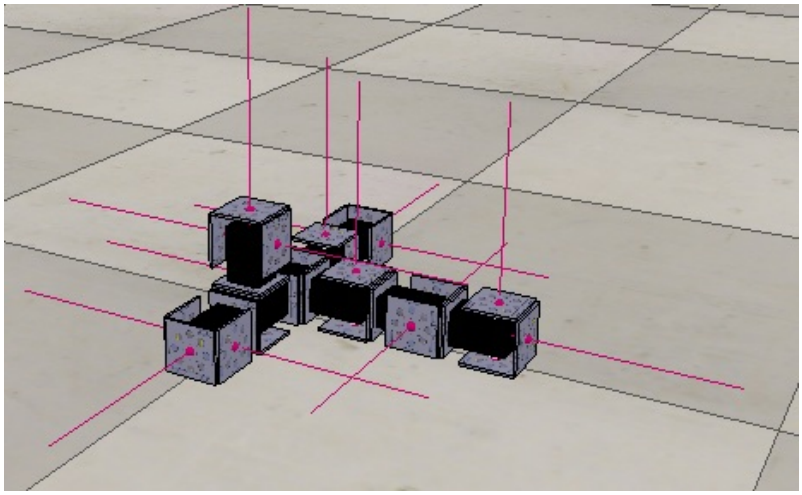


FIGURE 5.21. T-shaped topology in simulation.

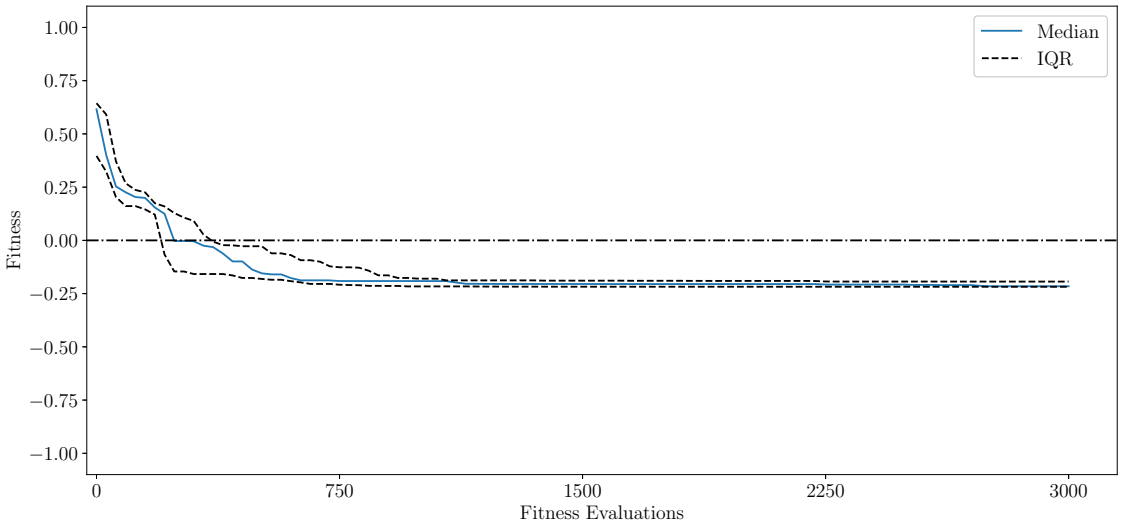


FIGURE 5.22. Performance of controllers generated using 10 seeded executions of HAEA with a T shaped morpholgy. Bars show the inter-quartile range

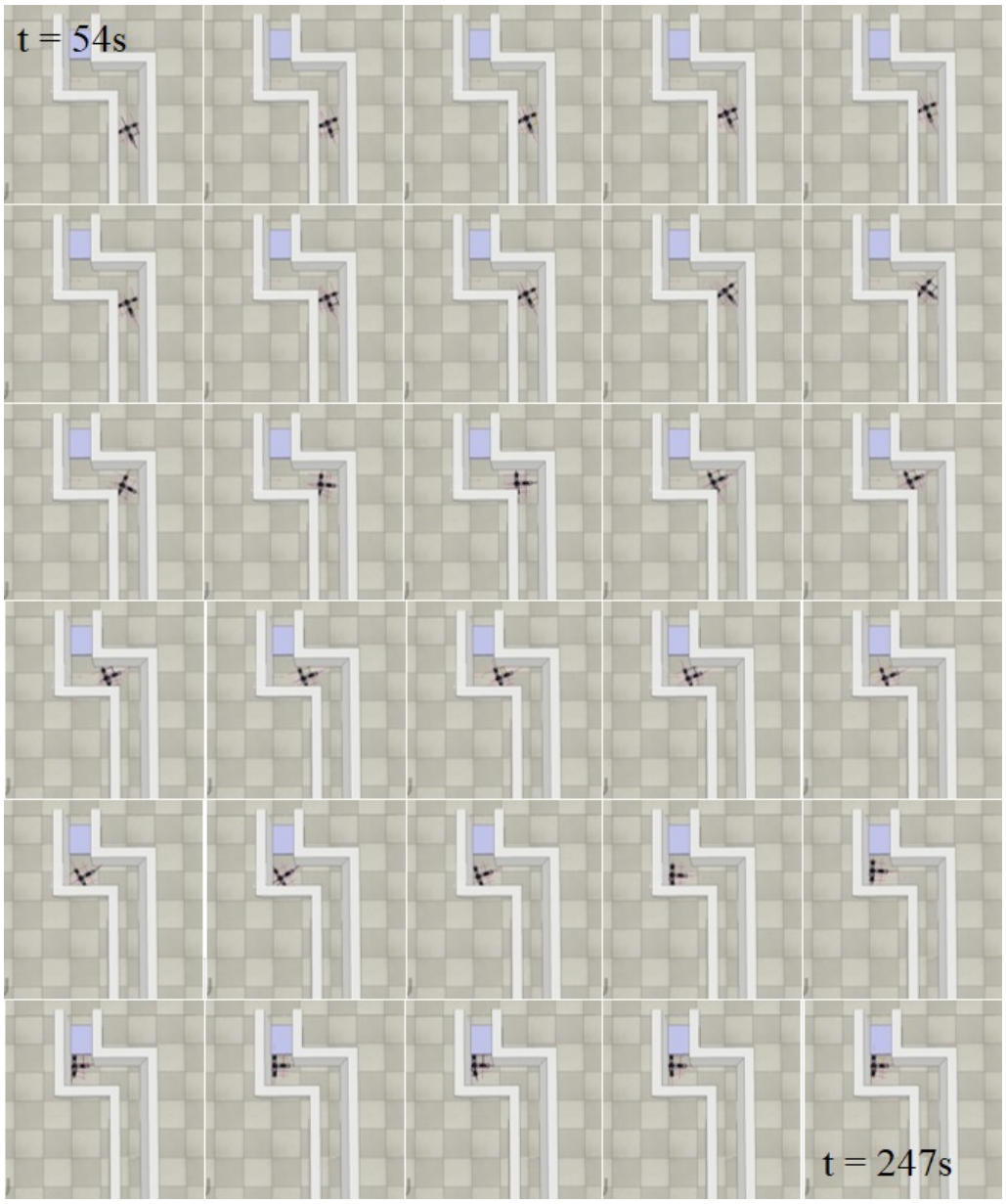


FIGURE 5.23. Simulated robot in a T-Shaped morphology running a controller with sensors generated using HAEA in the $l \oplus r \oplus b$ environment.

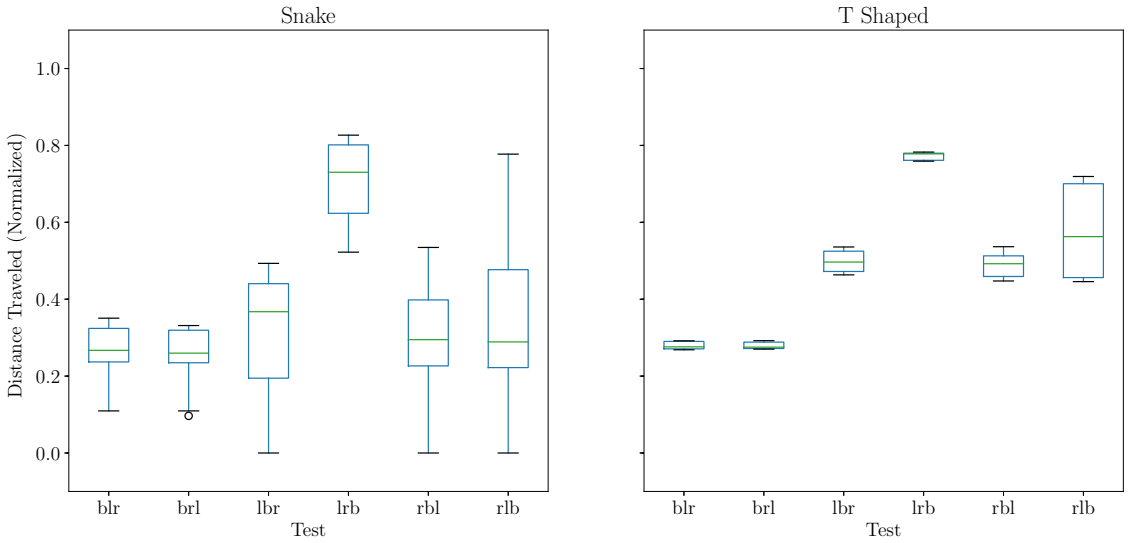


FIGURE 5.24. Overall distance traveled, by test environment, by the best controllers with sensors generated for a snake morphology an a T shaped morphology when tested in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

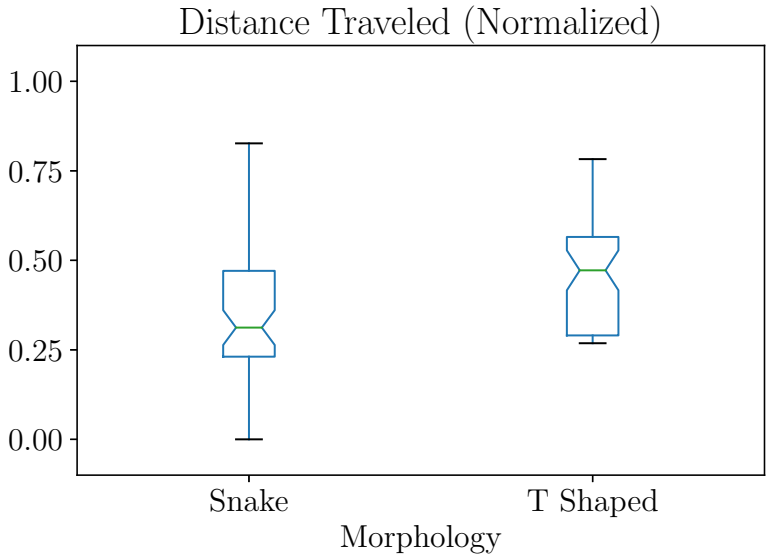


FIGURE 5.25. Overall distance traveled by the best controllers with sensors generated for a snake morphology an a T shaped morphology when tested in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

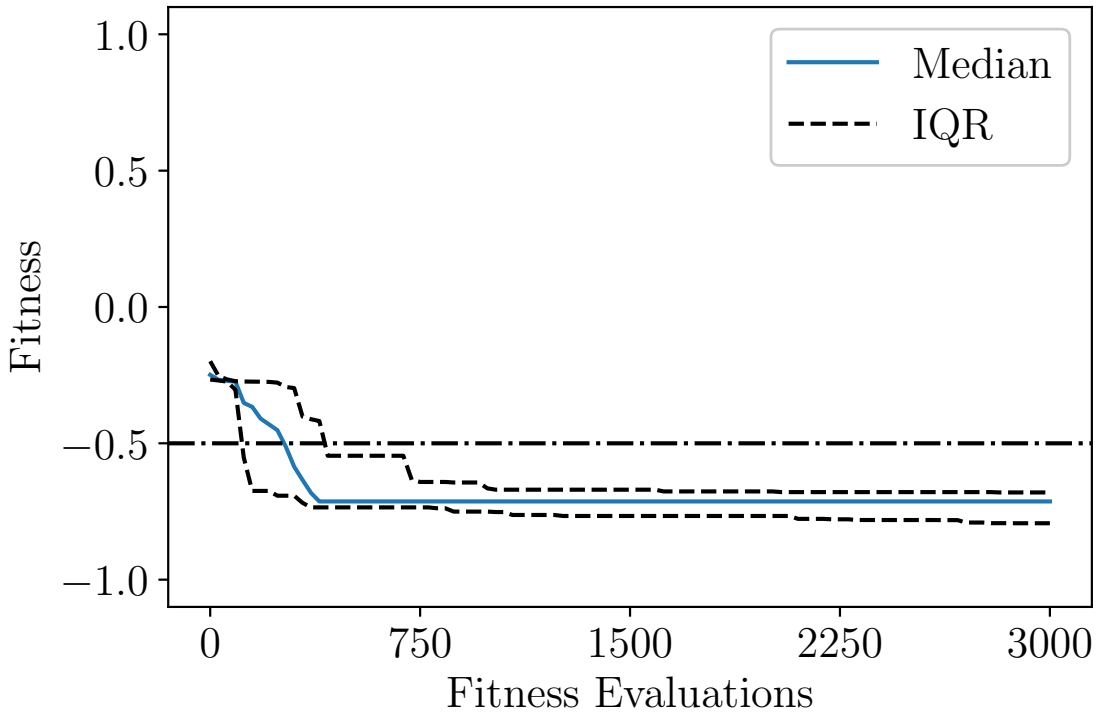


FIGURE 5.26. Performance of controllers generated using 10 different executions of HAEA with a Snake morpholgy and an initial population generated from HAEA-NS-CPG controllers. Bars show the inter-quartile range.

over the bump *primenv*, mainly due to, again, the shape of the robot. Using the initial well coordinated seed controller idea, the test is repeated for the Snake morphology. The initial population is generated from ANNs trained to output the CPG parameters of the best controllers with no sensors, that were previously found (HAEA-NS-CPG). Figure 5.26 shows the controller performance for the seeded test(HAEA-S-CPGSeed). Figure 5.27 shows a robot running a controller generated using this method, movements appear to be more well coordinated than in the HAEA-S case.

Training the ANNs for the initial population introduces errors in the controllers. As a result, initial controllers are not able to get to the exit of the $l \oplus r \oplus b$ environment on their own, as HAEA-NS-CPG controllers did. Despite this, resulting controllers are able to clear the training environment. Figure 5.29 shows the overall distance traveled by the best controllers in the CPG seeded test in all combination environments. This figure suggests that, although there is not a statistically significant difference between the two groups, controllers generated using the CPG seed are slightly better than their CPG only counterparts when moving in almost all combination environments. This behavior is also evidenced on figure 5.28, in which again no statistically significant difference is found between the two groups.

A well coordinated initial population provides a better chance for the sensor mechanism to improve the generalization ability of the controllers. This increase can be attributed

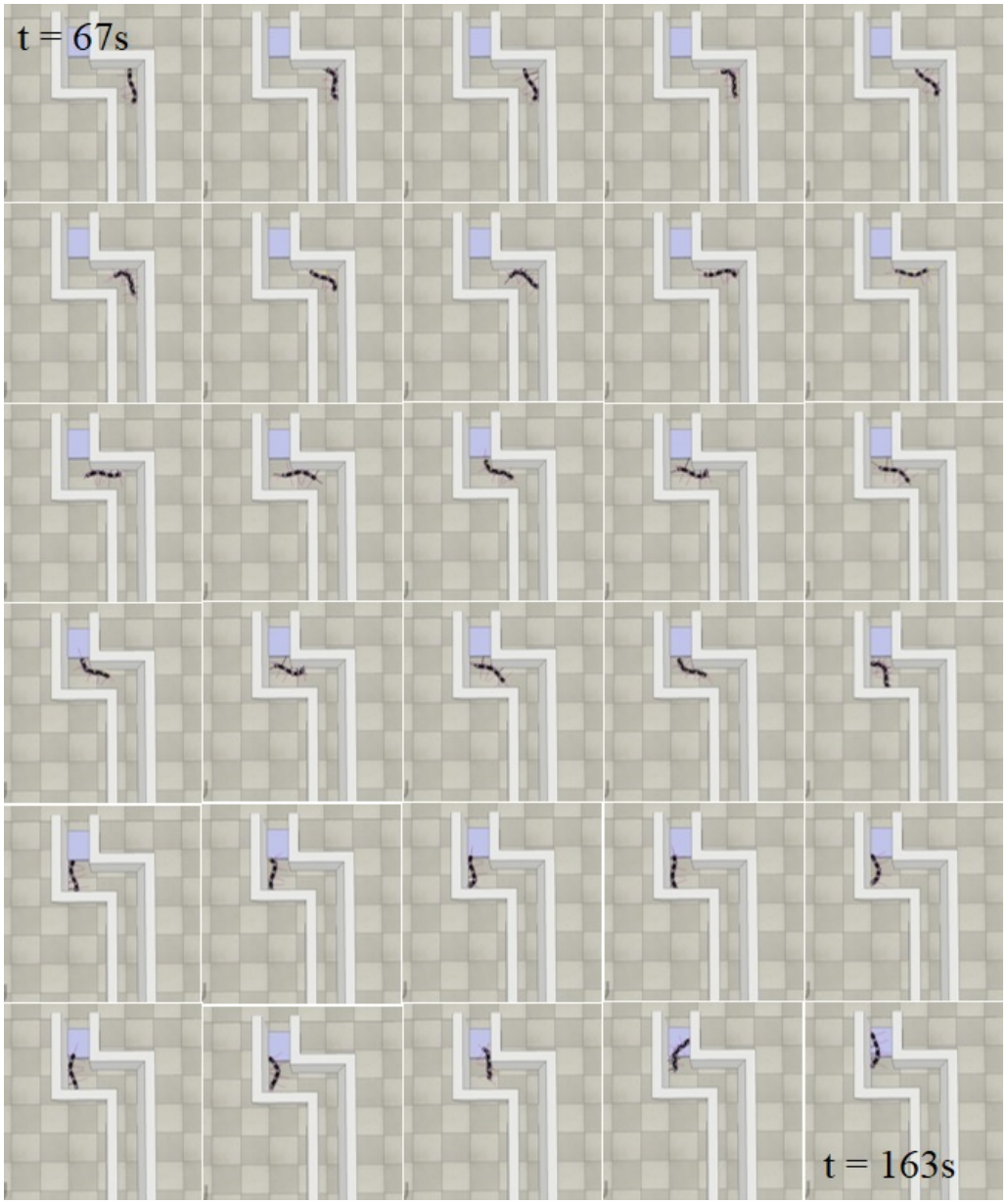


FIGURE 5.27. Simulated robot running a controller with sensors generated using HAEA and using a initial population of previously found CPG controllers (HAEA-S-CPGS) in the $l \oplus r \oplus b$ environment.

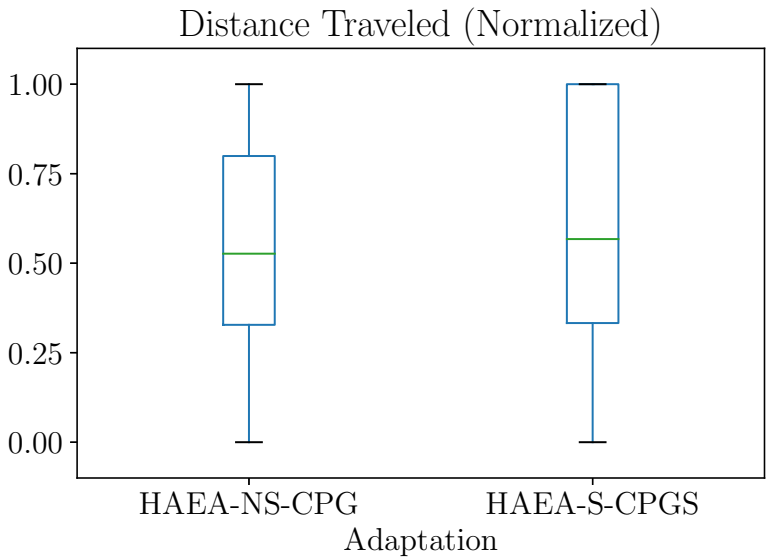


FIGURE 5.28. Overall distance traveled by the best controllers with sensors generated using a CPG seed (HAEA-S-CPGS) and controllers generated using the only CPG coordination mechanism (HAEA-NS-CPG), in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

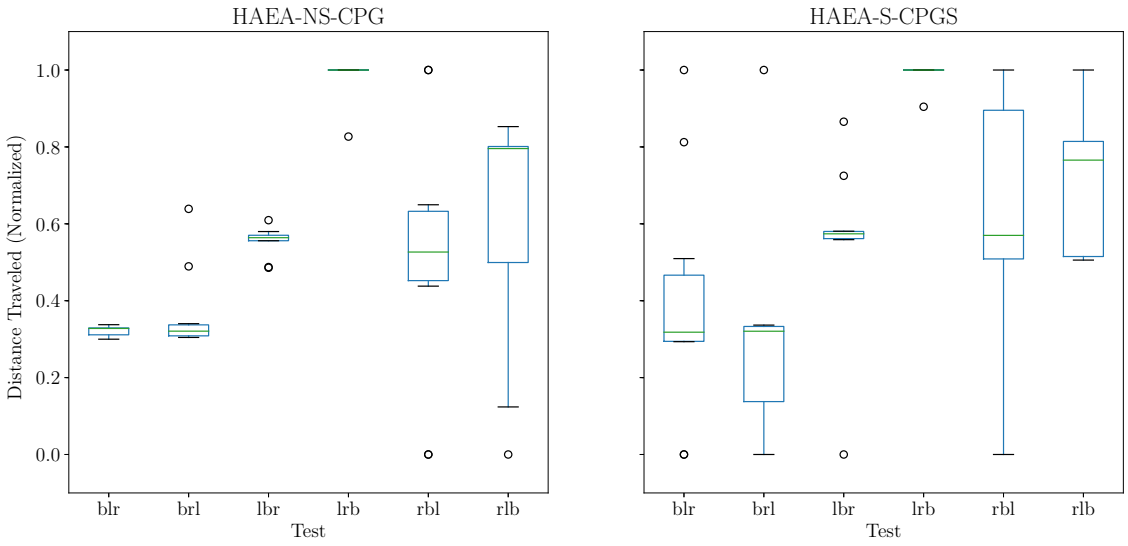


FIGURE 5.29. Overall distance traveled, by test environment, by the best controllers with sensors generated using a CPG seed (HAEA-S-CPGS) and controllers generated using the only CPG coordination mechanism (HAEA-NS-CPG), in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

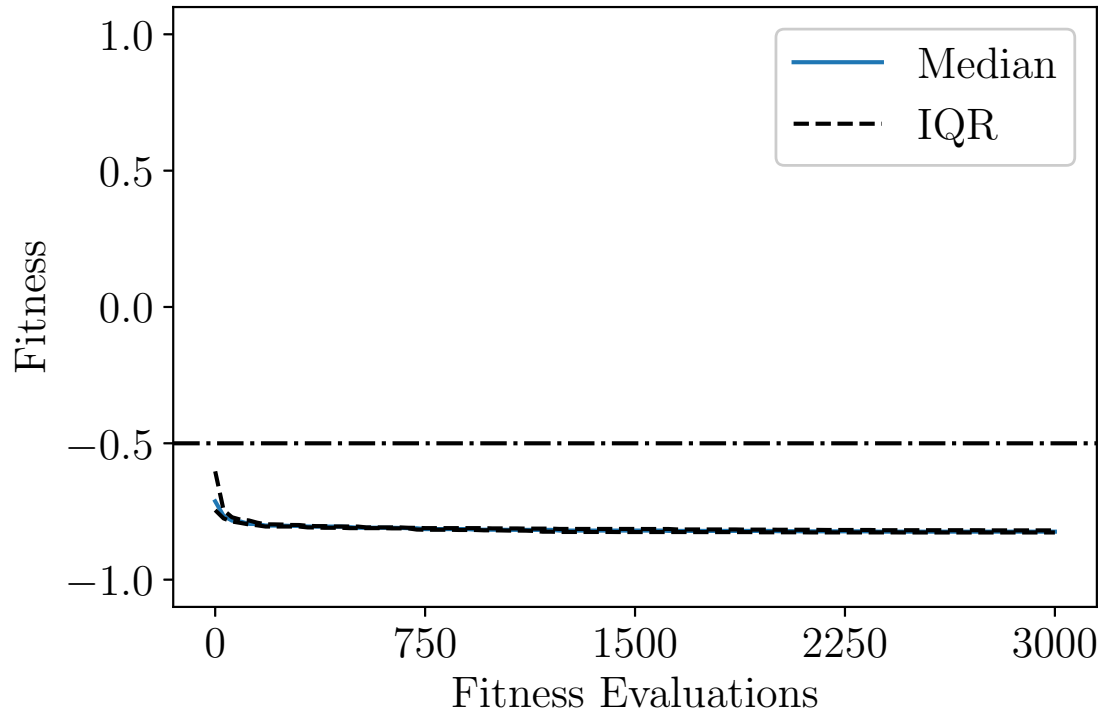


FIGURE 5.30. Performance of controllers generated using 10 different executions of HAEA with a Snake morpholgy and a manually generated seed for the initial population.

to two reasons: (1) It makes controllers move less erratically (Video in [74]), and (2) it increases the chances for the sensor system to find movements useful for tackling other *primenv*s, as indicated by figure 5.28. Thus, results suggest that a well coordinated set of movements should exist for a sensor message system, of the type proposed, to take advantage of them, and that is not trivial to generate these movements from scratch with the sensor message system in place.

The importance of having well coordinated movements for controllers with sensors to take advantage of is further evidenced when using a manual controller seed, built with coordination in mind. In particular, this seed allows the robot to move forward in a sinusoidal fashion with simpler and more ample movements. Figure 5.30 shows the performance of controllers generated using the manual seed.

In this last test, despite HAEA not being capable of improving the fitness of the controllers, generalization improves compared to that of controllers evolved with the CPG coordination mechanism only (HAEA-NS-CPG) and controllers evolved using the CPG seed. Figure 5.31 shows the overall distance traveled by the best controllers generated in all combination environments. A Kruskall-Wallis test shows a statistically significant difference on the median distance traveled ($p = 2.69866e^{-8}$). Figure 5.32 indicates that controllers generated using the manual seed travel more distance than their CPG only counterparts.

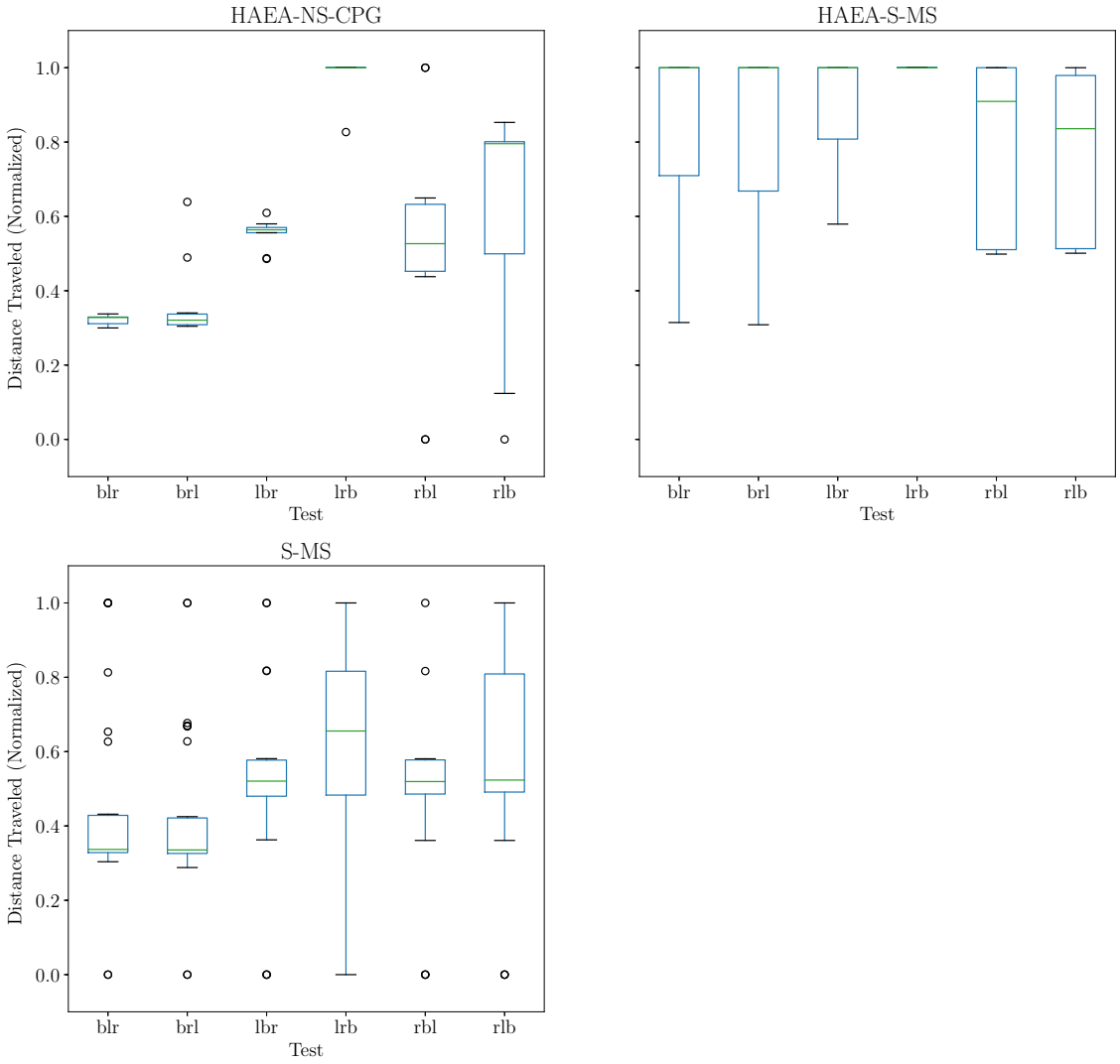


FIGURE 5.31. Overall distance traveled, by test environment, by the best controllers with sensors evolved using a manual seed (HAEA-S-MS), the initial population obained from the manual seed (S-MS) and controllers generated using the only CPG coordination mechanism (HAEA-NS-CPG), in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

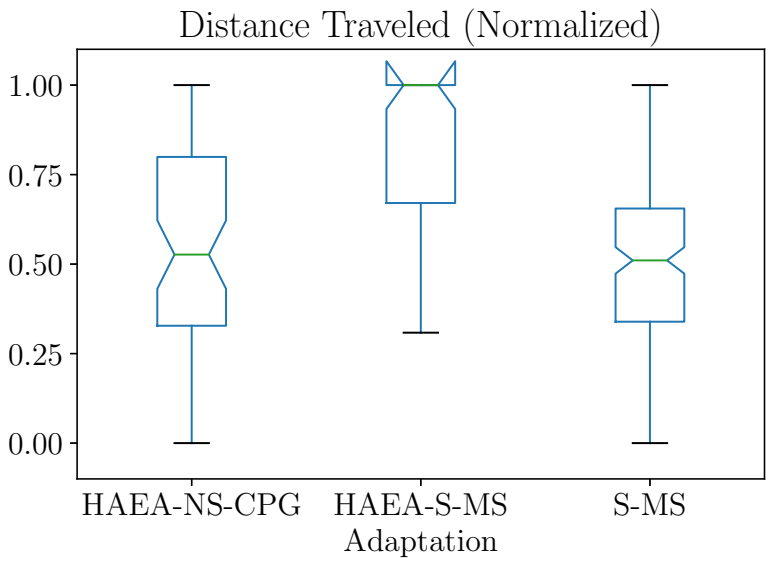


FIGURE 5.32. Overall distance traveled by the best controllers with sensors evolved using a manual seed (HAEA-S-MS), the initial population obtained from the manual seed (S-MS), and controllers generated using only the CPG coordination mechanism (HAEA-NS-CPG), in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

As mentioned before, the result of this last test stresses the importance of having well coordinated movements, which a sensor information handling mechanism and a decision mechanism can make use of. This is evidenced by the the difference in the overall distance traveled by controllers in the initial population generated from the manual seed controller before evolution (S-MS in figures 5.32 and 5.31) and controllers evolved using this same initial population (HAEA-S-MS). The simpler movements of the manual seed controller could have also aided the sensor message system to produce movements better suited for the different *primenv*s. Results also indicates that the evolution process, despite not improving the fitness of the controllers, is able to modify them and find ANNs that can move in different environments using sensor information. Further testing is needed to find out the exact way in which these ANNs make use of the information coming from the environment, which sensor messages are prioritized and which are shunned when selecting the parameters of the underlying CPGs.

5.5 Short Challenges

To further test the importance of having well coordinated movements when training a robot with sensors, an incremental evolution approach is proposed for training a robot from scratch. As mentioned before, incremental evolution gradually changes the task a robot is tested in so that it acquires the ability to perform more challenging tasks. Following this idea, the incremental evolution approach proposed involves a series of short challenges that would allow a robot to gradually obtain the movements necessary to move in the different *primenv*s presented.

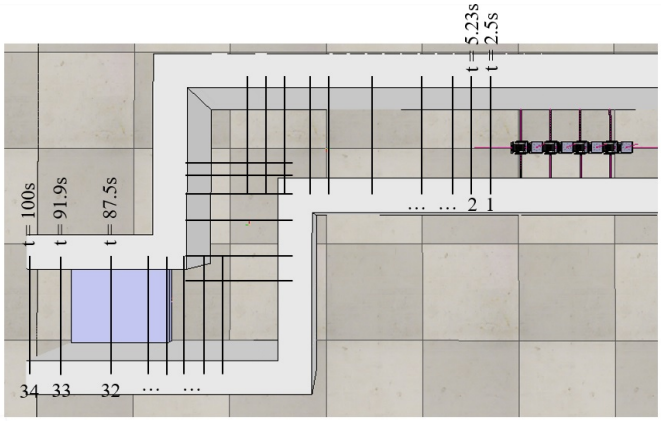


FIGURE 5.33. Short challenges in the $l \oplus r \oplus b$ combination environment. Black lines represent the goal of each challenge and numbers indicate the challenges sequence.

Short challenges can be defined over any *primenv* or normal environment. At first, a fraction of the total distance of the environment is shown to the controller being trained over a small amount of time. If the controller is able to cover the initial fraction a larger fraction of the environment is shown and more time is given to the robot. The process is repeated until the whole environment is covered. Short challenges also help to test the controller in new parts of the environment in a more controlled way so the decision mechanism is not overwhelmed by sensor information. In other words, the adaptation process is able to test individuals on very similar situations, e.g. with similar sensor inputs, using more evaluations, and the situation is only changed if an individual with suitable movements appears. Figure 5.33 depicts the short challenge incremental evolution scheme. Black lines represent the goal of each challenge and numbers indicate the challenges sequence.

With the same combination environment used in last section ($l \oplus r \oplus b$), a short challenge fitness approach is used to evolve controllers for robots in the snake morphology (Equation 5.1). The straight strip at the beginning of the *primenv* combination is eliminated in this test and instead the initial population of controllers is obtained from a previously evolved set of controllers with sensors, trained in a straight *primenv*. Still, these controllers trained in the straight *primenv* are evolved from scratch, i.e. without regard for having well coordinated movements.

The fractions of the environment and the times used for each short challenge in the $l \oplus r \oplus b$ *primenv* are shown on table 5.12. A challenge is completed if one of two conditions are met: (1) an individual that travels the total distance of the challenge appears, (2) evolution under the challenge conditions reaches the max number of generations. Each challenge is evolved with a population of 30 individuals for a max of 20 generations. The full population found at the end of one challenge is used as the initial population of the next challenge.

Figure 5.34 shows that in the left and right *primenv*s the incremental evolution approach is able to find controllers that solve all challenges. However, in the bump *primenv* controllers are not able to overcome challenges after a certain point. Nevertheless, at least

TABLE 5.12. Short challenges for the $l \oplus r \oplus b$ environment in the form (fraction,time(s)). Transitions indicate where each *primenv* starts.

Challenges	(0.016,2.5),(0.056,5.23),(0.11,10.2),(0.15,13.9),(0.18,16.6),(0.22,20.5), (0.23,21.5),(0.25,23.4),(0.26,24),(0.27,24.9),(0.30,27.6),(0.33,30), (0.35,32.5),(0.39,35.23),(0.44,40.2),(0.48,43.9),(0.51,46.6),(0.55,50.5), (0.56,51.5),(0.58,53.4),(0.59,54),(0.60,54.9),(0.63,57.6),(0.66,60), (0.68,69.5),(0.7,70.2),(0.72,72.23),(0.74,74.5),(0.76,76.6),(0.81,80.9), (0.85,83.6),(0.89,87.5),(0.93,91.9),(1,100)
Transitions	l:0,r:12,b:24

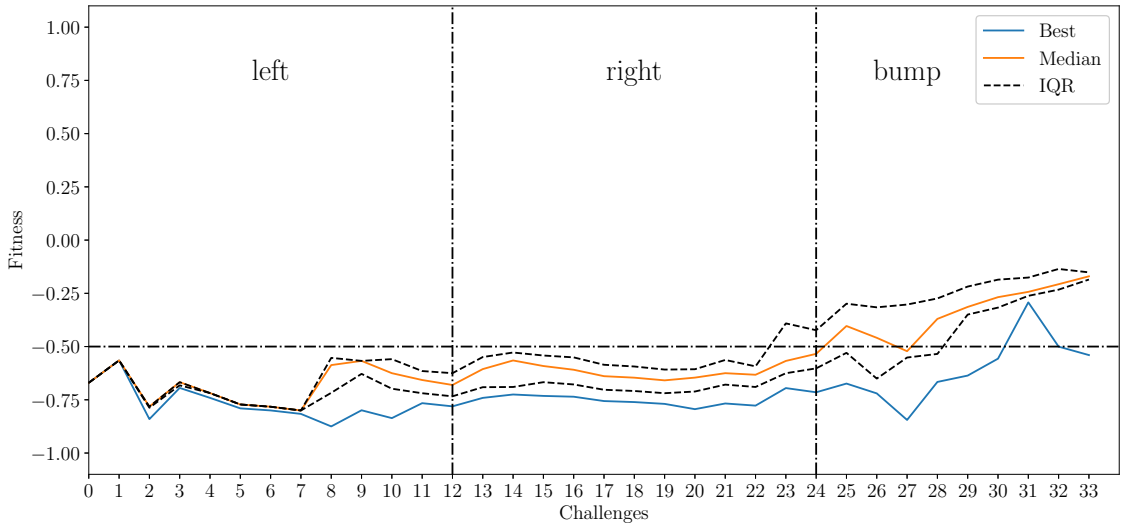


FIGURE 5.34. Performance of the best controllers generated by 10 different executions of the short challenge incremental approach. A fitness below −0.5 indicates the challenge being completed. Vertical lines indicate where each *primenv* starts. Bars show the interquartile range.

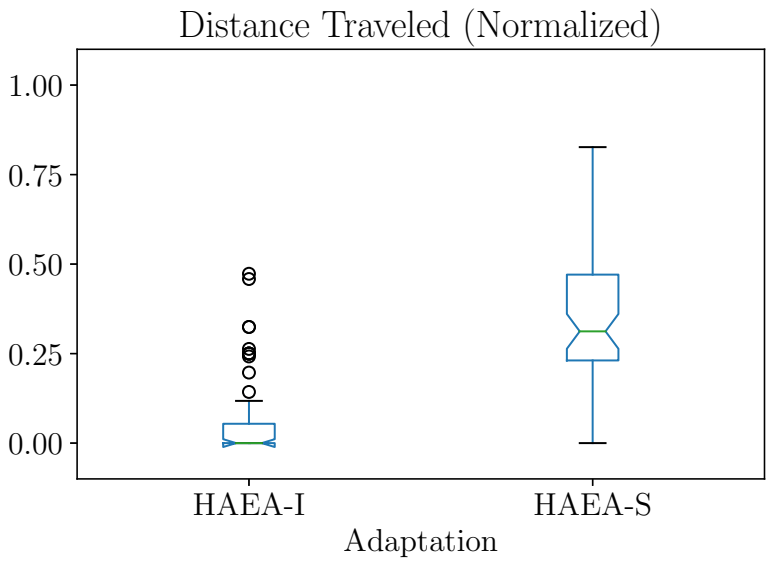


FIGURE 5.35. Overall distance traveled by the best controllers with sensors generated using the incremental short challenge approach (HAEA-I) and normal evolution without seeds (HAEA-S), in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

one controller is able to go over the bump *primenv* as indicated by the "Best" line in figure 5.34.

Despite enabling at least one controller to go over the bump, controllers generated in this test suffer from a loss in generalization ability. When measuring the overall distance traveled by these controllers in all 6 *primenv* combinations, the median distance traveled is not as high as with controllers using a normal evolutionary approach (Figure 5.35). This indicates that controllers generated with the short challenge approach specialize more than their normal evolution counterparts. One possible explanation for this behavior is that as each challenge is completed the information about the search space represented in each subsequent population is further reduced. Additionally, although exposed for a greater number of evaluations to the same situation, controllers do not seem to produce well coordinated movements. This could be a consequence of controllers facing easier tasks that can be solved with erratic movements, thus demonstrating that it is not trivial to find controllers with well coordinated movements, even when using an incremental system. This prolonged exposure could also increase the chances of specialization.

In spite of the specialization problem, and advantage of the short challenges approach is that it reveals the places where the adaptation process is having problems. To get more detail on the behavior of controllers in each *primenv* the same test is repeated with separated *primenv*s, that is, first, a set of short challenges is run in one *primenv*. Next, another set of challenges is run in the next *primenv* using the last population found on the last *primenv* as the initial population, and so on. The same order of *primenv*s is used as in the last test (first left, next right, and last bump). Table 5.13 shows the fractions

TABLE 5.13. Short challenges in the form (fraction, time (s)) for the sequence left-right-bump of separated *primenv*s .

primenv	**Challenges**
left	(0.05,2.5),(0.17,5.23),(0.33,10.2),(0.45,13.9),(0.55,16.6),(0.67,20.5), (0.7,21.5),(0.75,23.4),(0.78,24),(0.82,24.9),(0.91,27.6),(1,30)
right	(0.05,2.5),(0.17,5.23),(0.33,10.2),(0.45,13.9),(0.55,16.6),(0.67,20.5), (0.7,21.5),(0.75,23.4),(0.78,24),(0.82,24.9),(0.91,27.6),(1,30)
bump	(0.05,2.5),(0.1,3.2),(0.17,5.23),(0.24,7.5),(0.3,9.6),(0.45,13.9), (0.55,16.6),(0.67,20.5),(0.8,24.9),(1,30)

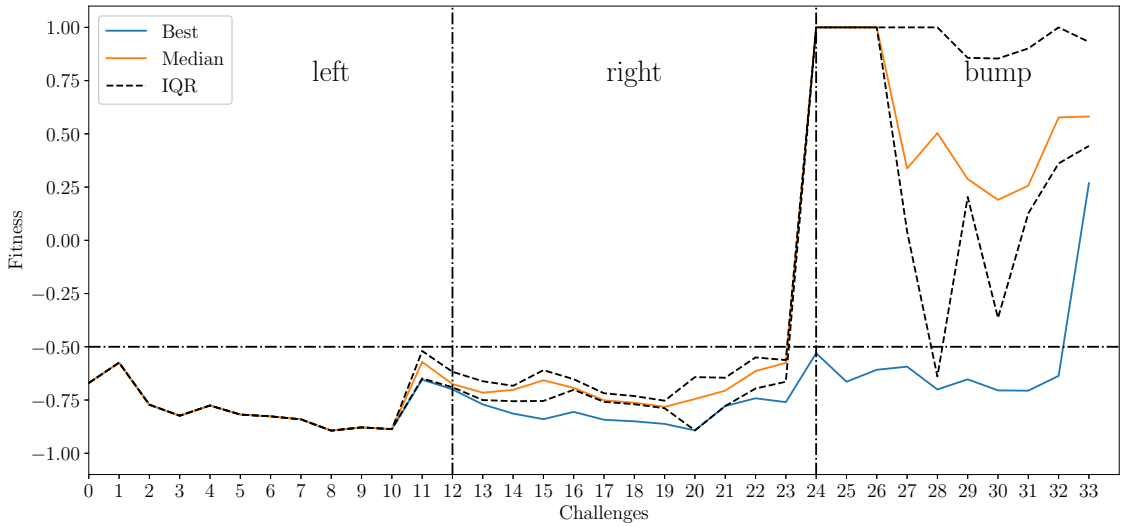


FIGURE 5.36. Performance of the best controllers generated by 10 runs of using the short challenge incremental approach with a sequence of separated *primenv*s (l-r-b). A fitness below −0.5 indicates the challenge being solved. Vertical lines indicate where each *primenv* starts. The full population at the end of one challenge is used as the initial population of the next one. Bars show the inter-quartile range.

and times used for each challenge in each *primenv*. Figure 5.36 shows the result in each *primenv*.

Results show that when using separated *primenv*s in sequence, controllers get worse fitness when transitioning to another challenge, which is the case when going from right to bump. As a matter of fact, figure 5.36 clearly shows that finding a good controller for going over the bump *primenv* is a very difficult task. In spite of this, at least one controller is able to overcome it. Erratic behavior can also be observed in the controllers found as in the $l \oplus r \oplus b$ environment case.

Controllers for robots using the T shaped morphology (Equation 5.6) are also evolved using the short challenge approach, both using the $l \oplus r \oplus b$ combination environment and the sequence of separated sub-environments. Figure 5.37 shows the best controllers generated by 10 different executions of the short challenge incremental approach in the $l \oplus r \oplus b$ environment. Figure 5.38 shows the result in each *primenv* when using the separated sequence of *primenv*s.

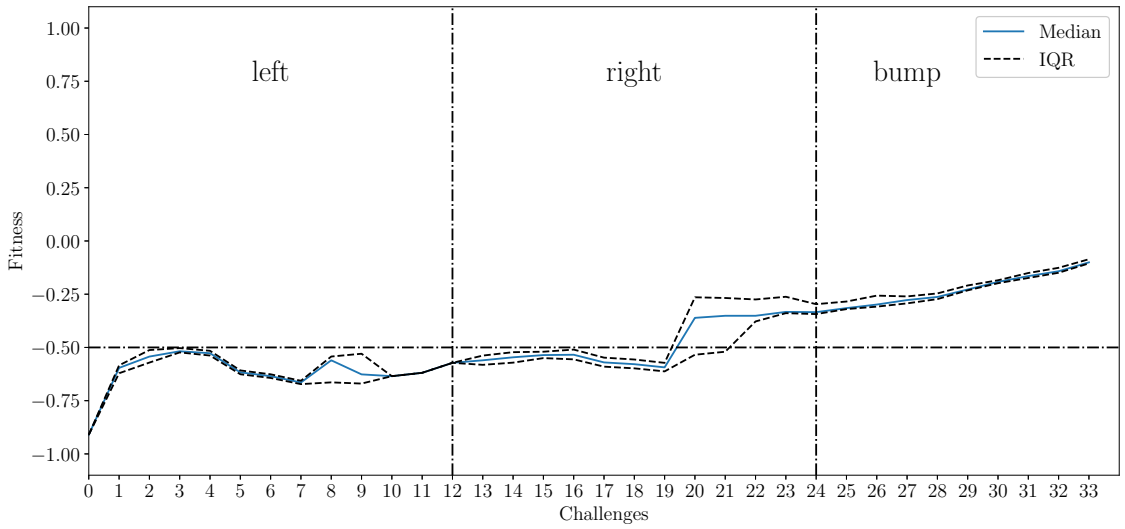


FIGURE 5.37. Performance of the best controllers generated for the T shaped morphology by 10 different executions of using the short challenge incremental approach. Vertical lines indicate where each *primenv* starts. Bars show the inter-quartile range.

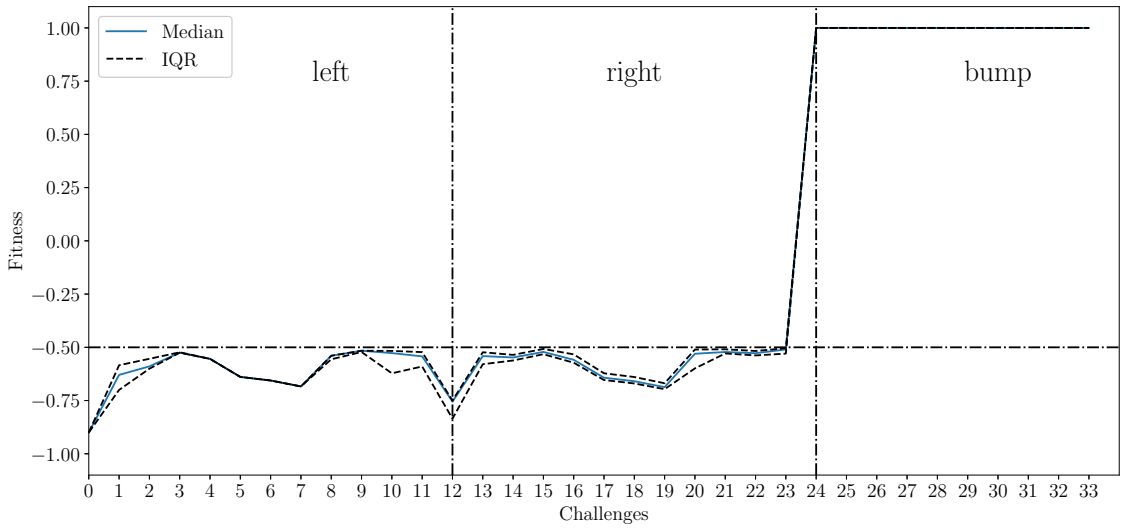


FIGURE 5.38. Performance of the best controllers generated for the T shaped morphology by 10 runs of using the short challenge incremental approach with a sequence of separated *primenv*s (l-r-b). Vertical lines indicate where each *primenv* starts. Bars show the inter-quartile range.

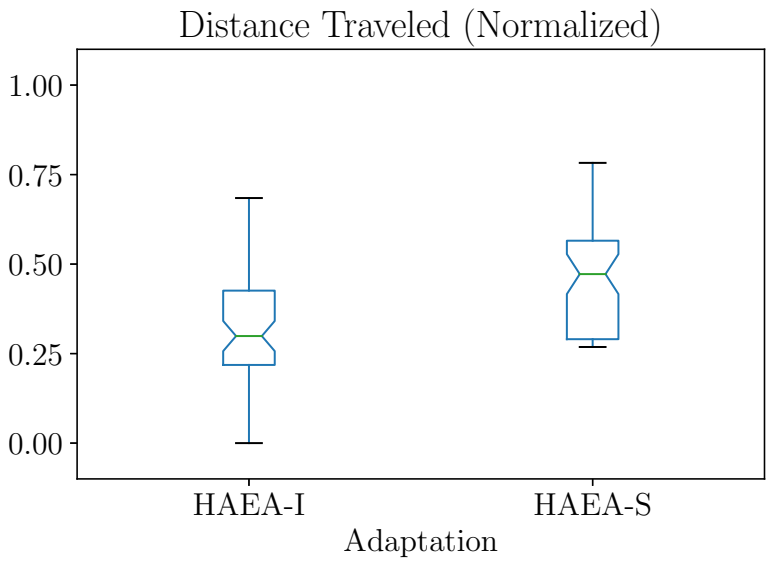


FIGURE 5.39. Overall distance traveled by the best controllers generated for the T shaped morphology using the incremental short challenge approach (HAEA-I) and using normal evolution (HAEA-S), tested in the 6 combination environments. Whiskers represent $1.5 \times IQR$.

Figures 5.37 and 5.38 show a very similar behavior to their snake morphology counterparts. In particular, when using the $l \oplus r \oplus b$ environment controllers are not able to cover the whole distance of the challenge in the right turn, this is a consequence of the shape of the morphology itself, which occupies almost all the space of the turn and is more prone to get stuck. The controllers generated for the T shaped morphology also show less generalization as less distance is covered in all 6 combinations of *primenv*s compared to a normal evolution (Figure 5.39).

5.6 Hardware Experiments

Previous sections have demonstrated the use of an adaptation mechanism, with a configurable environment approach, and a controller scheme, combining coordination, sensor and decision mechanisms, to generate controllers capable to travel through different environments. However, all tests have been performed in simulation and, despite being very useful for identifying the main issues and possible solutions, they may not completely correspond to what happens in reality.

5.6.1 Transferring controllers to reality

To show the difference between simulated and real EMERGE modules, the best individuals obtained in the simulation process are transferred to reality and compared to their simulation counterparts. A straight strip *primenv*, of length 1.5m, width 0.3m and walls

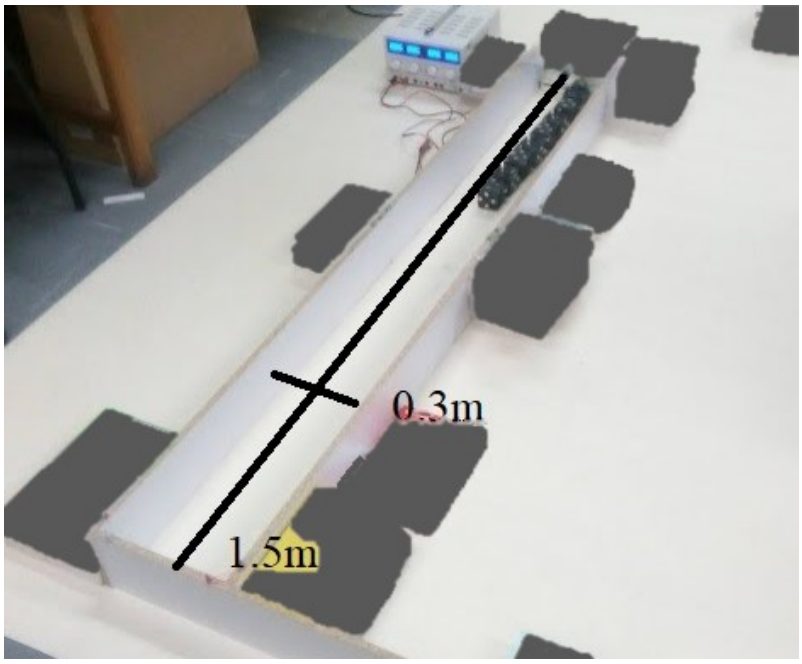


FIGURE 5.40. Straight *primenv* in reality.

TABLE 5.14. Performance of transferred controllers of simulated robots. Time (s) to reach the 0.5 mark in a straight $1.5m \times 0.3m$ environment. An empty distance means the robot did not move forward or moved backwards.

Controller	**Simulated**	**Real**
1	33	303
2	47	507
3	224	–
4	35	–
5	88	451

0.14m high is built (Figure 5.40). The same snake topology of equation 5.1 is used to build a robot out of EMERGE modules. Modules are powered using an external voltage source through an electrical cable. The controller and the position of the module in the morphology are programmed into each module using a microcontroller dedicated programmer (Cypress KitProg). Controllers running in the real EMERGE modules are almost identical to their simulated counterparts, even using the exact same parameters as in section 5.4. Robot morphologies are positioned at the start of the environment, powered on and tested for 10 min. Table 5.14 shows the time for each robot to reach the 0.5m mark. The reduced number of tests is due to the high cost of evaluating controllers in reality. Figure 5.41 shows a robot moving forward in the straight *primenv*.

Individuals that moved in the straight *primenv* are also tested in a right corner *primenv* (Figure 5.42). In [74] the best individual runs in the right corner are shown. Table 5.15 shows the times obtained by each individual to get to travel to the other side of the environment. Figure 5.43 shows a robot moving forward in the corner *primenv*.

FIGURE 5.41. Robot moving forward in the real straight *primenv* in a 10 minute test. Frames go from left to right starting in the upper-leftmost one.

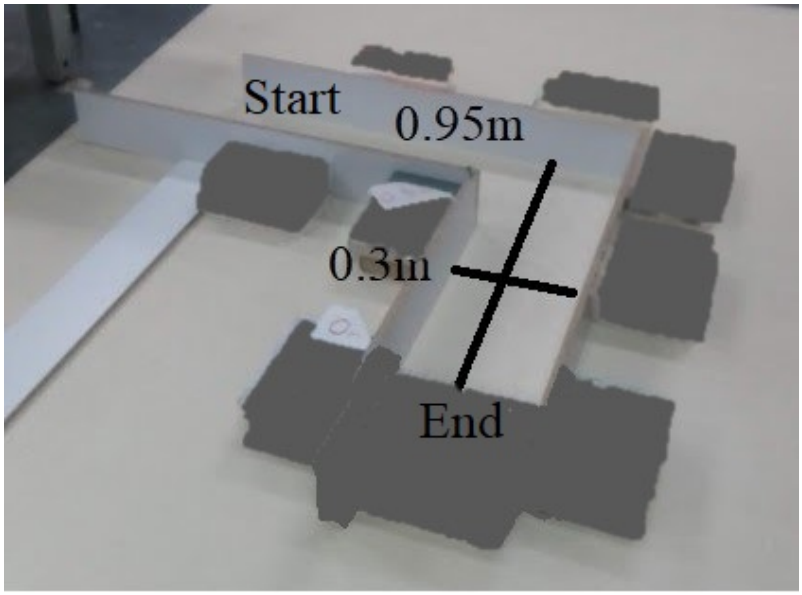


FIGURE 5.42. Right corner *primenv* in reality.

TABLE 5.15. Performance of transferred controllers of simulated robots, in the right corner *primenv*. Distance traveled (m) in 10 minutes and time (s) used for gettin from start to end. An empty distance and time means the robot did not move forward or moved backwards.

Controller	**Distance (m)**	**Time(s)**
1	1.19	600
2	0.89	600
3	1.49	549
4	–	–
5	1.45	395

FIGURE 5.43. Robot moving forward in the corner straight *primenv* in a 10 min test. Frames go from left to right starting in the upper-leftmost one.

While the reality gap is a very studied problem, differences found when transferring simulated controllers to reality are very big. Differences with simulation start arising when transferring the control scheme to the real EMERGE robot. As real EMERGE modules do not have an internal orientation sensor a special accessory is designed to measure the current orientation of the module. Such accessory reads an analog accelerometer (ADXL335) every second with a 12 bit resolution and determines its orientation relative to the ground. The accessory is attached to a module and broadcasts the orientation information to all modules in the morphology. Each module is in charge of transforming the global orientation measure into its local orientation using topology information. As a result, the mechanical structure of an EMERGE robot morphology is slightly modified, something that affects its movement. Another factor affecting the mechanical structure of the robot is the internal connection (cables, connectors, etc..) that can prevent modules from reaching their full movement range [74]. Differences in servo-motor torques among modules, environment friction forces, magnet forces, dimensions, etc, all contribute to make the movements performed by real modules different to those of simulated modules.

On the software side, the reduced processing power of the microcontroller in each robot affects how fast calculations can be carried out and introduces lag in the movement too. However, real controllers are mostly affected by the limited number of physical and software buffers used in communication. While simulated modules can virtually receive an infinite number of messages from each neighbor in each time step, real modules are limited to only 3 per neighbor. This is another reason for the implementation of attenuation and forward propagation in the sensor information handling mechanism, which is also present in the simulation experiments. In general, the control scheme is also designed to be as light as possible taking into account the main limitations of the real modules. The effect of these factors on the performance of controllers could be reduced by evolving directly on the real EMERGE modules.

5.6.2 Testing the framework

In order to obtain better controllers in reality, the full locomotion training framework is implemented. A robot topology with only three modules (Equation 5.7 and figure 5.44), is used in this case, the smaller topology reduces the sources of sensor messages and makes coordination easier. Simulation is not completely discarded but is instead used, with some tweaks, to initialize the system. In last section, transferred simulated controllers were not able to move real modules as well as simulated ones. One of the main observations was that modules actuators could not reach their full movement range due to their internal connections. In this test, the movement range of simulated modules is tweaked so that it corresponds more closely to the one observed in the real robots. Simulated controllers are generated using 10 executions of HAEA with the three module topology.

$$T = \{\langle 1,2,2,1,2\rangle, \langle 2,1,1,2,2\rangle, \langle 2,3,2,1,4\rangle, \langle 3,2,1,2,4\rangle\} \tag{5.7}$$

FIGURE 5.44. Three module topology in reality.

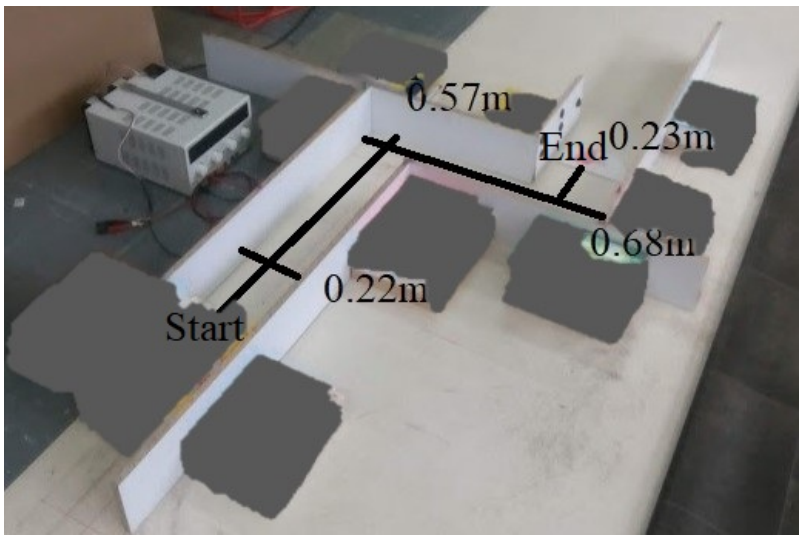


FIGURE 5.45. Environment $r \oplus l$ in reality.

Using the best controllers obtained in simulation as initial population, HAEA is used to evolve controllers directly in the real robots. For this purpose a configurable environment with only left and right *primenvs* is built. Specifically, environment $l \oplus r$ is used to evolve controllers. The environment dimensions can be seen on figure 5.45. The evolutionary setup uses the exact same parameters as in section 5.4, with the exception of the max number of generations, which is now 10, and the number of individuals, which now is 5. This reduction in the number of generations and individuals used is due to the high cost of evaluating the robots in reality (3 weeks were necessary to run all tests). Distance is measured manually on the environment and time is measured using a stopwatch. The maximum time allowed (T) is 300 seconds (5 minutes) per evaluation (each evaluation takes around 15 minutes to complete). In [74] the process of measuring both variables is shown. As always, the fitness function portrayed in equation 5.5 is minimized. Figure 5.46 shows an individual run in the $l \oplus r$ environment. The best individuals obtained can be seen in [74].

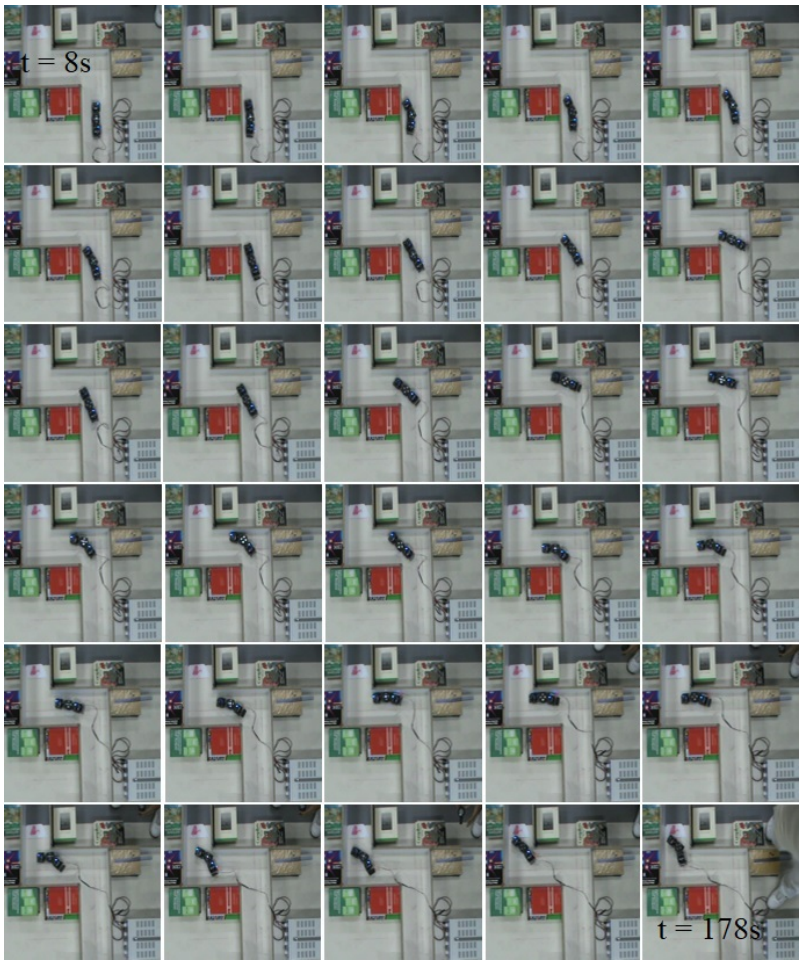


FIGURE 5.46. Robot moving in the $l \oplus r$ environment in a 3 minute evaluation. Frames go from left to right starting in the upper-leftmost one.

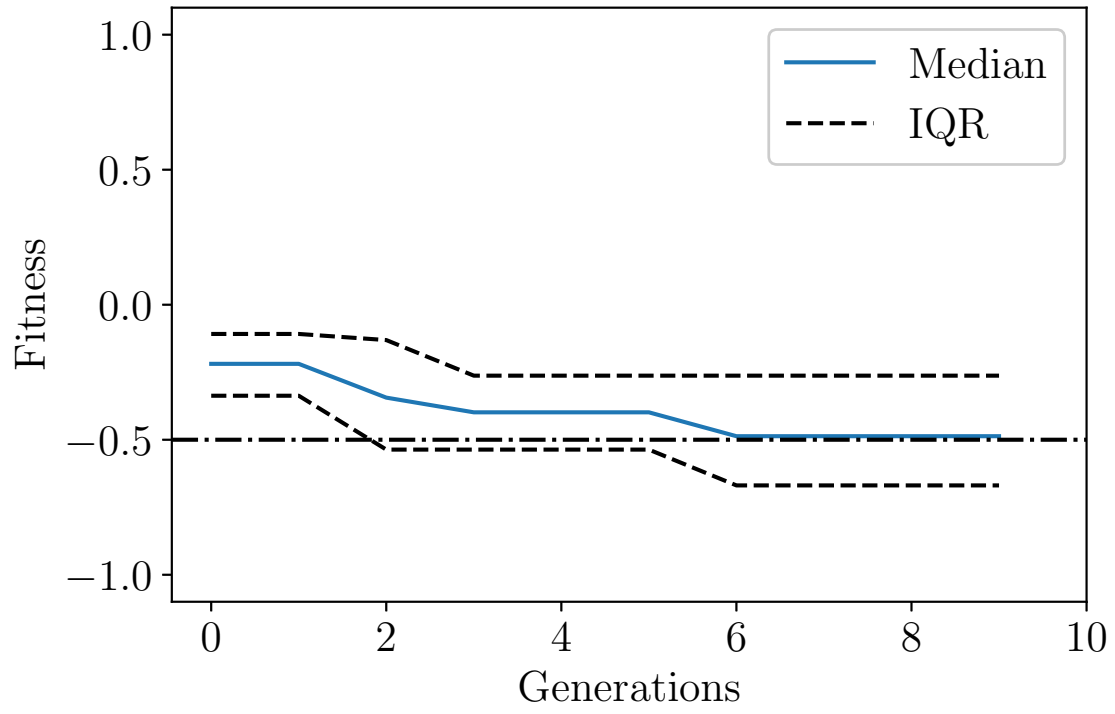


FIGURE 5.47. Performance of controllers generated using 4 executions of HAEA and a simulated seed for real EMERGE modules. Bars show the inter-quartile range.

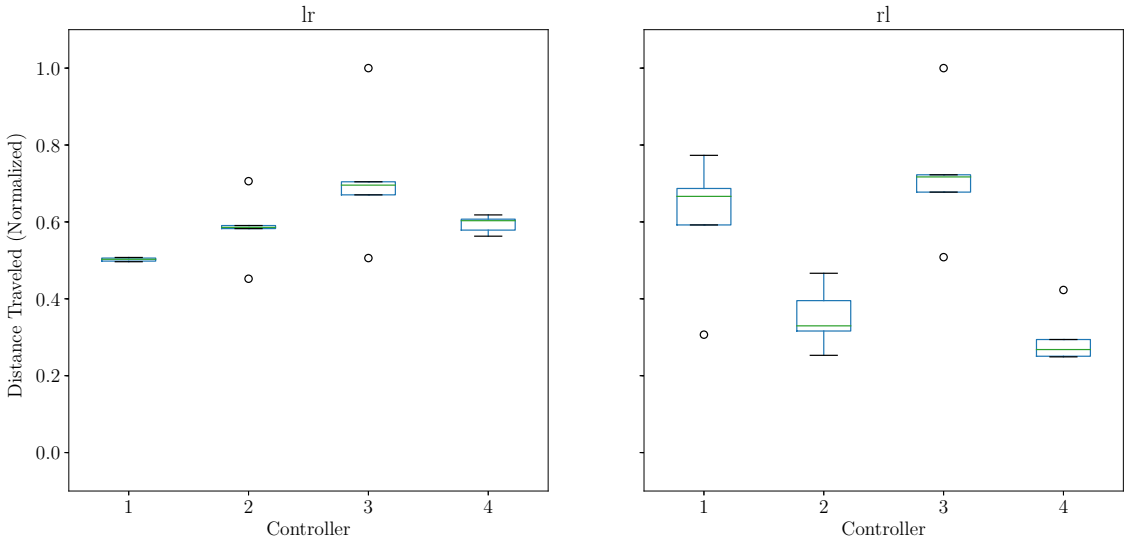


FIGURE 5.48. Overall distance traveled by the best controllers generated using HAEA and a simulated seed for real EMERGE modules, tested in environments $l \oplus r$ and $r \oplus l$.

Results show that some simulated individuals transfer better to reality after the reduction in the number of modules and the change in the module movement range, as can be seen from the performance of the initial population in figure 5.47. In spite of the reduced number of generations and individuals, in some cases, HAEA is able to make some individuals reach the exit of the environment. But, in general, it is not able to improve the initial population fitness by much.

The best controllers are not always repeatable, that is, they travel similar distances in around the same time each time they are tested under similar conditions. However they not always reach the exit of the environment, even if they did in the evolution run. When the environment is changed, individuals that traveled far in the training environment ($l \oplus r$) are able to also travel consistently in the other combination ($r \oplus l$). Some controllers that performed not that well in the training environment are able to improve in the other combination environment as can be seen on figure 5.48, mainly due to them producing movements more suitable for this other combination from the start. Again, problems with irregular friction forces, magnetic forces, and small differences among modules affect the capability of the adaptation mechanism to improve controllers. In addition, the reduced number of individuals and generations used also hinder the capacity of the evolutionary algorithm to find better controllers.

5.7 Conclusion

In this chapter various experiments testing the locomotion training framework of chapter 3 are described. Results show that the framework is able to make a modular robot with and without sensors move in different environments.

The specific way in which *primenv*s in the configurable environment model are used in conjunction with evolutionary algorithms affects the resulting controllers, even when only part of the controller defined is being used. This is the case for the tests of section 5.2, in which controllers using only the CPG coordination mechanism are evolved in sequences of *primenv*s or in separated *primenv*s related only by an aggregative measure of fitness. The *primenv*s presented in sequence showed more consistency when generating controllers that exited all *primenv*s used. The specific sequence of *primenv*s also affected resulting controllers performance in different environments made from joining *primenv*s, hinting that some environments are more difficult to travel in than others, an idea that is worth exploring and for which tests can be easily defined using the configurable environment approach. It can be seen that when using an aggregating measure of fitness, the average *primenv* fitness can be a deceiving measure, but in this case it performs better than the worst *primenv* fitness measure in evolving controllers capable of reaching the exit of all *primenv*s in all executions. This may be due to the worst *primenv* measure giving bad fitness to controllers that perform well in almost all *primenv*s but perform poorly in one. This condition is relaxed with the average *primenv* fitness measure by which this kind of controllers get a better fitness.

Measuring the generalization ability of controllers generated using the locomotion training framework allows to study the conditions (environment used for training, adaptation mechanism parameters, etc..) that produce more capable controllers. Generalization ability is measured in a set of special environments built using all possible combinations of *primenv*s.

As stated before, the more features of the environment shown to a training controller the better its performance in variations the environment built using the same features. This is confirmed also in section 5.3. Controllers trained using only one combination of *primenv*s are not able to generalize as well as controllers trained using more combinations.

Well coordinated movements turn out to be very important for controllers using sensors to improve their generalization ability when being adapted. Controllers with sensors evolved to move from scratch perform worse (do not move as far), even on the environment used for training, than evolved controllers without sensors. In contrast, controllers with sensors evolved with and initial population built using the movements obtained in controllers without sensors are able to increase their generalization ability slightly. This assumption is confirmed when using the short challenge incremental approach (Section 5.5), in which even if controllers are trained in smaller portions of the environment they fail to obtain well coordinated movements. As a consequence controllers evolved using short challenges over-specialize to the specific environment conditions that they saw in the last challenge.

Differences between simulation and reality make controllers that performed well in simulation behave completely different in reality. This can be seen when transferring some of the best controllers obtained in simulation to the real EMERGE modules. This holds true even when controllers are evolved directly in the real modules. The effect of the gap can be reduced by adjusting the simulated module to take into account physical limitations of the real modules more closely, but no matter how closely adjusted, the gap will ever be present.

CHAPTER

Conclusions and Future Work

Making a modular robot automatically learn the movements necessary to locomote in different environments involves coping with issues inherent to this kind of robots. For example, since modular robots can be reconfigured, the morphology of the robot can be arbitrary and not known beforehand. Locomotion includes coordinating different parts of the morphology (the modules), to move in a certain direction. It also involves adjusting the movements of the robot based on sensor information coming also from the modules, which can have an arbitrary set of sensors. In this context, the control and learning strategies used for tackling this problem in this work allow a modular robot to move in different environments regardless of its shape or sensing capabilities.

To achieve this, previous control and learning strategies were reviewed. Previously existing controller designs and automatic controller generation strategies have, in most cases, only been tested with a small number of sensors in pre-specified positions. This is mainly a consequence of the explosion in the number of controller parameters that appears when working with a high number of sensor sources and different morphologies (Chapter 3). This not only affects controller designs but also the different learning strategies used. For instance, the arbitrary morphology of modular robots makes providing examples to supervised learning techniques a very difficult task, since "good" movements for some morphologies are counter intuitive. Classical unsupervised learning also presents a problem: assigning reward and defining state-action pairs is not straightforward (which sets of sensor inputs, different from others, can be considered a state?). Thus, a simpler way of telling that it is doing well and also allowing the controller some freedom of action to get the reward, are desirable features in an adaptation/learning technique.

Locomotion Training Framework

This work proposes a locomotion training framework for modular robots, composed of four main parts: (1) A control strategy, capable of moving a robot in different shapes with an

arbitrary set of sensors, (2) an adaptation mechanism , which uses evolutionary algorithms to adapt controllers to different environments, (3) a configurable model of the environment which allows the framework to train robots in different features of the environment, and (4) a modular robot platform, which is easy to assemble and open for anyone to use and modify.

- **Control Strategy:** Controllers are composed of three main parts: the first part is a coordination mechanism, which uses a distributed technique (CPG) to asynchronously coordinate neighboring modules, determining their behaviour. This mechanism helps getting coordinated movements for different morphologies, however, some coordinated movements are more suitable to move in different environments than others. This is evidenced in the difference of performance of controllers with sensors generated using a well coordinated seed and controllers with sensors generated with no seed, being the first ones better than the latter ones (Section 5.4).

  The second part is a sensor information handling mechanism, which aggregates and filters sensor signals coming from all modules, and the third part is a decision mechanism which receives its inputs from the sensor information handling mechanism and uses them to determine the behavior of the coordination mechanism. Different kinds of decision mechanisms are possible (Section 3.1.3). Yet, a decision mechanism with very complex structures, like if statements, causes problems when paired with automatic adaptation techniques, like evolutionary algorithms. This is because it is very difficult for these techniques to make meaningful changes that would eventually improve the controller. For this reason, the decision mechanism complexity must match the capacity of the adaptation technique to make meaningful changes that improve the controller.

- **Adaptation Mechanism:** Evolutionary algorithms are chosen as adaptation mechanism due to the simple way in which to tell whether a controller is doing well, which is a desirable feature in a learning technique for this problem, as mentioned earlier. It also has the advantage of allowing several different controllers to be tested, as these algorithms are population based.

- **Configurable environments:** Defining the control strategy is only part of the problem. Modeling the environment and picking which features to show a robot in training is a matter of careful consideration. As more environment features and variations of these features are used to train a robot, the better the robot will be able to move in similar environments. Previous works often test controllers in simple environments like flat surfaces. In this work a configurable environment model is proposed in order to present more complex and varied environments to a learning robot. Although limited only to structured environments, for the sake of simplicity, it allows an environment to be separated in its main features, which are organized in units containing only one feature and clear boundaries. These units, called primitive environments or *primenv* are then combined in different ways in a training process.

As it has been shown, the specific way in which *primenv*s in the configurable environment model are used in conjunction with evolutionary algorithms affects the resulting controllers, even when only part of the controller strategy defined is used. This is the case for the tests of section 5.2, in which controllers using only the CPG coordination mechanism are evolved in sequences of *primenv*s or in separated *primenv*s related only by an aggregative measure of fitness. The *primenv*s presented in sequence showed more consistency when generating controllers that exited all *primenv*s used. The specific sequence of *primenv*s also affected resulting controllers performance in different environments made from joining *primenv*s, hinting that some environments are more difficult to travel in than others, an idea that is worth exploring and for which tests can be easily defined using the configurable environment approach.

Another advantage of using sequences of *primenv*s is a reduction in the simulation time required to get controllers that exit all environments. This can be seen in controllers that are evaluated in *primenv* sequences (Section 5.2): controllers are only evaluated in the next *primenv* if they exit the last one, this means that controllers that do not move are never evaluated in all *primenv*s. Although the total simulation time saved has not been tallied, in this case individual evaluations simulation time can be reduced up to a third for individuals that do not move past the first *primenv* compared to evaluations of individuals that move in all three *primenv*s. However, the fitness measure used is very complicated and difficult to compare to other results. For this reason sequences of *primenv*s are replaced by environments built connecting *primenv*s in the same order of the sequence in subsequent tests. Future work could concentrate in simplifying the measure of fitness for sequences of *primenv*s so that it can be used to speed up results.

- **Modular robot platform:** Most existing modular robots to date are still in the prototype stage. Additionally, module prototypes often use automatic connectors and other specialized parts that are difficult to assemble. As a result procuring a fully functioning modular robot to test in is still very difficult and expensive, and the few cheap, open and commercial models available do not posses the sensing or communication/control capabilities necessary to implement the control strategy proposed. For this reason, this work also introduces a new modular robot prototype, the EMERGE module (Chapter 4). This module prototype is designed to be easy to build using off-the-shelf parts and its design is open for everyone to use or modify. Thanks to the use of magnetic connectors these modules allow the quick assembly of module morphologies that are used in this work to test the sensor capable controllers.

Reconfiguration

The use of magnetic connectors makes the assembly and disassembly of EMERGE modules easier, which also makes evaluating new morphologies in reality easier. Taking adavantage

of this feature, a reconfiguration method using an external robotic manipulator is proposed as a practical alternative to self-reconfigurable robots and manual reconfiguration systems. Results show that this method, although having some limitations, is feasible and that the assembly process is enhanced by the magnetic forces, but the disassembly is hindered by these same forces. A magnetic connector force analysis confirms that EMERGE magnetic connectors produce a self centering force that helps reduce the error when aligning and connecting modules, but that also moves a morphology being disassembled from its reference position (Section 4.4). This indicates that a module localization system, like machine vision, is a must for reconfiguration using an external agent to work properly. Future work will concentrate in addressing the system limitations and improving it to the point of doing morphology evolution experiments fully automatic.

Generalization

Measuring the generalization ability of controllers generated using the locomotion training framework allows to study the conditions (environment used for training, adaptation mechanism parameters, etc..) that produce more capable controllers. Generalization ability is measured in a set of special environments built using all possible combinations of *primenv*s. In this way, a controller is faced with all the possible conditions the environment can offer. Using this measure of generalization and controllers using only the CPG coordination mechanism, in section 5.3 evolutionary algorithms are found to generate controllers that travel more distance than those generated by a gradient based optimization technique (Hill Climbing) (Figures 5.14 and 5.20).

As stated before, the more features of the environment shown to a training controller the better its performance in variations of the environment built using the same features. This is confirmed also in section 5.3. Controllers trained using only one combination of *primenv*s are not able to generalize as well as controllers trained using more combinations. Future work may be able to determine the right amount of variation in the environment to get controllers with better generalization ability.

Sensors

When going from controllers without sensors, used when testing the generalization measure (Section 5.3), to controllers with sensors (Section 5.4), the sudden increase in the number of controller parameters (from 3 to 234) shows how the curse of dimensionality creeps in modular robot systems. Using the same decision mechanism (ANN) in all modules helps reduce the impact of this change, which could render controllers impossible to adapt in the long run. Better techniques that can cope with even more parameters could be used to study the effect of using individual decision mechanisms in each module.

In this context, well coordinated movements turn out to be very important for controllers using sensors to improve their generalization ability when being adapted. This

can be seen in section 5.4 tests. Controllers with sensors evolved to move from scratch perform worse (do not move as far), even on the same environment used for training, than evolved controllers without sensors. In contrast, controllers with sensors evolved with and initial population built using the movements obtained in controllers without sensors are able to increase their generalization ability slightly. A subsequent test demonstrates that if a manual seed, with well coordinated and ample movements, is used to evolve controllers with sensors, the generalization ability of the resulting controllers improves more.

Even with the CPG coordination mechanism in place, if well coordinated movements are not found before implementing sensor mechanisms in the controller, these mechanisms may not improve the generalization ability of the robot. Future work could implement techniques to ensure that a robot is coordinating its modules well, before using sensor systems. This is confirmed when using the short challenge incremental approach (Section 5.5), in which even if controllers with sensors are trained in smaller portions of the environment they fail to obtain well coordinated movements. As a consequence controllers evolved using short challenges over-specialize to the specific environment conditions that they saw last.

This over-specialization of controllers in the short challenge approach is also consequence of the simple architecture of the ANN used. As a first approximation the simple reactive, hidden layer topology of the network allows it to be sufficient for the task of moving a robot with incoming sensor information. However, different neural structures (cyclic topologies, different types of activation functions, etc..), can be used to improve the ability of the network to, for example, memorize and re-utilize movements that were good in the past.

Reality

Differences between simulation and reality make controllers that performed well in simulation behave completely different in reality. This can be seen when transferring some of the best controllers obtained in simulation to the real EMERGE modules (Section 5.6). The effect of the gap can be reduced a little by adjusting the simulated module to take into account physical limitations of the real modules more closely.

Evaluating controllers in real world tests is very expensive compared to doing the same in simulation, this is evidenced by the reduction in the number of generations and individuals used in section 5.6 tests. In overall, the full set of tests took about 3 weeks to run, with each individual evaluation of fitness taking up to 15 minutes, including the time spent preparing the robot, running the evaluation itself and taking measurements.

The locomotion training framework could be fully implemented in a real setting, and all its parts worked as expected. Robots reacted to the environment through their, more limited, sensors and even the configurable environment model was useful to measure generalization of the evolved controllers. The only part that was not very effective was the

adaptation mechanism, which, by the very nature of the evolutionary algorithms used, needed a higher number of generations and individuals to work properly. This indicates that an adaptation mechanism which uses less costly evaluations in reality would help improve the limitations of this system. An adaptation mechanism with this characteristics is explored in [41]. Future work includes using the automatic reconfiguration approach, using and external robotic manipulator, also proposed in this work, to speed up experiments done in reality.

Wrapping up

In summary, a modular robot platform, the EMERGE modular robot, is created. EMERGE modules are easy to build, reconfigure and repair, which allowed morphologies to be built for all the tests performed in this work, be them in simulation or reality. This ease of use has also allowed EMERGE modules to be used in other Master and Ph.D projects in different parts of the world [122, 41, 65].

The configurable environment approach constitutes the first time, that we know of, that the problem of locomotion is tackled from the point of view of the environment. This approach not only allowed for a measure of generalization to be defined, but also provided a practical way of building primitive environments for real tests, which can be used also for training other kinds of robots with different low level definitions. Finding out how many combinations of primitive environments should be presented to a robot in training for it to generalize well in other combinations could help expand the types of environment the robot could move into and will be explored in future works. Sequence evaluation will also be explored as it may improve evaluation time when testing robots in simulation and reality.

Using bio-inspired techniques: sensor messages, which are a modification of hormone inspired messages, artificial neural networks, central pattern generators, and evolution, allowed controllers for modular robots to move in different environments while, also for the first time, integrating, processing and in general, managing information coming from distributed sensors in all modules of a morphology. Strategies using similar parts can be used not only for modular robots but also for different robots that also posses several sources of sensor information, like soft-robots, in which sensors can be defined even using the mechanical structure itself. Of course, finding controllers that learn to manage all this sensor information was shown to be not that easy, well coordinated movements should be available for a sensor system of this type to improve the performance of the robots, thus future work must look more into mechanisms to check for this type of movements.

Several types of ANNs can be used as decision mechanism, as they can make the architecture defined behave in different ways, which is also worth looking into in future studies. Transferring not only the controller but also the adaptation mechanism to reality has been the subject of a master work derived from this work [40] and will also be studied in future works. NEAT [107] and other incremental approaches will also be implemented

in order to better manage the increase in complexity when going from controllers without sensors to controllers with sensors.

Automatic reconfiguration was shown to be feasible using an external robot manipulator and a localization system. This provides a very practical way of reconfiguring EMERGE modules and eventually other kinds of robots made from detachable parts. The reconfiguration strategy proposed is also a first step towards fully automating experiments, which will help reduce evaluation time in real tests (i.e. the hardware tests in this work) from months to days. Reality gap reduction methods can also be implemented along the reconfiguration strategy to improve the performance of simulated controllers in reality which will be studied in future experiments.

Finally, as a general conclusion, the architecture defined allows a modular robot, built using EMERGE modules, to learn controllers that let it move in different environments. These environments represent different features of a more general structured environment in which the robot is expected to move. The controllers generated using the framework take into account sensor information, integrating and processing it and deciding their individual actions for all modules to locomote as a whole, regardless of morphology. In conjunction with the modular robot reconfiguration strategy, that uses an external manipulator, robots with different morphologies can be tested automatically in future experiments. This opens the way to the possibility of evolving robots with different morphologies, and thus different sensing capabilities, in completely automatic experiments performed directly in reality and completely autonomous learning architectures for real robots.

Bibliography

[1] AIST, *M-tran(modular transformer)*, `http://unit.aist.go.jp/is/frrg/dsysd/mtran3/index.htm`, Accessed: 2007.

[2] E Alpaydin, *Introduction to machine learning*, 2 ed., MIT Press, 2010.

[3] J. Auerbach, D. Aydin, A. Maesani, P. Kornatowski, T. Cieslewski, G. Heitz, P. Fernando, I. Loshchilov, L. Daler, and D. Floreano, *RoboGen: Robot Generation through Artificial Evolution*, Proceedings of the 14 Alife Conference (New York, NY, USA), The MIT Press, 2014, pp. 136–137.

[4] Joshua Auerbach and Josh C. Bongard, *How robot morphology and training order affect the learning of multiple behaviors*, CEC '09 (Trondheim), IEEE, may 2009, pp. 39–46.

[5] F.H. Bennett and EG Rieffel, *Design of decentralized controllers for self-reconfigurable modular robots using genetic programming*, Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware, IEEE Comput. Soc, 2000, pp. 43–52.

[6] Josh Bongard, *Behavior chaining: incremental behavioral integration for evolutionary robotics*, Artificial Life XI, no. 1976, 2008, pp. 64–71.

[7] Josh C. Bongard, *Incremental Approaches to the Combined Evolution of a Robot's Body and Brain*, Ph.d, University of Zurich, 2003, p. 193.

[8] ———, *Morphological and environmental scaffolding synergize when evolving robot controllers*, GECCO '11 (Dublin, Ireland), ACM Press, 2011, p. 179.

[9] Geoffrey Boothroyd, *Assembly automation and product design, second edition (manufacturing engineering and materials processing)*, CRC Press, Inc., Boca Raton, FL, USA, 2005.

[10] David Brandt, David Johan Christensen, and Henrik Hautop Lund, *ATRON robots: Versatility from self-reconfigurable modules*, 2007 International Conference on Mechatronics and Automation, 2007, pp. 26–32.

[11] L. Brodbeck, S. Hauser, and F. Iida, *Morphological evolution of physical robots through model-free phenotype development*, PLoS ONE **10** (2015), no. 6, 1–17.

[12] L. Bruzzone and G. Quaglia, *Review article: locomotion systems for ground mobile robots in unstructured environments*, Mechanical Sciences **3** (2012), no. 2, 49–62.

[13] Pilar Caamano, Rafael Tedin, Alejandro Paz-Lopez, and Jose Antonio Becerra, *JEAF: A Java Evolutionary Algorithm Framework*, CEC '10 (Barcelona), no. December 2007, IEEE, jul 2010, pp. 1–8.

[14] a. Castano, a. Behar, and P.M. Will, *The Conro modules for reconfigurable robots*, IEEE/ASME Transactions on Mechatronics **7** (2002), no. 4, 403–409.

[15] G S Chirikjian, *Kinematics of a metamorphic robotic system*, 1994 IEEE International Conference on Robotics and Automation, 1994. Proceedings., 1994, pp. 449–455.

[16] H C H Chiu, M Rubenstein, and W M Shen, *Multifunctional superbot with rolling track configuration*, IROS 2007 Workshop on Self-Reconfigurable Robots & Systems and Applications, 2007, pp. 50–53.

[17] H. S. Cho, H. J. Warnecke, and D. G. Gweon, *Robotic assembly: a synthesizing overview*, Robotica **5** (1987), no. 2, 153165.

[18] David Johan Christensen, Ulrik Pagh Schultz, and Kasper Stoy, *A distributed strategy for gait adaptation in modular robots*, 2010 IEEE International Conference on Robotics and Automation, IEEE, may 2010, pp. 2765–2770.

[19] ———, *A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots*, Robotics and Autonomous Systems **61** (2013), no. 9, 1021–1035.

[20] D.J. Christensen, *Evolution of shape-changing and self-repairing control for the ATRON self-reconfigurable robot*, Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., no. May, IEEE, 2006, pp. 2539–2545.

[21] Susan Craw, *Manhattan distance*, pp. 639–639, Springer US, Boston, MA, 2010.

[22] A Crespi and A J Ijspeert, *Online optimization of swimming and crawling in an amphibious snake robot.*, IEEE Transactions on Robotics **24** (2008), no. 1, 75–87.

[23] Alessandro Crespi, Daisy Lachat, Ariane Pasquier, and Auke Jan Ijspeert, *Controlling swimming and crawling in a fish robot using a central pattern generator*, Autonomous Robots **25** (2008), no. 1-2, 3–13.

[24] Massimiliano D'Angelo, Berend Weel, and A. E. Eiben, *Online gait learning for modular robots with arbitrary shapes and sizes*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **8273 LNCS** (2013), 45–56.

[25] Jay Davey, Ngai Kwok, and Mark Yim, *Emulating self-reconfigurable robots-design of the SMORES system*, Intelligent Robots and Systems (2012), 4464–4469.

[26] Stéphane Doncieux, Jean-Baptiste Mouret, Nicolas Bredeche, and Vincent Padois, *Evolutionary Robotics: Exploring New Horizons*, New Horizons in Evolutionary Robotics: Extended Contributions from the 2009 EvoDeRob Workshop (Stéphane Doncieux, Nicolas Bredèche, and Jean-Baptiste Mouret, eds.), Studies in Computational Intelligence, vol. 341, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 3–25.

[27] D Duff, M Yim, and K Roufas, *Evolution of polybot: A modular reconfigurable robot*, Proc. of the Harmonic Drive Intl. Symposium, Nagano, Japan, Nov, Citeseer, 2001.

[28] A. Faia, F. Bellas, F. Orjales, D. Souto, and R.J. Duro, *An evolution friendly modular architecture to produce feasible robots*, Robotics and Autonomous Systems **63** (2015), 195 – 205, Cognition-oriented Advanced Robotic Systems.

[29] T Fukuda and Y Kawauchi, *Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator*, Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on, IEEE, 1990, pp. 662–667.

[30] Faustino Gomez and Risto Miikkulainen, *Incremental Evolution of Complex General Behavior*, Adaptive Behavior **5** (1997), no. 3-4, 317–342.

[31] Jonatan Gomez, *Self adaptation of operator rates in evolutionary algorithms*, Genetic and Evolutionary Computation – GECCO 2004 (Berlin, Heidelberg) (Kalyanmoy Deb, ed.), Springer Berlin Heidelberg, 2004, pp. 1162–1173.

[32] ———, *unalcol java library*, https://github.com/jgomezpe/unalcol, 2015, Last accessed on Feb 26, 2016.

[33] G Granosik, M G Hansen, and J Borenstein, *The OmniTread serpentine robot for industrial inspection and surveillance*, Industrial Robot: An International Journal **32** (2005), no. 2, 139–148.

[34] D.J. Griffiths, *Introduction to electrodynamics*, Pearson Education, 2014.

[35] S Grillner, *Biological pattern generation: the cellular and computational logic of networks in motion*, Neuron **52** (2006), no. 5, 751–766.

[36] Evert Haasdijk, Andrei A Rusu, and A E Eiben, *HyperNEAT for Locomotion Control in Modular Robots*, Evolvable Systems: From Biology to Hardware, proceedings of the 9th International Conference on Evolvable Systems (ICES 2010) (Gianluca Tempesti, Andy M. Tyrrell, and Julian F. Miller, eds.), vol. 6274, Springer Berlin Heidelberg, 2010, pp. 169–180.

[37] Heiko Hamann, Jürgen Stradner, Thomas Schmickl, and Karl Crailsheim, *Artificial Hormone Reaction Networks: Towards Higher Evolvability in Evolutionary Multi-Modular Robotics*, (2010), 1–18.

[38] M. D. Hancher and G. S. Hornby, *A modular robotic system with applications to space exploration*, 2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06), July 2006, pp. 8 pp.–132.

[39] Jeff Heaton, *Introduction to neural networks for java, 2nd edition*, 2nd ed., Heaton Research, Inc., 2008.

[40] Henry Hernandez, *Estimacion de una serie de movimientos utilizando un algoritmo de optimizacion bio-inspirado para la operacion autonoma y on-line de una plataforma multi-robot (caso robot modular)*, Universidad Nacional de Colombia, 2018.

[41] Henry Hernandez, Rodrigo Moreno, Andres Faina, and Jonatan Gomez, *Design of a bio-inspired controller to operate a modular robot autonomously*, Advances in Artificial Intelligence - IBERAMIA 2018 (Cham) (Guillermo R. Simari, Eduardo Fermé, Flabio Gutiérrez Segura, and José Antonio Rodríguez Melquiades, eds.), Springer International Publishing, 2018, pp. 314–325.

[42] F Hou and W M Shen, *Mathematical foundation for hormone-inspired control for self-reconfigurable robotic systems*, Proc. 2006 IEEE Intl. Conf. on Robotics and Automation, 2006.

[43] Feili Hou, Nadeesha Ranasinghe, Behnam Salemi, and W M Shen, *Wheeled locomotion for payload carrying with modular robot*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008, Ieee, sep 2008, pp. 1331–1337.

[44] Feili Hou and Wei-Min Shen, *Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots*, Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, Ieee, may 2008, pp. 3135–3140.

[45] Rob J. Hyndman, *Moving averages*, pp. 866–869, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[46] Auke Jan Ijspeert, *Central pattern generators for locomotion control in animals and robots: a review*, Neural Networks **21** (2008), no. 4, 642–653.

[47] A. Jacoff, E. Messina, B.a. Weiss, S. Tadokoro, and Y. Nakagawa, *Test Arenas and Performance Metrics for Urban Search and Rescue Robots*, Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 4, IEEE, 2003, pp. 3396–3403.

[48] Nick Jakobi, *Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis*, Adaptive Behavior **6** (1997), no. 2, 325–368.

[49] P Jantapremjit and D Austin, *Design of a modular self-reconfigurable robot*, Australian Conf. on Robotics and Automation, Sydney, Australia, Citeseer, 2001.

[50] Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit, *Accomplishing High-Level Tasks with Modular Robots*, (2017).

[51] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji, *Automatic Locomotion Design and Experiments for a Modular Robotic System*, IEEE/ASME Transactions on Mechatronics **10** (2005), no. 3, 314–325.

[52] Akiya Kamimura, Haruhisa Kurokawa, Eiichi Yoshida, Kohji Tomita, Satoshi Murata, and Shigeru Kokaji, *Automatic locomotion pattern generation for modular robots*, IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA'03, vol. 1, 2003.

[53] John N. Karigiannis and Costas S. Tzafestas, *Robustness and generalization of model-free learning for robot kinematic control using a nested-hierarchical multi-agent topology*, 2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob), IEEE, jun 2012, pp. 1140–1147.

[54] H Kimura, Y Fukuoka, Y Hada, and K Takase, *Adaptive dynamic walking of a quadruped robot on irregular terrain using a neural system model*, Robotics Research (2003), 147–160.

[55] Dennis Krupke, Florens Wasserfall, Norman Hendrich, and Jianwei Zhang, *Printable modular robot: an application of rapid prototyping for flexible robot design*, Industrial Robot: An International Journal **42** (2015), no. 2, 149–155.

[56] Rahul Kumar, Sunil Lai, Sanjesh Kumar, and Praneel Chand, *Object detection and recognition for a pick and place Robot*, Asia-Pacific World Congress on Computer Science and Engineering, APWC on CSE 2014, 2014, pp. 1–7.

[57] Tüze Kuyucu, Ivan Tanev, and Katsunori Shimohara, *Genetic transposition inspired incremental genetic programming for efficient coevolution of locomotion and sensing of simulated snake-like robot*, ECAL-2011 (Paris), MIT Press, 2011, pp. 439–446.

[58] Polymorphic Robotics Laboratory, *Superbot*, `http://www.isi.edu/robots/superbot/`, Accessed: 2008.

[59] Steven M LaValle, *Planning algorithms*, Cambridge university press, 2006.

[60] Wei-Po Lee, John Hallam, and HenrikHautop Lund, *Learning Complex Robot Behaviours by Evolutionary Computing with Task Decomposition*, Learning Robots (Andreas Birk and John Demiris, eds.), LNCS, vol. 1545, Springer Berlin Heidelberg, Berlin, Heidelberg, jun 1998, pp. 155–172.

[61] Dan Lessin, Don Fussell, and Risto Miikkulainen, *Open-ended behavioral complexity for evolved virtual creatures*, GECCO '13 (New York, New York, USA), ACM Press, 2013, p. 335.

[62] Jens Liedke, Rene Matthias, Lutz Winkler, and Heinz Worn, *The Collective Self-reconfigurable Modular Organism (CoSMO)*, 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics: Mechatronics for Human Wellbeing, AIM 2013 (2013), 1–6.

[63] Quentin Lindsey, Daniel Mellinger, and Vijay Kumar, *Construction with quadrotor teams*, Autonomous Robots **33** (2012), no. 3, 323–336.

[64] H Lipson and J.B. Pollack, *Automatic design and manufacture of robotic lifeforms*, Nature **406** (2000), no. 6799, 974–978.

[65] C. Liu, J. Liu, R. Moreno, F. Veenstra, and A. Faina, *The impact of module morphologies on modular robots*, 2017 18th International Conference on Advanced Robotics (ICAR), July 2017, pp. 237–243.

[66] Daniel Marbach and Auke Jan Ijspeert, *Co-evolution of Configuration and Control for Homogenous Modular Robots*, Proceedings of the eighth conference on intelligent autonomous systems (IAS8), 2004, pp. 712—-719.

[67] ———, *Online Optimization of Modular Robot Locomotion*, Proceedings of the IEEE International Conference on Mechatronics & Automation, no. July, 2005, pp. 248–253.

[68] T. Matsumaru, *Design and control of the modular robot system: Tomms*, Proceedings of 1995 IEEE International Conference on Robotics and Automation, vol. 2, May 1995, pp. 2125–2131 vol.2.

[69] C McGray and D Rus, *Self-reconfigurable molecule robots as 3d metamorphic robots*, 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1998. Proceedings., vol. 2, 1998.

[70] G.T. T McKee, J.a. A Fryer, and P.S. S Schenker, *Object-oriented concepts for modular robotics systems*, Proceedings 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems. TOOLS 39, Published by the IEEE Computer Society, IEEE Comput. Soc, 2001, p. 229.

[71] Yan Meng, Yuyang Zhang, and Yaochu Jin, *Autonomous self-reconfiguration of modular robots by evolving a hierarchical mechanochemical model*, IEEE Computational Intelligence Magazine, vol. 6, 2011, pp. 43–54.

[72] ModLab, *Self-assembling modular robot for extreme shapeshifting (smores)*, `http://www.modlabupenn.org/2016/06/18/smores-ep/`, Accessed: 2018.

[73] R. Moreno, F. Veenstra, D. Silvera, J. Franco, O. Gracia, E. Cordoba, J. Gomez, and A. Faina, *Automated reconfiguration of modular robots using robot manipulators*, 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Nov 2018, pp. 884–891.

[74] Rodrigo Moreno, *Thesis video repository*, `https://sites.google.com/view/emergemodular/videos/morenothesis`.

[75] Rodrigo Moreno, Andres Faiña, and Kasper Støy, *Evolving Robot Controllers for Structured Environments Through Environment Decomposition*, Applications of Evolutionary Computation: 18th European Conference, EvoApplications 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings (Copenhagen) (M. Antonio Mora and Giovanni Squillero, eds.), vol. 1, Springer International Publishing, 2015, pp. 795–806.

[76] Rodrigo Moreno and Jonatan Gomez, *Central pattern generators and hormone inspired messages: A hybrid control strategy to implement motor primitives on chain type modular reconfigurable robots*, ICRA '11 (Shanghai), IEEE, may 2011, pp. 1014–1019.

[77] Rodrigo Moreno, Ceyue Liu, Andres Faina, Henry Hernandez, and Jonatan Gomez, *The emerge modular robot, an open platform for quick testing of evolved robot morphologies*, Proceedings of the Genetic and Evolutionary Computation Conference Companion (New York, NY, USA), GECCO '17, ACM, 2017, pp. 71–72.

[78] Rodrigo Moreno, Frank Veenstra, David Silvera, Julian Franco, Oscar Gracia, Ernesto Cordoba, Jonatan Gomez, and Andres Faina, *Automated reconfiguration of modular robots using robot manipulators*, `https://vimeo.com/292404982`.

[79] Rodrigo Moreno Garcia, *Diseño y Simulación de un Algoritmo para el Control de un Robot Modular tipo Cadena*, Master thesis, Universidad Nacional de Colombia, 2010.

[80] N. Mukosaka, I. Tanev, and K. Shimohara, *Performance of Incremental Genetic Programming on Adaptability of Snake-like Robot*, IES2013 **24** (2013), no. 0, 152–157.

[81] S Murata and H Kurokawa, *Self-reconfigurable robots*, IEEE Robotics & Automation Magazine **14** (2007), no. 1, 71–78.

[82] ———, *Self-Organization of Motion*, Self-Organizing Robots, Springer, 2012, pp. 173–209.

[83] S Murata, H Kurokawa, and S Kokaji, *Self-assembling machine*, 1994 IEEE International Conference on Robotics and Automation, 1994. Proceedings., 1994, pp. 441–448.

[84] S Murata, E Yoshida, K Tomita, H Kurokawa, A Kamimura, and S Kokaji, *Hardware design of modular robotic system*, Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000), vol. 3, Ieee, 2000, pp. 2210–2217.

[85] EH Ostergaard and HH Lund, *Evolving control for modular robotic units*, Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium (Cat. No.03EX694), vol. 2, IEEE, 2003, pp. 886–892.

[86] Esben Hallundbæk Østergaard, Kristian Kassow, Richard Beck, and Henrik Hautop Lund, *Design of the ATRON lattice-based self-reconfigurable robot*, Autonomous Robots **21** (2006), no. 2, 165–183.

[87] M. Pacheco, M. Moghadam, A. Magnusson, B. Silverman, H. H. Lund, and D. J. Christensen, *Fable: Design of a modular robotic playware platform*, Proceedings - IEEE International Conference on Robotics and Automation, IEEE, 2013, pp. 544–550.

[88] PARC, *Parcs modular robotics chain systems-polybot*, `http://www2.parc.com/spl/projects/modrobots/chain/polybot/index.html`, Accessed: 2004.

[89] Rolf Pfeifer and Josh Bongard, *How the Body Shapes the Way We Think: A New View of Intelligence*, MIT Press, 2006.

[90] Raja Humza Qadir, *Self-Sufficiency of an Autonomous Reconfigurable Modular Robotic Organism*, Springer, 2015.

[91] Nadeesha Ranasinghe and WM Shen, *The Surprise-Based Learning Algorithm*, USC ISI internal publication (2008), no. April.

[92] L Righetti and A J Ijspeert, *Design methodologies for central pattern generators: an application to crawling humanoids*, Proceedings of Robotics: Science and Systems (Philadelphia, USA), 2006, pp. 191–198.

[93] Eric Rohmer, Surya P. N. Singh, and Marc Freese, *V-REP: A versatile and scalable robot simulation framework*, IROS '13 (Tokyo), IEEE, nov 2013, pp. 1321–1326.

[94] C. Rossi and a. E. Eiben, *Simultaneous versus incremental learning of multiple skills by modular robots*, Evolutionary Intelligence **7** (2014), no. 2, 119–131.

[95] D. Saldana, B. Gabrich, M. Whitzer, A. Prorok, M.F.M. Campos, M. Yim, and V. Kumar, *A decentralized algorithm for assembling structures with modular robots*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 2736–2743.

[96] B Salemi, M Moll, and W M Shen, *SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system*, Proc. 2006 IEEE/RSJ Intl. Conf. Intelligent Robots Systems, Citeseer, 2006, pp. 3636–3641.

[97] B Salemi, P Will, and W M Shen, *Autonomous discovery and functional response to topology change in self-reconfigurable robots*, Complex Engineered Systems (2004), 364–384.

[98] A C Sanderson, *Modular robotics: design and examples*, 1996 IEEE Conference on Emerging Technologies and Factory Automation, 1996. EFTA'96. Proceedings., vol. 2, 1996.

[99] Thomas Schmickl, Heiko Hamann, and Karl Crailsheim, *Modelling a hormone-inspired controller for individual- and multi-modular robotic systems*, Mathematical and Computer Modelling of Dynamical Systems **17** (2011), no. 3, 221–242.

[100] W M Shen, B Salemi, and P Will, *Hormones for self-reconfigurable robots*, Proceedings of the 6th International Conference on Intelligent Autonomous Systems, Citeseer, 2000.

[101] Wei-Min M Shen, Maks Krivokon, Harris Chiu, Jacob Everist, Michael Rubenstein, and Jagadesh Venkatesh, *Multimode locomotion via SuperBot reconfigurable robots*, Autonomous Robots **20** (2006), no. 2, 165–177.

[102] Wei-Min M Shen, Yimin Lu, and Peter Will, *Hormone-based control for self-reconfigurable robots*, Proceedings of the fourth international conference on Autonomous agents (New York, New York, USA), ACM, ACM Press, 2000, pp. 1–8.

[103] Satoshi Shiba, Masafumi Uchida, Akio Nozawa, Hirotoshi Asano, Hitoshi Onogaki, Tota Mizuno, Hideto Ide, and Syuichi Yokoyama, *Autonomous reconfiguration of robot shape by using Q-learning*, Artificial Life and Robotics **14** (2009), no. 2, 213–218.

[104] Thierry Siméon, Jean Paul Laumond, and Carole Nissoux, *Visibility-based probabilistic roadmaps for motion planning*, Advanced Robotics **14** (2000), no. 6, 477–493.

[105] Karl Sims, *Evolving virtual creatures*, Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (New York, NY, USA), SIGGRAPH '94, ACM, 1994, pp. 15–22.

[106] Geum-Beom Song and Sung-Bae Cho, *Combining incrementally evolved neural networks based on cellular automata for complex adaptive behaviors*, First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. (San Antonio, TX), IEEE, 2000, pp. 121–129.

[107] Kenneth O. Stanley and Risto Miikkulainen, *Evolving neural networks through augmenting topologies*, Evol. Comput. **10** (2002), no. 2, 99–127.

[108] Peter Stone and Manuela Veloso, *Layered Learning*, Machine Learning (Ramon López de Mántaras and Enric Plaza, eds.), LNCS, vol. 1810, Springer Berlin Heidelberg, Berlin, Heidelberg, jan 2000, pp. 369–381.

[109] Rainer Storn and Kenneth Price, *Differential evolutiona simple and efficient heuristic for global optimization over continuous spaces*, Journal of global optimization **11** (1997), no. 4, 341–359.

[110] K Stoy, W M Shen, and P Will, *Implementing configuration dependent gaits in a self-reconfigurable robot*, IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA'03, vol. 3, 2003, pp. 3828–3833.

[111] K Stoy, Wei-min M Shen, and Peter M Will, *Using role-based control to produce locomotion in chain-type self-reconfigurable robots*, IEEE/ASME transactions on mechatronics **7** (2002), no. 4, 410–417.

[112] K Stoy, WM Shen, and P Will, *On the use of sensors in self-reconfigurable robots*, 7th international conference on simulation of adaptive behavior SAB, 2002, pp. 48—-57.

[113] Stoy, Kasper and Brandt, David and Christensen, David J and Brandt, *Self-reconfigurable robots: an introduction*, Mit Press Cambridge, 2010.

[114] Ioan a. Sucan, Jonathan F. Kruse, Mark Yim, and Lydia E. Kavraki, *Reconfiguration for Modular Robots Using Kinodynamic Motion Planning*, ASME 2008 Dynamic Systems and Control Conference, Parts A and B, ASME, 2008, pp. 1429–1431.

[115] Gentaro Taga, *A model of the neuro-musculo-skeletal system for anticipatory adjustment of human locomotion during obstacle avoidance*, Biological Cybernetics **78** (1998), no. 1, 9–17.

[116] Yuzuru Terada and Satoshi Murata, *Automatic modular assembly system and its distributed control*, The International Journal of Robotics Research **27** (2008), no. 3-4, 445–462.

[117] Michael T Tolley, Student Member, Jonathan D Hiller, and Hod Lipson, *Evolutionary Design and Assembly Planning for Stochastic Modular Robots*, New Horizons in Evolutionary Robotics, no. c, Springer, 2011, pp. 211–225.

[118] Fernando Torres and J Zagal, *Automated Synthesis of Locomotion Controllers for Self-reconfigurable Modular Robots*, From Animals to Animats 12 (2012), 412—-420.

[119] P. Varshavskaya, L. P. Kaelbling, and D. Rus, *Automated Design of Adaptive Controllers for Modular Robots using Reinforcement Learning*, The International Journal of Robotics Research **27** (2008), no. 3-4, 505–526.

[120] P. Varshavskaya, L.P. Kaelbling, and D. Rus, *Learning distributed control for modular robots*, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 3, IEEE, 2004, pp. 2648–2653.

[121] Paulina Varshavskaya, Leslie Pack Kaelbling, and Daniela Rus, *Efficient Distributed Reinforcement Learning through Agreement*, Distributed Autonomous Robotic Systems 8 (2009), 367.

[122] Frank Veenstra, *The watchmaker's guide to artificial life: On the role of death, modularity and physicality in evolutionary robotics*, IT-Universitetet i København, 2019.

[123] V Vonasek and Martin Saska, *Global motion planning for modular robots with local motion primitives*, International Conference on Robotics and Automation (2013), 2465–2470.

[124] Vojtech Vonasek, Sergej Neumann, David Oertel, and Heinz Worn, *Online motion planning for failure recovery of modular robotic systems*, 2015 IEEE International Conference on Robotics and Automation (ICRA) (2015), 1905–1910.

[125] Justin Werfel, Kirstin Petersen, and Radhika Nagpal, *Designing collective behavior in a termite-inspired robot construction team*, Science **343** (2014), no. 6172, 754–758.

[126] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone, *Evolving soccer keepaway players through task decomposition*, Machine Learning **59** (2005), no. 1-2, 5–30.

[127] T L Williams, K A Sigvardt, N Kopell, G B Ermentrout, and M P Remler, *Forcing of coupled nonlinear oscillators: studies of intersegmental coordination in the lamprey locomotor central pattern generator*, Journal of neurophysiology **64** (1990), no. 3, 862.

[128] M Yim, *A reconfigurable modular robot with many modes of locomotion*, Proc. of Intl. Conf. on Advanced Mechatronics, 1993, pp. 283–288.

[129] M. Yim, *New locomotion gaits*, 1994 IEEE International Conference on Robotics and Automation, 1994. Proceedings., IEEE Comput. Soc. Press, 1994, pp. 2508–2514.

[130] M. Yim, Wei-Min Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G.S. Chirikjian, *Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]*, Robotics Automation Magazine, IEEE **14** (2007), no. March, 43–52.

[131] M Yim, Ying Zhang, David Duff, and B Y Mark Yim, *Modular robots*, IEEE Spectrum **39** (2002), no. 2, 30–34.

[132] M. Yim, Ying Zhang, K. Roufas, D. Duff, and C. Eldershaw, *Connecting and disconnecting for chain self-reconfiguration with polybot*, IEEE/ASME Transactions on Mechatronics **7** (2002), no. 4, 442–451.

[133] Mark Yim, Kimon Roufas, David Duff, Ying Zhang, Craig Eldershaw, and Sam Homans, *Modular reconfigurable robots in space applications*, Autonomous Robots **14** (2003), no. 2, 225–237.

[134] Mark H Yim, David Goldberg, and Arancha Casal, *Connectivity planning for closed-chain reconfiguration*, Intelligent Systems and Smart Manufacturing, International Society for Optics and Photonics, 2000, pp. 402–412.

[135] Keisuke Yoneda, Ikuo Suzuki, Masahito Yamamoto, and Masashi Furukawa, *Acquisition of Adaptive Behavior for Virtual Modular Robot*, Advances in Artificial Life. Darwin Meets von Neumann **5777** (2011), 181–188.

[136] E. Yoshida, H. Kurokawa, a. Kamimura, K. Tomita, S. Kokaji, and S. Murata, *Planning behaviors of a modular robot: an approach applying a randomized planner to coherent structure*, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566) **2** (2004), 2056–2061.

[137] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji, *Evolutionary synthesis of dynamic motion and reconfiguration process for a modular robot M-TRAN*, Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics

and Automation for the New Millennium (Cat. No.03EX694), vol. 2, IEEE, 2003, pp. 1004–1010.

[138] Eiichi Yoshida, Satoshi Murata, Akiya Kamimura, Kohji Tomita, Haruhisa Kurokawa, and Shigeru Kokaji, *Evolutionary Motion Synthesis for a Modular Robot using Genetic Algorithm*, Journal of Robotics and Mechatronics **15** (2003), no. 2, 227–237.

[139] Chih-han H Yu and R Nagpal, *Sensing-based shape formation tasks on modular multi-robot systems: A theoretical study*, Proc. Intl. Conf on Autonomous Agents and Multi-Agent Systems (AAMAS), no. Aamas, 2008, pp. 12–16.

[140] Payam Zahadat, Thomas Schmickl, and Karl Crailsheim, *Evolving Reactive Controller for a Modular Robot: Benefits of the Property of State-Switching in Fractal Gene Regulatory Networks*, From Animals to Animats 12 (2012), 209—-218.

[141] V. Zykov, E. Mytilinaios, M. Desnoyer, and H. Lipson, *Evolved and designed self-reproducing modular robotics*, IEEE Transactions on Robotics **23** (2007), no. 2, 308–319.