



Norwegian University of
Science and Technology

Swarm intelligence in bio-inspired robotics

Jannik Berg
Camilla Haukenes Karud

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Pauline Haddow, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

One of the hardware challenges that the field of evolvable hardware aims to excel in is the achievement of adaptive hardware. Swarm robotics provides a forum where a single robot must adapt to its environment wrt the other robots of the swarm and wrt the physical environment. The project can be a software project running the intended algorithms on our robot simulators with a view to final implementation in hardware on real robots during the spring. The student(s) may select their favorite AI techniques as the basis of investigation.

Assignment given: 17. January

Supervisor: Professor Pauline C. Haddow, IDI

Abstract

This report explores the field of swarm intelligence in bio-inspired robotics. By taking inspiration from the social insect ants, this project aims to investigate collective behavior through a common box-pushing task and the use of nine autonomous mobile robots called e-pucks. The system is based on biological observations of ants, a behavior-based architecture from Rodney Brooks' subsumption, movement created from a pre-made artificial neural network and simple processing mechanism to create swarm behaviors in a physical multiagent environment.

Preface

This master thesis is written at the Department of Computer and Information Science at NTNU during the spring semester of 2011. The IDI faculty and professors at NTNU provide students with several different project options, which can be interpreted and built upon by students own creativity. This master thesis builds upon a specialization project of the fall of 2010 which had the following assignment:

”One of the hardware challenges that the field of evolvable hardware aims to excel is in the achievement of adaptive hardware. Swarm robotics provides a forum where a single robot must adapt to its environment wrt the other robots of the swarm and wrt the physical environment. The project can be a software project running the intended algorithms on our robot simulators with a view to final implementation in hardware on real robots during the spring. The student(s) may select their favorite AI techniques as the basis of investigation.”

During experimentation and research of the fall of 2010 we came across interesting studies of swarm intelligence and robotic projects. For our master thesis we aim to build upon biological studies of ants to make a biological plausible system of the ant retrieving model. Changing the control platform from remote-control (Bluetooth) to cross-compilation, and incorporating a behavior-based architecture together with capabilities of the e-puck will be the major challenges towards having a emerged self-organized system that collectively will be able to accomplish the box-pushing task.

Finally, we want to thank:

- Professor Pauline C. Haddow for her belief in this project, for the opportunities she has given us and for her support throughout the entire process
- Professor Keith Downing for introducing us to the e-pucks.
- Frithjof Christie Knudtzen for all the plastic material, necessary for building the box.
- Bjørn-Gunvar Nessjøen, Johannes Høydahl Jensen, Snorre Corneliussen and the people at the Omega workshop for guidance in electronics, the supply of components and assistance during construction.
- The people at the CRABlab at NTNU for inspiration, motivation, knowledge, discussions and the social events.

Trondheim, June 2011

Jannik Berg & Camilla Haukenes Karud

Contents

1	Introduction	1
1.1	Report Structure	2
2	State of the Art	3
2.1	Swarm Intelligence	3
2.1.1	Social Insects	3
2.1.2	Ants	4
2.2	Collective Behavior	4
2.2.1	Cooperation	4
2.2.2	Communication	5
2.2.3	Self-Organized Systems	8
2.3	Swarm Robotics	8
3	Background	11
3.1	Agent Control Systems	11
3.1.1	Deliberative systems	11
3.1.2	Reactive Systems	12
3.1.3	Hybrid	13
3.1.4	Behavior-Based Systems	13
3.2	Architectures	14
3.2.1	Brooks' Subsumption Architecture	14
3.2.2	TouringMachines	15
3.3	Artificial Neural Network	16
3.4	Machine Learning	17
3.4.1	Supervised Learning	17
3.4.2	Reinforced Learning	18
3.4.3	Unsupervised Learning	18
4	Robot Development	19
4.1	The E-puck Robot	19
4.2	Webots	21
4.3	Environment	22
4.3.1	Simulation Environment	22
4.3.2	Physical Environment	22
4.4	Early Experience & Analysis	23
4.5	Software to Hardware	24
4.6	E-puck Sensor Testing	25
4.6.1	Sensor Overview	25
4.6.2	Proximity Sensor Testing	27
4.6.3	IR Light Sensor Testing	30

4.7	Conclusion	33
5	Artificial Food Source	35
5.1	Designing the Box's Frame	35
5.2	IR Emitter	36
5.3	Circuit	37
5.4	Construction	39
6	Design & Implementation	41
6.1	CRABS	41
6.1.1	The Box-Pushing Task	42
6.1.2	System Overview	43
6.2	Code Structure	46
6.2.1	Accessing E-puck	47
6.2.2	Search & Avoidance	48
6.2.3	Converge & Push	49
6.2.4	Realignment & Reposition	50
6.3	The Behavior-Based System	53
7	Experiments, Results & Discussion	55
7.1	Experimentation	55
7.1.1	Setup	55
7.1.2	Goal	57
7.1.3	Realignment	63
7.1.4	Reposition	67
7.2	CRABS	70
7.2.1	Results	71
7.2.2	Analysis	72
7.3	Discussion	73
7.3.1	The Swarm CRABS	73
7.3.2	Brooks' Subsumption as a Social Insect Architecture	76
7.3.3	Simplicity & Scaling	76
7.3.4	E-puck as Swarm Robots	77
8	Conclusion & Further work	79
A	Project Settings	87
A.1	Operating Systems with Webots	87
A.2	Development Tools	89
A.2.1	Software Development Environment	89
A.2.2	Simulation Development Environment	90
A.3	User & Installation Guide	93
A.3.1	Installing Webots	93
A.3.2	Installing Eclipse	94
A.3.3	Using Python with Eclipse and Webots	97
A.3.4	Using C with Eclipse and Webots	101
A.3.5	Using the real e-puck with Webots	106

List of Figures

2.1	Honey bees	6
2.2	Ants	7
3.1	The sense-model-plan-act design.	11
3.2	The reactive design approach.	12
3.3	Horizontal version of the hybrid approach.	13
3.4	Traditional Design.	14
3.5	Brooks subsumption architecture.	15
3.6	Illustration of the TouringMachines architecture	15
3.7	Illustration of a network of neurons.	16
4.1	The e-puck.	20
4.2	The e-puck's eight IR sensors.	21
4.3	The simulated e-puck and the real e-puck	21
4.4	The simulation environment	22
4.5	The physical environment at the CRAB lab	23
4.6	Initial testing procedure.	25
4.7	Proximity detection.	26
4.8	IR light detection.	26
4.9	All the different LED lights	27
4.10	Average proximity values in dark conditions.	28
4.11	Average proximity values in bright conditions.	29
4.12	Average initial IR values in bright conditions.	31
4.13	Average initial IR values in dark conditions.	31
4.14	Average IR detection values, 30cm.	32
4.15	Average IR detection values, 0cm.	33
5.1	The first step towards an artificial food source	36
5.2	The first test circuit.	37
5.3	The IR emitter used in this project	37
5.4	The complete circuit design.	38
5.5	The components	39
5.6	The motherboard	39
5.7	The artificial food source.	40
6.1	Illustration of the environmental design.	43
6.2	System overview.	44
6.3	CRABS code structure.	47
6.4	The processes occurring during each time step.	47
6.5	The design of search and avoidance behavior.	48
6.6	The converge behavior.	49

6.7	The push behavio.	50
6.8	Illustration of the realignment design.	52
6.9	Illustration of the reposition design.	52
6.10	The visual feedback during recovery.	53
6.11	CRABS architecture, Brooks' subsumption.	54
7.1	CRABS during testing of the goal module.	57
7.2	The goal behavior.	57
7.3	Optimal positions for a group of three e-pucks.	59
7.4	Stagnation patterns for a group of three e-pucks.	59
7.5	Stagnation cases	61
7.6	Circular box movement	61
7.7	E-puck trapping	62
7.8	CRABS during testing of the realignment behavior.	63
7.9	The realignment behavior.	64
7.10	Stagnation occurences with realignment	65
7.11	Opposite forces stagnating.	66
7.12	CRABS behavior setup during reposition testing	67
7.13	The reposition behavior	68
7.14	Reposition avoidance	70
7.15	CRABS.	71
7.16	The performance of all the behaviors.	72
7.17	The average time spent of each behavior.	73
A.1	Eclipse workbench	89
A.2	Simulation and editor mode in Webots	91
A.3	Installing Webots.	93
A.4	The startup splash screen.	94
A.5	Installation of plug-ins.	95
A.6	Already imported software sites	95
A.7	Location and enabling of the subversive software site	96
A.8	Selection of the SVN Team Provider software	96
A.9	Selection of connectors	97
A.10	The location of the python environment.	98
A.11	Adding the python location into the environment path of windows	98
A.12	Testing the python installation	99
A.13	Adding pydev to the repository of avaiable software sites	99
A.14	Pydev certificate	100
A.15	The code structure of our python project	101
A.16	The setup of the e-puck.	101
A.17	Link to the latest MinGW release	102
A.18	Installing MinGW.	102
A.19	Adding the bin path	103
A.20	Testing the C compiler.	103
A.21	Controller.lib library	105
A.22	Makefile	105
A.23	Robot Window	106
A.24	Makefile.epuck cross-compilation	107

List of Tables

5.1	Dimensions and weights of the plastic box's frame	36
7.1	CRABS initial configuration	56
7.2	Results from the goal behavior.	58
7.3	Goal performance with three e-pucks	58
7.4	Goal performance with five e-pucks	60
7.5	Goal performance with eight e-pucks	61
7.6	Results from the realignment behavior.	63
7.7	Realignment performance with three e-pucks	64
7.8	Realignment performance with five e-pucks	65
7.9	Realignment performance with eight e-pucks	66
7.10	Results from the reposition behavior	67
7.11	Reposition performance with three e-pucks	68
7.12	Reposition performance with five e-pucks	69
7.13	Reposition performance with eight e-pucks	69
7.14	Results from food retrieval with CRABS.	71
7.15	Results	74

Introduction

This report presents the master thesis of swarm intelligence in bio-inspired robotics. The project description was given to us by Professor Pauline Haddow at the NTNU Department of Computer and Information Science.

In our search for making artificial intelligent (AI) systems we often take inspiration from biology and nature. Trying to recreate intelligence, like the one evolution has been developing for quite some time now, it would almost seem certain that answers could be found in our surroundings. Even though the optimal solution may not yet exist, and probably never will because of our ever changing environment, it provides a good starting point for further research.

There are multiple definitions of the word intelligence. When it comes to the history of AI computing, it seems that as systems are breaking new limits, we set new standards to what we look upon as an intelligent system. The first chess-playing computer that beat a reigning world champion was the Deep Blue in 1997. A system that could beat a world champion in chess would almost certainly be labeled as intelligent, but looking closer into how the system actually operates, it could be argued that it is mostly based on brute force and lack the ability of how humans play the game (especially the pattern recognition). It is kind of like discovering how a magic trick works; it is just not that impressive anymore. So whether a system is intelligent or not, is arguably ever changing, but it doesn't always depend of how complex it is. At least not if we look at the field of swarm intelligence.

From the simple social insects we find in nature, there can emerge an intelligent behavior beyond what could be achieved by one single insect. Even though each single individual has only a set of simple mechanisms and is only acting upon the local environment, together they manage to create a global structure and solve complex problems in their everyday lives.

The motivation behind this project was the fact that swarm intelligence is a vast and growing research field both theoretical and experimental wise, and its simplicity should be possible to replicate in small autonomous robots. The efficiency, flexibility and robustness of these collective behaviors is remarkable and can often be applicable to other systems. Its inspiration has already been implemented in system such as discrete optimization, graph partitioning, task allocation and collective decision

making, to name a few [23].

1.1 Report Structure

Chapter 2 presents the state of the art. It explores the field swarm intelligence with focus on the biological swarm aspects in social insects, their features and swarm robotics. Chapter 3 contains related background information researched with a focus on architectures and techniques. Chapter 4 introduces the robot we have used in this project, the simulation tool Webots, and the physical environment. It also describes some earlier experience and hardware tests on our e-pucks. Chapter 5 shows the construction of the artificial food source. Chapter 6 presents the design and implementation of our behavior-based system CRABS. Chapter 7 describes and analyzes the experiments and results, as well as discussing the challenges we have investigated. Chapter 8 concludes the report including thoughts about future work.

State of the Art

In the following sections we describe some of the key concept of swarm intelligence and explore the field of swarm robotics.

2.1 Swarm Intelligence

Swarm intelligence comes from the biological study of social insect and insight about how they manage to solve complex problems in their daily lives. Research fields as swarm optimization and distributed control in collective robotics was born from studies and observations on these tiny creatures, where collecting inspiration has been equally important as biologically replicating the observed system. Swarm systems are examples of behavior based systems and many of the behaviors can be adapted to different systems, optimizing or solving problems of a different nature [23].

An organization without an organizer, is perhaps the most notable factor of swarm behavior. Social insects, fishes, birds, etc. - all seem to collaborate as one unit without having a leader to follow. Strangely enough, the complexity of these behaviors is done by relatively simple mechanism in each individual. By processing their sensory inputs, which for any living thing is limited, and hence can be described as local information, they manage to coordinate and cooperate to do actions beyond that of a single unit. Also, encountering new problems and discovering new solutions, like ants building a bridge across water or gaps, shows how adaptive they can be. From all these actions based on local information, a complex global behavior arises [8].

2.1.1 Social Insects

During the past 20 years, social insects [8] have become a popular field in biological research. The four most known types of social insects are ants (*Oecophylla*), honey bees (*Apis mellifica*), wasp (*Polybia occidentalis*) and termite (*Macrotermes*), which all share the traits of living in large colonies. Characteristic for social insects is how they manage to solve complex tasks, using only simple mechanisms and no global perception. Of the four types of social insects, ants are the most popular research area.

2.1.2 Ants

A well-known myth about ants is that in an ant hill, there is only one queen. The queen's role is to distribute tasks, decide and command the rest of the ants. This is however not the case. An ant hill can have one or more, and in some cases not any queens within the colony. The queen's role is basically to produce offspring, while the ants work accordingly to whatever sense that is stimulated, which often results in maintaining the hive or finding resources to survive.

The general ant has mainly three different senses; smells, touch and taste, and is only able to perform a set of actions. These senses have limited range which makes it impossible for one single ant to guide or control an entire colony as it may consist of millions of ants. There is no leader who controls the actions of each ant; an ant do actions based on changes in the local environment. Ants do have roles, but still there exists no position or rank division among them. With none to lead or control, they still have a driving force to cooperate and solve problems as equals.[8]

2.2 Collective Behavior

Another widely used term in swarm intelligence is collective behavior (adaptation)[31]. Collective behavior can be defined as actions which differ from the already existing social structure of the group. There may be actions which are not in conformity with each other, or violates certain norms in the system. The fact that the actions violate the standards can be caused by different things. They may be unclear or contradict each other and from an outside perspective not make much sense. Such actions often occur when spontaneous changes happen in the environment and they quickly have to react. In nature and for ants, this could happen when a nest is destroyed the ants need to reconstruct it [23], or they get attacked by other insects and have to defend their nest, food or other resources.

2.2.1 Cooperation

Collective tasks are either non-cooperative or cooperative. Non-cooperative tasks are typically tasks which don't need any kind of cooperation between the individuals performing the tasks and can be successfully performed by one single individual given unlimited time [8][23]. It would however be more efficient if individuals cooperated. Examples of such non-cooperative tasks are: sorting [15][28], searching [52][39], map making [9][50], harvesting [52], vacuuming [40]. Cooperative tasks are tasks which can't be completed by one single individual. The goal can only be achieved by two or more individuals working together [21]. Examples of such cooperative tasks are: formation marching [14], material transport [18], tandem movement [22], box-pushing [34][46][17]. Cooperation (and in some cases competition) is the essential behavior of any type of swarm intelligence.

To survive in the world, the ants need to cooperate. The same applies when they are building the nest. Building a nest consist of two steps: **(1)** First, the ant need to find leaves. The leaves are too heavy for one ant to handle, so they need to make

a long line, and pull together. If it still is too heavy, more ants will help by making a second row of ants. They will connect with the ant in front of them, and they will together pull it as one bigger unit. They pull the leaf until it lies next to the other leaves that form some of the upcoming nest. **(2)** The next step is to use the silk which the larva produces by gluing the leaves together. Also here is one single ant not strong enough to carry one larva by itself. [21]

By looking at the behavior of ants, we can see how this constitutes to a swarm behavior. They depend on helping each other to solve problems like building nests, finding food and fighting against enemies, to name a few. One of the many assets to swarm behavior is that it relies on simple mechanisms, which is great for robotics, and that it can be applied and implemented in several different contexts. It is a great source for inspiration.

2.2.2 Communication

Communication is the most fundamental property that must be present for either competition or cooperation to occur between individuals. Communication can be both direct and indirect. Direct communication is exchanging information between individuals where all the participants are involved and are present at the same time. For us humans, this can be associated with things we say or write, and in social insects, transmitting signals through touch. Indirectly communication has a more defuse definition. It can be anything from body language (behavior), force, symbolic elements or time spent [34]. The research suggests that social insects base almost all exchange information through indirect communications, such as described about ants and honey bees later in section 2.2.2.

Stigmergy

Stigmergy [21][23] is a mechanism in indirect communication and in general, self-organized systems. It was first observed in social insects by Pierre-Paul Grasse in 1950's [24]. The concept of stigmergy is that an individual must always take into account the changes that have occurred in the environment and act on the basis of these. If one individual modifies the environment, the same individual or another individual get stimulated and responds to the changes by performing actions based on the new environmental settings [34]. Stigmergy is a necessity in any dynamic swarm system because of the scaling issue in direct communication.

Division of Labor

Division of labor[7][29] is a pattern of variation between individuals in a colony. Each individual eventually becomes specialized in parts of the tasks performed by the colony as they continue doing work which they are good at. Division of labor is both fundamental and necessary for a colony to function effective as a group. The determination of which tasks each individual will perform is based on two patterns: age and size/shape.

Pheromones

The most common form of communication among social insects are the use of pheromones [8][56][27]. Pheromones are chemical signals sent out from one individual to trigger a reaction behavior on the receiving individuals of the same species. These chemical signals can be used in many different occasions; trail-following [25], defense [16][29], mating and retrieving food.

Communication among Honey Bees

The communication among bees was discovered by Karl von Frisch, who got a Nobel Prize for it in 1973. He discovered that the bees are communicating with each other through the language of dance. A bee can perform different type of dances, which gives different information. The two most common dances are the round dance and the waggle dance. The round dance is often used when a food source is close to the hive, between 50 and 150 meters away. The way a bee performs the round dance, is by walking around in circles, suddenly turn 180 degrees, and then start walking forwards, see figure 2.1(a). This round dance says that the food source is nearby, but it doesn't say anything about where it is located. The waggle dance however, figure 2.1(b), is the most known. This is a complex dance, which describe both the direction and the distance. The dance is performed by walking around in two small circles and shaking its body whenever it is in the middle. Both dances are illustrated in figure 2.1. However, the main purpose of these dances is to recruit the other bees, and tell them where food sources or possible new nest sites are located. [45]

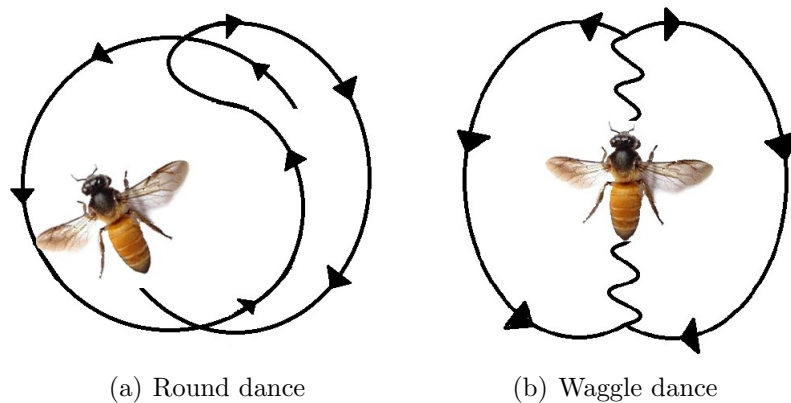
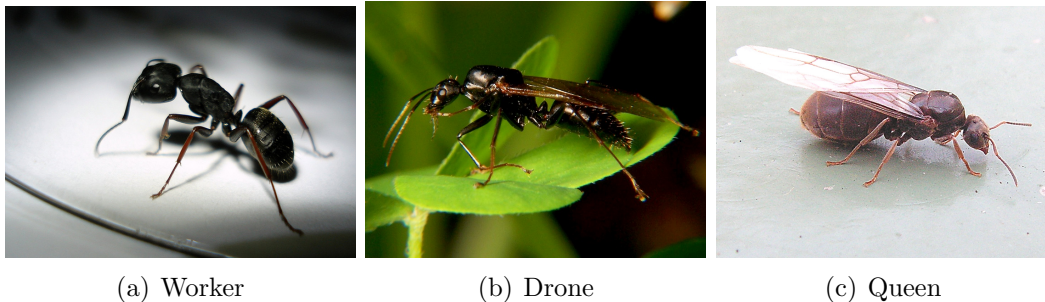


Figure 2.1: Honey bees

Communication among Ants

The structure of an ant colony is flat, which means that there is no leader in the group. Each ant perform actions based on observations in the local environment (the stigmergy principle), thereby creating a global structure, since all are following much of the same reactive patterns. Even though they perform action based on changes in the environment, there is a rough division among them. In a colony there are three main types of ants; queen, drone and worker, see figure 2.2. Common for all

ants is that they base communication on using pheromones, and detects them by using their antennae (similar to smell). Their antennae are equipped with features to both notice direction and intensity of pheromones. When an ant finds a new food source, it will lay down pheromones at the way back to the hill [25]. The intensity of the path created will increase as more ants follow the path and find the food source using the same principle of laying down pheromones on the way back. This principle will also lead to finding the shortest path when faced with several. However, the pheromones do not last forever. When the food source diminishes, fewer ants lay down pheromones, and the path will eventually disappear. [8][23]



(a) Worker

(b) Drone

(c) Queen

Figure 2.2: Ants

Shortest Path

The ants are able to always find the shortest path to a food source, by using the concept of pheromones. If there are two paths to the same food source, where one is longer than the other, the ants will soon find the shortest path. This happens when the ants put down pheromones on their way back from the food source to the ant hill. Since the longer path takes more time to walk, it will also take longer time to build up a solid pheromone path than it would do with a shorter path. This way it will start with a 50% chance of choosing the shortest path, and increase more as time goes by, and more ants put down pheromones on the shortest path.

An interesting point is where the bees and the ants are each trying to find the best food sources. Let us say that we have two food sources, one good and one bad and where both bees and ants discover them simultaneously. Then both bees and ant will eventually always choose the best food source. However, if they discover the bad food source first and then the good food source, ants will not be able to switch, but bees will. This is because of the pheromones that the ants lay down. Even though the ants discover the new and better food source, there still exists pheromones on the path to the poorer food source, and the ant will continue to go there until the food decrease. In other words, it is more difficult for the ant to quickly move from one food source to another, because of the pheromones they use.

2.2.3 Self-Organized Systems

Self-organization [21][10] takes form by processes performed by simple individuals which, by either competition or cooperation, create an organized global structure. Individuals make actions based on what they receive from the local environment, without actually knowing or concerning themselves about global structure. Since each action is a reaction to what is happening around the individual, it is not necessary to have a manager who is responsible for assigning tasks, which on the other hand would have been almost impossible in larger colonies. Examples of such self-organized systems are: fish schooling [47], formation [26], synchronization [14] and mapping [21].

Feedback: Positive and Negative

Self-organized systems builds upon two opposite forces; attraction and repulsion. These forces occur in biologically inspired systems when quantity of some sort is sent back into the system to increase or decrease the magnitude of that same quantity. By e.g. looking at the intake of food in a colony it is important to have a good balance. If they have too little, they will starve, but on the other hand, if they have too much food they have to throw the bad food away. Nature finds this balance by rewarding and revoking behaviors. This concept of attraction and repulsion is in biologically systems presented as positive and negative feedback.

2.3 Swarm Robotics

As a result of the increased interest of swarm intelligence, a new field emerged; swarm robotics. The expression "swarm robotics" is a term used for the coordination and behaviors occurring between larger groups of simple robots. The robots used in swarm projects [18][11] are often cheap (need several) and contain limited sensing features (biologically realistic). The amount of swarm robots within a group can consist of anything from 2-3 individuals and up, but most common at a size of 5-15 individuals.

General for swarm robotic systems is that they are all inspired by nature and how animals, especially social insects, behave in relation to each other, how the global and local structure differs from each other, and how they as a group manage to solve complex tasks. The Swarmlab [14] group from the Maastricht University in Netherlands is one of many research groups that focus on swarm intelligence and swarm robotics. Their goal and motivation is to build adaptive robotic systems capable of learning from the environment. They are inspired from the biology and especially social insects when it comes to swarm intelligence[4], reinforcement learning [30] and evolutionary algorithms [37] [38].

There are two main types of swarm robotic projects; those who use heterogeneous robots and those who use homogeneous robots. Heterogeneous swarm robot project [20][19] is necessary when the task requires different abilities from the robots. Homogeneous project [57][18] is where the robots don't need to be specialized in some

subparts in order to achieve the goal. In addition to distinguish projects of heterogeneous and homogeneous robots, there is also a difference of simulated [57] and physical swarm robotic projects[34].

In researching today, most groups creates their own designed robots from scratch, if the funding, material and knowledge are available.[34] The advantage of creating your own robots, is that you get to customize the robots to the extent needed. However, one of the pre-assembled robot that have received much attention through research the last decade, is the e-puck robot.[2] The e-pucks have been designed especially for learning purpose, with concerns of being cheap, flexible and robust, which has increased the interest of research among robotics and AI for students and scientists around the world.

Since the e-pucks are so cheap and contain several types of sensors, the robot is well suited for swarm based projects [12] [48] where high numbers of robots is required. One example of a project using the e-puck robots is: "Event detection and localization for small mobile robots using reservoir computing" [5] from the Ghent University in Belgium, from 2008. Through this project they have been investigating:

- Recognizing complex events in particular environments
- Determining the current robot location

Both tasks were simulated through a simple autonomous robot simulator, developed by Antonelo in 2006, and later in the physically realistic Webots simulator.

Box-Pushing with Kube & Zhang

C. Ronald Kube and Hong Zhang started in the early 90' at the University of Alberta in Canada a project called CRIP (Collective Robotic Intelligence Project) [33][32]. They wanted to examine the problems occurring when controlling multiple behavior-based autonomous robots. These autonomous mobile robots have been handmade especially for CRIP and have no centralized control or explicit communication. Their only feature is passive sensing through their integrated sensors. Each robot has two photovoltaic goal sensors which are used to locate the box (goal), two near-infrared sensors for avoiding other robots and one stagnation sensor to make the robots capable of getting positive/negative achievement feedback.

Their goal and motivation for this project is to get all the robots to form a homogeneous group of autonomous robots and through cooperation locate and move a box, which is unmovable for a single robot. They got their inspiration for this project by exploring social insects in nature with main focus on ants and their behaviors as a colony.

In 1993 C. Ronald Kube and Hong Zhang explored the idea of using an Adaptive Logic Network (ALN) for behavior arbitration in cooperation with Xiaohuan Wang [35]. The reason for using this approach was based on the results they got when

using a fixed priority scheme between the behaviors. For few behaviors this model worked very well, however, when the behaviors increased the difference between them decreased in terms of the priority. As a solution, they tried to transform it into a pattern classification problem, which an ALN can solve by training.

So far the CRIP experiment is a decentralized complex system of five mobile robots capable of achieving simple collective tasks as finding an object and avoiding other robots. These tasks are accomplished through cooperation without any explicit communication. Later in 1993, Kube and Zhang wanted to use the experience gained so far and make the robots perform the common box-pushing task both in simulation, with SimbotCity, and in real life [34]. They have only tested the control mechanisms for some of the behaviors on the real robots.

During the previous experiments with both the fixed priority scheme between the behaviors and the ALN arbitration, C. R. Kube and H. Zhang came across some problems related to stagnation and cyclic behavior. In 1994, they investigated a possible solution to this problem, in terms of their project with the box-pushing robots [36].

Background

This chapter present information about the general concepts of swarm intelligence, robotics and learning. It first elaborates on the matter of the design of an autonomous robot, moves deeper in the system to look at how we can achieve intelligence and ends up presenting the field of swarm intelligence.

Event though this report is focused on robotics, many of the sections in this background material will use the general term agents. The reason is that the information presented is not limited to robots, and can be useful in several different contexts.

3.1 Agent Control Systems

The control we have over our body comes from the central nervous system which humans possess. It allows us to send inputs and perform actions. Just like humans, intelligent agents need a mechanism to be able to coordinate inputs and actions. This section presents some of the most established design approaches towards such control systems [55].

3.1.1 Deliberative systems

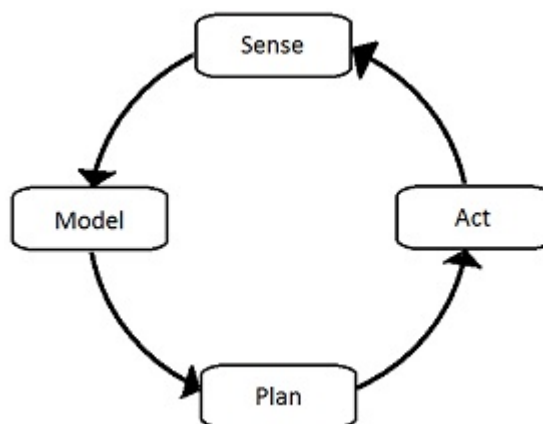


Figure 3.1: The sense-model-plan-act design.

A deliberative system was one of the first presented design approaches in the early days of AI. Today it is still a well known and used approach when designing agents.

For an agent to know which action it shall perform, it has to go through four phases; sensing, modeling, planning and acting. By using its sensors, the agent is able to collect a set of inputs which describes the environment in its current state. Based on this information, it will build a model of the environment, which will change during runtime. The planning module's task is to analyze the information from the model and create a sequence of actions that needs to be done in order to achieve the goal. When a suitable plan is found, the act module will transform the data so the agent is able to execute the plan.

A problem with this procedure is that it doesn't fit in a rapidly changing environment. The time it takes for the agent to process the input data and becoming ready to perform the action, can in many cases take too much time. When the agent receives the output from the system, the environment may already have changed in relation to the way it was when the model was build, making the purpose of the next action useless. This approach will work in most cases for smaller and simpler environments, but can become problematic for large and complex. For such an environment it would need access to more processing power and memory allocation to be able to build a fast and good model of the environment, in addition to create a plan for the agent and get the agent to execute that plan. Some systems, like simple robots would fall short in this situation, but a software program in today's computers may very well be able to handle the load.

3.1.2 Reactive Systems

The reactive system is a descendant of the deliberative system. In contrast of the deliberative system, an agent which uses a reactive approach doesn't plan every action it performs. Every action it performs is a response to what happens in the environment. Since it always maintains an ongoing interaction with its environment, and every input has a given output, it doesn't need a lot of memory or processing power. It doesn't matter how big or complex the environment is, since the input size will stay the same and the actions are static. Generally speaking, it distances itself from beliefs about the world where every input has a predefined correct response that will never change.

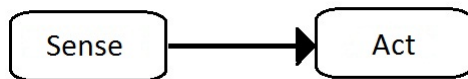


Figure 3.2: The reactive design approach.

By using this approach, it solves the problem of the deliberative design, because of the immediately reaction. However, there are some disadvantages with the reactive approach as well. It has no memory of previous experience, which makes it impossible for the agent to plan ahead, and it makes it difficult to implement in a

dynamic environment. As the reactive agent needs to be preprogrammed for every single encounter, it can become very static. If the system is not general enough, the agent can perform badly or even crash, especially if it encounters something that the designer didn't think of. The system will often never be able to handle changes, meaning that the purpose of the system will be narrowed.

3.1.3 Hybrid

Hybrid approach is an even younger design approach than the reactive. By looking at the advantages and disadvantages in terms of both the deliberative and reactive approach, there are clearly positive and negative sides with both. The hybrid approach combines already existing control systems together in a hierarchy as separated subsystems. Each subsystem would work independently of each other, but in this case they are brought together as layers to work as one big system. In cases where urgent response is needed, the reactive layer would be chosen, and cases where more complex action is needed the deliberative layer would be chosen. In this way a system would be more diverse, being able to handling inputs differently in different ways.

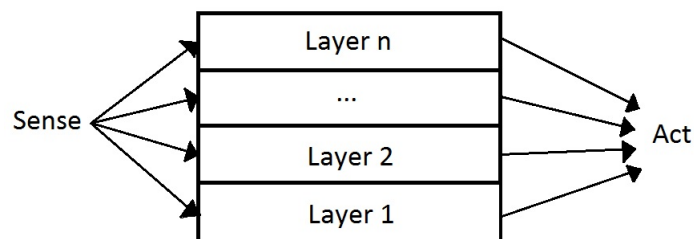


Figure 3.3: Horizontal version of the hybrid approach.

3.1.4 Behavior-Based Systems

A behavior-based approach resembles much of what we could find in a reactive system and some would even label it as a reactive subsystem. It was made famous by Rodney Brooks and his subsumption architecture, and became very popular in the field of robotics because it allowed for real-time processing systems that could work in complex environments.

A behavior is in most cases described as a sequence of interactions between e.g. an agent and its environment, where actions made by the agent affects future perceptions and actions. In this system the behaviors can be thought of as individual action selection functions, where each behavior module can perceive and map the input to a task-accomplishing action. Brooks claimed that intelligent behavior was created by a direct link between perceiving and acting (reactive) and that no reasoning was needed [21]. His view on intelligence would back up his statement on intelligent behavior. In his view, intelligence appeared in the eye of the observer as a result from multiple interactions in a situated and embodied system.

Even though Brooks design approach is very much used, his view on intelligence has

been more loosely kept throughout the years as people tend to incorporate memory and learning into the system.

3.2 Architectures

Based on the methodologies of the previous section 3.1, this section will elaborate on some of the architectures that have developed through use of these design approaches.

3.2.1 Brooks' Subsumption Architecture

The earlier dominant architecture of control systems for agents were usually divided into a sequence of functional modules such as perception, modeling, planning, task-execution and motor control (figure 3.4). This produced precise, controllable and predictable systems. It is a general architecture and can be applied to several instances of agents, and is especially popular in the field of robotics. A disadvantage is that it takes a lot of power for building models, and a failure at an early stage, can affect the entire system [21].



Figure 3.4: Traditional Design.

In 1986 Brooks proposed a behavioral decomposition of robot control, different from that of the functional. With a behavior-based architecture you would decompose the problem into task-achieving actions and have direct access to actuators and sensors, which would make the system able to operate in parallel to other behaviors. One of the key concepts behind this architecture is that the system is both situated and embodied. The system shouldn't be presented with abstract information, nor should it be simulated in something besides the world it runs in. It needs to directly experience and sense its environment hands on, and act upon it so forth.

Each layer in the new architecture is placed on top of each other and illustrates a stack of parallel behaviors, figure 3.5. The higher levels don't use the lower level as subroutines, but a set of preexisting competences. This is called the subsumption architecture.

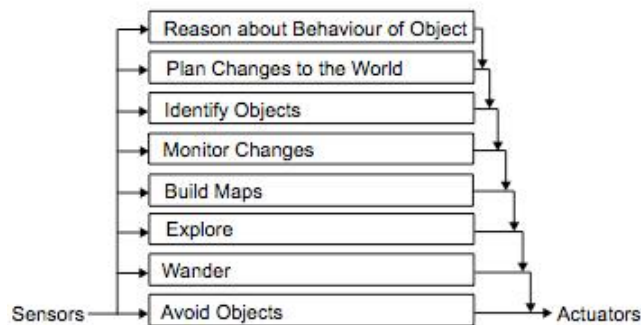


Figure 3.5: Brooks subsumption architecture.

The structure of Brooks' subsumption differs from other architecture structure in that it is vertical. In the process of making this system one should design, test, redefine until the layer works to the users satisfaction before moving on to next layer. When more layers are added, the system can take use of the competence from all. The lower, more basic layers should deal with the survival of the system, while the higher should be design to concentrate on goal achieving behaviors. The higher layer has priority over the lower layers, meaning that the higher layers can suppress a lower layers behavior. It is important to do an incremental design, so the higher layers won't be triggered unnecessary to ruin the survival of the system.

3.2.2 TouringMachines

The TouringMachines architecture[55] is divided into three separate subsystems, which all give an independent suggestion for what action the agent should perform next. A general TouringMachines architecture can be seen in figure 3.6.

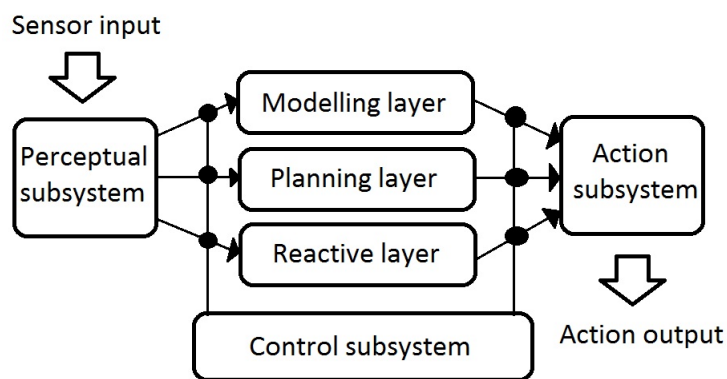


Figure 3.6: Illustration of the TouringMachines architecture

The reactive layer is very similar to Brooks' behavior approach, since the subsystem is implemented with a set of situation-action rules. The system is intended to give an immediate response to changes in the environment. There is no concept of the world inside this subsystem, only strict rules.

In the middle of figure 3.6 we find the planning layer. This is a subsystem similar to the deliberative system. Its main responsibility is to decide what the agent will do, give actions with a purpose towards a goal. Although this layer has a library of plans, it does not produce them, but considers which plan to use from the given input at run-time. The decision of which library or plan that should be executed would be the one who fits the criteria best to come closer to the agents' goal.

The third layer is the modeling layer, which represents concepts about the world including the agent itself. It is used to predict conflicts and create new goals or plans, which often is necessary in a dynamic environment. These plans are passed on to the planning library of the planning subsystem. This can be very useful in environments with several agents, where plans and goals may be in conflict of each other.

To manage all these individual control systems it is necessary to have an overall system to chose which layer should get the control of the agent. A general way of making this overall control system is to give a set of control rules. It can either suppress the inputs incoming to the layers or/and suppress the action going out to the agent.

3.3 Artificial Neural Network

Artificial neural networks (ANN) are software application (computational model) or hardware devices trying to recreate the behavior of a biological neural system. These systems are often built around the idea of connected neurons like the way human brains operate. Some of the neurons receive information directly from the environment (input layer), pass the signals on through a series of hidden or deeper located neurons (hidden layer), and ends up with the neurons that affect the environment (output layer). The connection between these neurons operates as the message path for signals. On its way through the system, different neurons will "fire" and tweak the signals, which will make them become a result/answer at the end. With the connection in place, one can produce, from different input signals, different output signals.

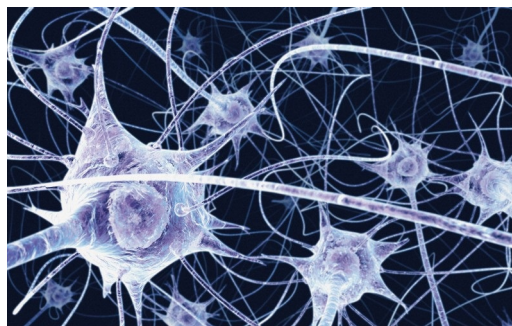


Figure 3.7: Illustration of a network of neurons.

Each neuron is set to active if the incoming signal is greater than its own threshold value. If a neuron is activated, it will send the information forward to all the neurons it is connected to. Each connection has its own weight, and can be either positive or negative.

In most cases the ANN becomes too complex to be able to make it statically. Like the brain which learns and changes continuously, it is usual to implement learning so the ANN can be trained to operate more correctly. One algorithm that is often used is the error backpropagation learning algorithm [53], which is a supervised learning algorithm.

The algorithm works as follows:

1. Initialization: In this first step, all the connections inside the network are initialized with random numbers inside the range of your activation function. These numbers are the weight of each connection.
2. Activation: The activation step, calculates the differences between the actual output you get from the network and the desired output.
3. Weight update: This step calculates the delta weights for all the connections in the network. It changes the weight on each connection by using a learning rule, for example the hebbian learning rule.
4. Iteration: If the requirements are not met, it goes back to step two. This requirement is often given as a threshold/a given limit. However, if it does manage to fulfill this requirement, the algorithm is done and the network is fully trained.

When it comes to making an ANN for complex problems, the answer to getting good results lays in the ways the designer is able to replicate the brain (more resembling factors of the brain often creates better results) and how well the designer is able to parameterize the problem [21].

3.4 Machine Learning

To be able to learn is one of the key concepts of strong AI. When it comes to artificial intelligent systems, some would argue that it cannot be describes as intelligent if it has no way of developing, reason, or possess the ability to learn. Machine learning is mostly concerned with design and development of learning algorithms and they can often be divided and labeled as one of these three groups; supervised-, reinforced- and unsupervised learning [21]. These algorithms apply for systems which can sense and affect the environment it is in.

3.4.1 Supervised Learning

Supervised learning can be looked at as having a dedicated teacher for the system, at all times explaining the correctness of a systems' output. The teacher is most

commonly a function that will interfere in some way with the system to modify a wrong response into a correct response. It is based on having a premade input-output set in order to predict the correct response for any valid input.

One of the problems of this type of learning is that it is not always possible or effective to make a complete premade set of input-output responses. An example of a supervised learning algorithm is the error backpropagation learning algorithm, which was described in section 3.3.

3.4.2 Reinforced Learning

It isn't always appropriate to use supervised learning. For example when it comes to learning from interactions. In such uncharted territory, the best way to do learning is by using reinforced learning, which means that the system learns from its own experience. Reinforced learning is where the learner gets a set of input values, and learns through evaluating the outcome of its outputs. The algorithm would have to include some sort of a value system which would make it possible to evaluate each reward (the outcome).

This type of trial and error search learning is targeted for situation which becomes infeasible for exact algorithms. With a general value system, one would have a highly adaptable system which can operate well in a dynamic environment. One of the problems is however making such value systems, since every previous action affects the state of the system and the next action.

3.4.3 Unsupervised Learning

The unsupervised learning differs from the previous learning rules in that it doesn't get any feedback from the user or the environment. The way this learning rule is able to learn, is to extract information from the input. It then classifies the information into different patterns, which is done by looking at common or distinctive features of the input information it receives.

Since this algorithm doesn't get any feedback from either the user or the environment, is it not as efficient as the other learning rules.

Robot Development

This chapter is intended to give an overview of the tools involved in our development of the swarm system. We will try to give insight in our choice of approaches and experience gained, by explaining what we have come across from research and testing.

4.1 The E-puck Robot

The robots involved in this project are homogeneous - the same kind. The robot is called e-puck and is a miniature mobile robot developed by EPFL [2] made for educational purpose. In recent years it has also been used in research because it is an open source project with low level access to electronic devices and has unlimited possibilities of extensions.

The reason for choosing to work with the e-puck robots were the obvious reason of availability. The Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU) in Norway, had already invested in nine e-pucks (which is a decent amount when exploring swarm behavior with physical robots) and a simulation tool with integrated e-puck features. Also, the e-puck is known as an easy to use robot with several different sensor and actuator options. The many features it provides would make it excellent in relating the robot to the simple social insects in nature. It has already been documented and used in a wide range of applications (see chapter 2), one of them being this projects research field; bio-inspired robotics.

Features

The e-puck has an integrated dsPIC microcontroller processor with 8kB RAM, 144kB flash memory, and is supported by a custom GCC C compiler. Besides the processor the e-puck is equipped with several sensors and actuators which enables flexible ways to interact and behave with the environment [1]:

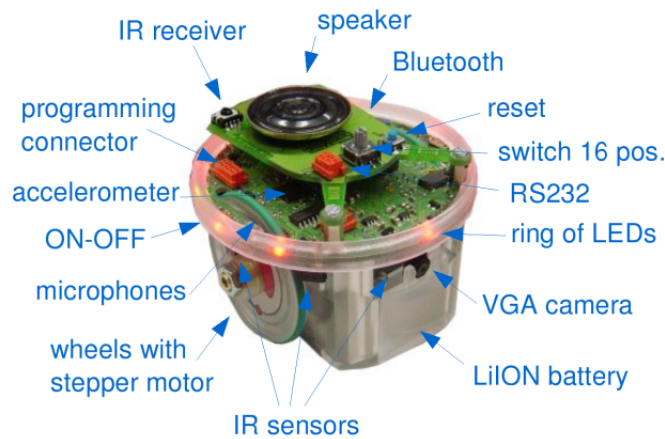


Figure 4.1: The e-puck.

- Eight infrared (IR) sensors placed around the robot which could be used to perceive close obstacles or intensity from IR light.
- A color camera which has a resolution of 640 x 480. Since the robot is equipped with a processor of 8k of RAM, the image rate and the resolution has to be reduced. With a resolution of 40 x 40 and colors, it is documented that it should be able to capture images at 4 frames per second. (8 with grayscale) [2].
- Two motorized wheels that can move forwards and backwards.
- Eight LED lights placed around the top of the robot.
- A speaker which can emit sound.
- Three microphones which can capture and localize sound.
- A 3D accelerometer for detecting position and rotation changes.
- Bluetooth radio link that enables remote control and communication.

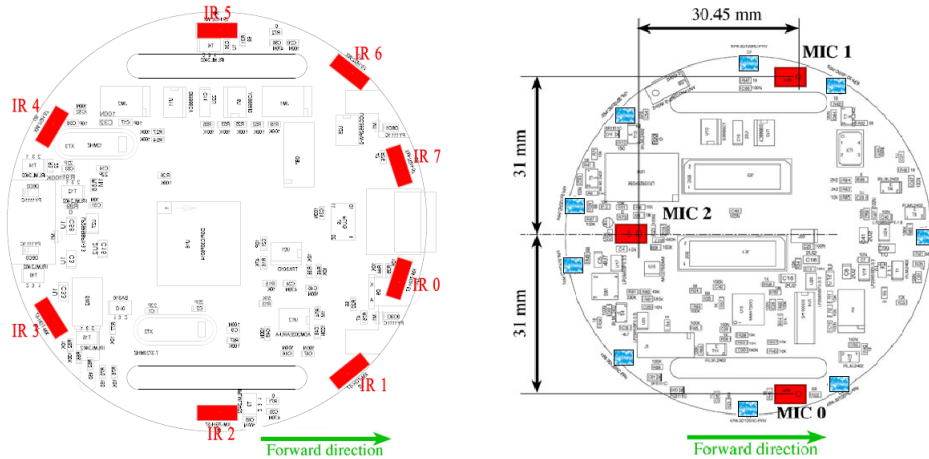


Figure 4.2: The figure to the left shows all the eight IR sensors, which also has the ability to detect distance to obstacles. The figure to the right shows where the three microphones are placed, represented by the red rectangles, and the nine led lights (only one controller for the two on the back equals eight), represented by the blue rectangles.

4.2 Webots

To rapidly design, test, and debug a hardware system, a software simulation tool named Webots was used to simulate the e-puck and the environment. The e-puck community themselves do not provide any type of simulation software, but it is well supported and integrated in Webots, created by Cyberbotics [13].

In a general view, Webots offers flexible and fast prototyping of mobile robots, flexible environment simulations and good precision that will enable you to build a software model which you can later transfer to a real robot. The rapid prototyping also provides a good starting point to brainstorm and test new ideas in robotics.



Figure 4.3: The simulated e-puck and the real e-puck

Webots added flexibility in development of the system. Time was saved when modeling the e-puck as it was already implemented. The high level of precision and detail, gave strength in designing an environment closely related to the physical. Along with simulation testing, it also provided easy to use remote control and cross-compilation options. Webots idea to make a general simulation platform does provide flexibility and many beneficial traits, but along with it limitations. The experience,

both benefits and drawbacks, is discussed and documented in the following sections. Appendix A.1 explains the setup of the projects system with concerns to Webots' remote-control and cross-compilation.

4.3 Environment

The e-pucks world takes place in a physical environment and in the software environment Webots. Much of the prior testing, early experience, was done in Webots, while the end performance of the system was tested on the real physical robots and its environment.

4.3.1 Simulation Environment

As previously described about Webots, it provides flexibility and options. You can give mass distribution to objects, set friction, give bounce parameters etc., all to make the simulation more realistic. One can also set constraints on communication abilities as for example limiting the range of signals. The idea is that one would be able to replicate the physical environment in such detail, that switching from software to hardware requires little or no changes to the system.

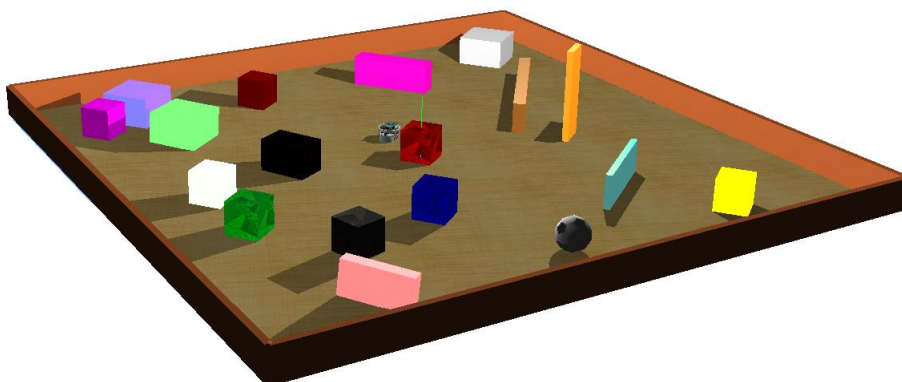


Figure 4.4: The simulation environment

4.3.2 Physical Environment

The physical environment is stationed in the lab and is a 124 x 152 cm board with 10 cm high walls. Beside the static grey ground and walls, we also have small wooden painted objects which can be placed to create new surroundings. Shape, color and light weight gave flexibility in creating different scenarios, which again helped establish good experimental conditions when testing different behaviors separately. Ultimately the objects were removed as they would only act as a random factor, making the box-pushing task unstable and difficult to evaluate.

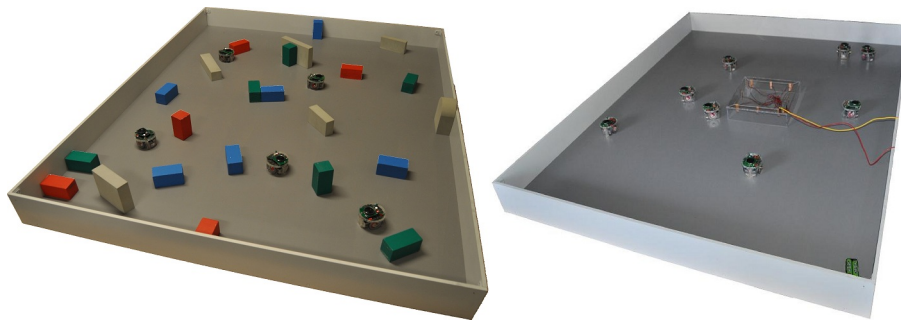


Figure 4.5: The physical environment at the CRAB lab

To give a description of the physical system we first define the e-puck as an agent, which gives us a multiagent environment as there would be several interacting with each other. Since the environment can only be changed by the action of our agents, it can be seen as static. It is also a non-deterministic environment, meaning that a given agent can't predict the next step of the system. The outcome from actions by one agent can affect the environment for others. The environment is only partly observable since it is impossible for any given agent to obtain accurate, complete, up-to-date information about the environment, and it is also continuous. There are only a fixed, finite number of actions that can be performed (discrete).

4.4 Early Experience & Analysis

Previous projects with the e-puck had already given some insight towards the capabilities of the e-puck [6], but some hypothesis still needed to be confirmed. The main focus in the beginning was to get an overview and understand how we could create a general swarm behavior. This section is meant to show the early experience from the first tests, the challenges that appeared and the analysis made to make the next step into design. As a side note when reading this, is that the e-puck and especially Webots are being improved regularly and that changes to the external tool system may have occurred. The e-puck community tends to give out new software libraries whenever a major public project on the e-puck has been developed. The Webots continues to improve their support for e-puck as it is not fully integrated with the system.

Previous Experience

In the spring of 2010 and the specialization project of fall 2010[6] we experienced some of the possibilities and limitations with the e-puck and Webots. As these projects were done in Python, controlling the physical e-pucks had to be done via Bluetooth (only C compiled code can run on the actual e-puck). This remote-control session would prove to be a limitation in itself. Bluetooth worked perfectly with Webots when it came to establishing connection with the e-puck, but it would very easily become overloaded. First of, by using remote control means you are using the computer for computation and not the microchip on the e-puck. The e-puck only executes the information given from the Bluetooth signals. A fair tradeoff for

getting to use the CPU from a PC, would be to expect some delay (in transfer of signals), which also was the case, but it would on top of that make the application crash at times. The delay was first handled by slowing the speed of the whole system down, but running several e-pucks at the same time from one computer would prove to cause delays that couldn't be tolerated by just decreasing the speed. It proved to be a tremendous challenge as when using as many computers as there were e-pucks, created a new situation where Bluetooth encountered interference problems. Bluetooth essentially became a unreliable bottleneck.

The experiments conducted were fairly small with testing the bare essentials for moving and taking pictures. In retrospect it became clear that using Bluetooth and Python was not applicable for swarm creation with e-pucks. It made us do a language and platform change by creating the system in C and cross-compile the code to the real e-pucks. Even though it would probably solve the limitations, more tests were still needed to make sure we could make a swarm system.

4.5 Software to Hardware

Running the system in simulation and in the real world may give two entirely different outcomes since it is difficult for any simulator to represent the real world. Also, moving from software to hardware there will always be differences, and in this case we would have to test the e-puck thoroughly to make sure that what was working in the simulator also worked on the hardware.

The goal of this project is to design and build a system where several physical e-pucks are achieving tasks collectively. For this reason and the time of four months, it was important to use the simulator in cases where it would be efficient. The first step was to verify what were representative in the simulator to the physical e-pucks. We would then be able to use it as a place to quickly test new configurations in control mechanism and behaviors, but also as a design guide for the physical system.

The e-puck community and the Webots designers themselves openly give an overview towards what e-puck sensors that are included in Webots' simulation, remote-control and cross-compilation. The simulator itself includes all features of the e-puck while the remote-control and cross-compilation can't through the normal C compiler make use of the speaker and microphones. As we were to make a system inspired by ants, complex sensing like hearing and actions like speaking is not a necessity to the system. The option of having full hardware access could be done by installing an IDE which included the custom GCC compiler pic30, but it would imply stepping away from Webots which essentially is the place to debug our software. Looking at it from a goal and design perspective, Webots didn't seem to cause any problems concerning the feasibility of the project, but there were of course challenges.

The observation of sensor and actuator differences between the two environments became the major factor to the Webots usage. Comparing the e-puck components measurements in remote-control (physical environment) and in simulation, the e-puck would operate in a more perfect world in simulation. As a consequence the

simulator became a tool for brainstorming new ideas, read sensor values, and debug software implementations.

4.6 E-puck Sensor Testing

The tested conducted in this section is done to give accurate results on how the sensors are operating. To control multiple behavior-based autonomous robots without centralized control or the use of direct communication, it is crucial that we have accurate information on how the e-puck responds. The information gather from the tests would also be of great significance when designing e-puck mechanisms and group behaviors.

The e-puck sensor tests presented is focused on how the sensors operates under different conditions, the differences between each e-puck and the difference from simulator to the physical environment.

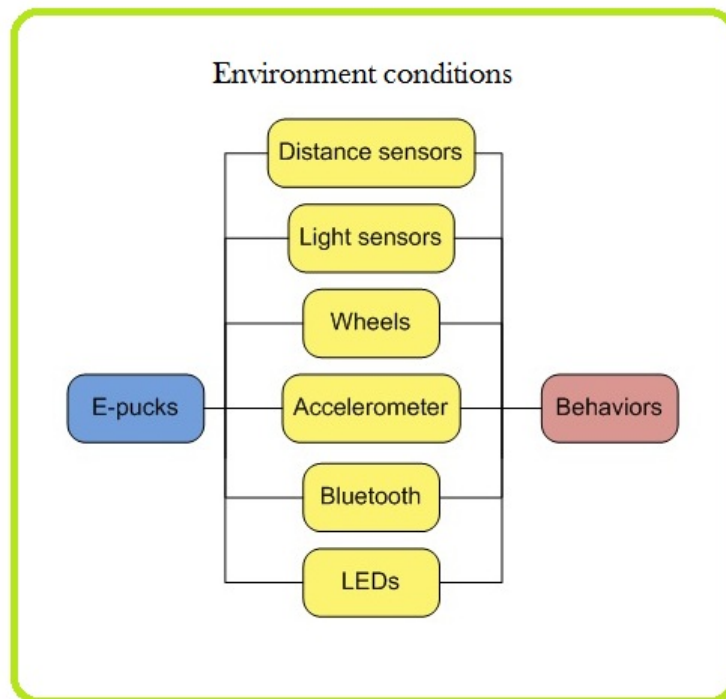


Figure 4.6: Initial testing procedure. Each component figured to be used in the design needed to be tested and evaluated. These components would act as building bricks for e-puck behaviors.

4.6.1 Sensor Overview

The overview of the used sensors is intended to give a short description of how each component operates.

- **Proximity:** As previously mentioned, the e-puck has eight proximity sensors spread around the upper part of its body. Each proximity sensors has an emitter and a receiver. The proximity data is collected by first pulsing an IR beam

to the surroundings and then measuring the difference of received IR light. This process is done several times per second. The values increases exponentially as an obstacle approaches. Since the sensors are placed in a height of 3cm, it is difficult for the e-puck to detect short obstacles.

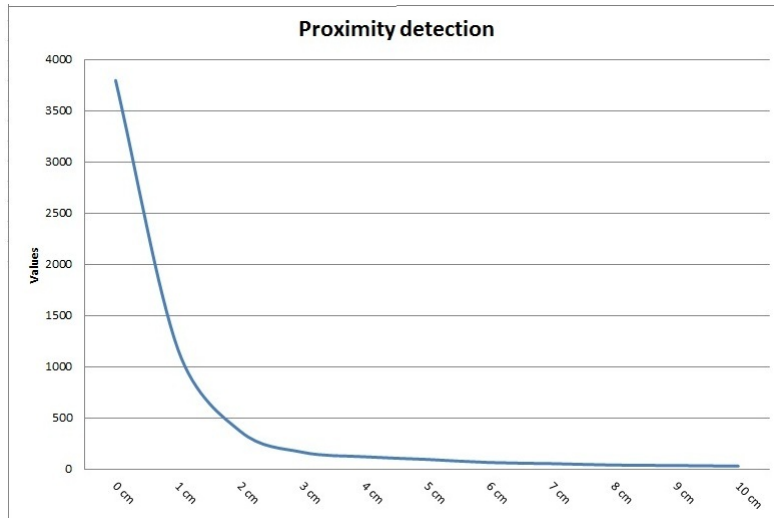


Figure 4.7: The general values for the proximity sensors. With no noise or interference, the values would initially be roughly 0 and increase exponentially to approximately 4000 as obstacles approaches.

- **Light:** The proximity sensors involves both measurements for distance and IR light. In principle, the proximity sensors can be used to detect ambient IR light as well as proximity of obstacles, but the proximity of obstacles values will be disturbed when close to a strong IR emitting source. This measurement is done several times per second, equally to the proximity. The values decrease as the IR intensity measured increases, starting at a value of ~ 4000 when there is no IR.

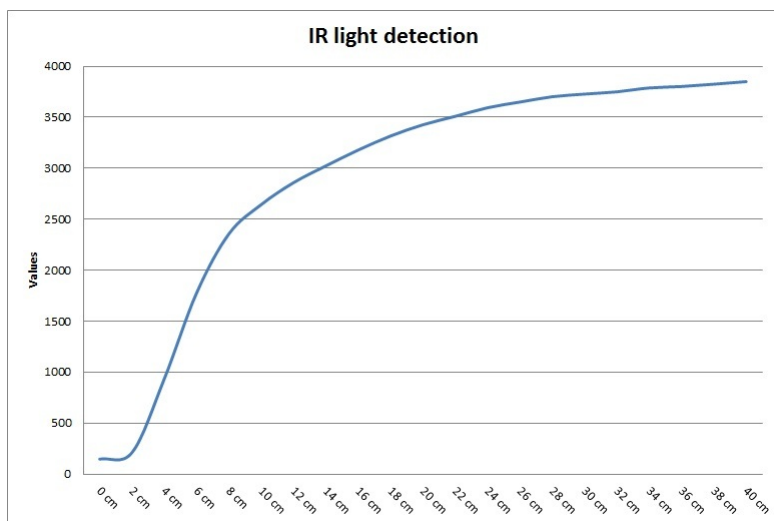
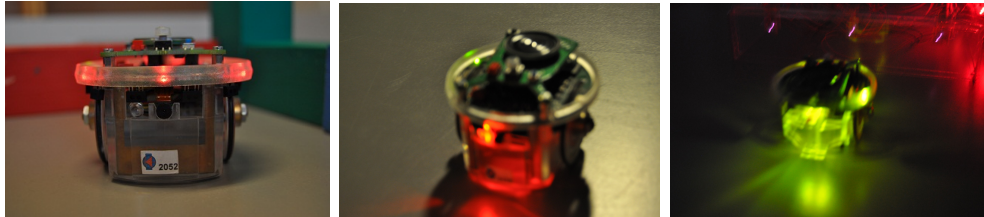


Figure 4.8: General value for IR measurements. These measurements are taken with the IR-diode used in this project. Value decreases as IR intensity increases.

- **LEDs:** In addition to the IR sensors, the e-puck is also equipped with ten LED lights. They are considered to be used for visual feedback to make it easier to see which state the e-puck is in. Eight of the LED lights are placed in the transparent plastic ring on top of the e-puck, figure 4.9(a). The ninth LED light is placed in front of the e-puck, figure 4.9(b). This LED light is also red, but has a more powerful intensity and a longer range, creating a beam. The last LED light has a green color, and is placed inside the e-puck 4.9(c). When this LED light is turned on, it will lighten up the whole e-pucks lower body with a light green color.



(a) There are eight small led lights around the top. (b) One red LED with longer range. (c) Inside the e-puck there is one green LED.

Figure 4.9: All the different LED lights

4.6.2 Proximity Sensor Testing

To navigate and move around without colliding with other e-pucks or obstacles, the e-puck needs to sense the environment. Just like the poor vision and short perception of most ants, the proximity sensors of the e-puck gives continuous distance information within short range. Since light from sources such as lamps, sun, etc. all emit IR to some degree, it is important to determine what impact it has on the e-pucks' sensors. The result data is also a necessity for designing an avoidance behavior.

Setup

There were two main experiments conducted on the proximity sensors to see how and if they worked:

- The sensor values during dark conditions with close to no interference from external factors.
- The sensor values when there is interference from external factors.

In both cases the e-puck would stand at the same open spot with no objects in range for it to detect. As in nature, no creature is identical, so it is necessary to perform the test on each e-puck and on each of the eight sensors. The sensor values were gathered by running the tests through remote-control.

Results

The first tests were performed at night in a room with close to no light and hence no external interference. This would show differences among e-pucks and the sensitivity of each sensor.

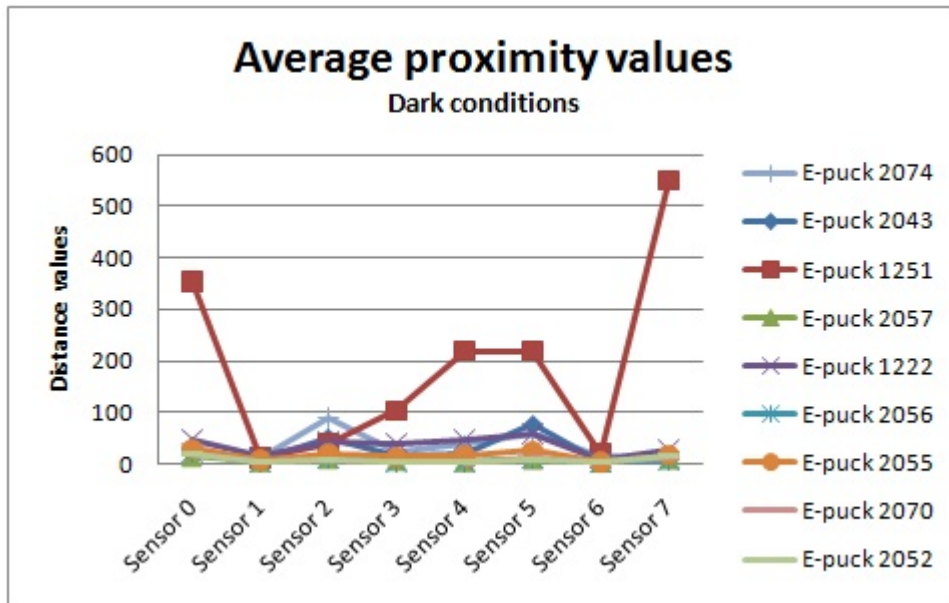


Figure 4.10: Result of each initial sensor value in dark conditions. The e-pucks are separated by their ID (i.e. 2074), and have a unique line. The graphs show average value of each sensor for 20 time steps (20 values).

The same experiment were conducted in the middle of the day, where both lamps and the sun had influence on the result. This would show the impact of proximity detection during bright conditions.

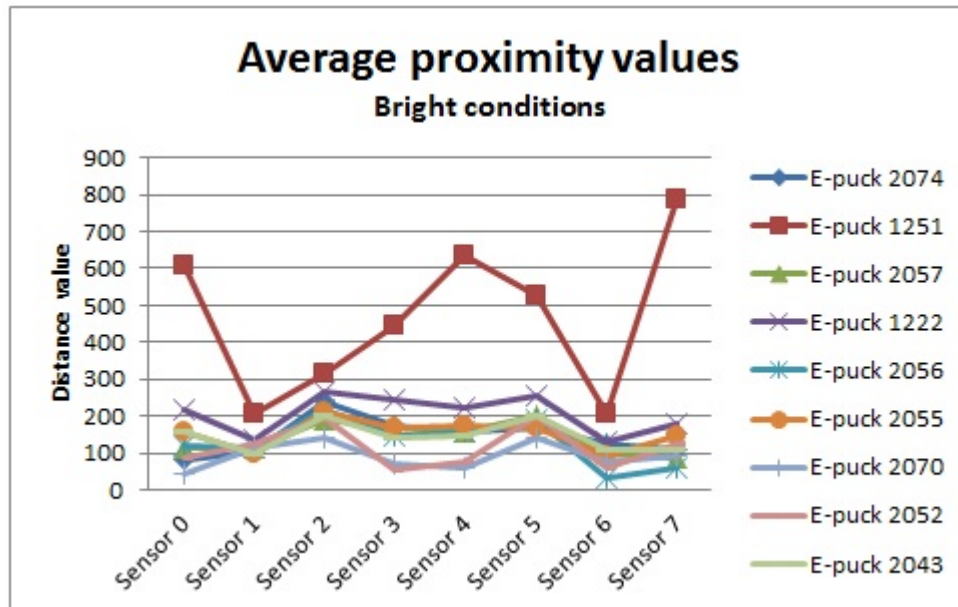


Figure 4.11: Result of each initial sensor value in bright conditions. In general, this bright environment affected the e-pucks with noise and gave a higher variances in the result compared to dark conditions.

Analysis

The tests in both dark and bright conditions showed good results and contained valuable information for the system design. All eight sensors have close to the same value (considering the range of 0-4000) when encountering the same environment and conditions. The differences between each e-puck were also relatively low, which increased the idea of having all e-pucks compiling the same configured system. The small differences are only biological realistic as differences between ants do exist.

E-puck 1251 did for none-environment reason give higher proximity values than the rest. This was the case for all eight sensors. The values between each sensor had a high variance (averaging from 200-800), which was the major factor for removing the e-puck from the system. The overall result was considered not applicable to the swarm system, as it would respond differently depending on sensors. E-puck 1251 was removed from further testing.

Bright versus dark enlightenment showed to what extent the environmental condition affected the sensors. Excluding 1251, having bright conditions exposes the e-puck to ambient IR which is picked up as noise, making values range from ~ 40 -260. The dark condition on the other hand showed to give more accurate information, only ranging from ~ 10 -100. It could be argued that extracting these initial average values from each sensor (removing the noise) would make the e-pucks respond equally in both conditions; however, the less variance in dark conditions displays more precise data and henceforth becomes the most optimal choice for detecting obstacles. Further tests in IR would display much of the same traits.

4.6.3 IR Light Sensor Testing

Collecting food requires the ability to locate the source and retrieve it from one location to another. The e-puck lack sensor units for detecting odors, like how ants detect pheromones, but it do however possess the ability to separate between objects that do and don't emit IR frequency light. This separation method can be used similar to how ants locate food, by creating an artificial food source which emits IR. Using the same proximity sensors, this experiment tests the e-pucks capabilities of detecting IR lights.

Setup

The tests were conducted on all eight sensors for all remaining e-pucks. The experiment would be useful when designing the artificial food source and for the design of behaviors (locating, pushing, realigning etc.) The main tests were:

- The initial state. Difference when operating with interference and without.
- Awareness state. The max range of detection.
- Pushing state. The values when standing close to an emitting IR diode.

The configuration of the IR diodes can be found in section 5.2. All tests were performed with one e-puck at a time in remote-control.

Initial State Results - Bright Conditions

With the goal of having the same interference as the proximity test, bright conditions involve having a lit up environment from the sun and lamps. This test did not rotate each sensor to the same location, but read as it was placed. Each e-puck had the same position. This would give a clear indication to the sun's impact on the sensors, as well as any malfunctioning hardware.

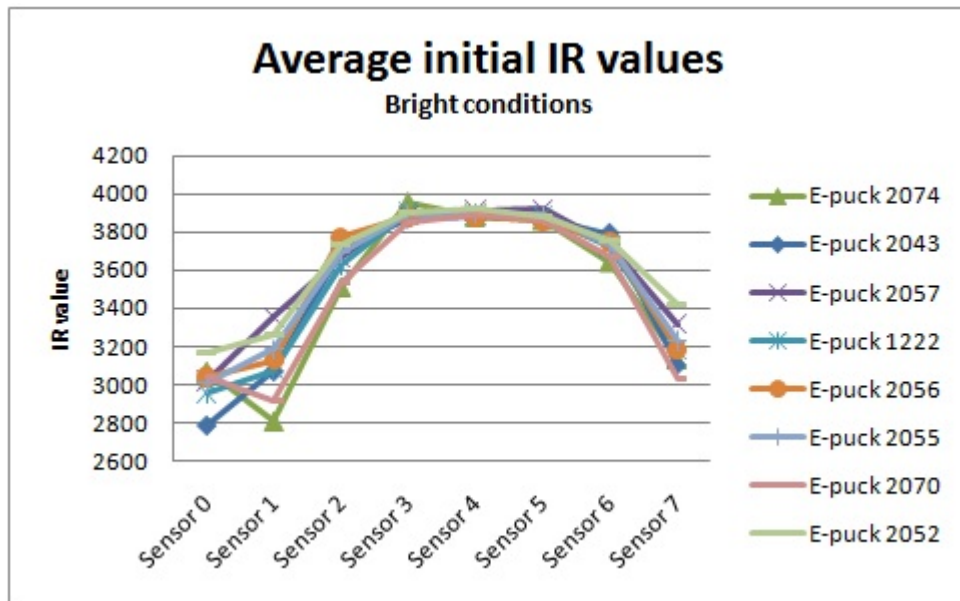


Figure 4.12: Average initial IR values in bright conditions. Sensors 0, 1 and 7 were faced towards the sun which shows it had a bigger impact than the ambient light from lamps.

Initial State Results - Dark Conditions

This test had the same setup as the bright condition experiment, but with no external interference. This would show differences in bright to dark conditions and differences between e-pucks.

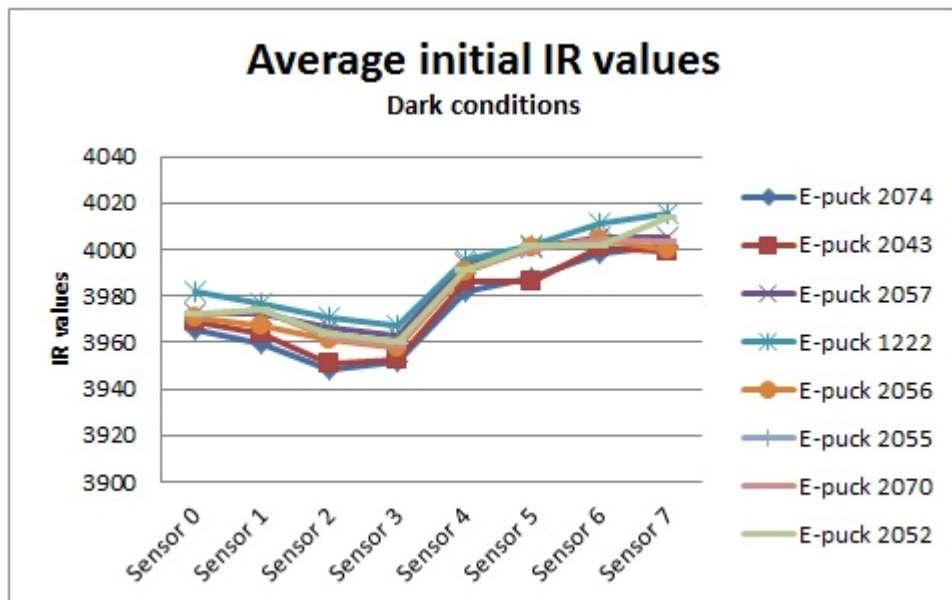


Figure 4.13: Average initial IR values in dark conditions. Compared to bright conditions the results were consistent and had a low variance.

Condition Analysis

The e-puck had already shown signs of being affected by the ambient IR that occurs during daytime. The IR detection differences from bright to dark show much of the same traits. There are definitely much more noise on the sensors during bright conditions. The values shows a clear distinction between dark and bright, as the overall values are generally lower (detecting IR frequency light) during the day and more precise in a room with close to no light. This difference can't be handle in the same way as the proximity detection (see section 4.6.2), with removing the average noise, since the sun has a major impact at certain angles. Therefore, the system should operate under no sunlight and less to no light to get accurate input. Further tests were only conducted in dark conditions.

Awareness State Results

Running the system in remote-control gives the ability to read sensor values on the fly, or step one single time step at a time. The process of finding the maximum range of IR awareness therefore became based on a try and fail procedure. Having the emitting source stationed, the e-puck was manually moved back and forth to a range where each sensor could, with a clear margin, be separated from the initial values. Even though the detection could be seen in most cases at a range of 34 cm, the range of 30cm had a clear differential margin from the initial values.

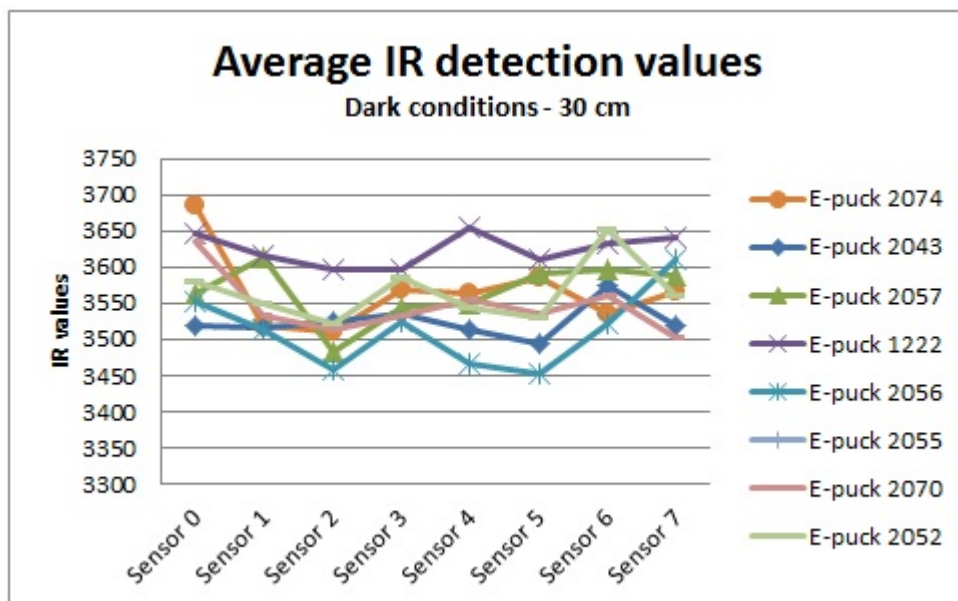


Figure 4.14: Average IR values when the sensor is 30 cm from the IR diode. All e-pucks shows a clear margin of difference from the initial value of 4000.

Pushing State Results

This experiment was conducted to obtain the IR sensor values when the e-puck would stand close to an emitting source. Each e-puck was rotated accordingly to the sensor being monitored in order to beam each with the same intensity. The results would indicate a threshold for when a pushing behavior could be triggered.

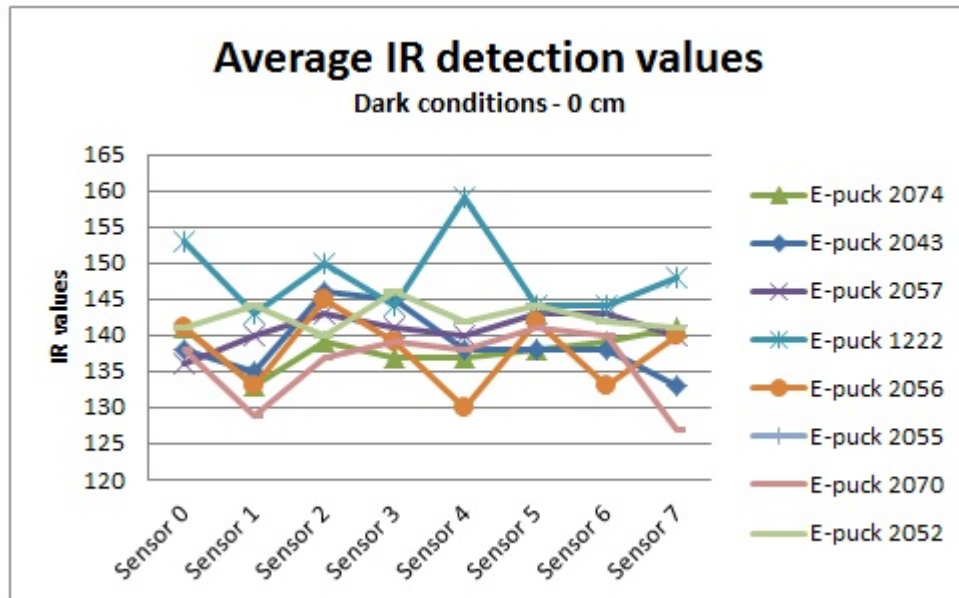


Figure 4.15: Average IR values when the sensor is right in front of the IR diode

Result Analysis

The results are showing the average values from 20 time steps (sensors are being read each time step.) The general result from all of these tests is that dark conditions provide accurate values (low variance). Each e-puck is responding with same values to IR intensity, which means we don't need to design a software with separate configurations for every single e-puck. The awareness tests gave a high margin difference from the initial values already at 30cm. This tells us with confidence that at a value of 3700, each e-puck is detecting IR. The pushing results on the other hand provides us with values that can let the e-puck know of far or close it is to a IR diode.

4.7 Conclusion

This chapter has presented information on the e-puck robot, the simulation tool Webots and the testing environment. Experience from earlier projects has been discussed and sensor testing analyzed. It has revealed some of the design challenges, but also insight to how we should proceed.

The e-puck together with Webots has shown to give some hardware limitations, but access to the hardware devices we feel represents mechanisms in ants. The possibilities of a potential swarm system has been found, which relies on eight robots that can move, detect obstacles and detect IR lights.

Artificial Food Source

The ants in the nature use chemical signals, called pheromones, to indirectly communicate with each other. These signals can among other things be used as a guidance path to food sources. This concept was a big inspiration for the project. By constructing an object with a form of chemical signal attached to it, it would be possible to create an artificial food source. In this case, a transparent square box with emitting IR lights.

5.1 Designing the Box's Frame

In nature, food sources varies in both size, shape and weight. Ants have several ways to retrieve these items, like to drag, carry or push. The e-puck on the other hand has very few options when it comes to interacting with the environment. With no possibility to grab or pick up an item, the retrieving part had to be done by using the wheels and its body to apply a pushing motion. To easily stay aligned with the object, the item was designed as a square box.

The e-puck at full speed apply force varied by friction of the surface it is driving on. The e-puck itself is constructed to be light and the wheels designed to spin when colliding with heavy object in order to not overload the stepper motors. The development of the box became very much affected by these observations. The e-puck wouldn't be able to handle much weight, in fact not more than 150g of the material we decided to use on the physical surface.

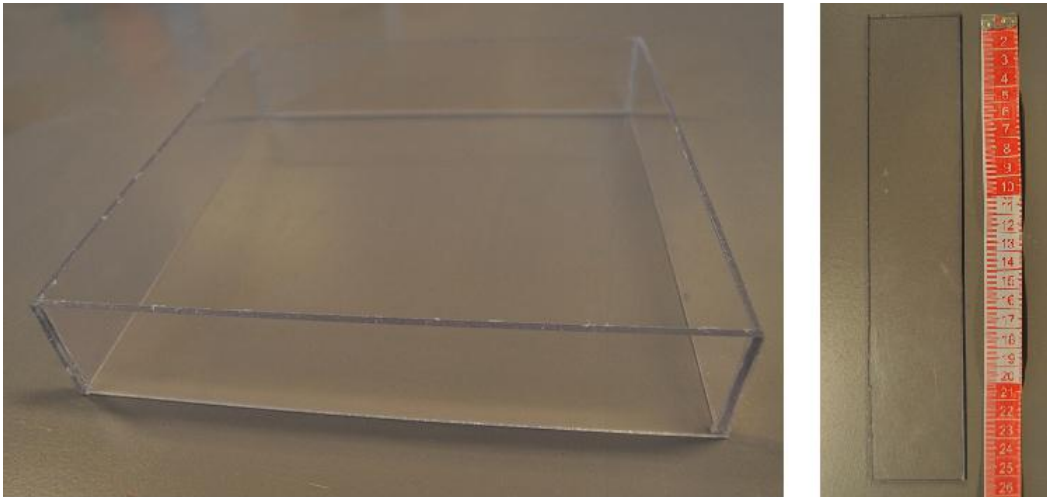
To keep the box as light as possible and having to emit IR lights, the material chosen was solid plastic. It would be solid, which wouldn't deform when force was applied. It would also be transparent which meant light could pass through. Measurements showed that 1cm^2 of solid plastic had a weight of 0.346g.

Based on calculations and result shown in table 5.1 it would be optimal to have length size with enough room for at least three e-pucks, which would be able to handle weight up to approximately 450g. Measurements of the e-pucks height implied that the box height should be at a minimum of 5 cm. By setting up three e-pucks beside each other, it became clear that it would be necessary to have a box longer than 20cm. Each e-puck has a width of 7cm and with the importance to have

Table 5.1: Dimensions and weights of the plastic box's frame

Length	Height	Weight sides
30cm	6cm	249,12g
	5cm	207,6g
	4cm	166,08g
25cm	6cm	207,6g
	5cm	173g
	4cm	138,4g
20cm	6cm	166,08g
	5cm	138,4g
	4cm	110,72g

some space to realign, a length of 25cm would give a decent space margin (see figure 5.1).

**Figure 5.1:** The first step towards an artificial food source

5.2 IR Emitter

The choice of using IR emitters inside the box as enticements for the e-pucks is a result of studying the e-pucks and their functionalities. In order for the e-pucks to manage aligning correctly and to push the box as a swarm, the design involved having three equal IR emitters attached to each side of the box. The IR emitter is a SFH 485 P diode[42] and has a maximum current of 100mA and a maximum voltage of 3V. To test the IR emitters and how it would work with the e-puck, a simple circuit with one IR emitter, a battery which produced 9V, and a resistance of 60Ω was created, figure 5.2. The resistance was calculated using Ohm's law, equation 5.1:

$$Resistance = \frac{Voltage}{Current} = \frac{(9V - 3V)}{100mA} = 60\Omega \quad (5.1)$$

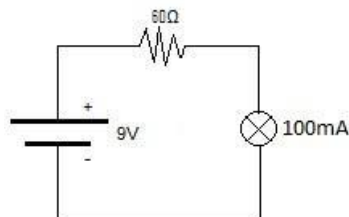


Figure 5.2: The first test circuit with one IR emitter, a battery of 9V, and a resistor of 60 ohm.

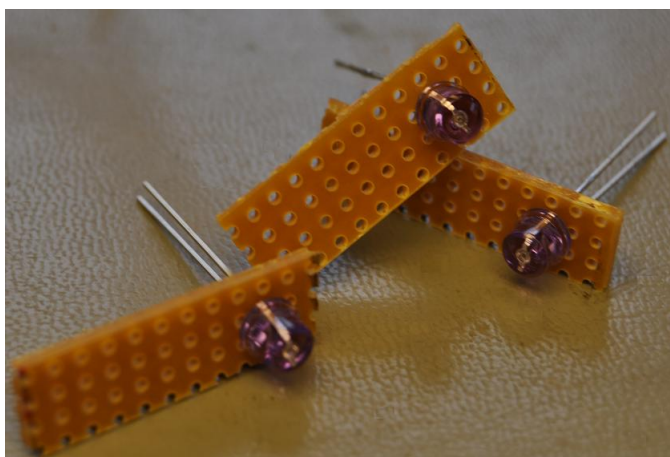


Figure 5.3: The IR emitter used in this project

5.3 Circuit

The choice of implementing three IR emitters on each side, twelve as a whole, gave challenging circuit design. The circuit, consisting of battery, wires, resistors, jumpers, IR and boards, couldn't be too heavy. Also, the circuit had to withstand movement and deal with current overload. The solution became creating a circuit where all the twelve emitters were connected together in parallel from a small motherboard. The battery was too heavy and had to be separated from the box, which was done by attaching long wires. Figure 5.4 shows an illustrated version of the circuit. By having all the IR emitters connected in parallel, it was possible to prevent any overload if an emitter were to fail. Based on the capacity of the battery and the inability to charge, the 9V battery was replaced by smaller rechargeable batteries with lower capacity (1,4V each). Combining four of these new batteries in a series, the circuit would have a capacity of 5,6V, equation 5.2. According to Ohm's law each emitter needed a resistor of 27Ω, equation 5.8

Voltage in series:

$$V_{total} = V_1 + V_2 + V_3 + \dots = 1,4V + 1,4V + 1,4V + 1,4V = 5,6V \quad (5.2)$$

Current in parallel:

$$I_{total} = I_1 + I_2 + I_3 + \dots = 100mA * 12 = 1200mA \quad (5.3)$$

Resistance in parallel:

$$R_{total} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \dots} \quad (5.4)$$

$$Resistance = \frac{Voltage}{Current} = \frac{5,6V - 3V}{1200mA} = 2,16667\Omega \quad (5.5)$$

$$R_1 = R_2 = R_3 = \dots = R_{12} \quad (5.6)$$

$$2,16667\Omega = \frac{1}{\frac{12}{R_1}} \quad (5.7)$$

$$R_1 = \frac{1}{\frac{12}{2,16667\Omega}} = 26\Omega \approx 27\Omega \quad (5.8)$$

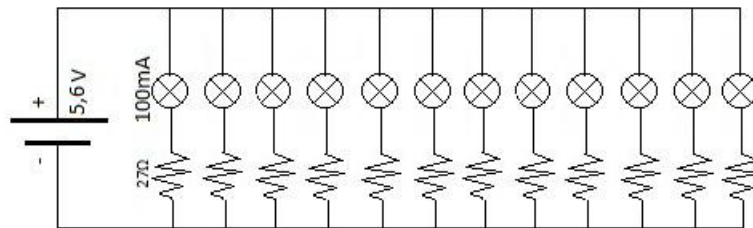


Figure 5.4: The complete circuit design with 12 IR emitters, 4 batteries of each 1,4V, and 12 resistors of 27ohm.

The prototype board used to solder the components (see figure 5.5) was originally 16x10cm. However, since the circuit design involved 12 IR emitters being spread equally around the box and a resistor attached to each of them, the board was divided into small 1x3cm pieces. These boards were connected together by wires.

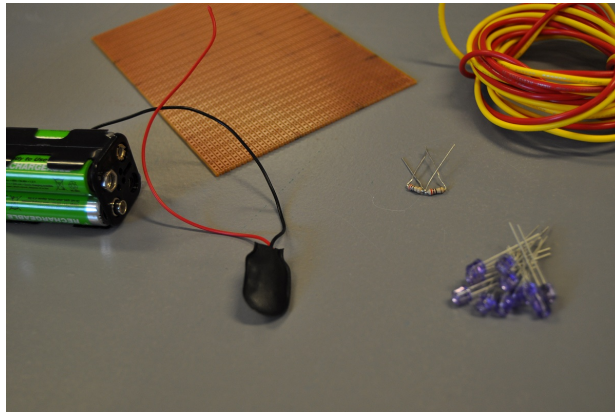


Figure 5.5: The components used to build the circuit; Battery, prototype board, resistors, wires, emitters.

5.4 Construction

To easily split power to each of the 12 IR emitters, a motherboard was made, see figure 5.6. The motherboard has 14 jumpers and 12 resistors. Two jumpers were used as connection to the battery, and the rest as separately powering one emitter each.

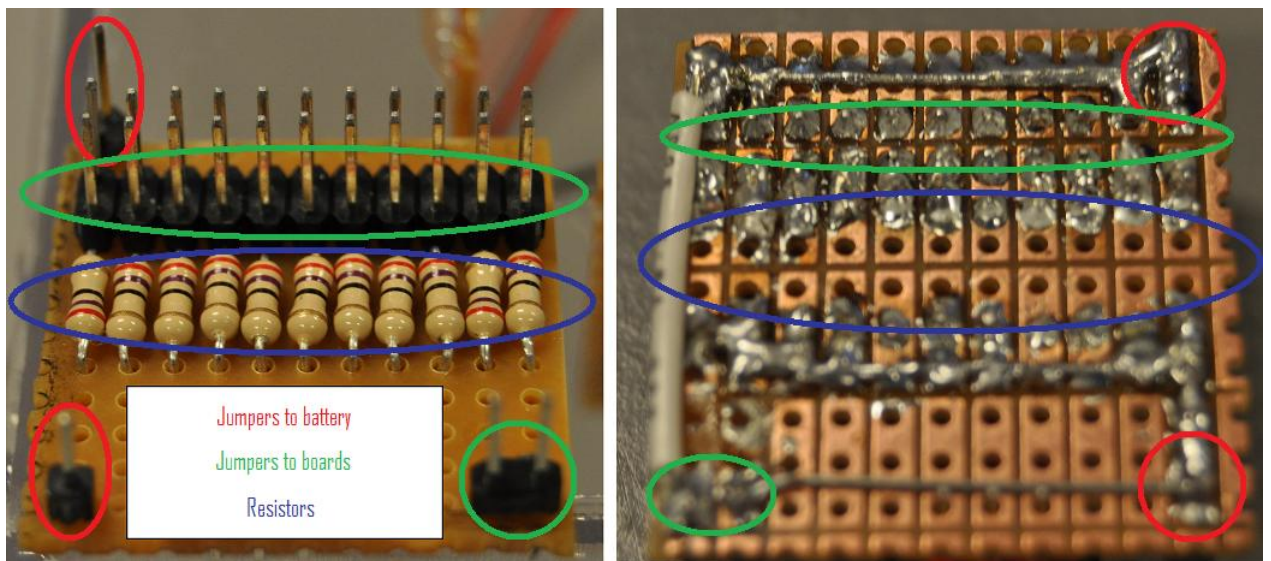


Figure 5.6: The motherboard

Attaching the boards to the box did add some work. The emitters couldn't be placed directly on the side of the box as the angle could be too narrowed and hit between two e-puck sensors when pushed. Without adding too much weight, we added some small solid plastic lists along the top of the box, and then glued the boards on in an upright position (figure 5.7). The lists are 25x2,5cm and added a total weight of 79,58g.

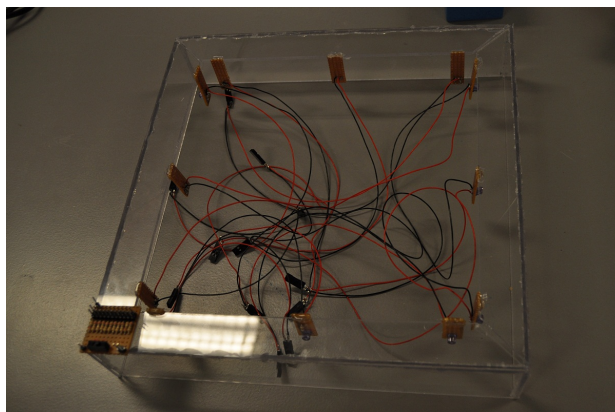


Figure 5.7: The artificial food source - the square IR light emitting box

After all components were attached, the box had a total weight of 324g. A quick test confirmed our prior testing and design. Three e-pucks pushing in the same direction could easily move the box from one side to the other. Two e-pucks with the right alignment only made initial small movements at impact, while one e-puck didn't move it at all.

Design & Implementation

This chapter describes the design process of making the e-puck system CRABS and environment. For further details on how to setup and run such a system see the appendix A.1

6.1 CRABS

Collective Robotic Autonomous Behavior-based System (CRABS) is our software system developed to explore cooperative transport of prey/food with e-pucks. CRABS is inspired by cooperative behaviors in social insect, with a main focus on making a model similar to the collectively retrieving system in ant colonies.

Ants of various species have been studied and reported to collectively transport and retrieve prey and larger items of food. This behavior arises as a prey or food source is too heavy for one single ant to retrieve. A basic scenario would be an ant which locates a food source, but finds it impossible to move. Either through direct contact or trail laying of pheromones, the ant will try to recruit nest mates to come participate. If nothing seems to work for a certain time, specialized workers are sent in to cut the item into smaller pieces.

During research it became clear that this ant approach to problems of cooperative behavior has been an inspiration for many systems (see chapter 2), but they most often contain features such as direct communication, global planning and memory. These are abilities beyond the studied ant model. Kube and Zhang [34], Stilwell and Bay's [51], have however been more strict in their ant-based approach and with CRABS we pursue an even more biological plausible approach with hardware.

The e-pucks problem domain revolves around swarming an item and collectively retrieving it back to the hive - box pushing. The task at hand could be describe as basic and not the most exciting problem to solve, but with focus on making a closely inspired biological approach with e-pucks, it will provide further insight to several challenges of swarm robotic systems:

- **The design of collective problems-solving robotic systems;** There is no formal description of the biological food retrieval by ants, and the underlying cooperative mechanism to when and how it is done remain seemingly unclear.

It is however well studied on a higher level, which do create guidelines and restrictions to the event. Section 6.2 portraits these challenges in detail and show how we are able to replicate ant behaviors in e-pucks (staying within the ant model).

- **Performance and capabilities of real physical robots;** Robotics projects have a clear tendency to end after simulation stage. Proof of concept seems often validated without having to run any form of physical tests. The CRABS will be tested on physical e-pucks in a physical environment.
- **Swarm relations with e-pucks;** The robots in physical robotics projects are often made for that particular experiment, with designated sensors designed for the specified task. E-pucks are as mentioned in section 4.1, made for educational purposes and does not include specialized features for swarm behaviors.
- **The systems simplicity;** Characteristic challenge is being able to make a robust and flexible system from simple behaviors.

6.1.1 The Box-Pushing Task

The box-pushing task is pretty straight forward. The e-pucks have to locate the artificial food source, converge, and collectively retrieve it back to the hive. In this case, the "hive" is any of the four walls surrounding the environment. One single e-puck can't complete the task alone, and two will struggle, which leads to cooperation of minimum three e-pucks to reach the goal. Configuration of the environment will always initially start out with the artificial food source in the middle with e-pucks randomly distributed in the environment. Figure 6.1 illustrates one case of the initial environment design of the event.

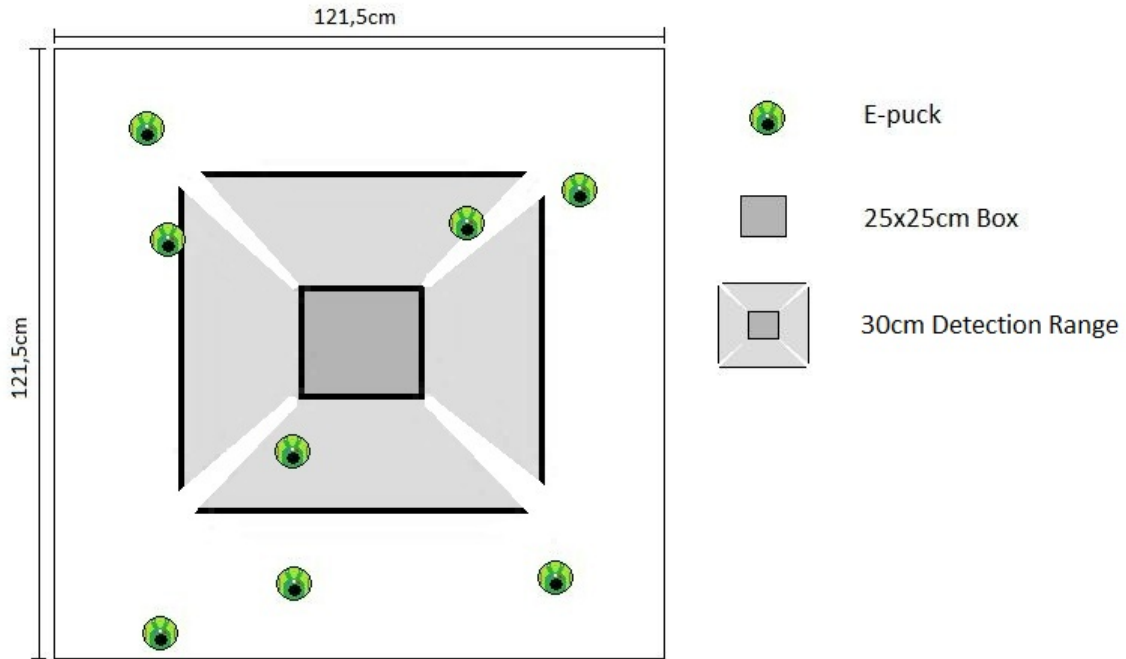


Figure 6.1: Illustration of the environmental design. The grey square in the middle represents the artificial food source (box), and the lighter areas around shows the range for e-puck detection. The whole environment is quadric, making it equally difficult to reach the goal.

To elaborate in more detail on the experience we seek from exploring swarm intelligence with box-pushing, we highlight the following:

- We intend to investigate cooperative behavior on a physical level, replicating the event to the extent of our knowledge. The strategy is to create a decentralized system invoking group behavior through simple mechanisms which, if successful, leads to an emergent self-organized system - a global structure. This is to be achieved through the guidelines of the ant retrieving model, with no direct communication, no internal memory, no control mechanism, and with only the use of two types of sensors (proximity and IR) and two actuators (left and right wheel). The system will be experimented, measured and validated in a physical dynamic environment, on eight e-pucks.

6.1.2 System Overview

The swarm system can be stripped into two main parts - software and hardware. This section explores the challenges for each part in detail. Some of the hardware design and challenges has already been revealed and discussed in section 4.6 and the focus in this chapter will be on swarm feasibility. The software will examine the design to controlling multiple behavior-based autonomous robots.

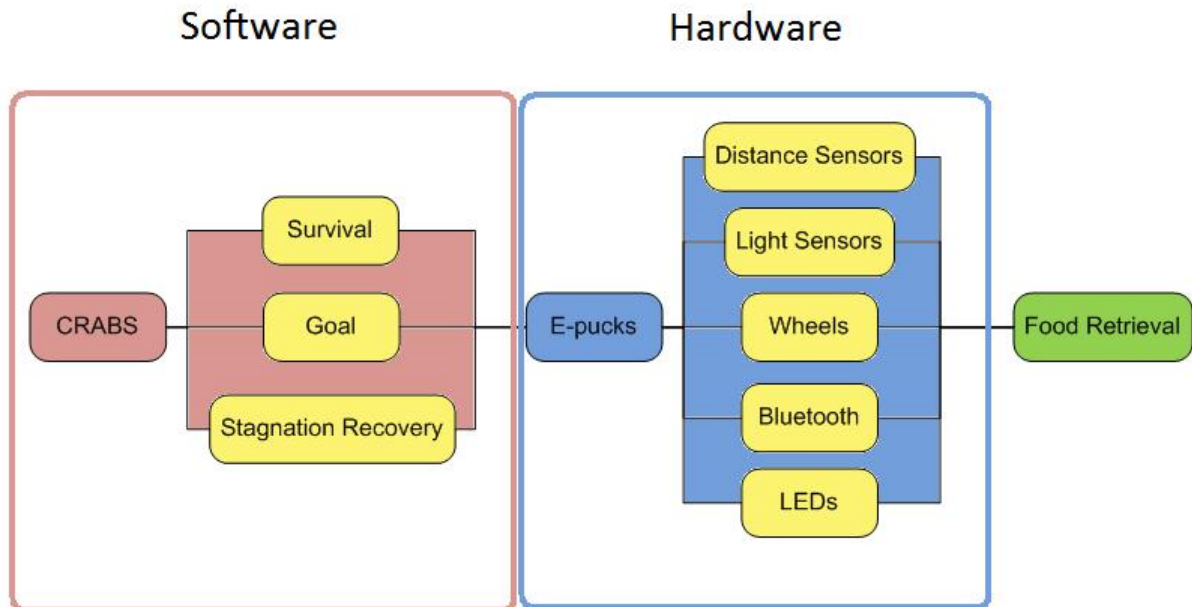


Figure 6.2: System overview. This illustrates the main features used in the software and hardware part of CRABS. Together they form a collective transportation of food.

Software Overview

The approach for controlling multiple autonomous robots relies heavily on the research made about e-pucks, similar robotics projects and ant observation. There lacks a common theory on the behavioral biology that explains the collectiveness in social insects. However, roboticists have gone further than biologist trying to model cooperative transport, and especially Kube and Zhang [34] have influenced our work.

The behavioral design pattern we felt represented ants, was the one presented by Moser [44]. He explains an ant's action as a result from the input of the receptor being most stimulated. The response could be triggered by pressure, sound, light, heat and chemicals. If not confirmed, it can surely be observed in Wilson's odor experiments [54].

- Ants carry their dead nest mates to refuse piles in order to keep their nest clean and Wilson wanted to experiment on stimulating ants to invoke this corpse clustering behavior. He first experimented with putting acetone extracts from dead corpses onto small pieces of paper. The experiment was successful as the papers were picked up and transported to the pile. Better yet, he started to experiment on putting acetone extract on live ants, which also invoked that very same behavior. The now smelly live ants were picked up and dumped in the refuse pile by nest mates.

Our architecture and behaviors are inspired by these observations.

Architecture

When designing the robot control system, we looked at architectures which seemed similar to the simplicity of ants and still able to arbitrate on conflicting commands. It became clear that Brooks' subsumption network, which is commonly used in robotic projects, was most fitting. Other architectures, like the Touring Machine, section 3.2.2, are based on too complex factors. It would seem highly unlikely that ants would have a reasoning system (a separate control system), that enabled them to plan or even model their actions. From the observation already presented, it would seem more correct to build a fixed priority scheme, where the highest stimulated behavior is the one acted upon. This implies a design with prior decision making on the behavior arbitration, which is presented in more detail in the behavior-based system section 3.1.4.

Collective Behaviors

As shown in figure 6.2, the behaviors designed to achieve the box-pushing task are divided in three modules. The implementation details is described in Section 6.2.

Survival is the module containing behaviors on a proactive level. Even though ants seems to work all the time, they do rest, but what stimulates them to wake up and work? In our design it becomes a very low form of reacting to the environment, which is why the term proactive is more appropriate. The e-pucks need food to survive, so they are constantly looking for food to renew their energy, when nothing else stimulates them to do otherwise. This search for food needs to be accommodated with an avoidance behavior to avoid obstacles and avoid becoming stuck.

Goal is a module made for behaviors invoking group behavior, which essentially emerges from robots sharing the same goal. The mechanisms are designed to trigger as a food source is detected. It needs to converge and align with the item before starting a push behavior to retrieve it.

Stagnation Recovery is a module made for behaviors which simply keeps groups behavior progressing. At some part of the retrieving event, the item's motion may stop and progress yield as ant forces are applied from opposite directions which cancel one another. This can also occur when the group encounters obstacles. Whatever the case, e-pucks have to solve it the same way ants do it - by spatial rearrangements. Through continuous ways of realigning and repositioning, the ants continue their progress towards the hive.

Hardware Overview

The research and sensor testing has been discussed in chapter 4. This section looks at the approach and challenges of the box-pushing task.

The sensing requirements to achieve the goal of the software modules above is divided into three abilities; the ability to sense the box, sense other objects (including other e-pucks as well), and sense task progression. Another aspect of these requirements

is that they have to follow restrictions, as towards staying true to the ant model. There is no leader, no global planning, no memory, and no direct communication. To make the e-puck system become a good food retrieving model, we went back to the observed ant model.

Ants can get directional information and navigates using the sun. They can even navigate on overcast days, as ants can sense ultraviolet light. The IR detection sensors on e-pucks resemble this ability and the design became to use them to detect and navigate to the box. This implies that the box emits IR lights.

Although ants don't have a form of memory, it do make use of feedback which share much of the same traits. Not incorporating reasoning into the e-pucks, but to get the right sensor stimulated from the environment, we would have to rely on negative and positive feedback. This would enable the ability to sense task progression.

Another trait the design would make use of, was stigmergy. Even though this e-puck setup lacks some sort of force sensor, the intensity of the lights emitting from the box could be used to tell which way the box is moving. E-pucks could almost like with pheromones, indirectly communicate through sensing neighbors. "E-pucks around me, tells me (and the ones around me), that we are working on making the food source move in the same direction."

Everything is based on local information, and each step acts only on the behavior being triggered this exact moment.

6.2 Code Structure

This section will describe the development of the software system CRABS and the underlying mechanism in each behavior in detail. The whole system is made in the programming language C. It is made with focus on building the architecture of a subsumption network and being able to add or subtract behaviors for testing purposes. The system is realized through CRABS and the use of the software simulation tool Webots.

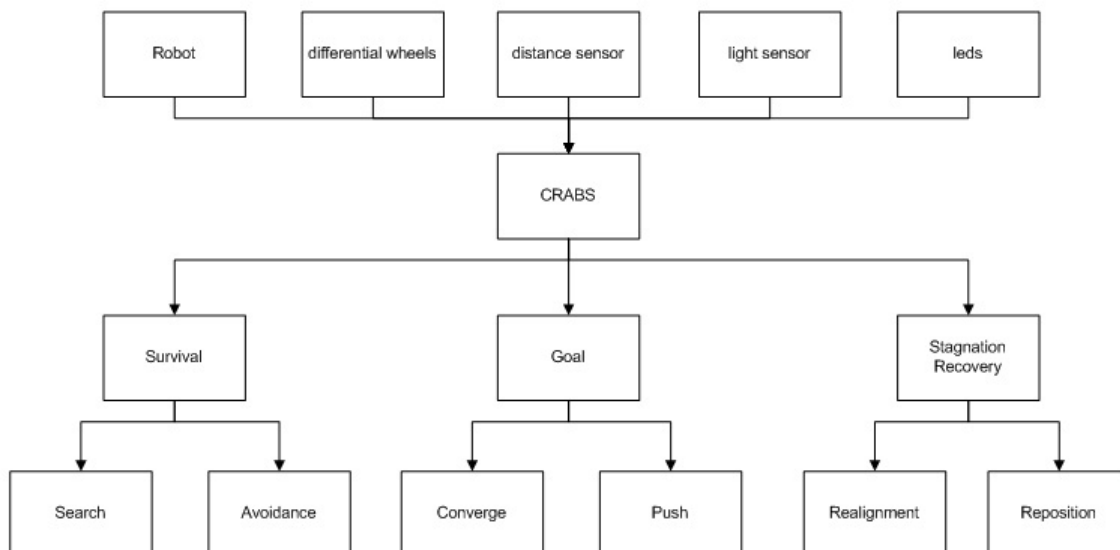


Figure 6.3: CRABS code structure. Using pre-defined headers files from Webots to get access to sensors and actuators, behaviors made as modules accessed through a subsumption network in the main file CRABS.

6.2.1 Accessing E-puck

As shown in the figure 6.3, access to sensor and actuators on the e-puck is gathered from the header files and the e-puck library from Webots. The robot file includes initialization of a robot event, while the remaining top files give access to the hardware related to their filename (differential_wheels gives access to the right and left wheel and so forth).

The overall design is like a two-way dialog between the e-pucks brain(CRABS) and its body. The body reactively sends the newest update about the environment with a question mark at the end, and the brain responds with an answer. As illustrated in figure 6.4, one time step consist of a sense and an act part. The e-puck sensory information is collected and process by CRABS, before it sends a command back to the actuators. The action of an e-puck continues to be preformed until a new command presents itself next time step. One time step is set to take 64 milliseconds.

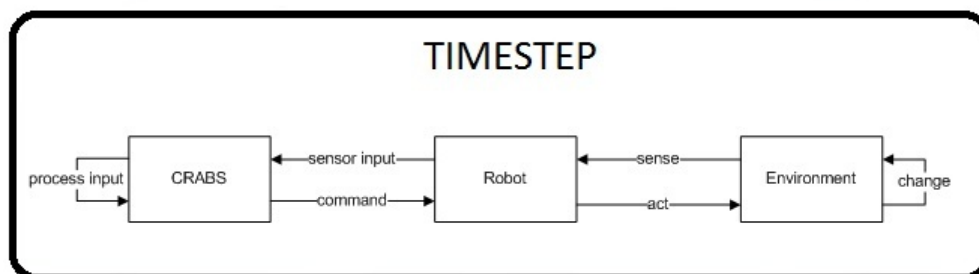


Figure 6.4: This shows the processes occurring during each time step. Depending on the sensor input, a behavior will trigger in CRABS which outputs a command regarding the actuators.

6.2.2 Search & Avoidance

The survival modual contains two behaviors, search and avoidance, where both operates on the input from the distance sensors. They are closely related and complement each other, but they do function individually. However, the survival design does not account for extracting one of the behaviors, as search would collide and possibly get stuck, and the system as a whole would not progress.

Movement

In the previous project [6] with e-pucks, we experimented on creating movement and avoidance from an artificial neural network 3.3. This was programmed in Python and mainly tested in simulation. The successfulness of these experiments led us to reuse much of the results and setup, but without the process of a neural network. With focus on swarm intelligence, we rather designed a fixed input-output combination.

The search and avoidance behavior is based on the four front distance sensors of the e-puck. The sensor input value is translated to be either free passage or an obstacle (0 or 1), which gives 16 different input combinations. Given the results seen in chapter 4, figure 4.10 and the actual distance in figure 4.7, e-pucks distance threshold to obstacles was set to a value of 250. This accounts for any measured noise that may occur during runtime.

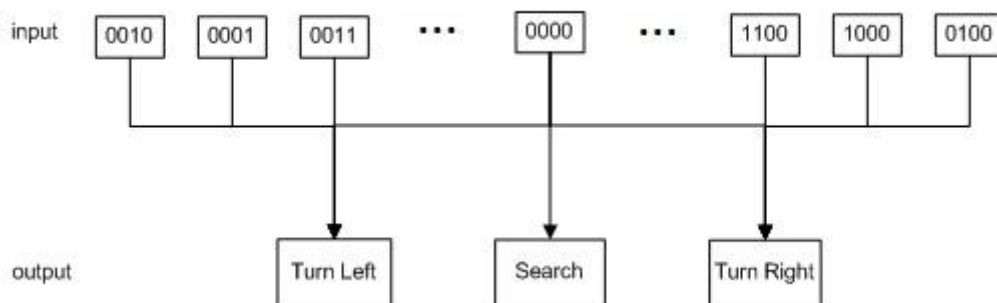


Figure 6.5: The design of search and avoidance behavior. A free passage (0,0,0,0) lets the e-puck explore, while a turn command will overrule if any obstacles are sensed.

Figure 6.5 illustrates how there is a direct connection between the sensors and wheels. The side with most 1's dictates the turning motion to avoid collision. If you feel something on the left side - turn right and vice versa. In the case of (0,1,1,0) or (1,0,0,1) or (1,1,1,1), which happens with bent shapes and corners, the e-puck will turn right. Avoidance is a pure reactive behavior which follows a static turn decision. To prevent collision as an e-puck turned, the turning motion involves reversing the wheel of which we are turning (left turn implies reversing the left wheel). Both right and left wheel turns with a static 70% and -30% of full wheel speed. This made the e-puck turn in place with a small amount of forward motion.

A free passage triggers the search behavior which is a random forward movement. Each wheel has a separate random function ranging from 50%-100% of full wheel

speed. The result of every time step (of free passage) calculating new wheel speeds, is the resemblance of a search motion. You look to the sides while moving forward, not concerned of the path you've already past. Recordings of this behavior has been made and published on youtube¹. To give a clear visual effect of the behavior we used the lower red light, which would continuously blink on and off each time step.

6.2.3 Converge & Push

The goal module represents behaviors containing actions which lead to movement of the box. The converge and push behavior are triggered when IR lights are detected by one or more of the eight light sensors. The environment is designed to have only one emitting IR source, which in the case of e-pucks is recognized as food.

Ready, Set ...

The threshold for detecting the box is set to 3700 giving a range detection of approximately 30cm (see section 4.6.1 for details). Upon detection of the box, the e-puck can be at any angle. It is important that the converge behavior sets the correct amount of speed on the left and right wheel in order to get close and align with the box. Similar to the movement design, we designed a cross connection between opposite sensors and wheels, but each sensor would have a different impact. As shown in figure 6.6, wheel speed is calculated each time step as a sum of sensors below the threshold limit. Left and right wheel speed is reset with every time step.

```

read ls // light_sensors
left_wheel , right_wheel = 0 // {0,1000}
repeat for all ls < 3700:
  if (ls0):
    left_wheel +=700
  else if (ls7):
    right_wheel +=700
  else if (ls1):
    left_wheel +=350
  else if (ls6):
    right_wheel +=350
  else if (ls2):
    right_wheel += -300
    left_wheel += 550
  else if (ls5):
    right_wheel+= 550
    left_wheel += -300
  else if (ls3):
    left_wheel +=500
  else if (ls4):
    right_wheel += 500

```

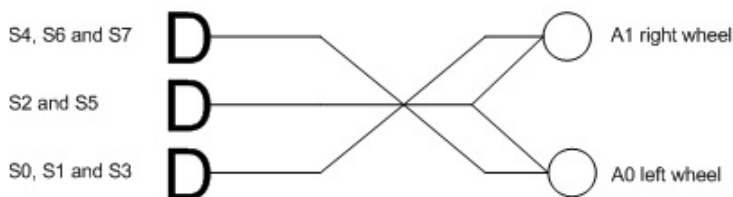


Figure 6.6: The converge behavior. A cross connection of opposite sensors and wheels are used to converge and align with the box. Sensor 2 and 5 affects both wheels to promote a quicker reaction, mastering a box on the move.

The design was initially tested using a Wii sensor (emitting IR light), and later

¹<http://www.youtube.com/watch?v=gPEhzisNOos>

tested on the actual box. With the Wii sensor we experienced that the converge behavior could also act as a follow behavior. Due to the quick reaction; e-pucks had no problem following a moving Wii sensor around in the environment. The threshold for detection was visualized through the use of the upper e-puck LEDs. Each sensor below the threshold would trigger the led above. A recording of the converge behavior has been made and published on youtube¹.

...Go

As the e-puck approaches the box, the light sensor value decreases. At the point where the value of the two front sensors is below 500 (3cm or less from the box), the push behavior is designed to trigger. The push behavior is simply putting full forward motion on both wheels - ramming the box. Push is visualized through blinking of all upper led lights.

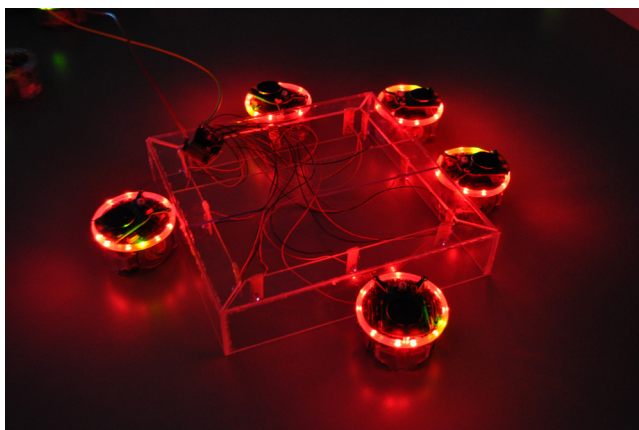


Figure 6.7: Push behavior. The LED lights will rapidly be turned on and off to visualize the behavior.

6.2.4 Realignment & Reposition

The e-puck will be able to push from any of the four different sides of the box. It becomes difficult to make a coordinated initial push using only stigmergy to communicate when e-pucks may detect the box simultaneously from different angles. There is therefore a high chance of stagnation occurring during the event, which have to be dealt with in order to progress.

The phenomenon of stagnation while engaged in group transport has been observed in ants [43]. For whatever reason, the item being transported has suddenly become stuck and the ants start exhibiting realigning and repositioning behaviors. The realignment occur more frequently than reposition; only when realignment seems insufficient does an ant try to find another spot. Our stagnation recovery module is inspired to handle box stagnation in the same way - realigning and repositioning.

¹<http://www.youtube.com/watch?v=G7TOMFxAbsk>

Stagnating

Stagnation detection is based on three factors: Positive or negative feedback, detection of box movement, and detection of nearby e-pucks. Positive and negative feedback dictates for how long an e-puck will try pushing before being stimulated to try something else. This is given as a function of time steps, where detection of the box starts an initial positive feedback for 150 time steps. At the end of each feedback run, the e-puck will review the progression through a series of tests to see if there exists box movement or nearby e-pucks. This review results in either a positive or negative feedback, increasing or decreasing the next epoch of time steps. As shown in the equation 6.1, the range of epochs is a minimum of 100 to a maximum 300 time steps. It involves increasing or decreasing the *feedback* variable, which can't go below 1 or above 8. The idea is that the e-puck should keep on going when things are working and review the situation less, while reviewing more frequently if stagnation occurs often. The function would also create a more dynamic push and recovery behavior.

$$timed_review = \left(\left(\frac{5}{10 - feedback} \right) * 100 \right) + 50 \quad (6.1)$$

The detection of box movement is handled by sensing the intensity of IR lights for a short period, but the value most "accurate" at that time is the distance sensors. As mentioned in section 4.6.1, the distance sensor does not detect proximity when high intensity IR is picked up by a sensor. In fact it detects nothing, with front sensor values ranging from 0-5 when close to the box. A small change in IR intensity increases distance value exponentially. We would therefore use the distance value to check for any box movement. At the end of each push epoch, the e-puck will stop its actions for a split second, recognizing any loss of intensity prior to stopping.

Detection of nearby e-pucks is the second method for getting positive feedback. If detection of box movement was false, the e-puck would check if it had company on either side. The design relies on the distance values on both sides of the e-puck, S2 and S5. The threshold is set to detect anything in range of 2.5 cm (distance value 300). If two neighbors were detected it would most probably be a wise decision to stay, while only one gave a 50/50 chance of staying put. If nothing became true then stagnation was imminent and a recover behavior would be stimulated to trigger.

```

read dsp // distance sensors prior to stopping
stop // wait 10 time steps
read ds
diff0 = dsp0 - ds0 // difference in right front sensor
diff7 = dsp7 - ds7 // difference in left front sensor
if(abs(diff7) and abs(diff0) > dist_thres):
    positive feedback // Keep pushing
else if(ds5 and ds2 > neighbor_thres):
    positive feedback // Keep pushing
else if(ds5 or ds2 > neighbor_thres):
    50/50
else:
    negative feedback // Recover

```

Recover

Negative feedback equals stagnation. It triggers the recovery methods of realignment and reposition. The design is inspired from the ant stagnation recovery method, with realigning happening more frequently than reposition.

Realignment is based on the current position of the e-puck. If the front sensors show equal amount of IR intensity, the e-puck will align either left or right as shown in figure 6.8. If the e-puck is facing left or right it will align directly at the box. During early test phases the e-puck started to show a non intentional behavior of moving from one end of the side to the other. The reason is that the box is too smooth and slippery for the e-puck. Whenever the e-puck would align left or right, it would also move slowly in that direction. However, we kept it as is, since it created a more dynamic behavior.

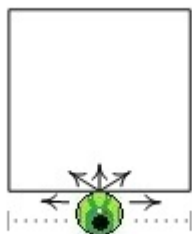


Figure 6.8: Illustration of the realignment design.

The reposition would trigger every third stagnation. Having tried to realign two times with no success, it would now be time to go to a completely new spot, changing to the side either left or right. The e-puck would first reverse, before making a turn either left or right and then turning the opposite way 6.10. During reposition the e-puck would have to avoid any collision with other e-pucks, which did create the possibility of losing contact with the box.

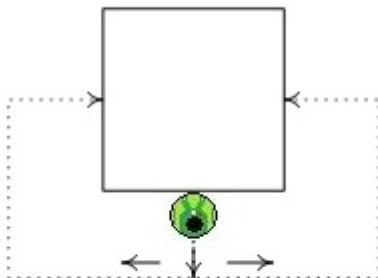


Figure 6.9: Illustration of the reposition design.

Recover consist of two quite different behaviors. Realignment is fast an abrupt,

while repositioning is a more drastic measure and taking more time. Both cases is visualized by a blinking green body light. A recoding of the behaviors can be found at youtube¹.

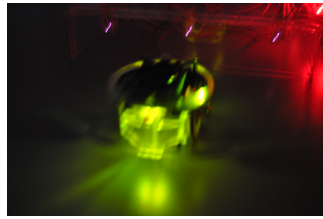


Figure 6.10: The green body light will blink during recovery.

6.3 The Behavior-Based System

Figure 6.11 shows the behavior-based flow of our inspired Brooks' subsumption architecture. Although each behavior has been explained in detail, this section shows an overview of how the system as a whole operates.

The strategy used in making the system was to follow the key concepts of Brooks' view on an intelligent system 3.1.4. In many ways it refers to the observed behaviors in ants. Intelligent behavior does not need reasoning, but needs only to react from a direct link between perceiving and acting. Each behavior in our architecture, figure 6.11, follows this pattern of being triggered only when sensory input stimulates it to react.

Starting from bottom in figure (6.11), each behavior has followed the continuous cyclic process of design and testing, before moving on to the next. This incremental process really showed its effect of creating a well balanced behavior-based system, and also how it resembles an observed biological ant model. Concentrating each design step on one single behavior, starting with the most basic and moving on to higher and more complex, became a very good process for creating a swarm intelligent system.

¹<http://www.youtube.com/watch?v=46IDt9igL-U>

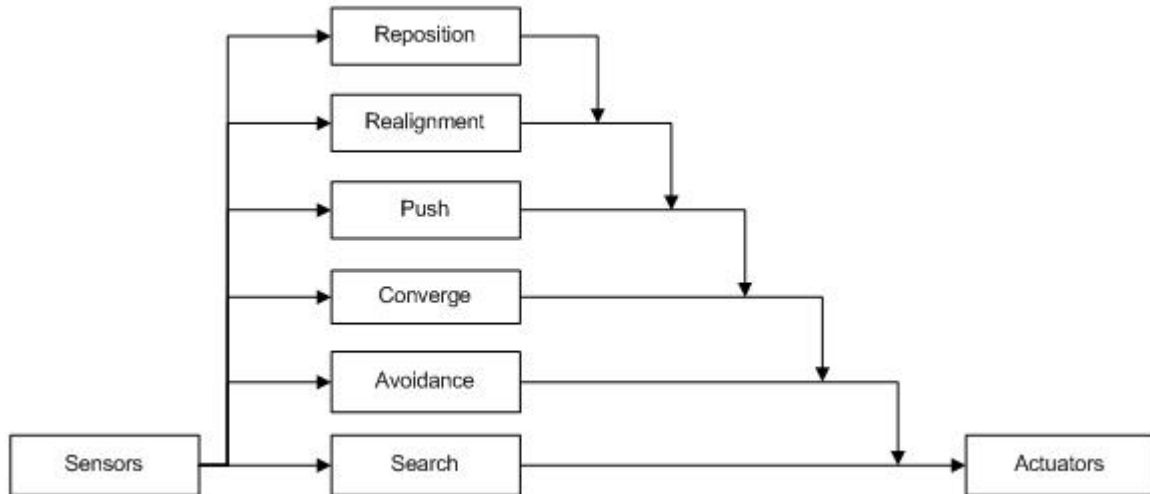


Figure 6.11: CRABS architecture, Brooks' subsumption.

With the higher behaviors subsuming the lower ones, this is how CRABS operates:

- Stimulated by the mere fact of being awake and in working modus, a clear path lets the e-puck search for food.
- If any danger for collision appears on the path in front of the e-puck, it will turn either left or right to avoid
- Sensing IR light is like ants pheromones; it triggers e-pucks to converge, heading towards that direction.
- Upon arrival at the artificial food source a push behavior will trigger to retrieve the box.
- During the retrieving event stagnation may occur and e-pucks will try realigning with the box, creating a different angle of force.
- If stagnation is still imminent after realigning the last option is to reposition; finding a new spot to apply force.

Experiments, Results & Discussion

This chapter presents the CRABS experiments. It focuses on individual behaviors as well as the entire system to explore and examine expectations and challenges presented in the previous chapters. Each experiment presents the necessary information for recreation, and each result is analyzed and discussed.

7.1 Experimentation

The behavior-based architecture had through the design phase been tested on reactions. This means that every behavior had gone through a series of proto-testing, to ensure that each behavior responded correctly according to environmental condition and that the action performed represented the intention of the design. With confidence in the design of the system, we started to experiment and measure this collective problem-solving robotic system.

7.1.1 Setup

The main investigation with these CRABS experiments is to explore cooperative behavior on a physical level, replicating the observed ant retrieving model on none tailored robots. As the system is based on the same observed mechanism we find in ants, an emergent swarm behavior would confirm and strengthen the thoughts on swarm intelligence in ants, Brook's view on intelligent behavior, and insight in collective problem-solving with e-pucks. Specific information about the task performance and optimal amount of e-pucks would also give deeper insight to each behavior, scaling issues and further work. The experiments are divided to explore the impact of each retrieving behavior, before measuring the performance of the whole system:

- *Goal* - How well does the system perform with no use of stigmergy or feedback?
- *Realignment* - Is small changes in direction of force enough to handle stagnation?
- *Reposition* - Does the negative feedback need a drastic change to turn into positive?

- *CRABS* - Can these simple mechanisms perform to such an extent that it will strengthen the idea of swarm intelligence in social insects and show how several simple robots can outperformed one complex? Does the system account for scaling issues?

All the experiments were performed under dark conditions, at night, on a 121,5cm quadratic board with no other static obstacles. In order to make each test as identical as possible, we drew a 25x25 cm square in the middle of the board to ensure the same starting position of the box. In addition, the initial e-puck state would be random. E-pucks where turned on searching for the box for 1 minute before the box was turned on. Each of the four experiment was tested on three groups of e-pucks; 3, 5 and 8, which were all tested in the environment 10 times. The system was given a maximum 3 minutes, from the time the box was detected, to accomplish the goal (retrieving the box to the wall).

The general configuration for all tests are listed in the following table 7.1. The details are explain in section 6.2.

Table 7.1: CRABS initial configuration

Variables	Values
IR Threshold	3700
Distance Threshold	250
Number of LEDs	10
Number of Light Sensors	8
Number of Distance Sensors	8

Since the e-pucks are sensitive to sunlight and other light sources the tests had to be performed at night to minimize IR interference. However, running the system during daytime gave a different swarm behavior, which in retrospect is a system similar to the research on clustering bees [49]. The sunlight hits the environment from one angle in our workspace, making all e-pucks run towards that corner. Closing in on the wall they turn around (sunlight is blocked) to avoid collision and start searching for what stimulated them just a second ago. Moving out from the wall they again pick up the scent which is now coming from behind, which makes them turn around and again move towards the wall. With a few tweaks, this cyclic behavior could easily show the clustering behaviors in bees.

With concerns to expectations and limitations, the system is constrained by the environment, shape of the artificial food source and the maximum number of eight e-pucks. Another issue that had to be taken into account was lifetime of the batteries on both the box and the e-pucks. The batteries on the box are four AA batteries and have a lifetime of about 1 hour with the circuit given in section 5.3. After 1 hour, the IR lights will be somewhat weakened, and the distance of where e-pucks sense the box may decrease. To get equal conditions for each test, it was important to always make sure the batteries were not used for more than one hour. The batteries

on the e-pucks on the other hand lasted slightly longer. By continuous driving, the e-pucks managed to last about 2 hours. However, the advantage of e-pucks is that they have an extra light that indicates when they start to run out of power. We are confident that each run is conducted under the same conditions.

7.1.2 Goal

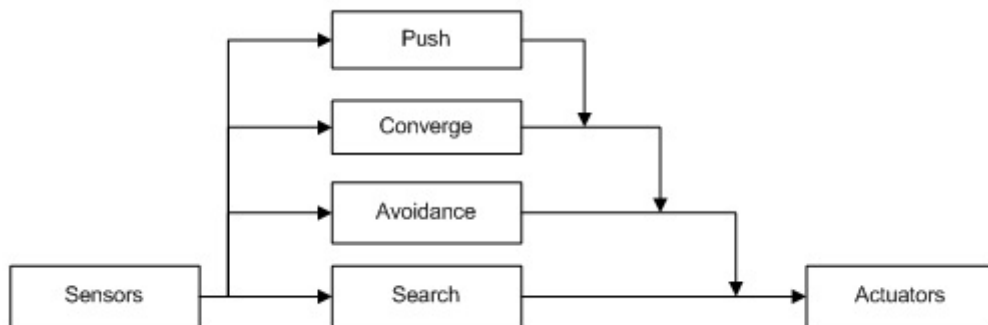


Figure 7.1: CRABS during testing of the goal module.

The lowest form of swarm intelligence emerges from sharing a common task. With only the basis of converge and push, we would test the system 10 times on each of the groups; respectively 3, 5 and 8 e-pucks. The box retrieving result is listed in table 7.2(a) and illustrated in figure 7.2. The systems performance is given as a combination of how far and fast the e-pucks were able push the box from its initial position. Distance is given as percentage and time in seconds. Figure 7.1 shows the behavior setup of the software system.

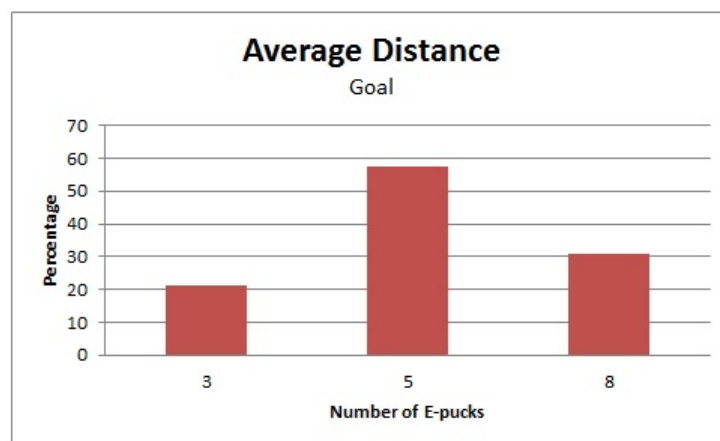


Figure 7.2: The goal behavior. This figure illustrates the average performance from all the 10 runs for each group of e-pucks.

Analysis

As one perhaps would expect from narrowed minded robots that are blinded by stimulation, they would either complete the task or stagnate forever. This section

Table 7.2: Results from the goal behavior - Table 7.2(a) shows for each group of e-pucks and for each run, how long distance, in percentage, they were able to push the box. Table 7.2(b) shows how long time the e-pucks used in each run.

(a) Performance				(b) Time Spent in Seconds			
Runs	E-pucks			Runs	E-pucks		
	3	5	8		3	5	8
1	2,06%	100%	0%	1	180	19	180
2	0%	100%	0%	2	180	29	180
3	2,06%	96,91%	21,65%	3	180	180	180
4	0%	17,53%	0%	4	180	180	180
5	100%	2,06%	54,64%	5	46	180	180
6	2,06%	0%	86,60%	6	180	180	180
7	0%	100%	6,19%	7	180	33	180
8	100%	49,48%	56,70%	8	50	180	180
9	2,06%	100%	52,58%	9	180	26	180
10	6,19%	7,22%	29,90%	10	180	180	180
Avg	21,4%	57,3%	30,8%	Avg	153,6	118,7	180

brakes down each of the three groups and takes a closer look at their performance.

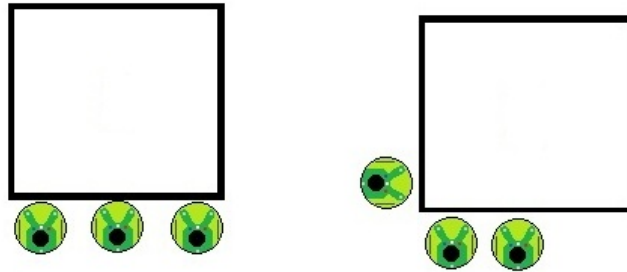
3 e-pucks

Table 7.3: Goal performance with three e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	2,06%	0%	2,06%	0%	100%	2,06%	0%	100%	2,06%	6,19%	21,4%
Time Spent	180	180	180	180	46	180	180	50	180	180	153,6

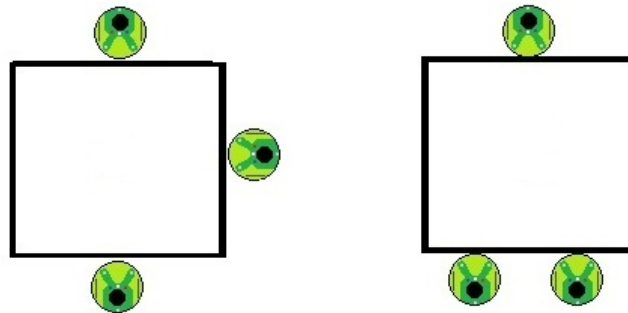
The first experiment performed involved three e-pucks with only converge and push behavior implemented (search and avoid is also present). Running the tests with only three e-pucks is a special retrieving case as everyone needs to participate in order to complete the task. Observing each run, we notice that this setup was all about initial configuration - the starting position of the e-pucks. In order to move the box, all e-pucks had to be placed on the same side, or the case shown in figure 7.3. In runs where it did happen, the amount of time used in retrieving the box was low - always less than one minute. Remember, it could take some time before all three detected the food source. If these cases did not occur, they would quickly be in a stagnation state which they couldn't recover from, see figure 7.4.

The chance for an e-puck to choose one of the four sides is equal, 25%. The probability of getting three e-pucks on the same side is 1.6%, which is quite low. They do however in reality have a higher chance as there exist cases like in figure 7.3 where they also are successful. From table 7.2(a) we can see that they manage to complete the task in 2 out 10 tries, 20%. This result showed us that to get accurate results we



(a) All e-pucks are placed on (b) This setup also creates the same side, creating maximum enough force to move the box. force in that direction.

Figure 7.3: These two cases illustrate two patterns which creates enough force to move the box.



(a) Lack of cooperation

(b) Opposite forces

Figure 7.4: These two figures illustrate two stagnation cases that may occur during runtime.

should run these experiments a hundred times, preferably a thousand, but it would not be feasible with night conditions, batteries and time. We are however able to observe the main features of the system. The result of 2.06% is the initial push from two e-pucks on the same side. This occurs because an e-puck who is ramming the box hits with a bigger force than an e-puck who is pushing. As a sum up, our three e-pucks preformed beyond our expectations, and one can argue that they during these 10 runs where quite lucky.

5 e-pucks

Table 7.4: Goal performance with five e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	100%	100%	96,91%	17,53%	2,06%	0%	100%	49,48%	100%	7,22%	57,3%
Time Spent	19	29	180	180	180	180	50	180	26	180	118,7

The experiments with a group of five e-pucks were quite similar to the experiment with three. The systems success was still very much bound by the initial starting position of the e-pucks, but there would be more force in the environment with two more robots. By looking at table 7.4, we can see that the success rate increased from two to four runs compared to the previous tests. These four successful runs were also accomplished in a very short amount of time. They only used a maximum of 40 seconds, which is 10 seconds less than the other experiment and the first run was done in 19 seconds. The reason for this decrease in time is mainly based on the amount of e-pucks participating. As long as there are no opposite forces, the e-pucks will be able to move the box.

Running the goal behavior with five e-pucks showed the general feature of having a faster system. Five e-pucks would be quicker in accomplishing the task, but it would also be quicker at running into a stagnation state. With five e-pucks we also observed a new emergent cyclic behavior, where the e-pucks would put force near each corner and begin pushing the box in a circular motion (see figure 7.6). The circular motion would not in most cases be completely circular and the box would move in a direction, but as seen in run 3 (table 7.4) it could be very inefficient, not making it in 180 seconds.

E-pucks are not always helping when they apply force. In two occasions, some e-pucks would get the box to move, before another e-puck on the opposite side suddenly would detect it and stop the movement. As shown in runs 4 and 8, the e-pucks managed to push the box respectively 17,53% and 49,48%, before getting into a stagnation state (see figure 7.5).

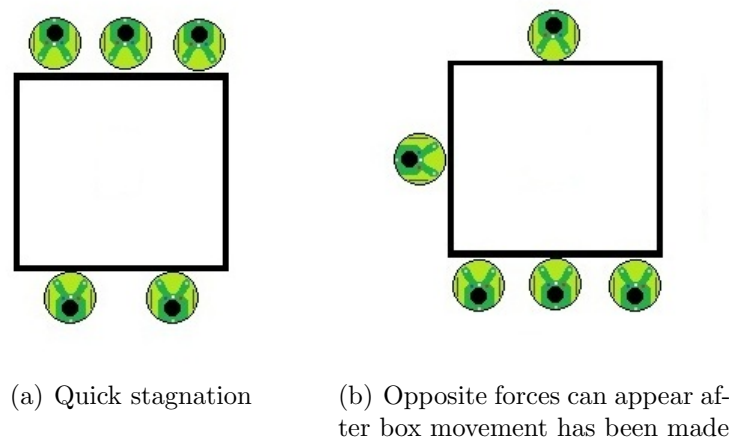


Figure 7.5: These figures illustrates some of the stagnation cases with five e-pucks

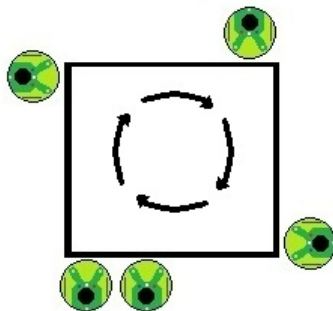


Figure 7.6: Illustrating a circular movement of the box, which did occur with five e-pucks

8 e-pucks

Table 7.5: Goal performance with eight e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	0%	0%	21,65%	0%	54,64%	86,60%	6,19%	56,70%	52,58%	29,90%	30,8%
Time Spent	180	180	180	180	180	180	180	180	180	180	180

By adding additional three e-pucks to the group could in theory give a better solution as more e-pucks could get involved and override if one e-puck detected the box at the opposite side, but it could possibly also create more stagnation. As already seen with five e-pucks, stagnation could occur after getting box movement, since other e-pucks could discover the box later, ending up applying force in the opposite direction.

An observation we came across when running tests with eight e-pucks was the potential of e-pucks becoming trapped between the box and the wall. Since the amount

of e-pucks where so high, a force in one direction could overrule the opposite force, eventually trapping the ones applying less force. This would prevent the group from finishing the task, which occurred during run 6 (table 7.5). Figure 7.7 illustrates the situation.

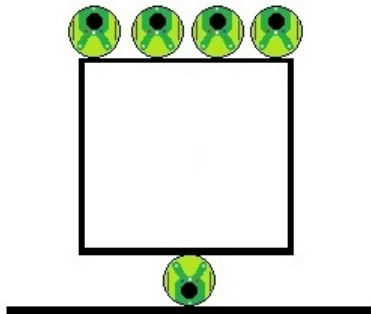


Figure 7.7: A massive force in one direction could overrule a weak opposite force eventually trapping the e-puck.

The environment was quite small for eight e-pucks. Collision was not an issue, but there is a certain amount of spots on the box. An e-puck in pushing state would stop the IR lights from being detected by others. We observed several occasions where there would be four e-pucks at one side, preventing others from helping. The others would keep on searching, increasing the risk of stagnation. The general observation was that adding more force did not necessarily increase the systems performance.

Compared to the experiments with three and five e-pucks, this experiment would also get in a stagnation state very quickly. By looking at the results in table 7.5 stagnation occurred in both the first, second, fourth and seventh run. The major difference from three and five e-pucks was that eight e-pucks with a goal behavior were never able to complete the task. The risk of stagnation seemed too high. The closest they came was in run 6 with 86,60%.

Like in the experiment with five e-pucks, this experiment also had the possibilities of being trapped in a cyclic motion. With the cyclic motion and the high risk of stagnation, these runs spent the overall most time trying to complete the task.

Overall

Experimenting with only the behavior of sharing a common goal gave valuable information about the system. From the observed ant model, it shouldn't be enough to only share the task, which also was the case with CRABS. Initial starting position was the major factor when running these tests.

When the number of e-pucks is low, the e-pucks are either able to complete the task 100% in a short amount of time, or they get in a stagnation state after less than 10 seconds and are not able to push the box more than small initial pushes. Increasing

the number of e-pucks results in higher risk of stagnation and the possibility of circular box motion which may complete the task given more time.

7.1.3 Realignment

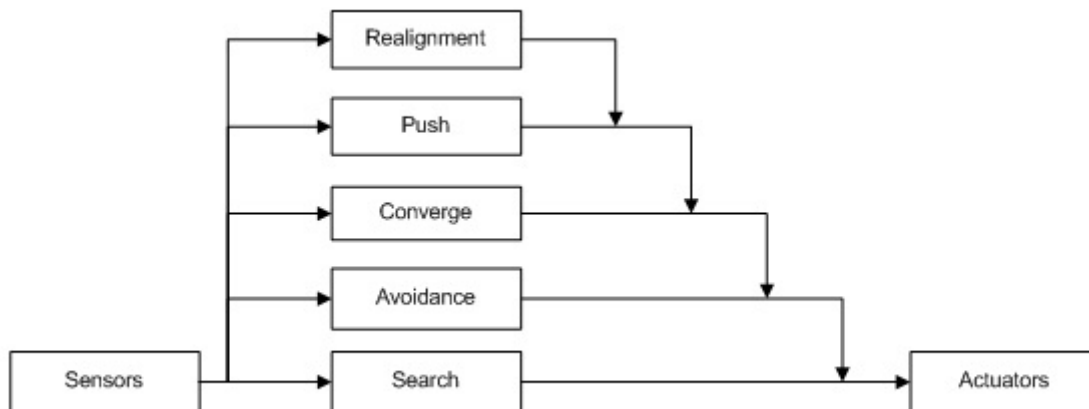


Figure 7.8: CRABS during testing of the realignment behavior.

The realignment behavior is one of two stagnation recovery behaviors implemented in CRABS. Adding only one recovery behavior would give insight to the impact it has on the system performance. Comparing it to the goal and later the reposition behavior would also show what small directional force changes could do. The results are illustrated in figure 7.9 and shown in table 7.6(a).

Table 7.6: Results from the realignment behavior - Table 7.6(a) shows for each group of e-pucks and for each run, how long distance, in percentage, they were able to push the box. Table 7.6(b) shows how long time the e-pucks used in each run.

(a) The Performance				(b) Time Spent in Seconds			
Runs	E-pucks			Runs	E-pucks		
	3	5	8		3	5	8
1	0%	60,82%	10,31%	1	180	180	180
2	0%	0%	100%	2	180	180	95
3	0%	8,25%	100%	3	180	180	155
4	46,39%	0%	40,21%	4	180	180	180
5	0%	10,31%	44,33%	5	180	180	180
6	2,06%	91,75%	84,54%	6	180	180	180
7	0%	100%	60,82%	7	180	84	180
8	0%	100%	100%	8	180	46	36
9	90,72%	8,25%	57,73%	9	180	180	180
10	40,21%	72,16%	100%	10	180	180	66
Avg	17,9%	45,2%	69,8%	Avg	180	157	143,2

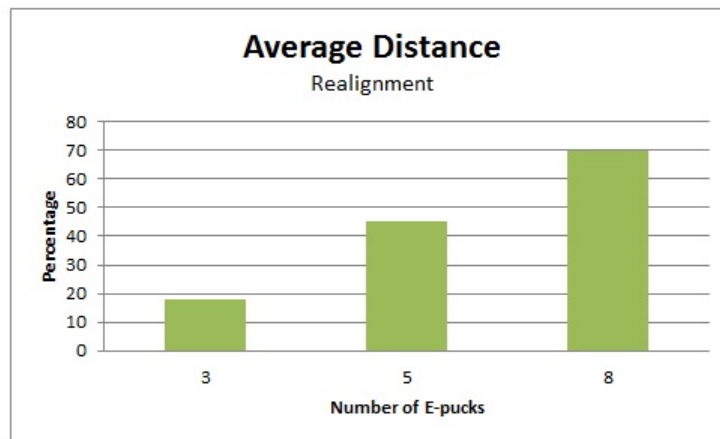


Figure 7.9: The realignment behavior - This figure illustrates the average performance from all the 10 runs for each group of e-pucks.

Analysis

Bringing positive feedback and stigmergy into the system, and applying it to a behavior which applies small directional force changes during stagnation, should in theory perform better than a narrowed minded robot. Looking at the overall result in figure 7.9 it could actually seem it did not make the system improve in all areas. A closer look into each test of groups will shed some light on the matter.

3 e-pucks

Table 7.7: Realignment performance with three e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	0%	0%	0%	46,39%	0%	2,06%	0%	0%	90,72%	40,21%	17,9%
Time Spent	180	180	180	180	180	180	180	180	180	180	180

The realignment behavior follows the same rules as the goal behavior - three e-pucks close to each other is needed to complete the task. The success rate of this setup would also be dependent on the random initial starting positions. The positive thing is that if there exists no opposite forces they could in theory realign to eventually get closer to each other and then get the box to move. Starting on opposite neighboring sides, they could after some time end up like figure 7.3(b) illustrates and accomplish the task.

Observing the runs it became clear that getting positive feedback would in some cases be problematic. If all three were on the same side they would get positive feedback from having neighbors and from small stops where a box on the move would continue moving slightly away before stopping (at that point there would only be two pushing). However, if the e-pucks had the setup of figure 7.3(b), there would be less chance of getting positive feedback from neighbors, and from a moving box. The speed of the box would not be exactly directional to the e-pucks force, which at times triggered them to realign when they shouldn't.

The realignment behavior showed early signs of being able to handle more situations, but it couldn't make the e-pucks complete one single run during the 10 tests. The closest they came was in run 9 with 90.72%. We are however confident that it would outperform the goal behavior if these two systems were tested more times. It showed us that the initial e-puck position still had too much impact on the results, but realignment did more often than goal get the box to move a significant distance.

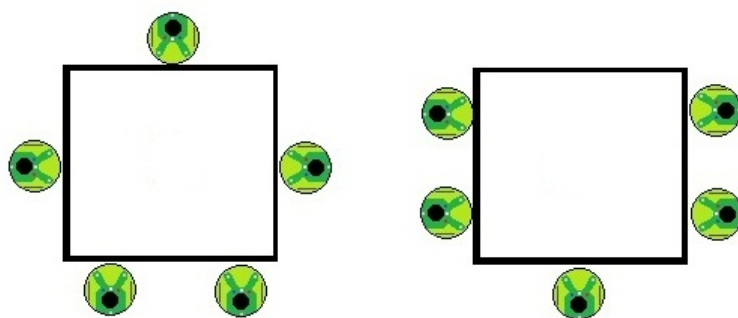
5 e-pucks

Table 7.8: Realignment performance with five e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	60,82%	0%	8,25%	0%	10,31%	91,75%	100%	100%	8,25%	72,16%	45,2%
Time Spent	180	180	180	180	180	180	84	46	180	180	157

The experiment with five e-pucks is also similar to the corresponding experiment with the goal behavior. The chance of completing the task increased, the time spent decreased and the overall distance they moved the box increased. The only difference we could observe was that opposite forces needed to be in place before the event would actually be in a stagnation state as illustrated in figures 7.5(a) and 7.10. If there were no opposite forces they could (given some more time) recover.

This would imply that realignment improved the system to handle more situations, but again it didn't show in the results. In the sense of not repeatedly state that more runs should show a difference, five e-pucks could be the case where realignment doesn't have a clear advantage to goal. Eight e-pucks on the other hand would show realignments true capabilities.



(a) Opposite forces causing stagnation (b) Opposite forces causing stagnation

Figure 7.10: Stagnation occurrences with realignment

Table 7.9: Realignment performance with eight e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	10,31%	100%	100%	40,21%	44,33%	84,54%	60,82%	100%	57,73%	100%	69,8%
Time Spent	180	95	155	180	180	180	180	36	180	66	143,2

8 e-pucks

The last experiment on the realignment behavior really gave some unexpected good results. Even though 4 out of 10 might not be a huge increase from the experiment with five e-pucks, the overall distance was. They were always able to get the box moving and during the event they seemed to cooperate in a bigger way. Part of that was because opposite forces could have even smaller impact now that the direction of force could switch from being direct to being sideways. This is illustrated in figure 7.11.

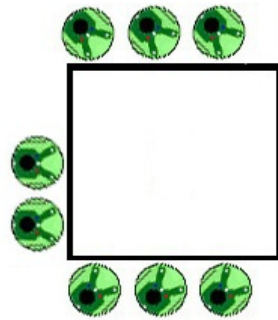


Figure 7.11: Realignment makes the e-puck switch force direction if negative feedback occurs, potentially solving the issue of opposite forces stagnating.

Having a minimum completeness of 10% and an average of almost 70% also showed that realignment wasn't enough to make sure the food source got retrieved. A part of that was of course the chance of coming in a complete stagnation scenario where each side had equal, or close to equal, amount of force. It also struggled with getting positive feedback from sideways movement. However, it did show some real potential, but not as much as we expected.

Overall

The results were improved each time more e-pucks were added and the time spent decreased almost linearly as the number of e-pucks increased. It would have been great to test this behavior with more e-pucks to be able to clearly tell the optimal amount of e-pucks, but there is the issue of pushing space on the box. Anything above twelve e-pucks can with some certainty be said to be too much. This would most probably create more stagnation problems than movement opportunities.

7.1.4 Reposition

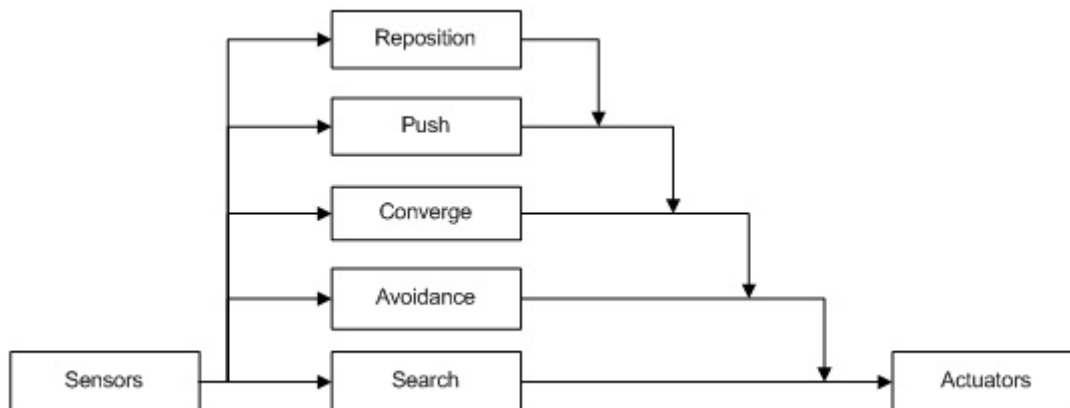


Figure 7.12: CRABS behavior setup during reposition testing

The last stagnation recovery behavior is the reposition behavior. Design to work with the realignment behavior, reposition will only trigger after *3 x negative feedback*. Removing realignment means that push behavior will keep running 2/3 times of negative feedback. Reposition is a behavior with drastic measures. When stimulated, the e-puck will start finding a new open spot on the box to solve stagnation. The results are illustrated in figure 7.13 and listed up in table 7.10.

Table 7.10: Results from the reposition behavior - Table 7.10(a) shows for each group of e-pucks and for each run, how long distance, in percentage, they were able to push the box. Table 7.10(b) shows how long time the e-pucks used in each run.

(a) The Performance				(b) Time Spent in Seconds			
Runs	E-pucks			Runs	E-pucks		
	3	5	8		3	5	8
1	100%	100%	100%	1	6	106	62
2	100%	29,90%	100%	2	134	180	67
3	60,82%	90,72%	100%	3	180	180	18
4	100%	100%	100%	4	179	64	85
5	64,95%	100%	32,99%	5	180	49	180
6	3,09%	100%	100%	6	180	157	93
7	100%	100%	100%	7	178	68	41
8	89,69%	100%	100%	8	180	91	94
9	90,72%	100%	100%	9	180	47	135
10	6,19%	100%	100%	10	180	78	110
Avg	71,5%	92,1%	93,3%	Avg	157,7	102	88,5

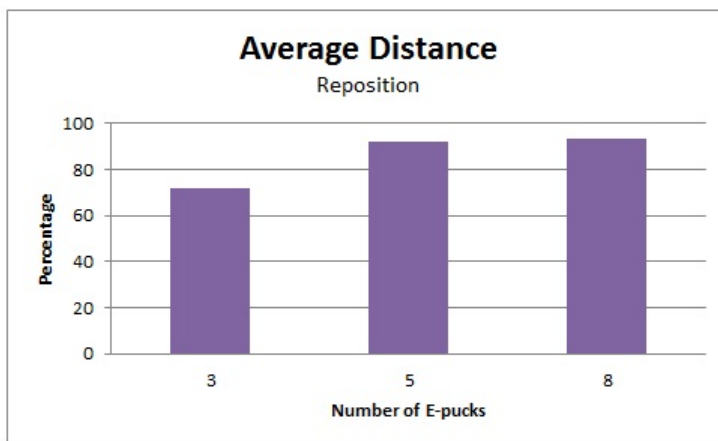


Figure 7.13: The reposition behavior - This figure illustrates the average performance from all the 10 runs for each group of e-pucks.

Analysis

The reposition behavior proved to exceed every other behavior by good figures. It is however difficult to pin point the reason and extract useful information about the behavior from the numbers shown in figure 7.13, which is why we take a detailed look at each group.

3 e-pucks

Table 7.11: Reposition performance with three e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	100%	100%	60,82%	100%	64,95%	3,09%	100%	86,69%	90,72%	6,19%	71,5%
Time Spent	6	134	180	179	180	180	178	180	180	180	157,7

It is clear that the reposition system has the same chance as others of accomplishing the task right away with the random initial starting positions. This was already observed in run 1 7.11. The next runs did not, but we observed a behavior which in a sense led to many "starting positions". Finding a new spot with two other e-pucks close by gives always a chance of getting three on one side. The drastic change of force showed some true recovery abilities with 4 of 10 times reaching the goal and only two low performances.

From the 10 runs it became clear that given unlimited time this system would retrieve the food source back to the wall every time. The reason for the large discrepancy of performance from 3 % to 100% is the lack of knowledge about forces during stagnation and the three minute mark. The positive and negative feedback is based only on movement because of limited sensing ability, which withholds information that could redirect the e-puck to make a more intelligent recovery move during stagnation. Having only the reposition behavior implemented it clearly improved the system, but it is also at times too slow and drastic in certain situations.

5 e-pucks

Table 7.12: Reposition performance with five e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	100%	29,90%	90,72%	100%	100%	100%	100%	100%	100%	100%	92,1%
Time Spent	106	180	180	64	49	157	68	91	47	78	102

Five e-pucks showed many similarities to the previous tests with three, but because of more force were able to complete the task 8 out 10 times! With five e-pucks it became easier to get box movement, which again results in more positive feedback. Positive feedback leads to longer durations of push behavior which was a nice asset to reposition. If every e-puck would have had the same timer to reposition, it would have been like resetting the test run. With neighbors and box movement, there emerged an intelligent unpredictable global structure, looking to be more complex than the simple mechanisms implemented.

Even though the system showed some real swarming abilities, there was occurrences in the event where actions made by the e-pucks would slow down progress and look unintelligent. The two runs that didn't make it in three minutes involved some unintelligent e-puck choices. Unintelligent is perhaps only correct when you look at the system from a global view. From the local information each e-puck sits with, it would seem to be the best choice. However, the situations where there is box movement and another e-puck starts converging on the opposite side, creates stagnation and slow down the progress tremendously. In some cases we observed that this situation would make e-pucks reposition and move the box "backwards" - covering distances they have already walked. The distance covered and box movement in run 2 7.12 was actually better than the performance numbers implies.

8 e-pucks

Table 7.13: Reposition performance with eight e-pucks

Runs	1	2	3	4	5	6	7	8	9	10	Avg
Performance	100%	100%	100%	100%	32,99%	100%	100%	100%	100%	100%	93,3%
Time Spent	62	67	18	85	180	93	41	94	135	110	88,5

Increasing the result with 1% from five to eight is revealing that the behavior is also working for bigger amount of e-pucks with less space. More force has already shown to give more box movement, but also higher risk of stagnation. The reposition behavior with eight e-pucks seems to surpass that problem. In the beginning there is a high chance that opposite forces occurs, but after getting some box movement, the side where e-pucks are pushing is filled with positive feedback which endures any incoming opposite forces. When the box moves, it is often at a fast pace since more force is operating on it, which again limits the area where opposite forces can occur.

Given unlimited time this behavior would always be able to finish, eventually getting the box to one of the walls. Even though the overall performance was better, taking less time and succeeding more often, more e-pucks did create some space issues. During repositioning the e-puck reverse and moves to a different location on the box as describe in section 6.2.4, but during that event others might as well. With higher layers of behaviors subsuming the lower ones and being more complex, repositioning would follow the same pattern as *avoidance* and turn before any collision. This would in most cases affect the repositioning behavior in a negative way, turning the e-puck away from the box (starting a new search) or turning it to the box (not relocating).

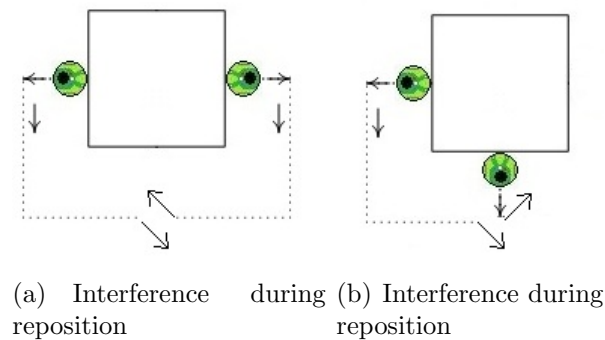


Figure 7.14: Avoiding other e-pucks during repositioning could at time have a negative impact on the behavior.

Overall

The repositioning behavior increased the overall performance of the system and showed to be a huge factor for e-pucks to make more intelligent choices during stagnation. The drastic change and time spent repositioning didn't slow down the system the way we expected it would. The action would take longer time to perform, but the benefit was greater, making the system accomplish the task faster, and given unlimited time - always!

7.2 CRABS

The architecture of Brooks' subsumption made it possible to test each behavior and the impact they had. The goal, realignment and repositioning module, all showed emergent swarm behavior from sharing a common goal, and the recovery module gave e-pucks a chance to progress when stagnating. This section presents the result from testing the whole system CRABS where all behaviors are combined. It also analysis observations made from specific runs.

7.2.1 Results

CRABS testing followed the same procedure as the previous experiments, with 10 runs on respectively 3, 5 and 8 e-pucks. Behaviors are designed to trigger according to how sensors are being stimulated and action subsumed according to CRABS architecture. We've already presented how each behavior performed and it is interesting to see how CRABS performs with all of them combined. The results are presented in table 7.14(a) and illustrated in figure 7.15.

Table 7.14: Results from food retrieval with CRABS - table 7.14(a) shows for each group of e-pucks and for each run, how long distance, in percentage, they were able to push the box. Table 7.14(b) shows how long time the e-pucks used in each run.

(a) The Performance				(b) Time Spent in Seconds			
Runs	E-pucks			Runs	E-pucks		
	3	5	8		3	5	8
1	90,72%	100%	100%	1	180	89	26
2	25,77%	100%	100%	2	180	156	27
3	100%	100%	100%	3	18	122	78
4	100%	100%	100%	4	117	29	151
5	53,61%	58,76%	100%	5	180	180	153
6	39,18%	100%	100%	6	180	74	38
7	23,71%	67,01%	100%	7	180	180	58
8	100%	100%	100%	8	169	34	37
9	100%	100%	100%	9	57	85	13
10	46,39%	100%	100%	10	180	72	77
Avg	67,9%	92,6%	100%	Avg	144,1	102,1	65,8

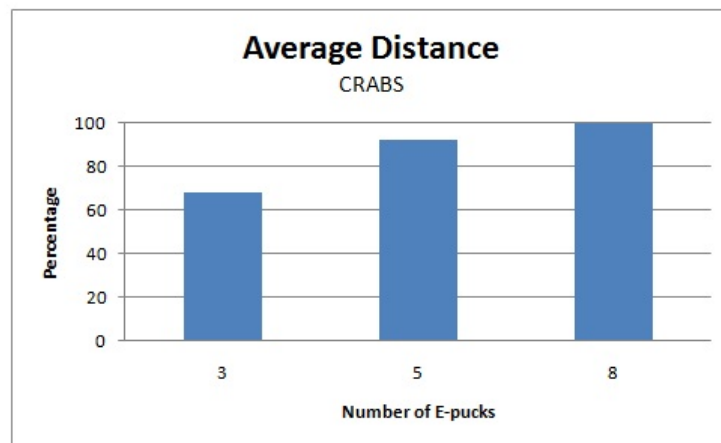


Figure 7.15: CRABS - This figure illustrates the average performance from all the 10 runs for each group of e-pucks.

7.2.2 Analysis

Brooks' architecture worked really well with the design of each behavior. Each e-puck was responding according to the static fixed priority scheme, but as a group they were dynamic and showed an emergent swarm behavior.

A combination of all behaviors, CRABS, had a better performance and less time spent than each behavior individually at every level. The only exception was the group of 3 e-pucks, which is deeply affected by the reposition behavior. As seen in figure 7.16, goal and realignment doesn't make an impact yet at this stage. Observing each run, we were pleased to see that the design of the system was working as intended and that each behavior made an impact from 5 and more.

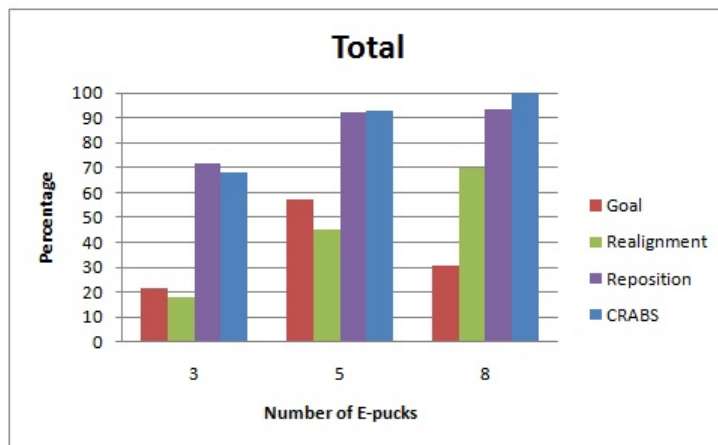


Figure 7.16: This figure illustrates the performance of each behavior and combined (CRABS). CRABS follow much of the same patterns as reposition, but is also benefiting from realignment, improving the performance of the system from 5 e-pucks and more.

CRABS takes the benefits and the drawbacks of each behavior. It still has problems with cyclic behavior, early stagnation from initial starting positions and reposition avoidance. However, the recover actions from realignment and reposition are not subsumed by one or the other. Both give separate quality effort to boost the system, where realignment gives small and quick force changes and reposition more of the opposite. It shows in the performance and quickness of CRABS.

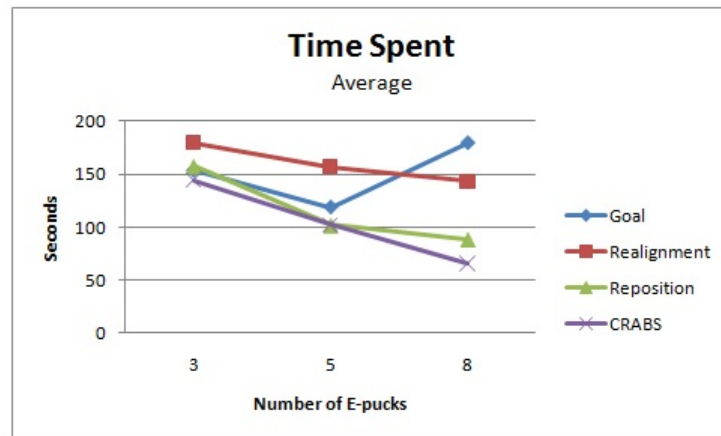


Figure 7.17: This figure illustrates the average time spent of each behavior and combined (CRABS). CRABS use equal or less time to complete the task.

To visualize how each behavior affects the system could be difficult, which is why we've released recordings of the system¹². Let's not forget that CRABS is a system operating on physical robots.

7.3 Discussion

The previous sections exposed CRABS to the box-pushing event and gave reason for the result. This section discusses and evaluates the system as a whole with concerns to the challenges.

7.3.1 The Swarm CRABS

Individual simple processing mechanisms within a group, proved to solve the complex problem of box-pushing. Robots sharing a common goal created swarm behavior, and recovery methods resulted in a robust and flexible system. The collective cooperative behavior formed a self-organized system which was based on observations in social insects. In retrospect - CRABS showed swarm intelligence.

... on Swarm Intelligence

Swarm intelligence emerged in the system because of two main factors - stigmergy and positive/negative feedback. Swarming emerges through acting on a common opinion, but CRABS showed us that intelligence first happen as each individual responds dynamically to the local information around them. Either through indirect communication (like having neighbors around) or feedback (box movement), the e-pucks would respond independently to optimize the performance of the system.

¹http://www.youtube.com/watch?v=9y_BbSY9.1M

²http://www.youtube.com/watch?v=G_ldCe8bDU4

...on Individual Behaviors

The goal behavior gave an unintelligent swarming behavior, and the result shows that converge, align and push is not enough to always retrieve an item, table 7.15(a). It does however show that "intelligence" in social insects could at times be a random factor. CRABS with only goal module implemented, is lucky with starting position each time the e-pucks are able to move the box.

Table 7.15: Results

(a) Average Performance

Behavior	E-pucks			Average
	3	5	8	
Goal	21,4433%	57,3196%	30,8247%	36,5292%
Realignment	17,9381%	45,1546%	69,7938%	44,2955%
Reposition	71,5464%	92,0619%	93,299%	85,6357%
CRABS	67,9381%	92,5773%	100%	86,8385%

(b) Time Spent in Seconds

Behavior	E-pucks			Average
	3	5	8	
Goal	153,6	118,7	180	150,767
Realignment	180	157	143,2	160,067
Reposition	157,7	102	88,5	116,067
CRABS	144,1	102,1	65,8	104

Realignment had an unexpectedly low performance versus the performance of reposition, but it was not the behaviors fault. The flaw was on the physical level. E-pucks have a round shape and the material of the box is very smooth. Those two does not combine to give realignment a chance to show its potential. Aligning sideways took away a lot of force, and realignment therefore needed more e-pucks to finish the job. Even though it was designed to be a quick recovery method, it used a lot of time stagnating, since many e-pucks needed to realign correctly to get box movement. It do show that realignment is more suitable for grab actions, like when ants try to lift there item instead of pushing or dragging it, which is something the e-puck simply can't do.

Reposition on the other hand was a great feature for box-pushing. It had the best performance of the behaviors and was also fast at it, table 7.15(b). Goal and realignment was either on the move or stagnating for a high amount of time, while reposition solved stagnation quickly, and spent more time pushing the box. We started to implement a more complex converge and reposition behavior that would reduce stagnation when there was box movement. This is explained in further work in chapter 8.

CRABS has over the course of this project shown to be robust and flexible. Should any of the hardware sensors malfunction, it would still be able to perform.

... as an Ant Retrieving Model

As mentioned throughout this project, there is no formal description on the ant retrieving model. The behavioral biology is well described on a observation level, but the biological details is not yet fully understood. CRABS has been designed to strictly follow the guidelines of the ant observation researched on this event. The restriction to the system has been the robots, which sense IR instead of actual food. With a decentralized system, indirect communication, simple mechanisms, and no memory, we were able to make a system that has a 100% retrieving success given eight e-pucks, table 7.15(a), or given unlimited time with three or more.

Using stigmergy and positive/negative feedback we have seen some ant-like behaviors during experimentation. The circular retrieving motion was truly remarkable. The problem we had with positive feedback creating an ongoing circular motion with the box, actually exist as a problem for ants as well. Known as the "circle of death", blind ants relying on the set of pheromones for direction of walking, may end up in an endless circular motion as shown in this clip ¹. CRABS portraits the same image. The e-pucks rely heavily on the set of IR and its intensity to get feedback, which gives no room for reasoning on a higher level than the local information around them at that exact moment.

Another behavior that show how e-pucks act similar to ants (act on the local information around them), is the avoidance behavior. As a child you've probably experienced ants avoidance behavior from trying to trap it. They will most often turn away or start to climb you. If you trap it enough times, you'll observe that it is doing the same thing over and over again. It is not thinking about previous action or what happened in the past, it only responds to the surroundings present right now. The recording of the avoidance design shows the same traits.

To claim that it is a biological plausible system would be taking it too far. Biological to the extent possible with e-pucks with concerns to the observed retrieving ant model is more correct. CRABS is a realistic system that shows how swarm intelligence in social insects emerge by using the platform of the box-pushing event.

... on Portability

The software of this system can easily be used in different context on different hardware. The actions from each behavior are not bound to input from e-pucks or Webots. The universal language of C and the separation of each behavior make it easy to port onto a different medium or reuse parts of the system in other software systems.

Optimal E-puck Amount

From table 7.15(a) we can see that every behavior performed better as more e-pucks participated in the box-pushing event, except for the goal behavior which had

¹<http://www.youtube.com/watch?v=prjhQcqiGQc>

an increase of stagnation as more e-pucks were tested. The optimal amount can't however be based on just the performance factor.

It is clear that eight e-pucks is a good candidate for the optimal e-puck amount if we look at performance and the time spent. It could be that nine or ten e-pucks would use even less time, but unfortunately we didn't have more robots. The results are also affected by the shape of the environment and box, so there are already boundaries to what can be optimal. If the box would have been round or space of the environment smaller, the results would be different. However, given this setup it comes down to the force and space necessary to perform well, which in this situation has to be between eight and ten. More e-pucks participating would only cause more stagnation.

7.3.2 Brooks' Subsumption as a Social Insect Architecture

The subsumption architecture has proven to be a great architecture for behavior-based robotic systems, and not just in this project. Researching on robotics and architectures, section 2, we were surprised to see the similarities of the purpose behind the design of subsumption and the thoughts on how ants are purely stimulated to perform certain actions.

Rodney Brooks stated that intelligent behavior is created by pure reaction to the perceived input and has no need for reasoning [21]. The subsumption architecture follows that pattern, which is also how Moser explained ants behaviors [44]. This is the main reason for why subsumption has worked so well with replicating behaviors of social insects. It goes hand in hand; the simple and few mechanisms in ants fit well into a fixed-priority scheme in a subsumption architecture where there should be no memory or reasoning, only a direct link between sensing and acting.

The simplicity of ants and Brooks' subsumption combines to be a low hardware demanding system, which is crucial for simple robots. It is difficult to see how other more complex architecture, such as TuringMachine, can give the same benefits and follow the pattern of how ants react. CRABS has perhaps the future potential of being scaled down onto a small chip in silicon-based units, because of the simplicity of this architecture. Although subsumption has proven its worth in a small social insect system, it should be mentioned that it is not easy to design. It gets increasingly more difficult to design the system as more behaviors are added, up to a point where it becomes nearly impossible.

7.3.3 Simplicity & Scaling

The true strength of the system has yet to be revealed as we have been limited to eight e-pucks. Following a biological pattern with decentralized control, no direct communication and no memory, we also made a system that will work with 5 and potentially a 1000 e-pucks without having to make changes to the system. It is only constrained by the space and shape of the environment.

Each e-puck is independent. It is dependent on others to complete the task, but not to make choices and actions. Systems that use direct communication or memory are constrained by bandwidth capacity or storage room on hardware devices. CRABS on the other hand makes no use of this and just like a colony of ants, we can potentially have a colony of e-pucks walking around in an environment retrieving food.

7.3.4 E-puck as Swarm Robots

One of the things that make swarm intelligence applicable to robotic projects is the few factors necessary in creating it. Using simple perceiving and action mechanisms, one is able to create a robust and flexible system. The hardware expenses of having complex systems and robots, is the major factor for why there aren't many. However, swarm intelligence has the simplicity that allows a project to rely on simple and cheap robots.

Nothing can compare to making your own tailored robot from scratch, but e-pucks general abilities and size actually fits quite well to swarm systems as this project has shown. The relatively cheap robot expense has allowed us to experiment on eight e-pucks, which has collectively accomplished the goal of retrieving a food source back to the hive. Not only has this project projected e-pucks swarm capabilities, but any simple robot with pushing and perceiving ability.

E-pucks have during this project proven to be feasible to work with in relation to swarm intelligence. The eight proximity and light sensors gives robustness, and the stepper motors haven't malfunctioned on any robots during this project. It is simple, but yet sufficient for incorporating stigmergy and feedback which has been most noticeable factor for this projects intelligence. Replicating ants through e-pucks has given us challenges with force and actions capabilities, but in general it provides a solid testing ground for any relatively simple systems mechanical wise.

Conclusion & Further work

In this report, we have explored swarm intelligence through a box-pushing task with physical robots called e-pucks. Research on social insects has been presented together with different ways of controlling autonomous robots, where combining this knowledge has been essential in our quest to make a biological plausible ant retrieving system.

Inspired by ants and behavior-based robotics, we have created the system CRABS. It is based on Brooks' subsumption architecture to control six different behaviors, from a fixed input-output scheme. The system is designed to easily handle adding or removal of behavior layers. Behavior modules can also be used separately and ported to other software or hardware platforms.

During this project we came across several hardware and software challenges investigating cooperative behavior. With the use of the simulation tool Webots, we were able to determine e-pucks' capabilities, and through this knowledge able to design and construct an artificial food source. This operated as the box-item in the box-pushing task.

Based on two types of sensors and two actuators (wheels), we had a strategy to accomplish the box-pushing task following the biological principles of social insects. The guidelines of the ant retrieving model made CRABS a self-organized system that given three or more e-pucks, will always succeed in retrieving the box back to the wall. The most remarkable view on this accomplishment is that is done through the use of only stigmergy and positive/negative feedback.

One of the things we've experienced throughout this thesis is that hardware is a more work demanding and inconsistent platform than your usual software simulation. Everything is not given, and although Webots provided helpful shortcuts, a lot of time and hard work was put down in order to get the system up and running. With that being said, we are pleased that we took the hardware rout and were able to test and validate our system on physical robots.

Further Work

As we observed CRABS emerge a global structure among the e-pucks we also saw possibilities to improve the system, and use it as a foundation to explore other factors of swarm intelligence.

Improvements to the system has already been designed, but still needs more tests and work before it becomes a part of CRABS. We tried to upgrade the intelligence of the converge behavior to lower the stagnation risk from when e-pucks apply force on the opposite side of a already moving box. The design was to always check for IR intensity changes at the moment of detection. By standing still for just a split second, the e-puck could tell if it should help from this angle or find a new spot. Early manual tests showed promising results as a moving IR target approaching the e-puck would make it turn and try to step out of the way. As mentioned, this is not a part of CRABS as configuration values to trigger the correct action and behavior still needs adjustments to operate dynamically for several scenarios.

We also tried to design a way to have an autonomous finish detection, instead of the three minute mark. Maybe because of lack in creativity, but it became a much bigger problem than we expected. The only good solution where we feel confident that each e-puck would stop retrieving the item when they actually finished, is to have a separate goal area which sends out a short range signal. This would imply extensions to the e-puck and environment, which we didn't have.

One of the things that would be interesting to see is how CRABS operates and how the e-pucks perform in different environments: In larger environments, with more food sources, larger groups, several goals, and with obstacles. Another interesting aspect is to try heterogeneous robots with concerns to exploring division of labor in social insect. The heterogeneous part could be done by limiting or implement extensions on some e-pucks. It would be interesting to see to what extent division of labor will emerge or would have to be statically implemented.

These explorations with CRABS might make it difficult to stay within the biological aspect, but stepping outside of the ant model has its benefits. One of the constraints with our subsumption architecture is that it is static and each time a behavior is added a more complex design is needed. If Brooks's subsumption could be adaptive in the sense of perhaps rearranging the priority of the behaviors during run time (or even when and how they trigger), you would have a dynamic system that could possibly adapt to changes in the environment.

References

- [1] The e-puck, a robot designed for education in engineering.
- [2] e-puck education robot. <http://www.e-puck.org/> (01.12.2010).
- [3] V-rep - Virtual Robot Experimentation Platform. <http://www.v-rep.eu/> (10.06.2011).
- [4] Sjriek Alers, Daan Bloembergen, Daniel Hennes, Steven De Jong, Michael Kaisers, Nyree Lemmens, Karl Tuyls, and Gerhard Weiss. Bee-inspired foraging in an embodied swarm (Demonstration). *Computational Intelligence*, pages 1311–1312, 2011.
- [5] E a Antonelo, B Schrauwen, and D Stroobandt. Event detection and localization for small mobile robots using reservoir computing. *Neural networks : the official journal of the International Neural Network Society*, 21(6):862–71, August 2008.
- [6] Jannik Berg and Camilla Haukenes Karud. Computer science - Specialization project Swarm robotics. Technical Report 16, Trondheim, 2011.
- [7] S N Beshers and J H Fewell. Models of division of labor in social insects. *Annual review of entomology*, 46:413–40, January 2001.
- [8] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence; From Natural to Artificial Systems*. Oxford University Press, New York, 1999.
- [9] R. Brooks. Visual map making for a mobile robot. *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, pages 824–829, 1985.
- [10] Scott Camazine, Jean-Louis Deneubourg, Nigel R Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau. Self-organization in biological systems, 2001.
- [11] Christopher M. Cianci, Xavier Raemy, Jim Pugh, and Alcherio Martinoli. *Communication in a swarm of miniature robots: the e-Puck as an educational tool for swarm robotics*, pages 103–115. Springer-Verlag Berlin, Heidelberg ©2007, 2007.
- [12] Nikolaus Correll, Christopher Cianci, Xavier Raemy, and Alcherio Martinoli. Self-Organized Embedded Sensor / Actuator Networks for ”Smart” Turbines. *Robotics*, page 5.
- [13] Cyberbotics. Webots. <http://www.cyberbotics.com/products/webots/> (19.12.2010).

- [14] Department Of Knowledge Engineering (DKE) Maastricht University. Swarm-Lab; Robots, Agents & Interaction. <http://www.unimaas.nl/swarmlab/> (10.12.2010).
- [15] C Detrainl, L Chrétienl, J L Deneubourgl, S Goss, and N Franks. THE DYNAMICS OF COLLECTIVE SORTING ROBOT . LIKE ANTS AND ANT . LIKB ROBOTS. *Small*.
- [16] P. D’Ettorre and J. Heinze. Sociobiology of slave-making ants. *Acta ethologica*, 3(2):67–82, April 2001.
- [17] Bruce Randall Donald. Artificial Intelligence On information invariants in robotics *. *Artificial Intelligence*, 3702(94), 1995.
- [18] Marco Dorigo, Elio Tuci, Roderich Groß, Vito Trianni, Thomas Halva Labella, Shervin Nouyan, Christos Ampatzis, Jean-louis Deneubourg, Gianluca Baldassarre, Stefano Nolfi, Francesco Mondada, Dario Floreano, and Luca Maria Gambardella. The SWARM-BOTS Project. *Robotics*, pages 31–44, 2005.
- [19] Frederick Ducatelle, Gianni A Di Caro, and Luca M Gambardella. Cooperative Self-Organization in a Heterogeneous Swarm Robotic System Cooperative Self-Organization in a Heterogeneous Swarm Robotic System. *Artificial Intelligence*, page 14, 1988.
- [20] Eliseo Ferrante. *The Design of a Modular Behavior-based Architecture*. PhD thesis, UNIVERSITÄ ’E LIBRE DE BRUXELLES, 2009.
- [21] Dario Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence*.
- [22] Nigel R Franks and Tom Richardson. Teaching in tandem-running ants. *Nature*, 439(7073):153, January 2006.
- [23] Simon Garnier, Jacques Gautrais, and Guy Theraulaz. The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31, July 2007.
- [24] David Gordon. *Collective Intelligence in Social Insects*.
- [25] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76(12):579–581, December 1989.
- [26] Tove Gustavi, Xiaoming Hu, and Maja Karasalo. Multi-Robot Formation Control and Terrain Servoing with Limited Sensor Information. page 6, 2005.
- [27] Ulrik Heger. *Verden er magisk*. Arkania forlag, Oslo, 2007.
- [28] Owen Holland, Chris Melhuish, Coldharbour Lane, and Bristol Bs. Stigmergy , self-organisation , and sorting in collective robotics Faculty of Engineering University of the West of England. *English*, pages 1–37.
- [29] Ge Julian and S Cahan. Undertaking specialization in the desert leaf-cutter ant *Acromyrmex versicolor*. *Animal behaviour*, 58(2):437–442, August 1999. <http://www.ncbi.nlm.nih.gov/pubmed/10458895> (31.08.2010).

- [30] Michael Kaisers and K. Tuyls. Replicator Dynamics for Multi-agent Learning: An Orthogonal Approach. *Adaptive and Learning Agents*, page 8, 2009.
- [31] James Kennedy, Robert C. Eberhart, and Yi Shi. *Swarm Intelligence*, 2001.
- [32] C. Ronald Kube. Kube's Research: Collective Robotics.
- [33] C. Ronald Kube and Hong Zhang. *Collective Robotic Intelligence*, 1992.
- [34] C Ronald Kube and Hong Zhang. Collective Robotics: From Social Insects to Robots. *Adaptive Behavior*, page 16, 1993.
- [35] C. Ronald Kube and Hong Zhang. Controlling Collective Tasks With An ALN, 1993.
- [36] C Ronald Kube and Hong Zhang. Stagnation Recovery Behaviours for Collective Robotics 1 Introduction 2 Group Transport by Ants. *Simulation*, pages 1–8, 1994.
- [37] Nyree Lemmens, Steven De Jong, and Karl Tuyls. Bee behaviour in multi-agent systems : A bee foraging algorithm. page 12, 2008.
- [38] Nyree Lemmens, Steven De Jong, Karl Tuyls, and Ann Nowé. Bee System with inhibition Pheromones. 2007:11, 2007.
- [39] V.J. Lumelsky. A comparative study on the path length performance of maze-searching and robot motion planning algorithms. *IEEE Transactions on Robotics and Automation*, 7(1):57–66, 1991.
- [40] Douglas C. MacKenzie and Tucker R. Balch. Making a clean sweep: Behavior based varcuuming. pages 1–6.
- [41] Stephane Magnenat, Markus Waibel, and Antoine Beyeler. Enki - The fast 2D robot simulator. <http://home.gna.org/enki/> (10.06.2011).
- [42] Wesentliche Merkmale. GaAlAs Infrared Emitter (880 nm) Lead (Pb) Free Product - RoHS Compliant SFH 485 P SFH 485 P. pages 1–7, 2009.
- [43] Moffett. Cooperative Food Transport by an Asiatic Ant. 4:386, 1988.
- [44] J. C. Moser. *Control of insect behavior by natural products*. 1970.
- [45] NC State University. The dance language of the honey bee.
- [46] F.R. Noreils. An architecture for cooperative and autonomous mobile robots. *Proceedings 1992 IEEE International Conference on Robotics and Automation*, (1):2703–2710, 1992.
- [47] Gustaf Olson. Emergent Schooling Behavior in Fish. pages 1–13.
- [48] Erol Sahin and Alan Winfield. Special issue on swarm robotics. *Swarm Intelligence*, 2(2-4):69–72, August 2008.

- [49] Thomas Schmickl, Ronald Thenius, Christoph Moeslinger, Gerald Radspieler, Serge Kernbach, Marc Szymanski, and Karl Crailsheim. Get in touch: cooperative decision making based on robot-to-robot collisions. *Autonomous Agents and Multi-Agent Systems*, 18(1):133–155, August 2008.
- [50] K. Singh and K. Fujimura. Map making by cooperating mobile robots. *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 254–259, 1993.
- [51] D.J. Stilwell and J.S. Bay. Toward the development of a material transport system using swarms of ant-like robots. *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 766–771, 1993.
- [52] Francisco Varela. *Toward a practice of autonomous systems : proceedings of the First European Conference on Artificial Life*. MIT Press, Cambridge Mass. [u.a.], 2. print. edition, 1994.
- [53] Bill Wilson. *The Machine Learning Dictionary*.
- [54] E. O. Wilson, N. I. Durlach, and L. M. Roth. Chemical Releasers of Necrophoric Behavior in Ants. *Psyche*, 65(4):108–114, 1958.
- [55] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, second edition, 2009.
- [56] Tristram D. Wyatt. Pheromones and Animal Behaviour - Communication by Smell and Taste. 13(4):28, December 2003.
- [57] Pengchong Zhao, Alei Liang, Liang Liu, Ying Chen, Haibing Guan, and Xinan Yan. An Exploration Algorithm for a Swarm of Homogeneous Robots. *2009 International Conference on Computational Intelligence and Software Engineering*, pages 1–6, December 2009.

Appendices

Project Settings

A.1 Operating Systems with Webots

For our program development we used Webots 6.3 beta 1. This can be downloaded and installed from Webots' webpage: <http://www.cyberbotics.com>. Our program has been developed by using Windows vista and Windows 7, but is also supported by other OS:

- Windows: Webots runs on Windows 7, Windows Vista and Windows XP.
- Linux: Webots is mainly supported by the latest Ubuntu release, but it also run on other larger Linux distributions like RedHat, Mandrake, Debian, Gentoo, SuSE, and Slackware.
- Macintosh: Webots is also supported of Machintosh such as Mac OS X 10.5 and Mac OS X 10.6.

A.2 Development Tools

This chapter presents all the different tools we have used in our project as well as options in terms of coding and simulators. We would like to highlight that the reader is in no way obligated to use the same tools to be able to build further on this project. We will however suggest it, as the experience presented in this report should help save time especially on the setup. If you want to go straight for replicating the setup of our projects, as almost all software we have used is open source and free, you can follow the user installation guide in section A.3.

A.2.1 Software Development Environment

The use of integrated debugging environments is up to the user, but using a well-known IDE always has its benefits. From writing in a clean editor like notepad where one has no help, there are several tools that will provide useful features when programming. Whatever the choice, try to make sure a source code editor, compiler/interpreter, build automation and a debugger are present. A helpful addition is to see if there is a version control system integrated or plug-in available.

Eclipse

The programming tool we choose for this project was the multi-language software development environment Eclipse which is an open source IDE. Booming from late 2001, the stated goals of Eclipse was to create a robust, fully featured, commercial-quality industry platform for the development of highly integrated tools, which they stayed and stays true to. Their focus on the workbench, plug-in-tools and technology research has really paid off all the way down to the average user.

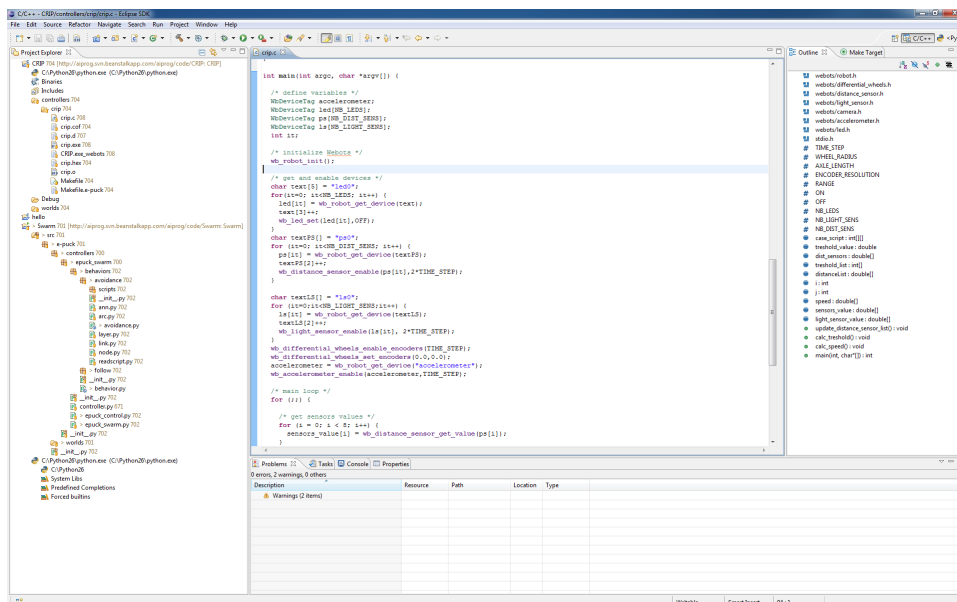


Figure A.1: Eclipse workbench

It contains the features you would expect from an IDE; syntax-highlighting editor,

thread-aware debugger, incremental code compilation, class navigator, file/project manager and code refactoring to name a few. The feature which has really helped this project is that Eclipse is platform and language neutral. From the integrated update/install new software option one can easily make projects with different programming language like C, C++, Python, Fortran, PHP, Ruby etc (default installation provides Java).

A.2.2 Simulation Development Environment

When operating from software to hardware, one of the essential processes is being able to prototype, debug and test the system at a early stage of the project as errors and faults can be very costly. Simulation software is a key in this development as it is based on imitating real phenomenon through mathematical formulas and should give a good indication of how the system is operating. Since we chose to use the e-pucks as our autonomous mobile robots, there were mainly three different simulators available to us with pre-defined modules implemented for the e-puck.

Enki

Enki is an open source robot simulator [41]. The simulation takes place in a two-dimensional world where collision and limited physics support are included. There are four standard robots implemented in Enki where the e-puck is one of them. Also creating your own custom robot is possible in Enki, but when it comes to e-puck, one can take advantage of these already implemented features:

- IR distance sensors
- Bluetooth
- Camera
- Scanner turret
- LED
- Wheels

Webots

Webots is a mobile robot simulator that is used to modulate, program and simulate mobile robots [13]. It has both similarities and differences compared to Enki. Both have some standard robots included in the system, like the e-puck. However, in contrast to Enki, Webots operates in a three-dimensional world with more physics, and henceforth is more similar to the real world. The features Webots can offer for the e-pucks are:

- IR distance sensors
- Light sensor
- LED

- Wheels
- Camera
- Accelerometer
- Bluetooth
- Speaker
- Microphone

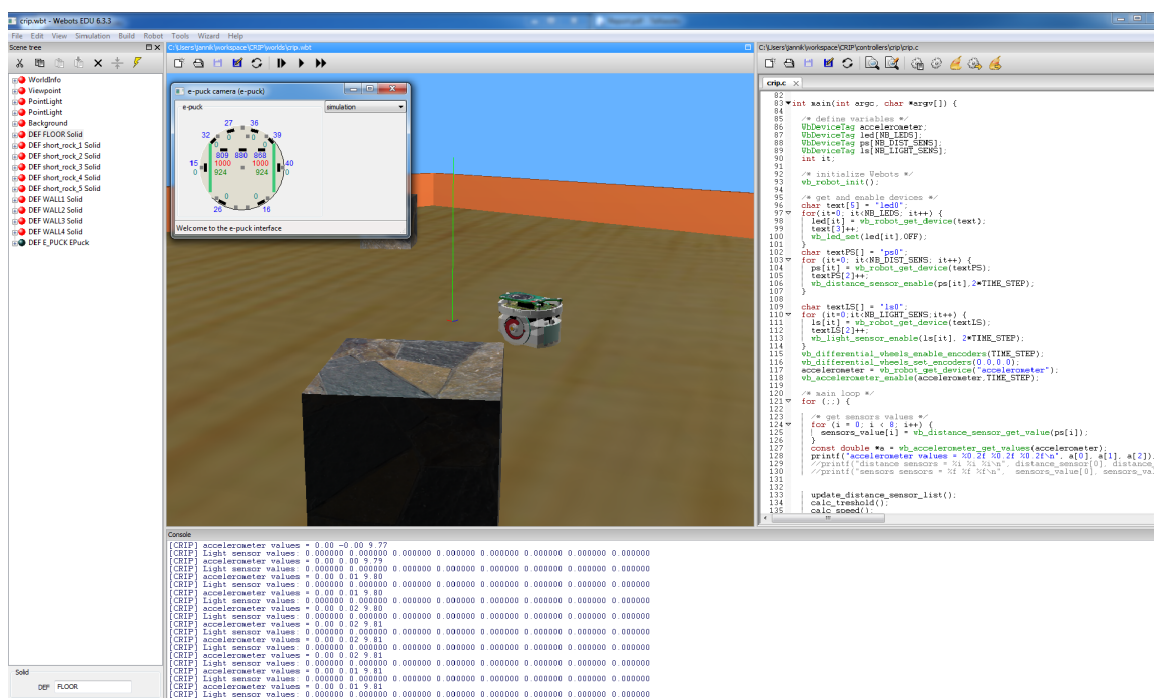


Figure A.2: Simulation and editor mode in Webots

Having only used Webots, it becomes perhaps easy to recommend, but it is truly a good tool. Webots provides a simulator that is very easy to understand, easy to use and contain support for several programming languages. One of the features we have used for this project has been the control interface of Webots. In the simulator you'll find a control window, which keeps track of sensors for every e-puck with a nice graphical user interface. You'll also find a Botstudio where one can graphically program the e-puck and programming graphically or not, you'll find already implemented example to get you started. For debugging, the remote-control possibility gives the user the chance to read real sensors values from the hardware environment. Note: C is the only language where you are able to cross-compile your code to an e-puck, other languages can only remote-control a physical e-puck through Bluetooth.

V-REP

V-REP (Virtual Robot Experimentation Platform) is another three-dimensional robot simulator[3]. The difference between this simulator and Webots is that V-REP is based on more distributed control architecture. This means that it is very versatile and handles multi-robot applications very well. V-REP also gives you the opportunity to work on smaller parts of the system, like sensors, mechanisms, etc. Being free of charge for students private computers, well documented and much used, this seems to be a great tool for robot development. However, in the case of this project we chose simplicity and own experience over complexity and unknowns.

A.3 User & Installation Guide

The start of any project requires a lot of research in the beginning, some more than others. When reading reports and articles from others over the course of this project, one of the things we came across as lacking, is an explanation of the setup of the project. This is of course something that might not be related or relevant to the scientifically cause of the project, as procedure and results usually would be, but something helpful for anyone interested in using this project in their work, or continue and be able to run this very project. Before starting to follow these instructions on how to setup this project, take a peek at the tool section A.2 to get an overview of your options. Remember, the following only presents our way of setting up the project with concerns to our code, but there is many ways to make it run. However, the webots guidance should be common no matter how the setup is done, so this guide could possibly give answer to any problems you may have encountered.

A.3.1 Installing Webots

- First step is to download the cyberbotics simulator Webots from <http://www.cyberbotics.com/download>. If you are a part of NTNU you can use the floating licenses which gives full access to Webots, but either way, cyberbotics provide a 30 day fully featured free trial.
- If you follow the standard installation procedure of Webots when using Windows, the default destination location will be either

C:\Program Files (x86)\Webots or C:\Program Files\Webots

depending on whether you are running Windows 64 or 32-bit. The reason is that cyberbotics has upon until version 6.3.3 only made Webots for 32-bit Windows.

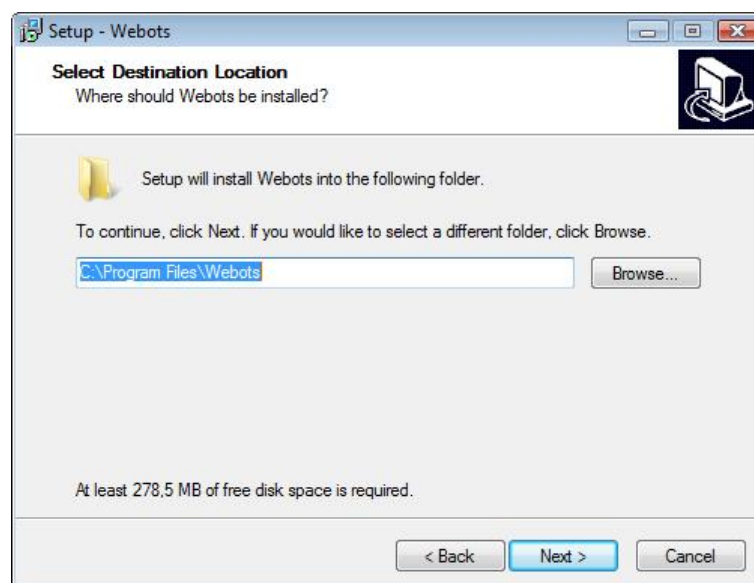


Figure A.3: Installing Webots into default location on Windows Vista 32-bit

- If you do have a license, the location of the file should be put inside the folder:

C:\Program Files\Webots\resources

A.3.2 Installing Eclipse

As explained in the tool development guide A.2 the Eclipse IDE is a very powerful, flexible and versatile software development tool. Seeing as Webots handles multiple languages, this installation guide will take a neutral side at first and expect that the classic version of Eclipse is downloaded. Downloading the classical version will give you the Eclipse platform, Java Development Tools and the most important part for continuing this guide; the Plug-in Development Environment. Remember that you have to have installed the Java Development Kit to be able to run Eclipse and develop applications, and in our case it will give us the ability to run the "plug-in installation" from the Eclipse workbench.

- Download Eclipse Classic from <http://www.eclipse.org/downloads/>
- If Eclipse complains about not finding a JRE/JDK/Java virtual machine - download and install it from: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Make sure you can run Eclipse and step into the workbench

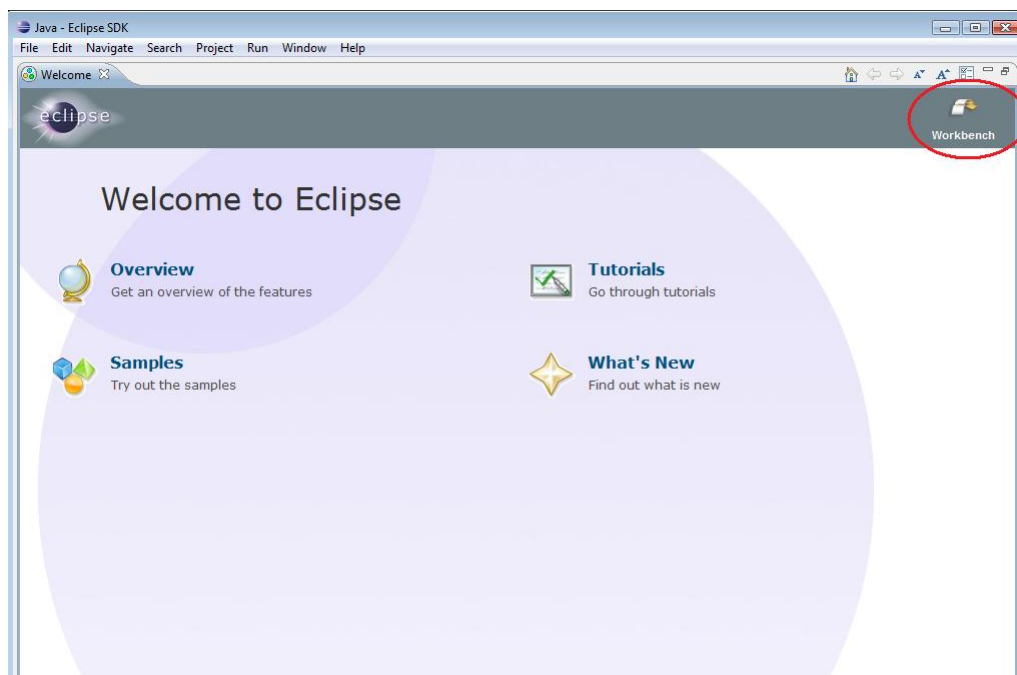


Figure A.4: The startup splash screen when running Eclipse for the first time

Subversive

This part will help you get the Eclipse subversive (SVN) which is a revision control system designed to work well for source code. It will be installed by using the integrated plug-in installation from the workbench.

- When inside the workbench and java perspective, locate the install environment: Help→Install New Software

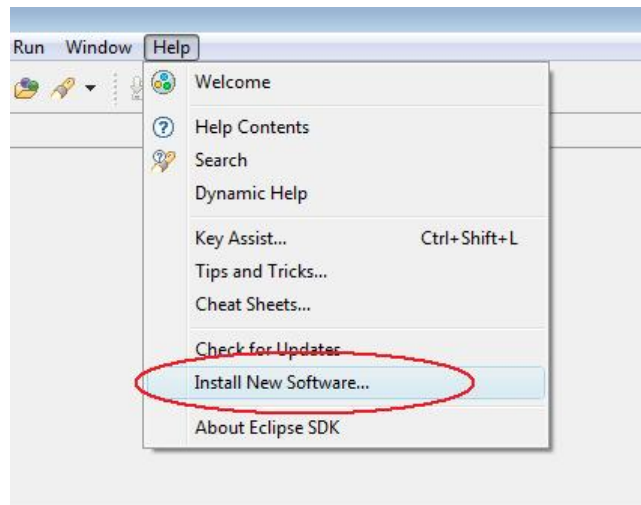


Figure A.5: When inside the workbench, plug-ins can be installed from the help menu

- As of 24.jan 2011, it is the subversive update site: <http://download.eclipse.org/technology/subversive/0.7/update-site/> which provides the latest release. In most cases this site is already imported by Eclipse and can be found under "Available Software Sites". If not, find the download site somewhere inside here <http://www.eclipse.org/subversive/> and use the "Add" function.

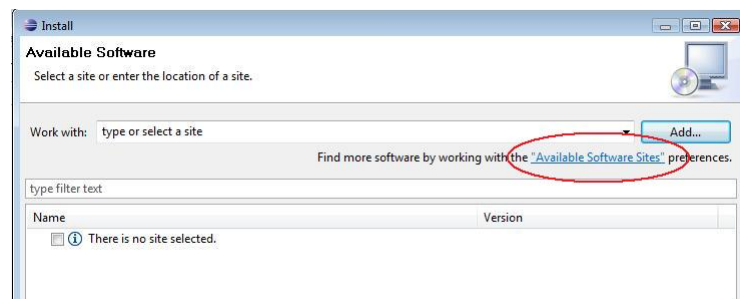


Figure A.6: Already imported software sites

- Enable the subversive software site and select it from the drop down menu.

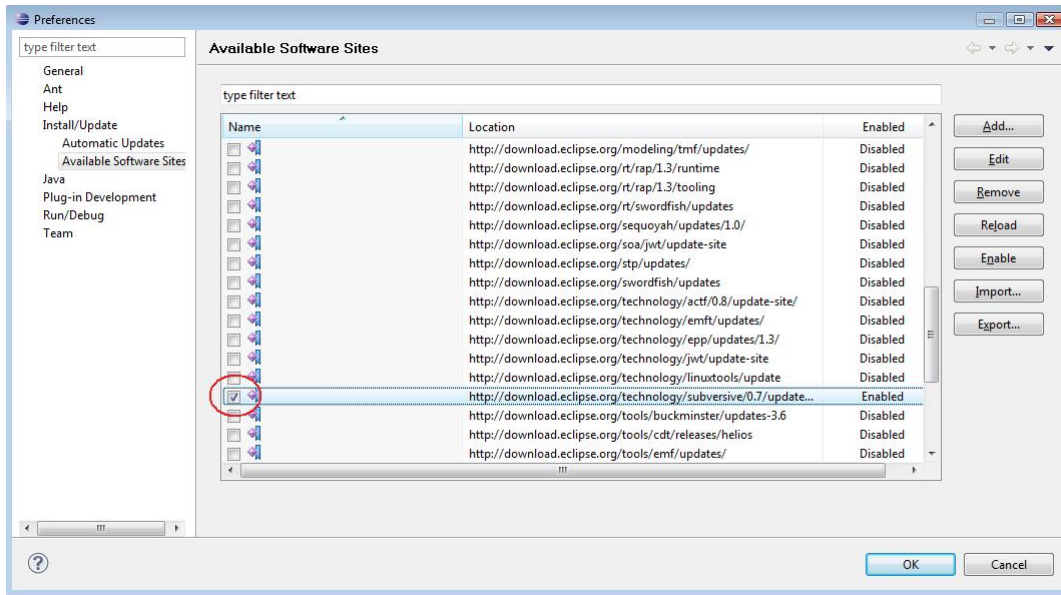


Figure A.7: Location and enabling of the subversive software site

- Although the site and install packages may change in the future there will most probably be optional extensions which is worth considering, but this guide will only choose the necessary parts to make it work as shown in Figure A.9. Only select the "SVN Team Provider" and install the software by pressing "Next".

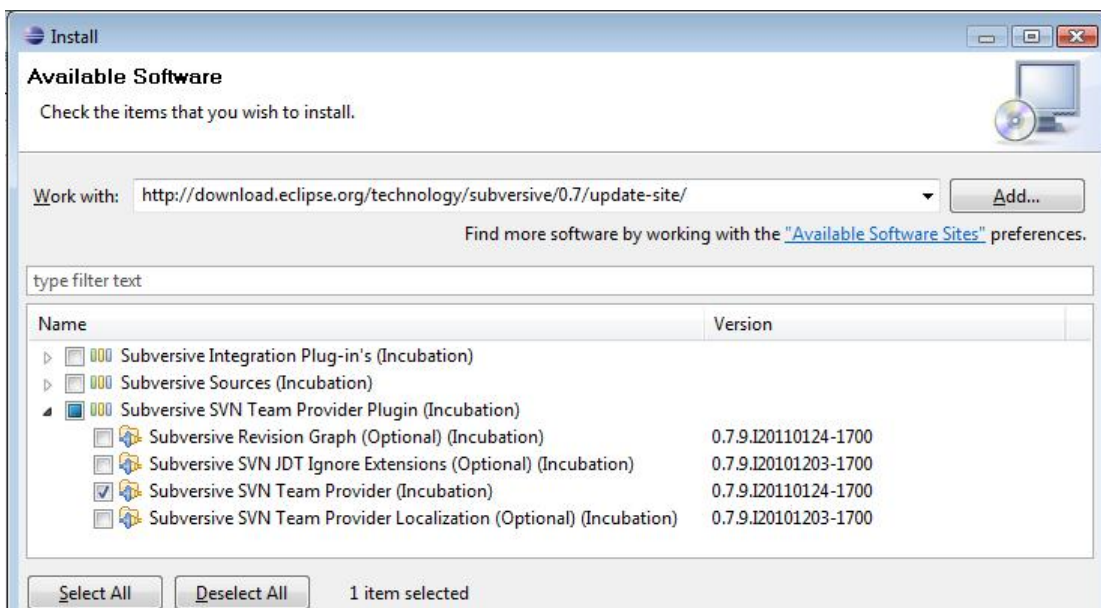


Figure A.8: Selection of the SVN Team Provider software

- To be able to use subversive you also need a subversive SVN connector which you will be prompted with the first time you try to use the subversive software. Make sure to choose the external connector which is compatible to your eclipse (with Helios 1.6.3 the SVN kit 1.3.X should work without any problems).

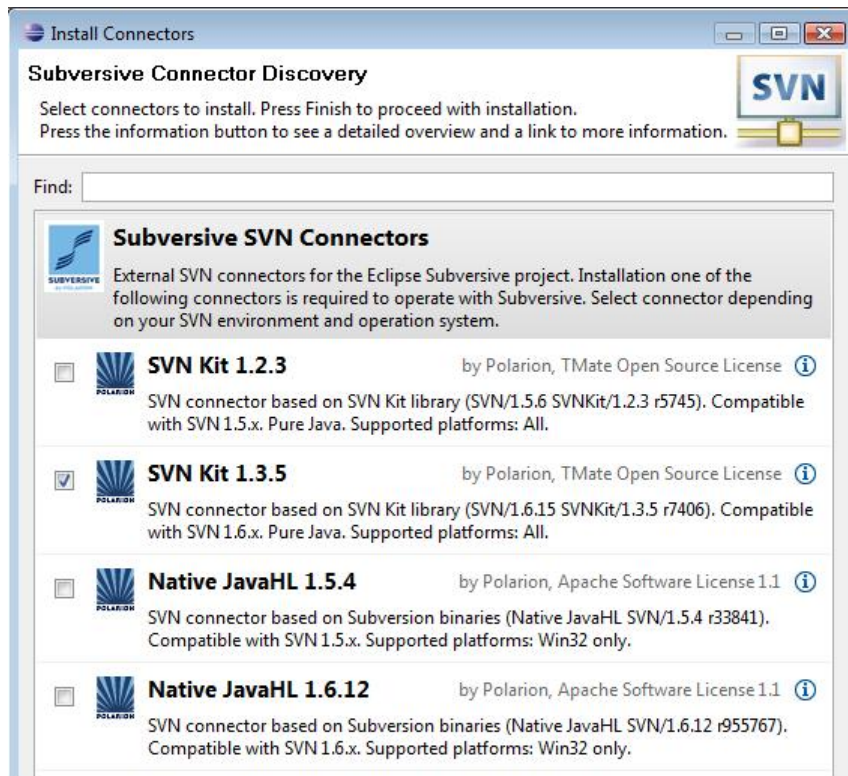


Figure A.9: Selection of connectors

- You should now, after following the installation procedure of the SVN connector, have a proper functional SVN ready to be used.

A.3.3 Using Python with Eclipse and Webots

As earlier experience with Webots was done with eclipse and python, we figured it could be useful with a setup guide for those who decide to work further with Python and Webots - although we do not recommend it for physical e-puck use, because of Bluetooth issues. This guide will first get python working with eclipse, before making the python project work with Webots.

Installing Python

- First step to get Python working with eclipse and Webots is to download the Python environment from: <http://www.python.org/getit/>.
 - Note: Python 2.7.1 does not yet work with Webots and you might have to choose a earlier version to make it work. Python 2.6.6 <http://www.python.org/download/releases/2.6.6/> is what we have been using.
- Follow the Python installation setup and keep in mind where you select the destination directory. The standard location works fine.

C:\Python26

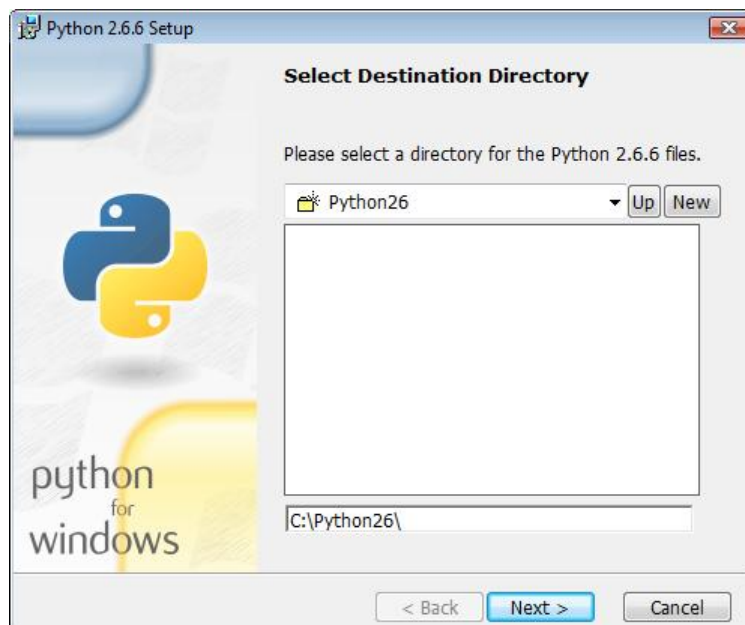


Figure A.10: Choosing the location of the python environment

- The next step is to make python work with windows, which is done through adding the python location path in the "Environment Variables".

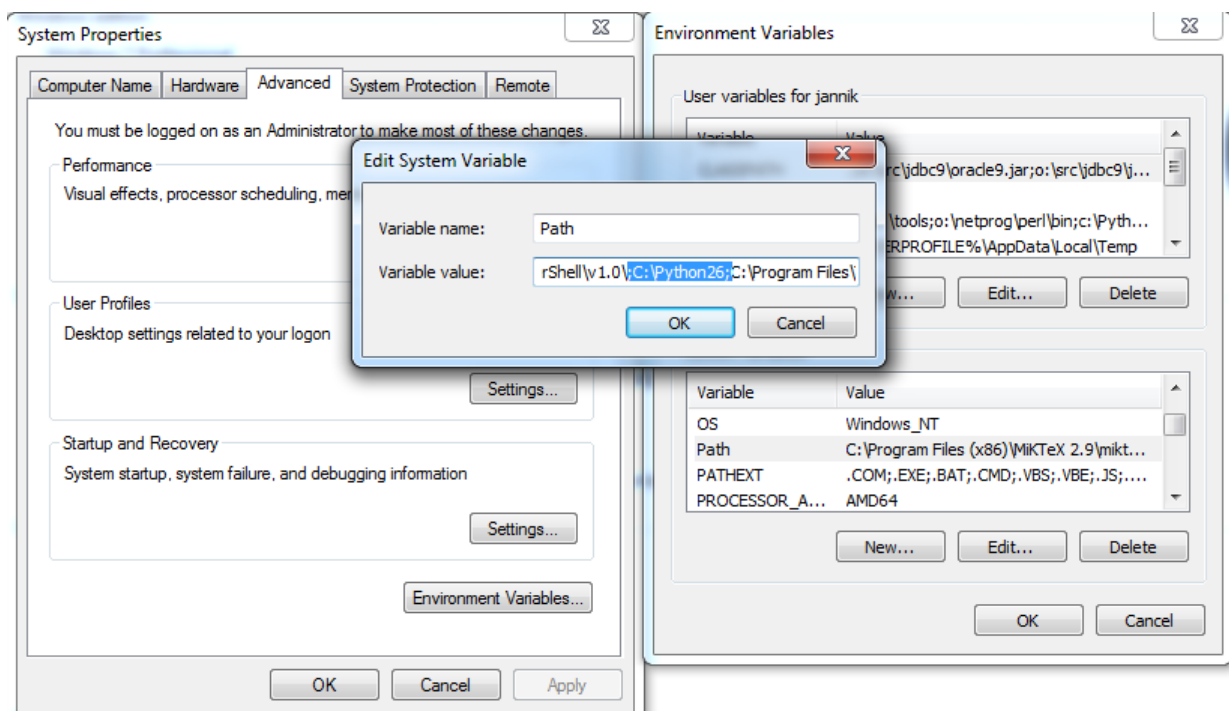
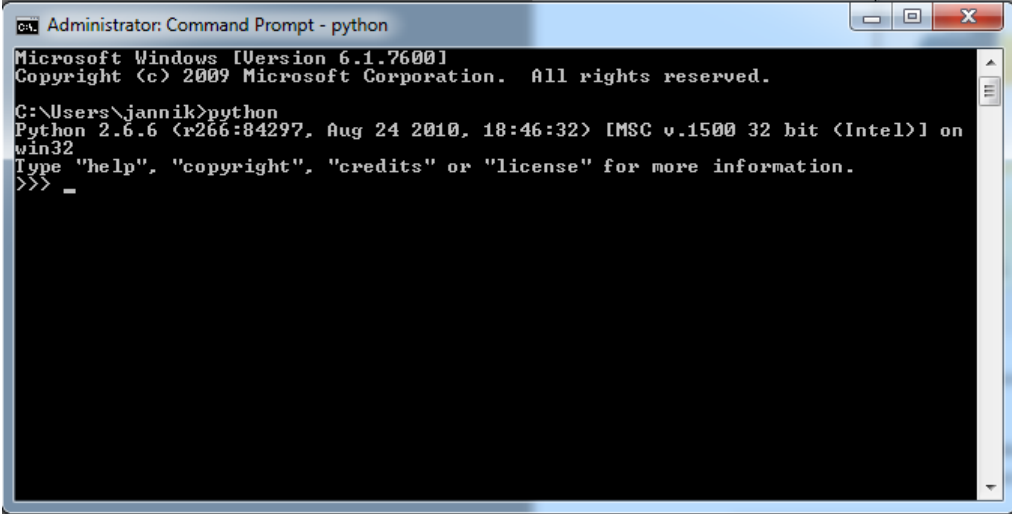


Figure A.11: Adding the python location into the environment path of windows

- As figure A.11 shows, you'll find the place to add the python path under: System Properties→Environment Variables→System variables→Path. Once inside the path frame, add the location of your python installation.

- All you have to do to see if the python installation is working correctly, is to type "python" in the command window.



```
Administrator: Command Prompt - python
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\jannik>python
Python 2.6.6 (x266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Figure A.12: Testing the python installation

- You should get a response like the one on the figure above, which means that windows recognize python and that it is ready to use.

Python with Eclipse

Getting python to work in Eclipse is done through using much of the same procedures as we have already shown with Subversive A.3.2, but some new configurations has to be made.

- Using the same procedure as for Subversive, download and install the python plug-in pydev <http://pydev.org/updates> (This have to be added manually and will not be found under "Available Software Sites"). For more information about the latest release see <http://pydev.org/download.html>

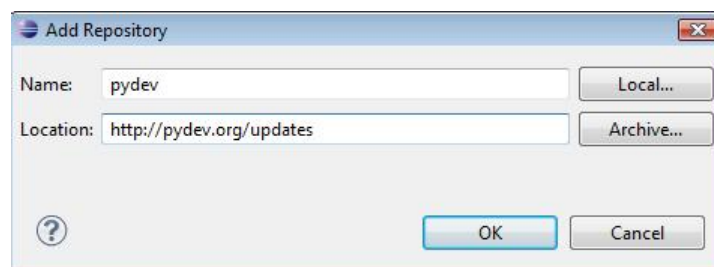


Figure A.13: Adding pydev to the repository of available software sites

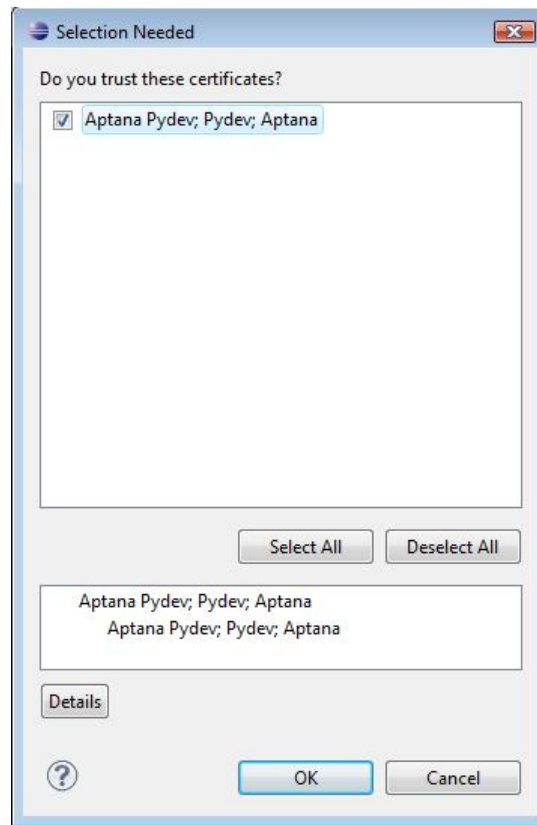


Figure A.14: Pydev certificate

- When installing the plug-in pydev, you are prompt with trust issues related to the pydev software. You will have to agree on this to be able to complete the installation.
- Once the installation is complete and Eclipse has restarted, the python platform perspective can be chosen under Window→Open Perspective→Other...→Pydev.
- On how to use python in general there are plenty to choose from on the web, but for starting up your first project this site will get you going: <http://technoticles.com/2010/04/13/pydev-installation-on-eclipse-tutorial/>

Python with Webots

- First, to make an e-puck run around in the environment of Webots you need to make a .wbt (world file) and make sure your code inherits from the controller.py file. The world file can be created by using the Webots editor, but the easiest way would be to copy one of the already existing e-puck world files located in

C:\Program Files (x86)\Webots\projects\robots\e-puck\worlds

- To initiate the code or even to be able to make code which can be used on the e-puck, you need take use of the controller.py file which can be found here:

C:\Program Files (x86)\Webots\lib\python

- The structure of the project is important to Webots. As shown in the figure below, the structure has to match the expectation of Webots, which is that every world file is located under a folder "worlds" and on the same level as the package of the controller.

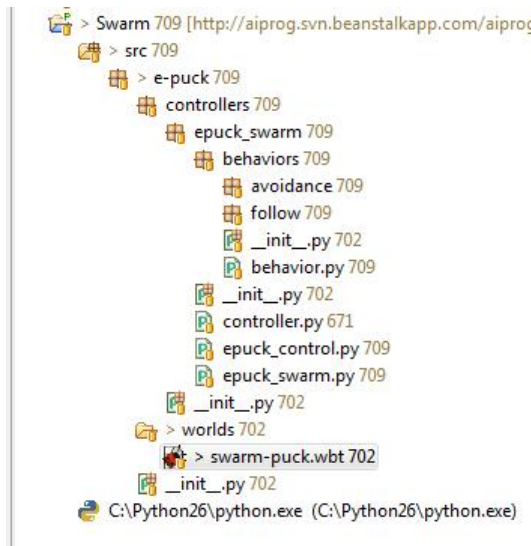


Figure A.15: The code structure of our python project

- It is also important that the controller name in the world file matches the.py file of where the controller is initialized.

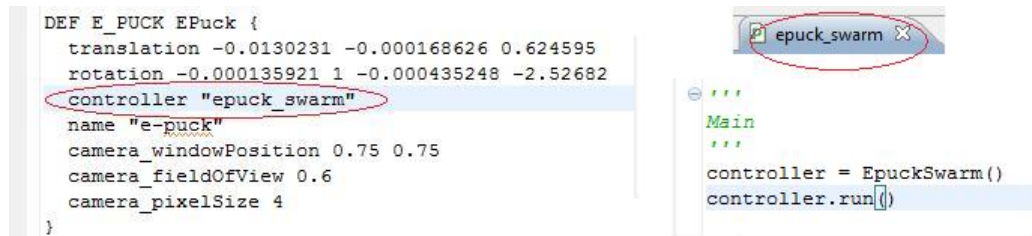


Figure A.16: The setup of the e-puck in the world file must correlate to the location of the main .py file

A.3.4 Using C with Eclipse and Webots

This project was done in C because of the experienced limitation of python on the real e-puck. We hope this can be used as a setup guide which will help you save time and continue working on this project. This guide will first get C working with Eclipse, before showing how a C project can be setup to run in Webots.

Installing C compiler

Getting C code to compile in Windows follows much of the same traits as the previously explained python installation. This guide will show you have to install

the GNU Compiler Collection (MinGW) which works really well with Windows. (Any C compiler will do).

- Download the latest installer release of MinGW from <http://sourceforge.net/projects/mingw/files/> and look for:

Looking for the latest version? [Download mingw-get-inst-20110211.exe \(575.4 KB\)](#)

Figure A.17: Link to the latest MinGW release

- When installing it, make sure you remember the destination of your installation. The default location is usually the one with least issues:

`C:\MinGW`

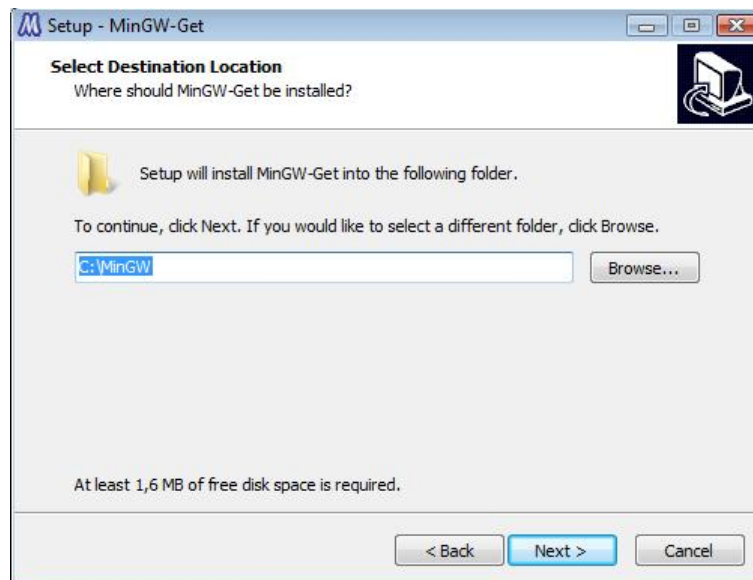


Figure A.18: Installing MinGW into the default location

- You also get the opportunity to install several compilers (C++, fortran, etc), but the default C compiler is enough for this project.
- The next step is to make it work with Windows, which is done by adding the MinGW bin path to the "Environment Variables". If you have installed MinGW in the default location, the path you must add is:

`C:\MinGW\bin`

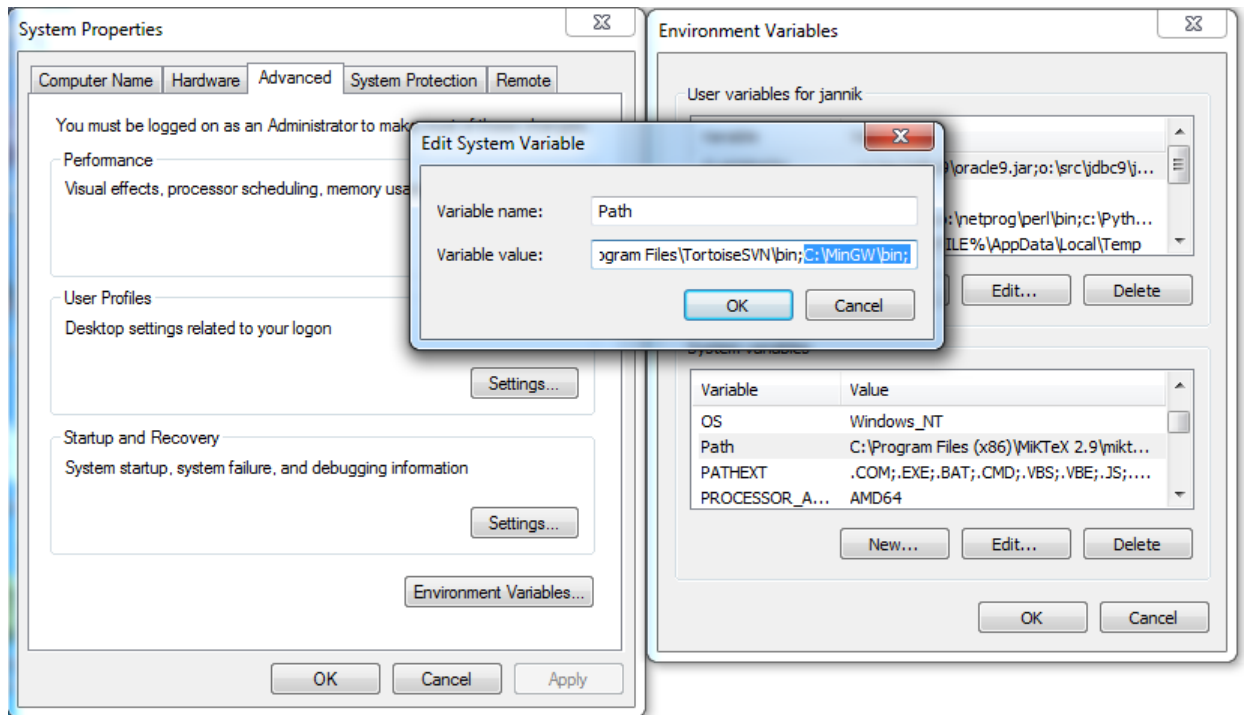


Figure A.19: Adding the bin path

- The final step is to check to see that it is working. This can be done by prompting the Command window with:

```
mingw32-make --version
```

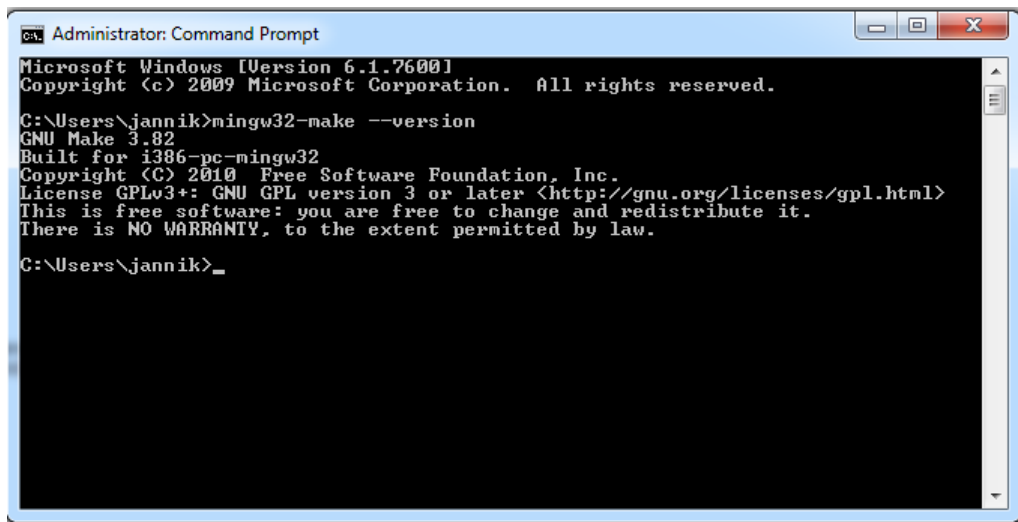


Figure A.20: The correct feedback from testing the C compiler

- The compiler is ready for use if the feedback relates to figure A.20.

C with Eclipse

Getting C to work with the classic Eclipse (Java), a plug-in installation is required, as explained for python in section A.3.3.

- The C plug-in project we have been using is the CDT project which is made for eclipse to integrate the C and C++ Development Environment. With the Helios installation of eclipse the plug-in can be found under "Available Software Sites" as <http://download.eclipse.org/tools/cdt/releases/helios>
- Once the installation is complete and Eclipse has restarted, the C/C++ platform perspective can be chosen under Window→Open Perspective→Other...→C/C++.
- If you are new to C and Eclipse, a thorough tutorial can be found here: <http://www.cs.umanitoba.ca/~eclipse/7-EclipseCDT.pdf>

C & Eclipse with Webots

Creating a Webots project with C from the Eclipse platform requires some few adjustments before making it work. One of the things that will help you create your own e-puck code with Webots is to take a look at the premade e-puck projects from cyberbotics. Several e-puck project can be found here:

`C:\Program Files (x86)\Webots\projects\robots\e-puck\controllers`

- To access the e-puck features (sensors and actuators) in Webots and on the physical robot, you need to include files from Webots:

`C:\Program Files (x86)\Webots\include\controller\c`

- These header files were included in CRABS:
 - include `webots/robot.h` - To initialize a robot/Webots event
 - include `webots/differential_wheels.h` - Access to right and left wheel
 - include `webots/distance_sensor.h` - Access to proximity sensors
 - include `webots/light_sensor.h` - Access to IR sensors
 - include `webots/led.h` - Access to all LEDs.
- To be able to compile your code you also need to add the *Controller.lib* library to your eclipse project:

`C:\Program Files (x86)\Webots\lib`

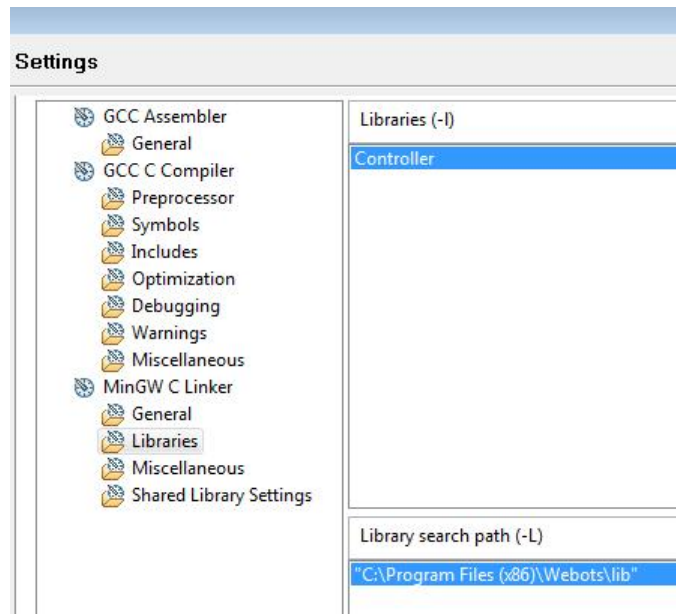


Figure A.21: Adding the path of the Controller.lib file is necessary to be able to compile the code.

- Make sure your code structure follows the expected setup for Webots (explained here: section A.3.3). Important factors are the location of your world file and the controller name inside it.
- The code can be opened and simulated in Webots by opening the .wbt file (world file).
- To run, Webots needs a makefile, which Webots will ask to make for you when you are trying to compile the code for the first time in Webots.
- The makefile is important. This file needs to be configured in order to be able to structure your code into several C-source files. Simply state the file name.c in the bracket C_SOURCES:

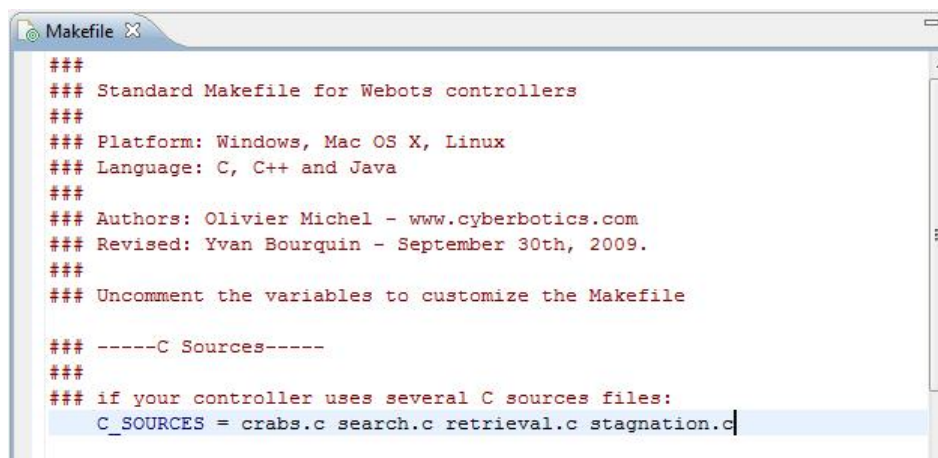


Figure A.22: The makefile is also essential for dividing the software into several files

- Everything should now be up and running in the simulator.

A.3.5 Using the real e-puck with Webots

As explained through this project there is two ways of interacting with the physical e-puck; remote-control and cross-compilation. Remote-control works for all languages, running the software on your PC, receiving and sending signals to the e-puck through Bluetooth (controlling more than one has shown to give interference problems). Cross-compilation is transferring your software onto the e-puck, letting the e-puck's microprocessor handle everything. (Code needs to be in C)

Remote-Control

When testing certain features with the e-puck, the remote-control interaction is a good Webots tool. Using Bluetooth you have to pair your e-puck and computer (this is outside of Webots and is dependent on your OS) and make sure the e-puck is turned on to make a connection once inside the Webots simulator. Once connected, one is able to read sensor values, debug the code and do single time steps. Setup:

- Pair up the e-puck. A passkey is needed, which is the e-puck's ID (e-puck 2055 has passkey 2055). If passkey fails at this stage, pair without a passkey, and enter the passkey later when Webots is trying to access it.
- To be able to connect to the e-puck using Bluetooth, the e-puck has to have the default firmware - *firmware-1.4.3.hex*. That file can be found here:

`C:\Program Files (x86)\Webots\transfer\e-puck\firmware`

- This file can be uploaded by selecting the paired e-puck inside Webots: Tools→Upload to e-puck robot... (the hex file is selected after selection of e-puck has been made).
- The blue reset button on top of the e-puck needs to be pushed when prompted by Webots.
- Once finished the e-puck is ready for remote-control session.
- To establish a remote-control connection simply double click the e-puck inside the simulator or go: Robot→Robot window, and choose the outgoing port to the paired e-puck.

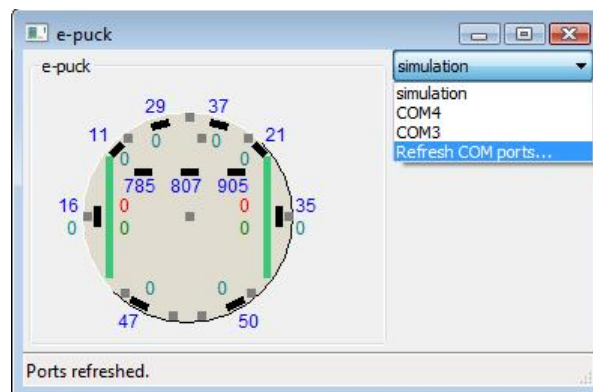


Figure A.23: The robot window of the selected e-puck. The correct COM port has to be chosen to establish a remote-control session.

- The robot window in Webots gives real e-puck sensor values when connected to a physical e-puck. Using the play, stop, step feature in Webots can also be used during remote-control.

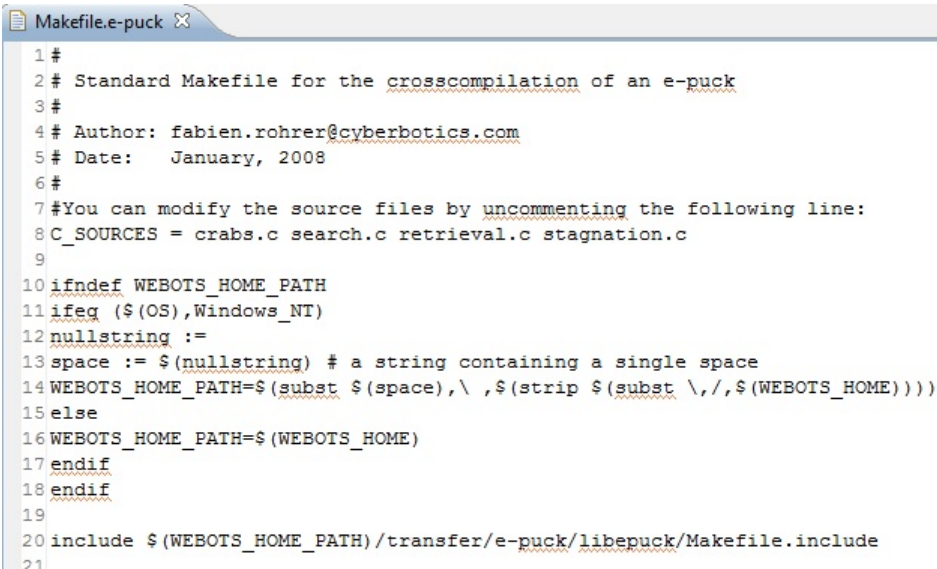
Cross-Compilation

Having several e-pucks interact with each other can cause interferences problems with the Bluetooth and result to lost connection. The solution is to use cross-compilation instead, where the software is run on the microprocessor of the e-puck. The drawback is that it is impossible to debug or read e-puck sensor values. Cross-compilation requires a separate makefile, generating a .hex file of your code and uploading it through Webots (Bluetooth is need for the uploading part).

- The makefile has to be named *Makefile.epuck* and be placed in the same folder as your normal makefile (should be the same folder as your code). A default *makefile.epuck* can be copied from this location:

C:\Program Files (x86)\Webots\projects\robots\e-puck\controllers\e-puck_cross

- Remember that this makefile also needs to include all separated source files.



```

1 #
2 # Standard Makefile for the crosscompilation of an e-puck
3 #
4 # Author: fabien.rohrer@cyberbotics.com
5 # Date: January, 2008
6 #
7 #You can modify the source files by uncommenting the following line:
8 C_SOURCES = crabs.c search.c retrieval.c stagnation.c
9
10 ifndef WEBOTS_HOME_PATH
11 ifeq ($(OS),Windows_NT)
12 nullstring :=
13 space := $(nullstring) # a string containing a single space
14 WEBOTS_HOME_PATH=$(subst $(space),\,$(strip $(subst \,/, $(WEBOTS_HOME))))
15 else
16 WEBOTS_HOME_PATH=$(WEBOTS_HOME)
17 endif
18 endif
19
20 include $(WEBOTS_HOME_PATH)/transfer/e-puck/libepuck/Makefile.include
21

```

Figure A.24: For Webots to be able to generate the .hex file, a *Makefile.epuck* is needed.

- To cross-compile your code into a .hex file, open the project in Webots and push the "cross-compile" icon or go Build→Cross-compile.
- A *projectname.hex* file should now be in your project folder, and can be uploaded to the e-puck through Webots: Tools→Upload to e-puck robot... The e-puck has to be paired with the computer as explained in the remote control section A.3.5.
- If there is no code errors, the e-puck should run the software once the upload has completed.

- The software is now on the e-puck, even if you turn it off/on. Follow the remote-control setup to reset the e-puck.

We hope that this guide has given an all in one package setup so you can get started without having to spend time figuring out all details from scratch.