

UNIVERSITY OF OSLO
Department of Informatics

Control System Development and Technological Investigation for a Climbing Robot in Offshore platforms

Master Thesis

Akbar Faghihi
Moghaddam
(Shahab)

February 2012



Acknowledgment

To my parents & family . . .

I want to first of all thank my parents and family whose sacrifices allowed me to be here and finish this work. I also want to specially thank my girlfriend, Yao Wang, who supported and helped me during the whole process of this work. In addition I would like to thank my supervisors and all those friends and classmates, whose fruitful discussions inspired me through my work. My dear classmates and close friends such as Magnus Lange, Mohammad Bagher (Puya) Afsharian, Aryan Esfandiari and Ashkan Mardanpour. At the end I would like to end this acknowledgment by mentioning my regards to Robotica Osloensis robotics student community whose members and resources were always to my help and inspiration.

Akbar Faghihi Moghaddam (Shahab), February 2011

Abstract

To improve human safety and environmental concerns, oil and gas industry is interested in using remote and autonomous robots instead of human workers on offshore platforms. This will also increase their revenue and allow operations in places where it is too difficult to operate in. This project is further development of a custom climbing robot called Walloid at University of Oslo, currently under development. Walloid is a 4 arm climbing robot with arms and grippers designed for possible later usage in offshore platforms. Through this project, as a contribution to the Walloid project, an end effector with gripping functionality and three climbing gaits with focus on optimization of speed were developed. Thereafter, the focus was on developing a control hardware capable of handling 12 motors and 24 encoders simultaneously. To achieve this a distributed embedded system consists of five micro-controllers (Arduino boards with Atmega AVR 8 bit) was designed and implemented with two interconnection protocols (ZigBee and RS-232). Based on the hardware design, a distributed control algorithm was designed to implement the earlier developed climbing gaits. This distributed navigation program supported remote controlling, semi-autonomy, repeating taught (logged) tasks, and power optimization algorithms to put idle parts into sleep mode. Due to absence of the physical robot, the evaluation of the work was done by self-developed simulation tools.

The power optimization algorithm, together with optimized climbing gaits reduced the power consumption of the system significantly.

Short Contents

Acknowledgment	i
Abstract	iii
1 Introduction	1
1.1 Introduction	1
1.2 Motivation	2
1.3 Summary	3
2 Background	5
2.1 Previous Work	5
2.2 Climbing Robots	7
2.3 Automation and Smart Agents	9
2.4 Offshore Platforms, Challenges and Opportunities	15
2.5 Summary	25
3 Walloid Robot	27
3.1 Ongoing Project	28
3.2 Walloid Hardware Components	28
3.3 Calculated Kinematics and Workspace	30
3.4 Review and Tech Upgrade of Walloid Robot	35
3.5 Summary	39
4 Top Level Perspective	41
4.1 Top Down Objectives	41
4.2 Analysis of offshore platforms as an area of application	42
4.3 Climbing Operation	42
4.4 Control Hardware, a Distributed Embedded System (DES)	43
4.5 Control Algorithm, a Distributed Navigation Program (DNP)	43
4.6 Summary	44

5	Development Process	47
5.1	Climbing Strategy and Design	47
5.2	Control Hardware, the Distributed Embedded System	58
5.3	Distributed Embedded System Design	66
5.4	Control Algorithm	75
5.5	Distributed Navigation Program (DNP) and Features	84
5.6	Simulation and conformability of data	92
5.7	Summary	96
6	Implemented Control Systems and Results	99
6.1	Offshore Industry Point Of View	99
6.2	Climbing Operation Results	100
6.3	Control Systems	103
6.4	Simulation	107
6.5	Summary	108
7	Robustness Issues	111
7.1	List of Issues	111
7.2	Blocked Paths	112
7.3	Positioning after Improper Shutdowns	113
7.4	Passive Joint Control	115
7.5	Power Interruption	116
7.6	Instability / Current Orientation	118
7.7	Offline Modus, Network-less Operation	119
7.8	Security Concerns	121
7.9	Summary	122
8	Conclusion	123
8.1	Conclusion	123
8.2	My Contribution	124
8.3	Further Works	125
A	Interviews	126
A.1	Anders Røyørøy	126
B	Visual Reports	128
B.1	3D Designs	128
C	Remainings	133
C.1	Climbing Operation	133
C.2	Hardware Issues	138
C.3	Software Issues	139

D Source Code	144
D.1 Control program, Java	144
D.2 Control algorithm, Arduino C	173
D.3 Simulation, Processing	192
D.4 Matlab, Workspace	203
 List of Figures	 217
 Bibliography	 225

Contents

Acknowledgment	i
Abstract	iii
1 Introduction	1
1.1 Introduction	1
1.2 Motivation	2
1.3 Summary	3
2 Background	5
2.1 Previous Work	5
2.2 Climbing Robots	7
2.3 Automation and Smart Agents	9
2.3.1 Autonomous Robots	9
2.3.2 Artificial Intelligence Methods	11
2.3.3 Embedded Systems	13
2.4 Offshore Platforms, Challenges and Opportunities	15
2.4.1 Motivations	15
2.4.2 Automation Opportunities in Offshore Platforms	20
2.4.3 Challenges in Automation of Offshore Platforms	22
2.5 Summary	25
3 Walloid Robot	27
3.1 Ongoing Project	28
3.2 Walloid Hardware Components	28
3.2.1 Encoder	29
3.3 Calculated Kinematics and Workspace	30
3.3.1 One Arm, Three Prismatic Joints	31
3.3.2 Calculations	31
3.3.3 Workspace	34
3.4 Review and Tech Upgrade of Walloid Robot	35

3.4.1	Adjoining Surface Climbing	36
3.4.2	Speed Issues	36
3.4.3	Design, Material and Methods of Production	37
3.5	Summary	39
4	Top Level Perspective	41
4.1	Top Down Objectives	41
4.2	Analysis of offshore platforms as an area of application	42
4.3	Climbing Operation	42
4.4	Control Hardware, a Distributed Embedded System (DES)	43
4.5	Control Algorithm, a Distributed Navigation Program (DNP)	43
4.6	Summary	44
5	Development Process	47
5.1	Climbing Strategy and Design	47
5.1.1	End Effectors	47
5.1.2	Climbing Gaits	50
5.2	Control Hardware, the Distributed Embedded System	58
5.2.1	On-board Motherboard	58
5.2.2	Micro-controllers, the embedded system	60
5.2.3	Centralized vs. Distributed Embedded Systems	61
5.2.4	Arduino Boards, the chosen embedded system	63
5.2.5	Alternatives	65
5.3	Distributed Embedded System Design	66
5.3.1	Cable Based Distribution	66
5.3.2	Wireless Distribution	70
5.4	Control Algorithm	75
5.4.1	Robot-Server-Client (RSC) Architecture and Development Tools	75
5.4.2	Hardware - Software Interaction	79
5.4.3	Development Process	80
5.5	Distributed Navigation Program (DNP) and Features	84
5.5.1	DNP Positioning Logics	84
5.5.2	Reading Sensor Data	84
5.5.3	Power Optimizer Feature	85
5.5.4	Remote Control System	87
5.5.5	Logging System	88
5.5.6	Semi-Autonomous System	91
5.6	Simulation and conformability of data	92
5.6.1	Implemented Simulations	92
5.6.2	Conceptual Simulations	95

5.7	Summary	96
6	Implemented Control Systems and Results	99
6.1	Offshore Industry Point Of View	99
6.2	Climbing Operation Results	100
6.3	Control Systems	103
6.3.1	Distributed Embedded System	103
6.3.2	Distributed Navigation Program	106
6.4	Simulation	107
6.5	Summary	108
7	Robustness Issues	111
7.1	List of Issues	111
7.2	Blocked Paths	112
7.3	Positioning after Improper Shutdowns	113
7.3.1	Zero Positioning	113
7.3.2	Current Positioning, the software oriented approach . . .	114
7.4	Passive Joint Control	115
7.5	Power Interruption	116
7.6	Instability / Current Orientation	118
7.7	Offline Modus, Network-less Operation	119
7.8	Security Concerns	121
7.9	Summary	122
8	Conclusion	123
8.1	Conclusion	123
8.2	My Contribution	124
8.3	Further Works	125
A	Interviews	126
A.1	Anders Røyrøy	126
B	Visual Reports	128
B.1	3D Designs	128
C	Remainings	133
C.1	Climbing Operation	133
C.2	Hardware Issues	138
C.3	Software Issues	139

D Source Code	144
D.1 Control program, Java	144
D.2 Control algorithm, Arduino C	173
D.3 Simulation, Processing	192
D.4 Matlab, Workspace	203
 List of Figures	 217
 Bibliography	 225

Chapter 1

Introduction

A journey of a thousand miles begins with a single step.
Lao-tzu, The Way of Lao-tzu, a Chinese philosopher (604 BC - 531 BC)

1.1 Introduction

No one could set a price on human life, neither health, nor being away from loved ones. Those who work in remote harsh working environments, endanger their lives, risk their health and spend most of their time away from their loved ones. Offshore platforms are one type of such working environment that endanger lives (61 deaths in 2 last incidents) [1,2]. This leads to an important question:

how could this danger be eliminated?

On the other hand, in 2006, Norwegian Oil Industry Association (OLF), published a report warning the offshore industry to start the process of Integrated Operation, where the main goal was to move staff to onshore. The report stated that in case of implementing this concept, they would face an increased revenue by 41.4 billion dollars, or by keeping today's trend face losing 10 billion dollars from their potential income in Norwegian shelf [3].

Tail IO was a reply to such demands and 4 out of 6 sub-projects were about robotics and ICT infrastructures in platforms (base for remote operation) [4]. This trend is not a local trend in Norwegian shelf as similar projects have started all around the world (E.g. Smart Field (Shell), Field of future (British Petroleum), i-field (Chevron) [5]. Robotic automation would be the key concept in such approaches as it could replace human workers onboard, allowing them to remote control the process through fixed or mobile robots. Tail IO has created a unique situation which today one could join extraction of petroleum in the North Sea, while having plans for dinner with friends at home after work [6]. However,

this is only the start and this new trend of automation on topside requires new technologies to be developed. [7]. Robotic automation with combination of remote control and autonomous robots, if implemented correctly, could be the right answer to the aforementioned question.

This work could be considered as an attempt in the same direction. Here an attempt is made to contribute in further development of an ongoing custom robot project at University of Oslo called Walloid [8]. Walloid is a 4 arms prototype climbing robot concept for offshore platforms. (figure 1.1). The necessary remaining parts of this climbing robot project were developed. This was done with regards to *offshore platforms requirements and specifications gathered from literature*. The following points present features and areas of interest in this work.

- **Offshore automation, challenges and opportunities for robotic automation (section 2.4)**
- **Optimized climbing strategies for Walloid robot (section 4.3)**
- **Distributed Control Systems (DES) to navigate Walloid on vertical surfaces (sections 4.4 and 4.5)**

1.2 Motivation

Personally I like creating new systems. Robotics combines different expertise in order to make new systems and solve real world problems. In addition, robotics saves lives by replacing humans in harsh environments. It also creates new opportunities by operating in areas that are not suitable for humans. Therefore, I am highly motivated to study robotics and work on related projects. I also had a weblog called Walloid during development of this project where I posted about my activities and my experiences with different technologies. Project timeline, animations of different climbing gaits, figures and web-based simulations can be found on this weblog (<http://walloid.blogspot.com>) [9].

Moreover, oil and gas companies have become interested in developing new technologies for platform's topside automation [7]. As mentioned earlier, this is due to the new demands in the industry. The final aim of such attempts is to develop solid technologies, capable of automating processes in offshore platforms (A.1) [4, 10, 11]. Motivations of the industry for such interests could be categorized in following way:

- **Cost and production efficiency of automation / integrated operation (2.4.1)**

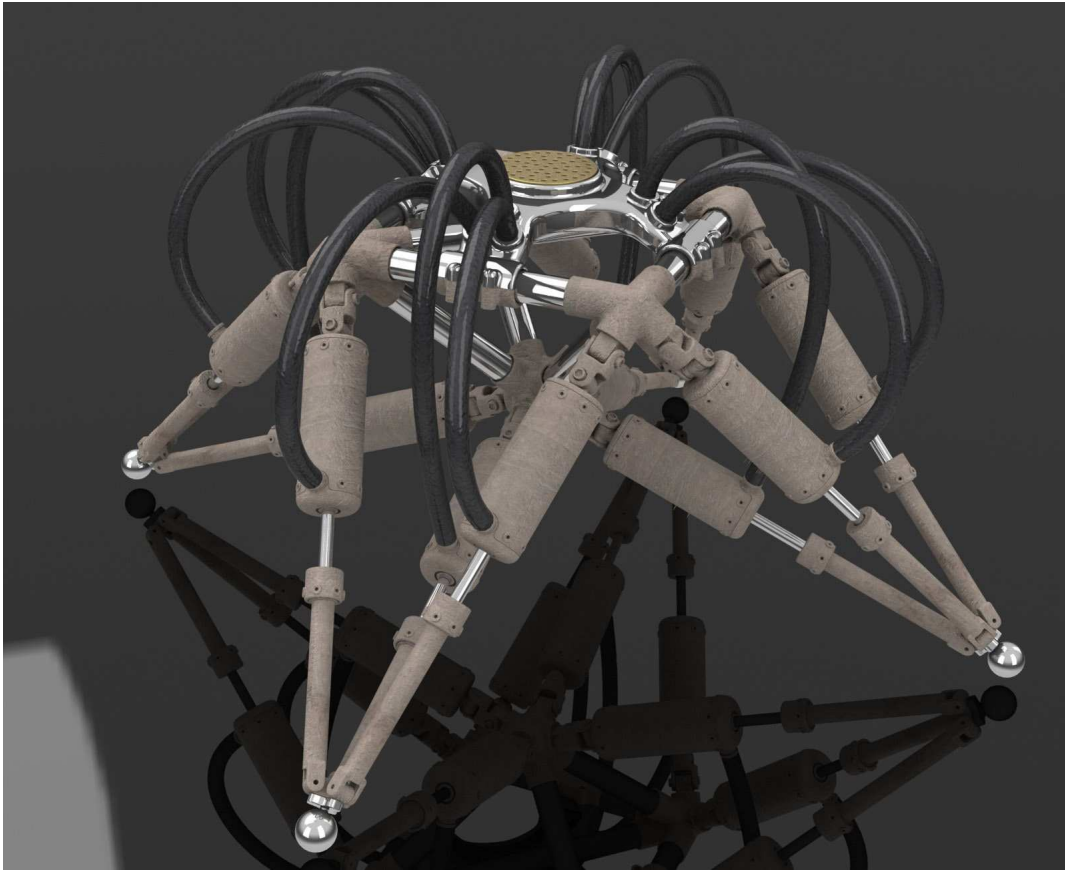


Figure 1.1: *Walloid Robot, an ongoing project at ROBIN group, University of Oslo*

- **Better Health, Safety and Environment (HSE) (2.4.1)**
- **Future platforms could not be built without newer technologies 2.4.1**

1.3 Summary

This work is divided into eight chapters. These chapters give an introduction and top-down perspective to what is done during the whole process. Later detailed development process and result are presented. Chapter 2, named Background, goes through technologies used in this work, where different aspects of these technologies are discussed. Topics such as climbing robots, automation, artificial intelligence (AI) issues, and embedded system design are discussed during this chapter. This chapter also contains an analysis of offshore platform challenges and possibilities for automation purposes. This is done by study-

ing the current literature around the subject (articles, previous academic works, governmental and industrial reports) (section 2.4). Chapter 3 is where a full status of Walloid project, the unit under test and the framework of this work, is presented and remaining topics are discussed. E.g. workspace and kinematics calculation are calculated. Chapter 4 presents a Top Level Perspective of the practical work. This is an attempt to create a top-down view to give a better overview of the later work. This plan is based on missing parts in Walloid project and the focus areas are climbing operation and control systems. In addition, it is decided to have an analysis of offshore platforms to have a better perspective toward this area of application (already presented in chapter 2). Chapter 5, called Development Process, is the most challenging and longest chapter of this report as it goes into details of all development processes (development of climbing strategies, control hardware and algorithm) through this work. Climbing gaits, design of end effector, Distributed Embedded System (hardware) and Distributed Navigation Program are topics of this work. Finally the topic of simulation is discussed, which tests system functionality and is the virtual presentation of the work done. Chapter 6, Implemented Control System and features, focuses on results achieved based on project objectives defined in chapter 4. This time the content is presented from a bottom-up perspective by going into details of results and charts and measurements around them. Chapter 7, Robustness issues, simply tries to have a risk analysis of developed parts. The concerns for improving the system robustness with solutions are presented in this chapter.

This work is finalized in Chapter 8, which presents the conclusion and contributions.

Chapter 2

Background

what is past is prologue.
William Shakespeare

Robots with that work, working in extreme and impossible conditions, create new possibilities. The high production rate, high yield, relatively inexpensive maintenance of robots and better HSE conditions are pushing industries towards robot workers and automation of their production lines [12] [13] [14]. This trend has already started in many industries, and is expanding everyday [12] [14]. The trend of robotizing daily lives does not stop there and lately it has even reached homes by autonomous vacuum cleaner robots produced by international scale companies, conquering the market (e.g. iRobot, Samsung, LG, Electrolux, etc). This promises a bright future for customers robot market and robotics technology as such companies invest money in developing new solutions and technologies to do their tasks better. [15].

2.1 Previous Work

Many climbing robots are developed for different purposes in academia and industry and several papers are dedicated to various types of efficient climbing strategies and gait planning. The main focus in developing climbing robots is to reduce the energy usage and balance the distribution of forces among the whole robot chassis [16]. Areas of applications of climbing robots can vary from case to case, but generally are categorized in inspection, verification and other services functions (tasks such as cleaning, welding, painting, etc).

Although there are numerous works about climbing robots, very few are specialized for offshore platforms. The offshore platforms are divided into two main parts, topside and subsea. Due to the limitations of operating in deep

waters, since the 1950's new subsea technologies and methods were developed and have now reached a very stable level [17]. This was not done on topside as changing the old routines were not necessary (automation) and lack of technology for operation in this section was never felt [18]. Based on earlier topics in introduction and similar studies in this field [18], the interest in automation in the industry is growing. Automation process for offshore platforms contains its own properties, limitations and challenges. Such challenges include extreme weather condition, vibration, and salt water, local banned radio spectrum, etc [11,18].

Similar studies show a wide variation of opportunities for robotics automation (mobile and fixed) in offshore platforms [11,18]. Same studies also underline that, looking closely at everyday work-plan of platform operators, most of their time is used on walking around the platform, transportation, regular inspection, maintenance assignments and other repetitive jobs that could be taken over by robots [11,18]. Such works could have been done by inspection, monitoring and detection robots equipped with proper sensors. It is important to mention that the complexity arises when one moves from inspection robots towards manipulator robots, especially mobile ones without a fixed origin. Such approaches taken by industry could increase the revenue and HSE level of work environment [3,11,18].

Many works and surveys have been done by governments and companies, in order to implement a new organizational chart called Integrated Operation [3]. The integrated operation tries to migrate the work force from offshore to onshore, and get the work done by minimum onboard staff [6,19] with helps from remote operation, robotics and automation on the platforms [4,19]. Based on such studies, there are projects that are trying to implement these ideas. E.g. TAIL Integrated Operation (TAIL IO), an ABB-led consortium attempting to develop new technologies and work processes for StatoilHydro. The aim of this consortium was to develop methods that could lead into an increase in production rate, while decreasing the costs of production and maintenance, human safety and environment issues and finally prolonging the life time of platforms [5]. However TAIL IO was not a direct attempt to automate the operation on platforms, but an attempt to migrate the workforce from offshore facilities to onshore, bringing real time remote controlling to the platforms. To this, ICT infrastructures were implemented that can be easily used as the base of real time controlling and automation in future (ICT infrastructure, Robotics on platforms, live streaming of sensors data to the experts, etc) [5]. This was a strategic step toward an automated future in oil and gas industry.

2.2 Climbing Robots

Climbing robots are a type of man-made intelligent machine with the ability to climb vertical surfaces [20]. The maturity and stability of climbing technologies have resulted in increasing numbers of climbing robots in industrial applications that help human workers in areas which is impossible, dangerous or too difficult for them to operate. Such operations are done either by remote control or automated methods. It is very challenging to develop fully autonomous robots for sensitive and complex tasks (too many unplanned events), but remotely controlled systems could have been much more reliable. There is also a middle solution called a semi-autonomous approach where tasks are automated, but very complex and sensitive operations are dependent on the operator [21].



Figure 2.1:

left to right Max V a chain-driven climber using vacuum cups, developed at University of Aalen - Rest six legged welding robot using magnetic force, Developed at CSIC Madrid - Roma grasping robot, specialized for inspection in steel bridge, developed at University of Madrid

The Climbing operation in nature is done mainly in two different methods:

- Quasistatic: Using slow static motion in locomotion to climb.
- Dynamic: Using fast dynamic motion in locomotion to climb.

Most of climbing animals (e.g. chimpanzee) and human use dynamic climbing method to climb. There are also animals that naturally use quasistatic slow climbing method (e.g. sloth). Dynamic climbers are fast and can overcome most

obstacles. On the other hand, most of climbing robots are quasistatic (also Walloid) due to the high complexity of design and control [A Minimalist Dynamic Climbing Robot: Modeling, Analysis and Experiment, A Spring Assisted One Degree of Freedom Climbing Model(book)]. No matter what methods of climbing is used, based on Locomotive abilities, climbing robots could be divided into three main classes: wheeled / tracked locomotion, legged locomotion and arms with grippers locomotion [20]. Different examples of such robots can be seen in figure 2.1. Usually robots developed with arms and grippers or legged locomotion, fits best in more complex surfaces (e.g. oil and gas platform with various surfaces), while the wheeled/tracked locomotion fits best even terrain like glass, concrete, brick , steal walls [20,22]. For these locomotion types, different types of adhesion forces could be used to keep the robot from falling off the wall. These adhesion forces can be categorized in following classifications [20].

1. Magnetic force (permanent and electrical)
2. Negative Pressure - Vacuum
3. Grasping
4. Pressing to the inner wall
5. Van der Wals force - Gecko

Moreover, it is now time to name some of the general critical requirements in the development of any climbing robot. Such requirements can be stability, flexibility (ability to handle a variety of terrains), surface contacts issues, power consumption, force distribution, overheating of motors, and climbing between adjoining surfaces [22, 23]. These requirements plus the additional specifications / local issues for each area of application are challenges that every project faces during development phase. However, it is difficult to satisfy all these requirements by only choosing one type of adhesion force. Therefore to reach the highest reliability, stability, flexibility and HSE concerns, a combination of locomotion and several types of adhesion forces should be used at the same time.

Table C.1 (had to move to Appendix due to size) shows a number of active industrial climbing robots which use a combination of different methods to robustly perform their tasks on vertical surfaces. As this table shows, available climbing industrial robots in the market focused on a very specific problem such as inspection or a specific service. This focus in addition to the fact that all of them are controlled by remote controls, underlines once again the complexity of building a versatile autonomous system that can result in reduction in reliability and redundancy in industrial automation process which is not acceptable by the industry.



Figure 2.2: *Operator works side by side with a climbing welding robot (on-site user).*

2.3 Automation and Smart Agents

2.3.1 Autonomous Robots

Autonomy is applied to a system, being able to operate and behave on its own, without external control power for an extended period of time [24]. Such systems are able to operate in dynamic environments, also adapt and respond to the changes forced to them and their environments [24]. Bringing this concept to man-made systems, it would be machines capable of reasoning and controlling their actions in their workspace (environment) with capability to perform specific assignments.

Such systems to some extents exist today. Fully automated machines can perform their assignments without any help from the operators. However, quality control monitoring is unavoidable based on complexity of the task. Today's technology in artificial intelligence (AI), data flow speed in both networks and internal buses and the high sensitivity of the sensors, combined together are capable enough to shape systems that can perform the expected tasks automatically. These fully automated systems are usually a combination of interconnected smaller systems. These systems, with their few capabilities, when

mounted and connected together, could work as a whole that is capable of much more (e.g. compare functionality of four arms getting together to shape a climbing system). Such systems are addressed later in this chapter under embedded systems (2.3.3).



Figure 2.3: *Autonomous cleaning climbing robot for glass and solar panels.*

Today's robotics technology is almost on the edge to present reliable autonomous products to the market to perform specific tasks. The extremely competitive market of smart vacuum cleaner robots is the beginning of a new era. The era of autonomous machines working instead of human, fully automated, but this time not only in harsh industrial environment, but also at homes. It's clear that expanding the limited tasks done by an autonomous robot, the complexity of task planning would rise. This complexity would also result in more issues in handling all upcoming situations, planned or unplanned, during operation. One solution to handle sensitive and complex tasks was to go for a middle approach, called Semi-Autonomy, where things mostly are automated while some few sensitive and complicated tasks are tagged to be remotely controlled. Beside the extra control over system, stability and reliability, such solutions do not need many operators as their presence is requested only in case of reaching the sensitive levels of the work or in case of emergency. No matter which approach or complexity level is chosen, AI is the part in charge of receiving the input, reasoning and decision making accordingly. This would be the topic of the next section of this sub-chapter.

2.3.2 Artificial Intelligence Methods

An autonomous system should be capable of deciding on its own and acting based on the decision making process. This means there is a need for intelligence in machine level which can receive the input data, reason based on it and act accordingly. This is the topic of artificial intelligence. Different philosophical definitions of AI can be given based on similarity in thinking and behaving like a human, or a system that thinks and acts rationally [25].

The second approach fits our purpose in discussing industrial automation best and therefore is preferred. According to this point of view, AI is the study and design of rational agents that can perceive, reason and act [25]. Agents are beings that can precepts the environment around it by their sensors and acting through their actuators, while the rationality means selecting the choice that maximizes the performance based on earlier precepts of the environment [25].

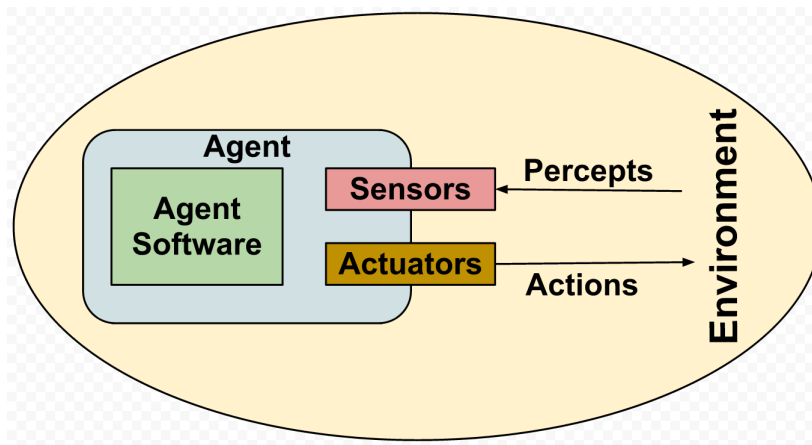


Figure 2.4: *Artificial intelligence and smart agents.*

The nature of environment is one of the key elements in developing a rational agent. The environment varies a lot and the complexity rises as a rational agent is supposed to perform in several environment. The characteristics of environment can be categorized in different sub topics (Table 2.1) [25].

Regardless of the environment types, the agent type defines the way they process the input data from sensors (perception of the environment). This difference in analysis of the data later affects the decision making process by choosing the feedback through actuators upon the environment. Table 2.2 tries to give a brief categorization of such agents differs by their processing methods. The complexity of methods rises from top to the bottom of the table [25].

The decision making method, combined with challenges and limitations of the working environment are gathered in a logical decision making of the nav-

Table 2.1: Environmental characteristics

Environmental Characteristic	Offshore Platform Environment
Single agent / Multi-agent	Single (later could be developed to several)
Fully observable / partially observable	
Deterministic / Stochastic	Deterministic as level of automation would be re-doing an already done task (blind copy with sensors monitoring)
Episodic / Sequential	Agent's experience categorized in episodes
Static / Dynamic	Dynamic
Discrete / Continuous	Distinct percepts and action
Known / Unknown	Known, but could be unknown due to dynamic environment

Table 2.2: Agent and the type of automation

Task	Area of Application
Table driven	Set of rules which guides the system after an implicit goal, usually a set of if structure, $state \leq INTERPRET(percept)$ $rule \leq RULE(state, rules)$ $action \leq rule.ACTION$
Model-based Reflex	Set of rules which guides the system after an implicit goal, ... $state \leq UPDATE-STATE(state, action, percept, model)$ $rule \leq RULE-MATCH(state, rules)$ $action \leq rule.ACTIO$
Goal-based	Rational agents choose actions based on contribution to an explicit goal
Utility-based	Alternative ways to reach a goal usually have different coast for the system
Learning	Learning is the key to achieve autonomy and improve of performance over time

igation program. Meaning the navigation program receives the perception of the environment from the sensors. These would be sent along to the AI unit (logical decision making unit). At AI unit, based on inputs, environmental constraints and the type of agent one feedback is chosen and is performed through actuators. In case of having a system which is a combination of interconnected smaller systems, these AI units can be divided into smaller units at each lower level system, to ease the decision making process and to distribute the processing power needed for it between different CPU's. This division of tasks between smaller independent units is the topic of our next section called Embedded Systems.

2.3.3 Embedded Systems

What is an embedded system and why is it important to mention it here? An embedded system is a computer system (hidden), which its task is to perform one or a set of limited dedicated tasks [26]. Simply this means breaking assignment of the whole system into smaller one which are simpler. Each embedded system would be only in charge of one or few simple task. E.g. control unit of one arm could be called an embedded system and a part of a bigger whole (climbing robot).

Embedded systems are usually real time and can either be a single independent unit or like the above example a part of a larger system (the above example). The processing unit of these systems can be microprocessors, micro controllers or Field-Programmable Gate Array (FPGA) and they have limited computing hardware resources such as memory, keyboard, screen, etc. Each embedded systems consists of the following three main components [27]:

1. Embedded hardware, with its special specifications (figure 2.5)
2. Main application software which may perform a series of tasks or multi-tasking at the same time.
3. Real time OS (RTOS).

Here in this project, the focus is more on Small Scale Embedded Systems (SSES) that have a small 8-16 bit micro-controller, are battery supported and can be programmed by a variation of C language family. SSES family have limited memory and CPU resources and in case of continuous running it is critical to keep track of memory usage, CPU usage and power consumption and limit the dissipation [27]. The application level of the embedded system in this family is programmed in C and is supposed to monitor and perform the expected tasks based on specifications and real time constraints. There are also more

complicated embedded systems such as Medium Scale Embedded Systems and Sophisticated Embedded Systems which have higher complexity in terms of software and hardware which is not the case here.

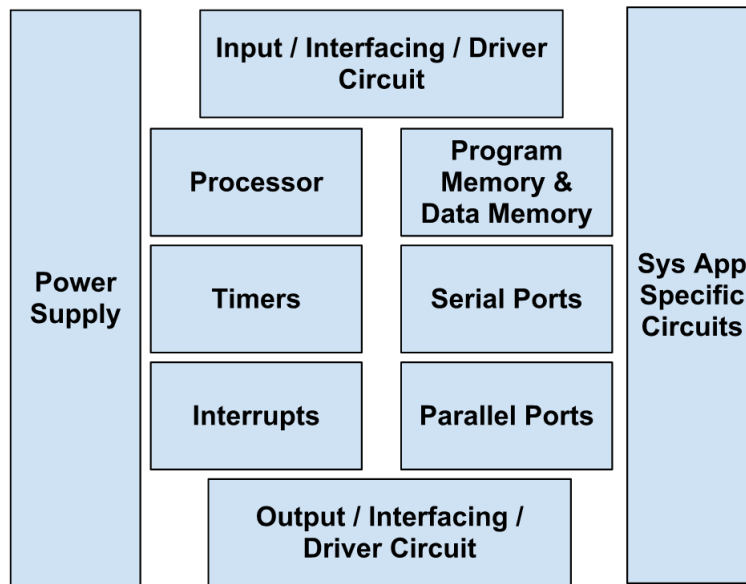


Figure 2.5: *The components of an embedded system [27]*

RTOS is a type of operating system (OS) that adds determinism to the system. Real Time here does not mean really fast, but means the ability to be able to determine when a section of code would run. RTOS can be divided into two categories of Hard Real Time systems such as flight control systems which are very restrict about the responses to happen in deadlines and Soft Real Time system such as Data acquisition systems (DAQ) that allow some response deadlines to be missed (slow degradation in system work, but not total failure) [26]. This is totally in contrast with general purpose OS (e.g. Windows) which operates on a fairness basis. Regardless of priority of applications in such systems, the CPU resources are fairly distributed between all running tasks. Meaning the anti-virus routine check could stop / delay a critical process. General purpose OS could also preempt processes based on their priorities, but there is no guarantee that processes end in the specified time [26].

Embedded system is an important definition and will be used frequently later. One embedded system is responsible only for one or a limited number of tasks and is supposed to perform it correctly and on time. The real time characteristic brings determinism to the system that one knows in each time what part of our code is being run on the system. Next part is the analysis of

offshore platforms where results from different articles, medical and industrial reports are gathered, to define opportunities and challenges in this field.

2.4 Offshore Platforms, Challenges and Opportunities

2.4.1 Motivations

Automation wave in industries with different types of robots has already begun for years with fixed industrial manipulators and is continuing in different fields with new types of mobile robots, for example cleaning, welding, painting and etc. Petroleum industry is not an exception in this trend and has already been benefited from underwater robots (ROV's) in the past and is planning to use mobile robots in future on their platforms A.1 [11, 18]. This new wave has led into new policies which prefer to shift from manual to automated production due to different reasons. *Such reasons can vary from reducing production cost, higher production rate, steady quality, easier production planning, less human resources issues and last but not least safety reasons* [6, 11, 18, 19]. Offshore applications are one of the earliest industries that started using mobile robots (ROV's) and also are among those who invested most on robotics technologies. However, a deeper investigation in the numbers and exact areas of investment shows that these attempts were focused on areas where human workers could not operate and the industry had no choice but to use submersibles, such as Remote Operated Vehicle (ROV) and recently Autonomous Underwater Vehicles (AUV) [18]. These investments and researches ended up in today's ROV technology which is mature and very stable [18]. Today ROV's are used all around the world both in building operations, under water studies and also for search and rescue operation in crisis time [28].

At the same time on the topside human operators were doing things in the traditional way and it just worked well enough. This is a very good example to show that the *industries do not change out functional systems easily. Redundancy* is an important factor in industrial application. *As long as there is no better redundant solution* which improves the situation, *the problem does not exist!* Until recently everything worked fine as although the alternatives did not match the reliable good work that human operators were doing. This was changed after the huge rises in oil price which made previous remote and non-economical projects (Shtockman, Sakhalin and etc) in the middle of Arctic and Alaska economical. *This change in the market plus lack of new resources near shores has made the industry to reconsider investing in future platforms which*

are placed even further in the sea (North Pole, deep sea, etc). The history is now repeating in offshore industry and these demands are now calling once again for new investments in bringing automation to oil and gas platforms, but this time to the topside [4, 18, 29]. Projects like Integrated Operation (StatoilHydro), e-Operation (Hydro), Smart Field (Shell), Field of future (British Petroleum), i-field (Chevron) [5] are some of these attempts. Later in this sub-chapter an overview of the demands for automation in offshore platforms and the challenges in implementing such ideas is given. Some of the motivations for such activities can be: [5, 11, 18]

1. *Cost and production efficiency of automation / integrated operation*
2. *Better Health, Safety and Environment (HSE)*
3. *Future platforms could not be built without newer technologies*

Cost and Production Efficiency

Working on offshore platform contains high wages, meaning tighter budget for projects and needs for higher investments. *Less profit* for investors could result in projects *being stamped as non-profitable and closed*. The ***robotics-automation can help in reducing cost in offshore production***. The financial benefits of automation with robots involved can be described in following three points [10, 11, 18]:

1. Due to the reduction in robotic automation prices, robots are now becoming a better alternatives than manual work and transportation of experts. The sudden investment in the automation can also be paid back as the production cost reduces overtime.
2. Robots work 24/7, are more precise and make less mistakes, meaning higher production quality, efficiency and flexibility. On the other hand, human workers, with high wages, are subject to stress and hard working conditions and their mistakes can result in accidents with big financial and environmental consequences.
3. Reliability and stability followed with robots rather than human operators reduces the possibility of un-planned shutdowns which is most costly for oil industry. Unplanned shutdowns occur due to bad weather conditions, lack of human resources at a time on site due to sickness, etc.
4. Reduce labor turn overs and the problems followed by recruiting new workers and training them.

5. Such solutions result in fewer work forces on site, which again results in saving accommodation space needed for housing workers.
6. Help senior workers to keep up with the junior inexperienced workers when doing heavy works.

All in all, involving robots in automation or semi-automation (remotely controlled robots by human from onshore) of the offshore work environment is the golden key for oil industry to increase their revenue, production rate and easier Human Resources (HR) procedures. Governmental economic reports state that in case of ignoring such solutions the industry faces reduction in revenue and other consequences in their organizational structure [3].

Health, Safety and Environment

Oil companies in general emphasize health, safety and environment (HSE) as an important issue due to heavy financial consequences of incidents caused by the lack of HSE concerns. The latest incident in Gulf of Mexico had catastrophic consequences as killed 11 workers and resulted in a huge economic and environmental catastrophe along the U.S. gulf coast [2]. The British Petroleum also lost billions of dollars due to this incident. Table 2.3 shows a timeline over the major offshore incidents in the history of oil and gas industry [30].

There is *a lesson in such incidents* which is to *replace human workers with robots in harsh environments as much as possible*. Offshore platforms are one of the most extreme work environments for human workers due to the harsh weather, unstructured environment and high concentration of dangerous and deadly gases (H₂S) [18]. *The robotic automation might be costly to begin with, but would pay back in preventing such incidents and also with earlier benefits mentioned.*

Beside all these incidents that can cost billions of dollars for oil and gas companies (visible costs), there are some HSE concerns which also results in *invisible costs* that the robotics-automation can help reducing them. The heavy nature of working on platform can result in *injuries during operation (muscle injuries, etc) and tier out workers out during years of work, resulting in early retirements* [31]. Robots do not suffer from such problems and their maintenance would cost much less than all sick leaves and early retirements that would *affect both national and companies' financial statement*. Figure 2.6 shows the number of incidents occurred during 2006 - 2010 at United Kingdom. These numbers clearly shows that *due to the usage of ROV's for under water operation, diving incidents are very low*, while on the other hand the number of incidents on the topside is very high. Based on all these it is obvious that HSE

Table 2.3: Major offshore accidents in the global oil industry [30]

Year	Damages	Area	Company	Causes
1969	Up to 100,000 barrels of crude oil leaked	The California coastline	Platform A offshore near Santa Barbara	Blowout
1979	Spewing 3 million barrel of crude oil	Campeche Bay of Mexico	The Pemex-operated Ixtoc I offshore well	Blowout
1980	123 death	The North Sea	Alexander Kjelland	Capsizal
1982	84 death	The coast of Newfoundland, Canada	The Ocean Ranger semi-submersible drilling rig	Huge storm
1984	Death of dozens of workers	the Campos Basin	Brazilian State oil company Petrobras	Blowout
1988	167 death	In the North Sea	Occidental Petroleum	Explosion
1989	More than 90 death	The gulf of Thailand	U.S. drilling Ship Seacrest	Typhoons
1995	13 death, many injured	The coast of Nigeria	Mobil oil rig off	explosion
2001	11death, sank off the coast of Rio De Janeiro, around 10,000 barrels of fuel	The Atlantic	Brazilian State Oil company Petrobras	Explosion
2005	12 death, and reduce the country's domestic output (15%)	The India's West Coast	ONGC	Fire
2007	21death, and fuel leaks	The coast of Mexico	The State Oil firm Pemex	Stormy weather
2009	Oil leaking, sank of drilling	The East Timor Sea near Australia	The West Atlas	Fire
2010 (April 20 th)	11 death	The U.S. Gulf Coast	Transocean Ltd.	Fire and explosion
2011	49 missing	Coast of Siberia	Russian oil and gas	Collision and collapse

concerns are very important in oil and gas industry and should be considered carefully while developing any system for offshore platforms. Such concerns sometimes create challenges and might even prevent developers to build machines capable of automating risky tasks in offshore (Welding robots, spark and NORSOK qualifications).

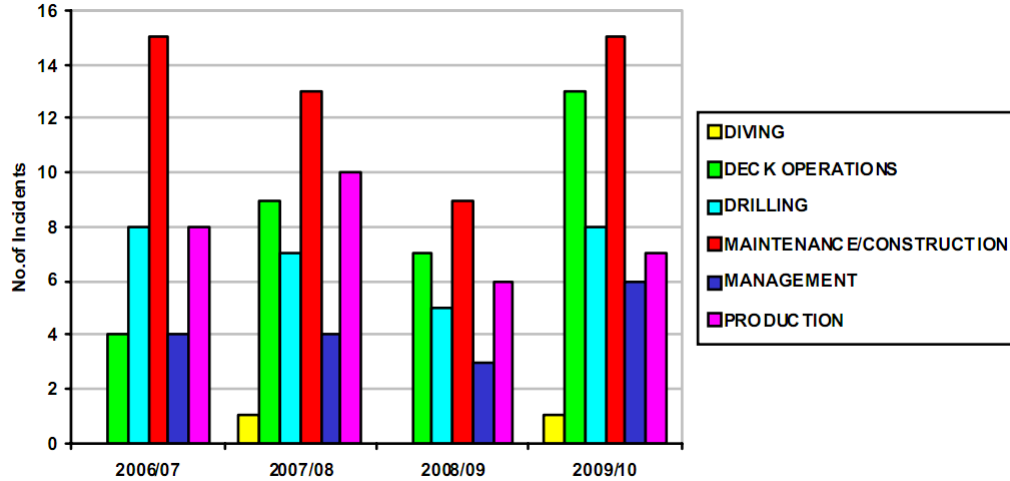


Figure 2.6: Number of incidents in UK offshore from 2006 - 2010 [32]

Future Platforms

Beside the financial benefits and HSE concerns in automation of oil platforms, there are other concerns that leaves oil and gas industry with no other option but to automate the process of oil production on the platforms. *Future platforms are different from current ones, not in building technology but in term of characteristics.* Combination of rising oil price in the international market and lack of available resources near the shores, takes the oil companies *even further in the middle of the sea* in search for oil [18]. It is obvious that frequent transportation of experts and supplies would become even more difficult and costly than now. These challenges leave the industry with no choice, but to develop new operation methods to make working in those remote areas possible (e.g. automation of the process, manual steering from onshore, etc). Ever rising oil price guarantees the revenue for such projects and the new technology that needs to be developed to reach that level of automation in near future [18]. Examples of such remote fields with extreme weather conditions are *Shtockman and Sakhalin in Arctic or Alaska*. Shtockman is located in the Barents Sea about 600 kilometers north of the Kola Peninsula and experience extreme

weather condition. This situation makes transportation of experts and supplies very difficult during six months of the year [18].

Oil companies are already investing in automation of their platforms. They are already planning for a day that there would be a minimum of workforce on their platforms and therefore trying to start by small models of such visions. Kristin platform of StatoilHydro is an example of such projects where the number of human workers on-board the platform was kept to minimum level of 27 people and the results were satisfying [6, 29]. The goal of almost zero workforce vision can only come true with robotics-automation. Integrate operation is a very good start to build the fundamental of such automation in the near future. However, the IO operations are just the beginning of making these dreams come true and the last step in such long term projects is to bring autonomous machinery to the picture. Machines that do their job independent of external control, and operators only need to interfere in case of emergency.

2.4.2 Automation Opportunities in Offshore Platforms

Robotics automation is already helping oil and gas industry in some areas applications. E.g. pipe inspection, submersibles and drilling [33] which are all impossible for human workers to operate. Studies done in categorizing the offshore oil and gas production fields into 5 different categories [11]:

1. Shallow water: Platforms in waters with maximum 200m depth, mostly two jackets (3-5 decks) connected with bridge.
2. Deep water: Platforms in waters with beyond 200m depth, with only one jacket with more than 5 decks.
3. Floating: Almost like a ship than platform and very flexible in changing locations.
4. Unmanned: There are several wells in big fields. These platforms are usually maintained every 2 weeks with a crew of 2 - 4 operators.
5. Subsea: Wells and installation which are mounted under the sea. These platforms are fully automated with use of ROV.

For a climbing robot, beside subsea platforms, other platforms could be a potential area of application. Such platforms are usually large facilities with equipment located all around and monitoring such vast ground would be time consuming. Therefore, mobile robots, and specially inspection climbing robots



Figure 2.7: *A:Bad weather condition | B:Shallow water platform*

would be very useful in such areas. This becomes even more important in unmanned platforms, as it could make the monitoring process live (24/7) and prevents long ship trips by the crew. On the other hand, for developing an inspection / monitoring robot for such areas, one should discover the expectations from the inspection operation in such environment. There are reports and articles which tried to identify such operations that were frequent enough to be automated. Such activities can be [11, 18]:

- Live video feed of environment
- Gauge readings
- Valve and lever position readings
- Monitoring gas level
- Acoustic anomalies
- Surface condition
- Check for intruders
- Gas leakage
- Fire detection and locating

One can see that all these operations are easily performable by an equipped climbing inspection robot. A robot equipped with proper set of sensors, all these tasks could be automated as most of them have a digital nature and are based on reading values from sensors (gas leakage, fire detection, etc), streaming of video and image processing of received pictures (valve positions, reading numbers, surface condition, etc) [11, 18, 34].

2.4.3 Challenges in Automation of Offshore Platforms

The challenges that one project might have in offshore platforms are addressed here. Beside these issues, one might wonder if HSE requirements are also a part of these challenges. Due to the fact that HSE was discussed enough before it was decided to look away from that.

Extreme weather conditions

Almost all offshore platforms do suffer from extreme weather conditions. The word extreme can vary from the freezing temperatures of Arctic area in North Pole, ever shining sun of Persian Gulf. This challenge gets even worse when it comes to the future platforms as these are even placed in more remote locations. When developing a robot to perform for this area of application, such natural phenomena like very low and high temperatures, ice, direct sun light, fog, alga and moss, and last but not least the salt water from the sea must be considered as a normal environmental issue that the robot should deal with daily.

E.g. direct sunshine and fog interrupts Rfeed signals. There is also a good chance that very low and high temperatures effects senders and receivers of radio signals. In case of using grippers ice (20mm ice is considered during design of platforms), alga and moss can make grippers job in grasping the bolts very difficult (slippery). Salt water, humidity and the rust are also other three destructive natural phenomena in offshore platforms. These points are discussed more in the following sections and are addressed and discussed in details later in following chapters where such concerns are considered while choosing different components / technologies.

Salt Water

Salt water can have destructive effects on the equipment over time. Therefore structures and machines are being washed by high pressure water regularly and this work is an essential part of maintenance work on platforms. A functional machine for offshore application is supposed to be designed to stand such condition. The salt water destructive effects should be considered both in design of the chassis and also in choice of electronics parts on board the robot.

The task of washing machines is one of the repetitive jobs on a platform that can be done by robots, but this task needs to be done by robots that are stable enough and could stand such load.



Figure 2.8:
After being submerged in concentrated salt water for 5,000 hours, the unprotected iron T-Bolt on the left is totally corroded and unusable.

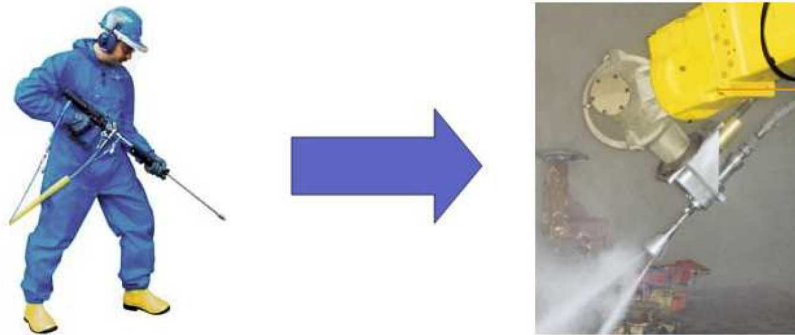


Figure 2.9: *Washing salt off equipment with high pressure water*

Rust

Rust is another destructive factor in the offshore platforms that needs close attention in designing process and also in choice of electronics devices used in the embedded system. Rust is permeable to air and water and even interior parts of a design can be exposed. Stainless steel creates a passivation layer made

of **Cr₂O₃** (Chromium three oxide) [35]. Similar protection can be achieved by magnesium, titanium, zinc, zinc oxides, aluminum, polyaniline and etc.

Offshore Standards

Existing standards and test methods in the offshore environment almost cover all discussed challenges with proper testing methods. Each equipment before being able to start their operation on platforms needs to be qualified by such standards (different national and regional requirements). The tests either qualify or deny a product to start operation on offshore applications. One of the most important and initial tests for such equipment as a mobile robots is explosion test. Here a list of different relevant standards with a short description is presented.

Norsok is a standard, developed by Norwegian petroleum industry, is a standard that contains a series of different standards that applies to both offshore and onshore installations.

DNV Sfc 2.4 - Environmental test specification for instrumentation and automation equipment.

This document specifies the environmental test specification applicable to all instrumentation and automation equipment such as: hydraulic, pneumatic, electrical, electromechanical and electronic equipment, including computers and peripherals that are to be installed on Ships, MOUs and HSLC with DNV Class. DNV Sfc 2.4 document

IEC 60945 - Maritime navigation and radio communication equipment and systems general requirements)

Prepare standards for maritime navigation and radio communication equipment and systems, making use of electro-technical, electronic, electroacoustic, electro-optical and data processing techniques for use on ships and where appropriate on shore. IEC 60945 document

US military spec MIL-STD-810 [36] - Department of Defense Test Method Standard for Environmental Engineering Considerations and Laboratory Tests

This test method standard is approved for use by all Departments and Agencies of the Department of Defense (DoD). Although prepared specifically for DoD applications, this standard may be tailored for commercial applications as well - The primary emphases are still the same – tailoring a materiel item's environmental design and test limits to the conditions that the specific materiel will experience throughout its service life, and establishing chamber test methods that replicate the effects of environments

on materiel rather than imitating the environments themselves. However, the "F" revision has been expanded significantly up front to explain how to implement the environmental tailoring process throughout the materiel acquisition cycle. MIL-STD-810

2.5 Summary

During chapter two, Background, an attempt was made to form a backbone for later practical work. Chapter two started with previous works and later tried to give clear definitions about different important concepts in this project. These concepts were climbing robots, automation, smart agents, and embedded systems. Later an analysis of benefits and challenges of automation in offshore platforms are discussed. Finally opportunities for robotics automation and practical challenges that one would face in the development of a semi-autonomous robot for offshore platforms were discussed and addressed.

Table 2.4: summary of chapter2

Concept	Description
Climbing Robots	Manmade machines capable of climbing vertical surfaces.
Autonomous Robots	Robots capable of reasoning and controlling their actions in their workspace (environment) with capability to perform some tasks.
Artificial Intelligence	An autonomous system is capable of deciding on its own and acting based on the decision making process. The decision making process is called an artificial intelligence.
Embedded Systems	A hidden computer system, which goal is to perform one or a set of limited dedicated functions.
Robotic Automation in Platforms	Using remote controlled or semi-autonomous robots to be able to automate the process in offshore platforms.
Benefits of Robotic Automation in Platforms	Robotic automation benefits both investors and workers by reducing the costs of projects, better production rate, improved HSE standards and last but not least allowing for operation in areas with extreme environmental conditions (difficult or impossible for human workers to operate).
Challenges Faced in Offshore Platforms	Such challenges are mostly environmental conditions imposed by the geographical location of the platform. Such problems are extreme weather condition (rain, snow, thunders, ice, alger, etc), salt water, rust and vibration. There are also many regulations that should be respected while trying to develop a robot for offshore platforms.

Chapter 3

Walloid Robot

The original plan in this project was to use a developed prototype at ROBIN group, called X2 [8] (figure 3.1, last picture on the right). However, this prototype was not precise or cost efficient enough. Therefore, the development of a new prototype, called Walloid, was started. This prototype was designed to be more cost efficient for production and also more precise than earlier X2 model.

Walloid project **contained**:

- **4 arms** climbing robot
- **3 prismatic joints** on each arm.
- **Robin developed encoder** solution with light fork sensors and a rotary joint that connect the motor shaft to the prismatic joint (section 3.2.1).
- **Very precise movement of prismatic joints (0.25 mm per encoder reading)** which make the movement of the end effector very precise (section 3.3.1).
- The prototype contained **only primitive hardware components design**, which needed further development to support a functional climbing robot (development process in section 5.2).

On the other hand, Walloid project **did not contain** the following points- An attempt was made to develop them in the mentioned parts of the work.

- **Adhesion method** was **missing** in the initial design.
- **End effector** (section 5.1.1).
- **Kinematics calculations** and **workspace** (section 3.3).
- Control Hardware (section 5.2).

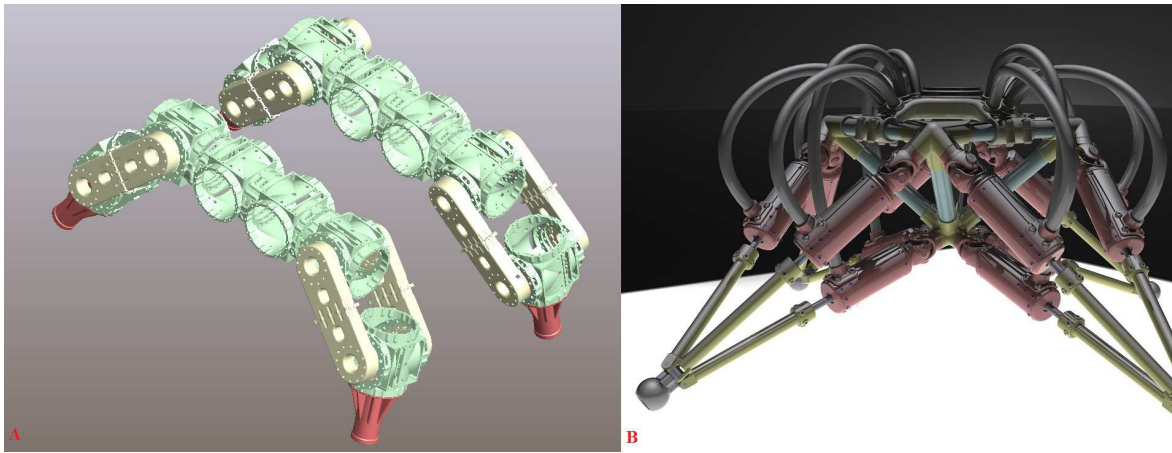


Figure 3.1:

From left to right: Walloid robot received part, Walloid 3D design, in-house developed encoder (rotor and sensors), the implemented version of in-house designed version of the encoder, prototype

- Navigation Program (section 5.4.3).

3.1 Ongoing Project

Walloid project is still an ongoing project at University of Oslo and has not yet been finished due to practical problems (figure 3.1, A). The first prototype that was received during this project was only a single prismatic joint. The initial work was started on this prismatic joint and later a total arm was received with three prismatic joints (one arm). The *received arm* was *printed by a 3D printer* at Robin group at University of Oslo (figure 3.2). This part eventually broke down due to *poor material quality and some design issues* (discussed in 3.4.3). *To reach project deadline, the process of fixing the arm was ceased* and therefore, *simulating Walloid robot was based on earlier experiments with the arm*. This resulted in the simulation becoming the test bench of the control software and hardware and the only way to present the climbing gaits (5.6).

3.2 Walloid Hardware Components

Each arm consists of three prismatic joints, which each of them includes one DC motor, one motor driver and one encoder (figure 3.2). The telescopic motion of the prismatic joint and the joint speed are controlled by the micro-controller

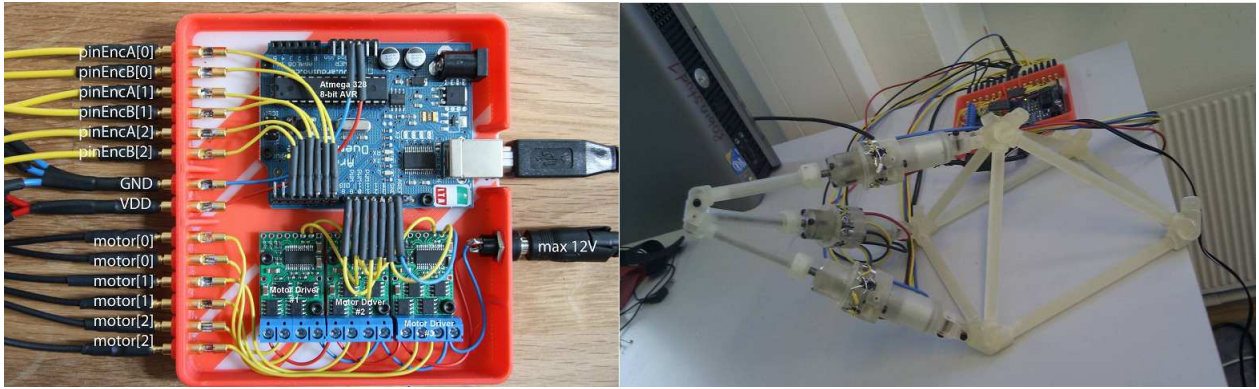


Figure 3.2: Left to right: 1: Control system hardware | 2: Walloid arm

Table 3.1: Primitive components of the received arm of Walloid robot

No.	Name	Manufacturer	Quantity
1	Micro controller, Atmega 328 (Arudino Duemilanove), 8bit	AVR	1
2	DC motors, 12 V	Elfa	3
3	Motor drivers, 15 Amp high-power motor driver	Pololu	3
4	Encoders, in house developed encoder consist of 2 light fork sensors-Optek Tech., Opto switch, Logic output	ROBIN	3

through motor driver (Enable/Disable signals, PWM values). The screw shaft opens and closes the joint. This movement would result in rotor to follow the rotation, interrupting the light sensors of the encoder. These interruptions generate hardware signals that were read by the navigation program and based on them the direction and speed of the movement was detected. These sensor readings were processed elsewhere in the navigation program into information like speed, position, RPM and etc.

3.2.1 Encoder

Encoders are a measuring tool for detecting the angle of motion with a 3-bit binary system [37]. One of the special characteristics of the robot used here is the encoder that was developed at Robin group at University of Oslo (figure 3.3). The idea with this encoder was to have a built-in system capable of doing two things at a time. 1, to be able to forward the motion from the motor shaft to the axel. and 2, to monitor the motion of the motor shaft (by light forks) (figure

3.3). This was done by designing a small connector with connective joints on both sides and a rotor for the encoder. The rotor part of this connector had a half circle edge whose task was to interrupt the two light fork sensors that were placed around the joint body (figure 3.3). The idea here is brilliant, but there are some design problems here that resulted into a series of problems while testing on received parts of Walloid robot. These challenges will be mentioned, discussed and finally a solution is presented in about the weakness section in 3.4.

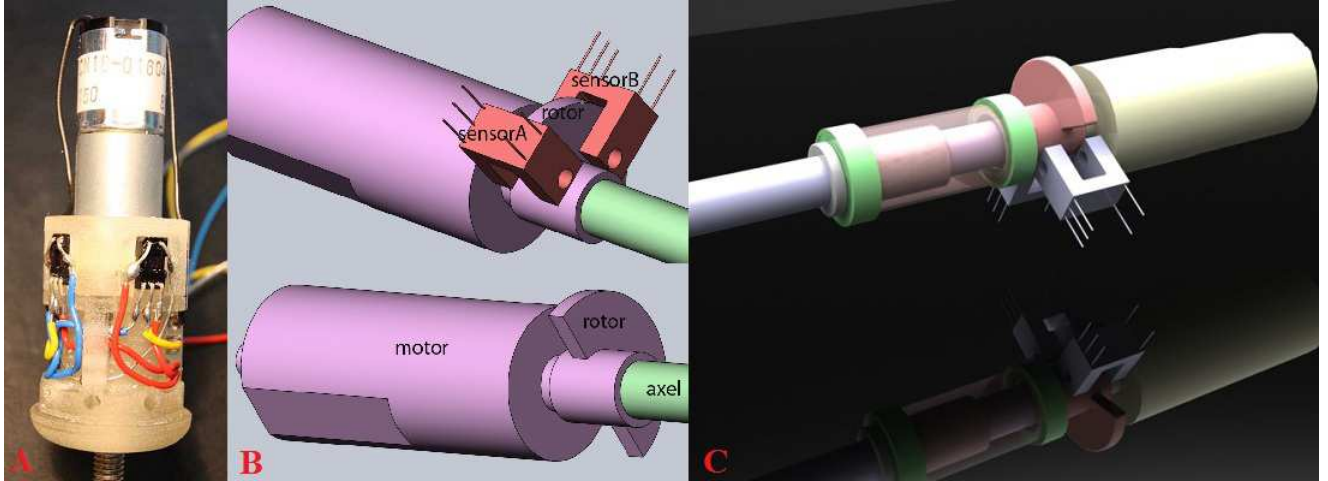


Figure 3.3:
In-house built encoder, the rotor and the Sensors (A and B), together both read the rotation of the rotor and transfer motion to the motor axel

3.3 Calculated Kinematics and Workspace

It was explained earlier that the Walloid robot had four arms. The structure of each arm is made of three different prismatic joints that provide the robot arm with a linear telescopic motion in three directions which gave the robot high flexibility in moving around in the workspace with high precision , ***0.25mm in each joint direction*** (figure 3.6). As stated, Walloid project did not contain any workspace or kinematic calculations; therefore, one had to calculate the kinematics of this robot to know the range of workspace and also to develop logic to reach one point in the workspace. This is presented in this section.

3.3.1 One Arm, Three Prismatic Joints

Walloid might be slow, but it is very accurate. As all prismatic joints, the joints of this prototype can move back and forth in a linear motion along the joint (figure 3.6). Each prismatic joint is 175 mm and when opening the maximum length becomes 229.4 mm. This means our variable distance is around 54.4 mm. It was discovered later in this project that it was possible to measure this distance in 217 countable steps (counter variable in control program). This means that the accuracy level would be around 0.25 mm (figure 3.6). Navigation and positioning of Walloid is discussed in section 5.5). In addition, each prismatic joint consists of *one DC motor, in-house developed encoder (light fork sensors), motor jacket, rotor (the interrupter of light sensors) and a screw shaft* (figure 3.5). Hardware components would be discussed in details in next chapter (section 3.2).

3.3.2 Calculations

Kinematic calculations were one of the most time consuming parts of this project. Although the geometrical concept of the Walloid arms was discovered right away as an irregular tetrahedron, to find the best solution to reach the kinematic calculations became time consuming. Factors like *frustration, tiredness, wrong measurement of the dimensions, small mathematical mistakes in formulas and lack of accurate tools* made this process even harder. The lessons learned from this process were *to first of all to have clear questions that clarify the problem, and try to remember them during the process of solving the problem (question-guided approach)*. Moreover, *right and precise tools, training to use the tools, documented facts of the problem and assumptions are vital in order to solve the problem in an easier way*. As mentioned in introduction, this project's process was documented in a *weblog*. This weblog was updated through the process. This was very useful as one could easily read through old posts *to see why and when one approach was chosen*. This was used several times during kinematic

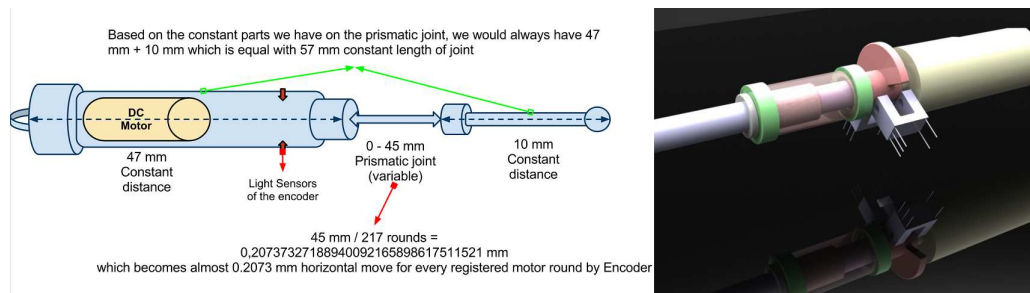


Figure 3.4: Walloid Prismatic joint

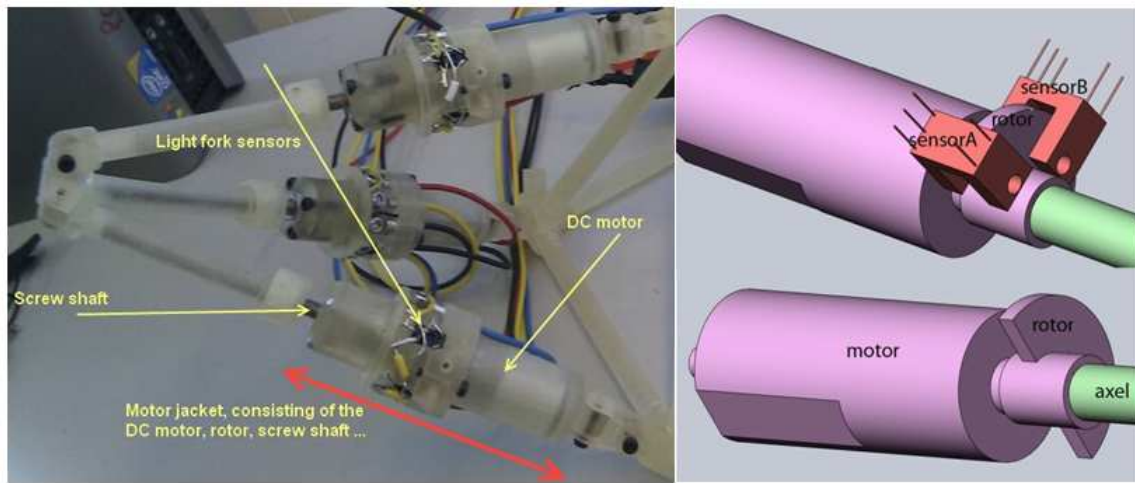


Figure 3.5: Walloid robot arm | 2: Motor-rotor-encoder design

calculations.

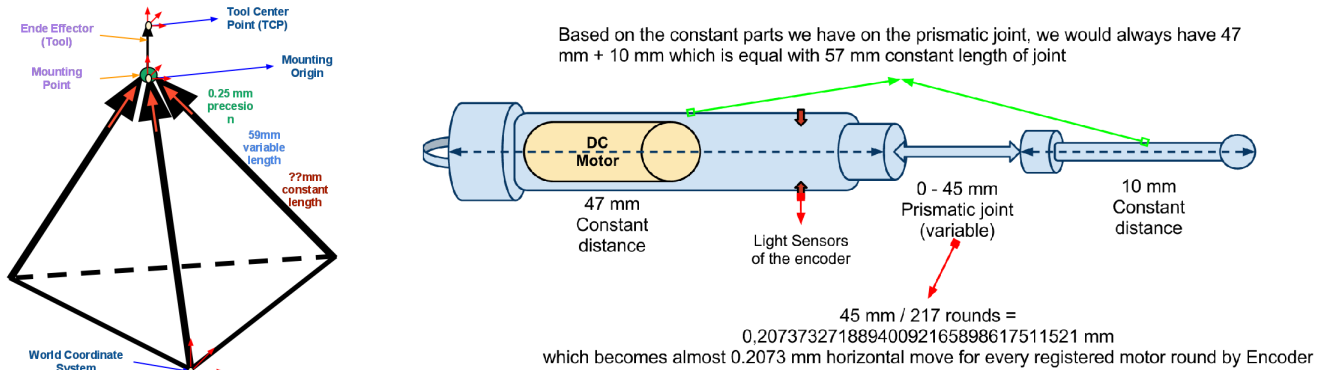


Figure 3.6:

Left to right: Arrow model of Walloid arm | Prismatic joint of the robotic arm

For calculating the kinematics of the three prismatic joints that are mounted together at a single point, it need to be looked as a geometrical shape (irregular tetrahedron). The early attempts were focused on solving the problem not through an irregular tetrahedron, which was complicated, but by breaking it into smaller simple geometrical parts (2D), which turned out to be useless. Later the irregular tetrahedron approach was taken. This approach became too complicated as well and still was not the easily computable logic that could be implemented in micro-controller level C.

The final approach in solving the problem was discovered accidentally by

reading through an article about Global Positioning System (GPS) which has exact similarity with Walloid arm design. GPS system needs one GPS unit to have contact with at least 3 satellites. Here it is assumed that starting point of each joint is the center points for a sphere around satellite (figure 3.7). The radius of this sphere is equal with the length of the prismatic joint (equivalent to distance to the GPS unit). First starts with two spheres (empty spheres). The intersections between them is a perfect circle. In case of knowing the radius of the third sphere (which is known in the robot), it would be possible to limit the intersected area to just two points. Finally GPS uses earth as the forth sphere, defining that only one of the two points can be on the surface of the earth, while here the orientation of origin and X,Y vectors can distinguish the right point, as the wrong one would fall into robot chassis (figure 3.7).

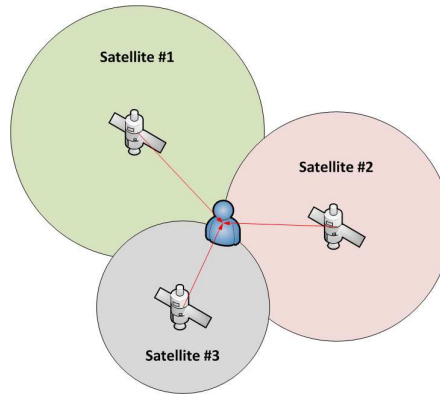


Figure 3.7: GPS positioning system

To solve this equation, it is necessary to set three sphere equations equal with the square of the length (L^2) of the prismatic joints. As expected solving these 3 quadratic equations results in two points, which one is not acceptable as the answer falls into the robot chassis (not physically reachable). This is solved by Matlab and the code is presented in Appendix. Later this approach was used to produce the workspace in 3.3.3 as shown in figure (figure 3.7). If R_1 , R_2 and R_3 are respectively the length of prismatic joints, and $[X_1, Y_1, Z_1]$, $[X_2, Y_2, Z_2]$ and $[X_3, Y_3, Z_3]$ are respectively starting point of each prismatic joint, then:

$$\begin{aligned} R_1^2 &= (X-X_1)^2 + (Y-Y_1)^2 + (Z-Z_1)^2 \\ R_2^2 &= (X-X_2)^2 + (Y-Y_2)^2 + (Z-Z_2)^2 \\ R_3^2 &= (X-X_3)^2 + (Y-Y_3)^2 + (Z-Z_3)^2 \end{aligned}$$

3.3.3 Workspace

To calculate the workspace, the length of each joint (equivalent to distance of satellite and GPS unit) is required. It was explained in Walloid specifications that the length of one prismatic joint (L) is the initial length ($L0$) of it plus the value of the encoder data (*counter, here shown by n*) multiply by 0.25 mm (each step in the counter is equal with 0.25 mm movement) 3.3.1. Experiments done during this project proved the range of counter (n) to be $[0-217]$, meaning maximum variable length to be 54.4 mm . The equation to calculate the length of one joint is shown in equation that follows:

$$L = L0 + (n * l)$$

Based on calculations done in section 3.3.2, with the help of previous equation calculating the length of one joint, the workspace was produced in Matlab. Figure 3.8 shows the workspace for the whole robot. This was implemented in Matlab.

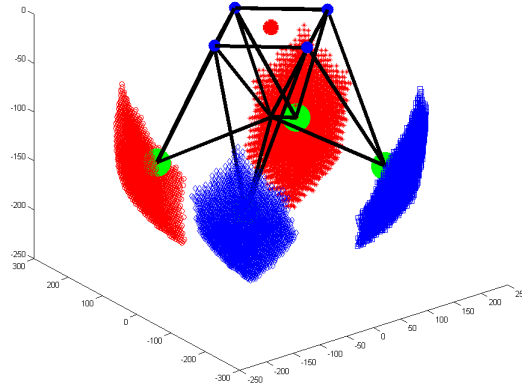


Figure 3.8:

Robot workspace calculated in Matlab by the logic discovered and developed here

The workspace of the robot is the area that the end effector can reach. However, the joints motion calculations were critical in calculating the workspace, while the fact of having an end effector connected to the mounting point was very important as well (different tool center point). Industrial manipulators solve this problem by having several convertible origin points [38,39]. Important origins in calculating the workspace are *world origin*, *mounting point* and

finally *one origin at the center of the end effector (tool)* which is called *Tool Center Point (TCP)*. The whole workspace can be calculated based on world and mounting point origins and later expand the calculation based on different types of end effectors (tools) used. Figure 3.9 shows the way these origins are chosen based on Walloid settings (world, mounting and TCP).

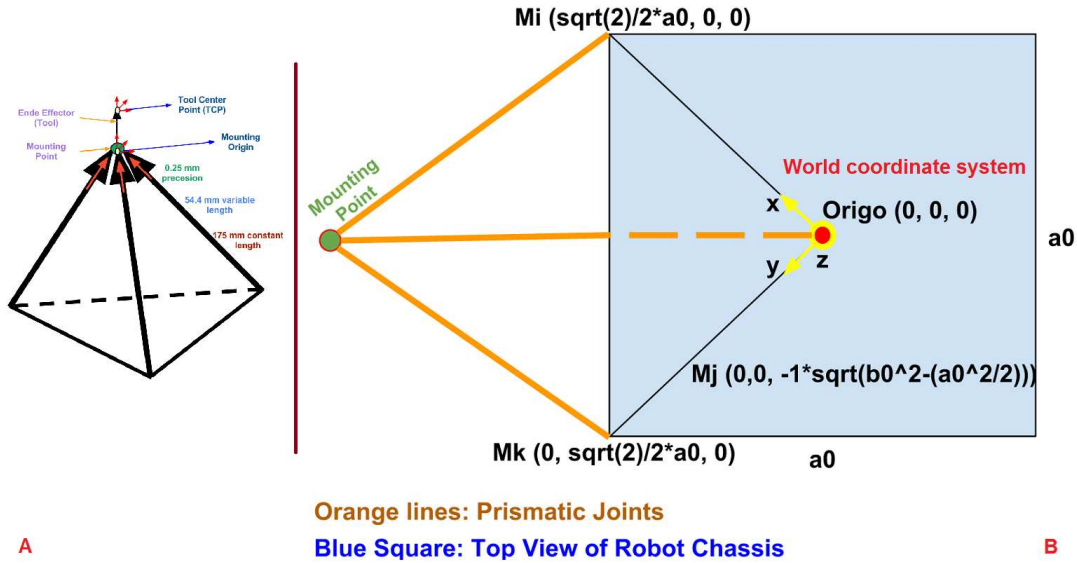


Figure 3.9:

A: Walloid arm | B: Top view of Walloid chassis with one arm(as the received part was)

At the end, it is important to mention that one can limit the workspace even more by setting limitations for the height of Walloid robot from ground (figure 3.10, left). When the length of arms are maximized, the robot has *least height from the surface and maximum workspace* and vice versa. Climbing, while using the maximum length of the arms, can reduce the chance to avoid and skip obstacles on the way (figure 3.10, left).

3.4 Review and Tech Upgrade of Walloid Robot

This section is dedicated to the review of the Walloid prototype. This section goes through *the discovered weaknesses* and tries to discuss the *upgrades (based on experiences gained during testing received parts)* that can help this ongoing project to become a better system. The weaknesses of Walloid prototype can be categorized in four different categories:

1. Crossing over adjoining surfaces

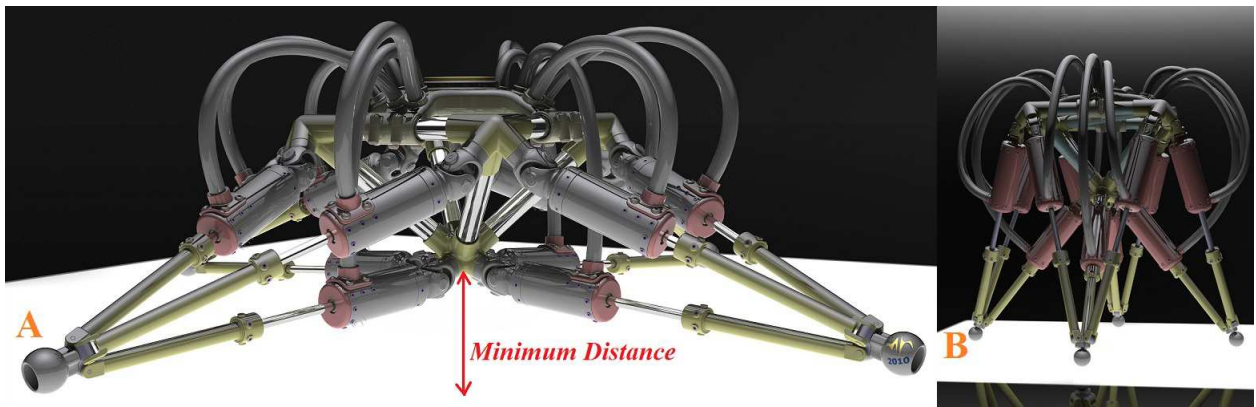


Figure 3.10:

The longer the feet extends, the lower the height of the robot gets (A) and this means less place for overcoming obstacles | B: Maximum height of Walloid

2. Materials and methods of production
3. Design weaknesses
4. Speed issues

3.4.1 Adjoining Surface Climbing

Climbing between adjoining surfaces (figure 3.11) is almost the biggest challenge that each climbing robot could face during the development phase. This problem is complex enough to be the subject of an independent work and therefore was only pointed out as a dilemma that should be considered as a further works point. The current design of the Walloid prototype might be able to climb some adjoining surfaces, but not all types.

3.4.2 Speed Issues

Although the Walloid was designed to be very accurate, it is too slow for an inspection / monitoring robot. Currently the *speed is around 0.9 mm/s* in prismatic joints direction and this is *too slow* for an inspection robot. This could be improved by *changing out the screw shaft with a hydraulic or air pump*. To be able to achieve this improvement, the encoder system should be re-designed from an angle measurement tool to a horizontal motion detector.

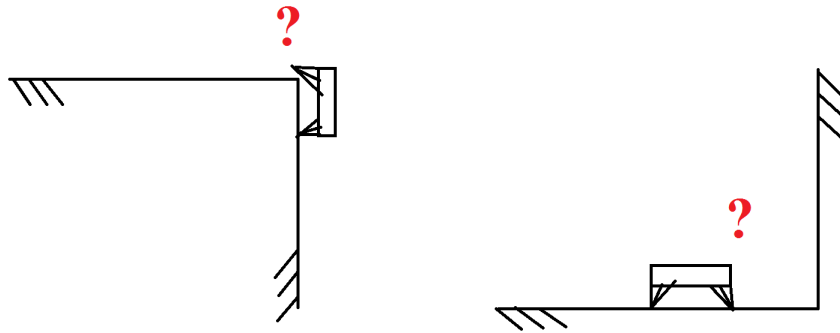


Figure 3.11:

Adjoining surfaces is a challenge and possibility can be a candidate for future works

3.4.3 Design, Material and Methods of Production

As mentioned before, the last received part of Walloid project was an arm. This part worked pretty well under tests, until the *disadvantages of 3D printing production* showed up once again. This was not the first time that quality of production would make problems during the project. The 3D printer that was used to produce these parts was a Connex500 model. This printer uses the polyjet technology which build up the model by adding layers of photopolymer on top of each other and use UV lamps to instantly cure them [40,41]. Lack of strength in 3D printer's products is a fact that needs to be considered in development process. Difficulties with 3D printers products have been reported in several occasions in academic reports from ROBIN students [37,42]. Some of these weaknesses in produced parts were *not only due to the production method, but design problems* (figure 3.12). The weak points in construction of the parts resulted in parts breaking exactly from the same spot in different occasions (figure 3.5).

Due to time constraints, and the time consuming process of re-designing, printing and fixing the prototype, it was decided to continue the work based on the assumption and experiences with the prototype in simulations (5.6).

Discussing these issues, one would wonder how these problems with the design could be avoided. The whole prismatic joint was placed in a jacket. The design of this jacket was very neat and small, but this fancy design led into some problems. 1. The *available room inside the jacket and the air circulation inside it were too small* for such purposes. 2. This lack of space and room for air circulation *left almost no place for the produced heat by the motor to be exchanged, neither for the expansion of overheated parts* (Figure 2.4). The other



Figure 3.12:

Left to right: 1: Part-1 is thinner than part-2 which imposes a weak point to the construction | 2: Weak points of rotor in ROBIN developed encoder | 3: Motor overheating and breakage results

design problem was in the rotor, where *differences in thicknesses, resulted it to be another exact weakness point* on rotor (figure 3.12) which broke several times during the process (including once during installation). It is believed that the jacket design can be perfected by some small design changes to remove the discovered weakness points.

Weakness spots (3.12) do not necessarily need to be there as it is imposed from the shape of the jacket that covers motor, rotor, encoder and the screw shaft. As changing the size of the jacket will not bring any restrictions, nor disadvantage to the current design, it is **strongly recommended to re-design the jacket** to prevent all the earlier mentioned problems. In the following 3D design one could see the changes in the jacket and then the rotor which is supposed to result in better construction strength (figure 3.13).

Table 3.2: Table of improvements and critical problems in Walloid robot design

Improvements	Critical problems
Removing the sharp edges as much as possible to reach better strength in robot structure	Bigger dimensions in the motor jacket to prevent <i>overheating</i> and <i>weak spots</i>
Screw shaft can be replaced by hydraulic or air pumps for speed improvement	simplifying the design of the rotor to reach more strength in design

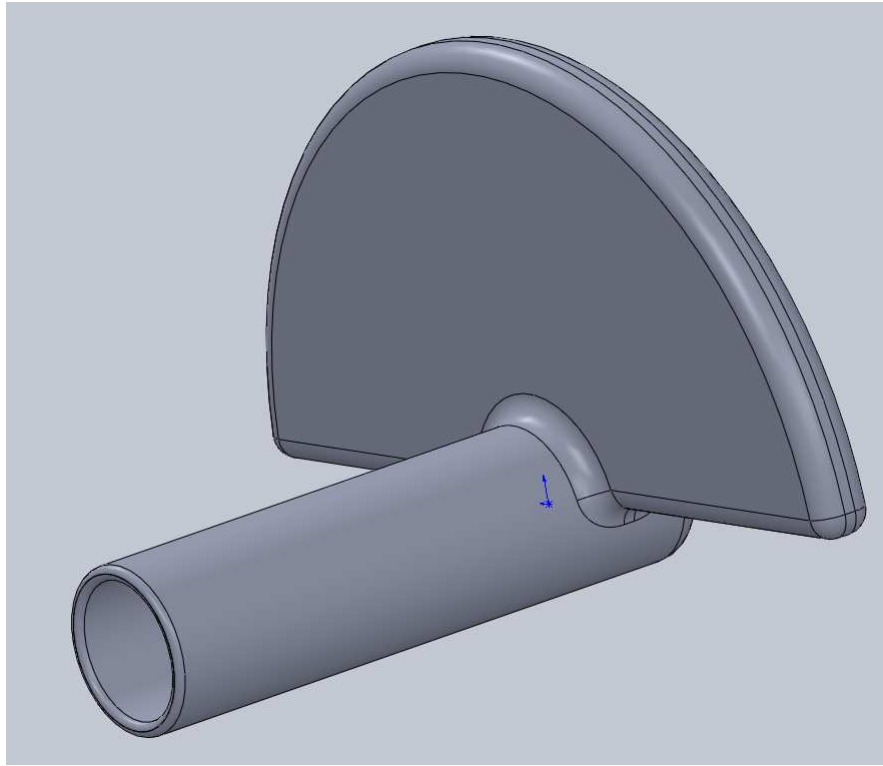


Figure 3.13:

Left to right: 1: Better construction as new rotor design with thicker walls, based on new jacket design | 2: The redesigned jacket giving more space

3.5 Summary

This chapter made an attempt in clarifying the status of the Walloid project in details. Different topics are discussed and already developed features such as "arms" and "grippers". After general specifications and configurations, the topic of calculating kinematics and workspace was addressed and was **calculated during this project**. These calculations were not a part of the features included with Walloid. During the process of calculation, the **similarity between the robot arm shape and the GPS system** was discovered (3.3). This benefited in shaping a logic in order to be used in control algorithm (section 5.5.1). The following table shows a summary of specifications of Walloid reviewed in this chapter.

Table 3.3: Summary of Walloid climbing robot configuration and specifications

Specification Name	Type
Robot Type	Climbing Robot
Locomotive Power	4xArms
Adhesion Solution	Grasping Arms
Micro-Contr0ller	Arduino Duemilanove
Motors	3 x DC motors
Encoder	In-house developed encoder (by Robin, University of Oslo)
Kinematics	*Missing. This was calculated during current chapter. *Robot arms positioning in the workspace was similar to GPS system
End Effector	To be developed
Climbing Gaits	To be developed

Chapter 4

Top Level Perspective

Questions provide the key to unlocking our unlimited potential.
- Anthony Robbins

4.1 Top Down Objectives

It's important to have clear questions before starting a process of solving a dilemma. Answers to these questions should clearly cover the solutions or at least the approach toward the solution. Applying this idea to further develop the Walloid robot project as an academic prototype for offshore platforms, one might wonder about a critical initial issue.

- What does Walloid project contain and what does it miss to be a semi-autonomous academic prototype for Offshore Platforms?

Chapter 3 tried to answer this initial and important question by naming detailed contained and missing features. Moreover one can categorize missing parts in following points:

1. Offshore Industry Point Of View
2. Climbing Strategy
3. End Effectors
4. Control System

Therefore an attempt began to cover missing parts to turn Walloid from a robot chassis to a climbing robot. However, as mentioned in 3.1, when the received parts broke down (3.4), the actual robot was not ready and the results

were only tested by self-developed simulations (5.6). The work which is done during this project can be categorized in three main categories, such as:

- Analysis of offshore platforms as an area of application
- Climbing Operation
- Control Hardware
- Control Algorithm

4.2 Analysis of offshore platforms as an area of application

This part was implemented by going through articles, master and PhD thesis's, industrial and governmental reports and standards. This application analysis goes through motivations for robotic automation in offshore platforms involving challenges and possibilities. This is presented in 2.4. The point of view obtained from this analysis makes one aware of robustness issues that were reflected in a separate chapter concerning robustness issues in development in chapter 7).

4.3 Climbing Operation

Simply a climbing robot would need four things to climb a vertical surface:

- Locomotive Power (already developed in Walloid)
- Adhesion Force (already decided in Walloid)
- End Effector, implementing the adhesion force (missing).
- Kinematics and Workspace (missing)
- Climbing Gait (missing)

Based on the list of points presented above, climbing operation work-plan was designed (5.1). Grasping arm according to above specifications was the base of the climbing process. Therefore, a series of attempts were taken to develop an end effector (gripper) and bolts. Before reaching the final step of climbing, calculations of workspace range and kinematics were required. These were calculated through this project and addressed in 3.3. Finally with regards to adhesion force and kinematics calculation, four climbing gaits were developed.

4.4 Control Hardware, a Distributed Embedded System (DES)

To have control hardware capable of controlling the Walloid with 4 arms, 12 motors and 12 encoders and in addition a number of sensors, further development of initial hardware component was necessary. Development process of this phase of the work is explained in 5.2. To achieve this a distributed embedded system approach was designed (figure 6.1), where one micro-controller was used as a master and four smaller micro-controllers were used as slaves, to control each arm (5.3). The distributed embedded design was implemented by using Serial communication (RxTx), I2C and ZigBee technology 5.3.

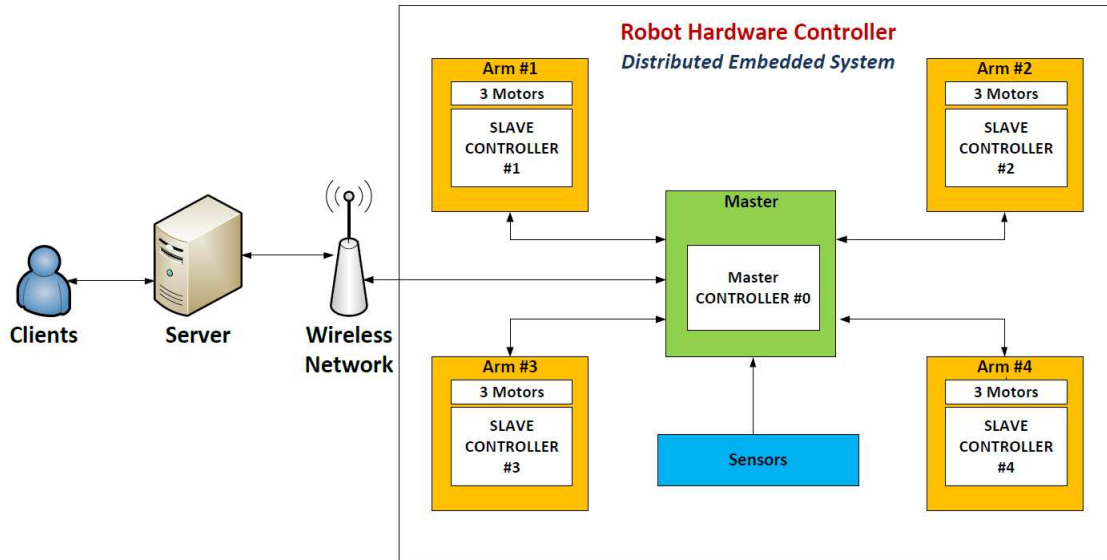


Figure 4.1: Top-Down view of DES, designed for Control Hardware System

4.5 Control Algorithm, a Distributed Navigation Program (DNP)

The top level design of the system presented in this project was changed during the whole process and landed finally on the server - client architecture (mature technology), which consists of three main nodes ensuring **automation, remote control, monitoring and high performance**.

- *Robot*

- *Server*
- *Client*

Regarding the DES designed in control hardware, a distributed Navigation Program (DNP) was developed 5.5 and divided between distributed between different embedded systems (micro-controllers) in the robot control hardware (figure 4.3).

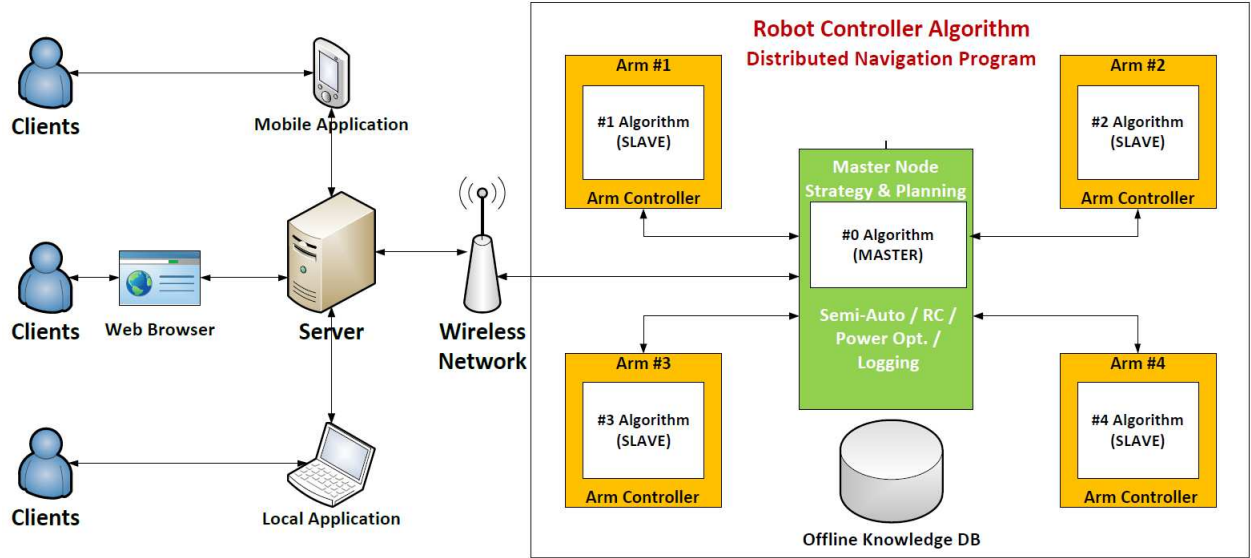


Figure 4.2: *Top-Down view of DNP designed, for Control algorithm*

To support both manual override and automation, DNP was planned to have three modes, Manual (Remotely Controlled), Auto (running automatically) and Teach (robot's learning mode). Walloid is supposed to be a semi-autonomous system. The level of autonomy in this project is simple and stable. Navigation's program machine learning process was designed under a very easy concept. If one steer Walloid through one process, Walloid, if in Teach mode, would be able to remember (store) all the positions it has been jogged to (steer to). Thereafter the robot, if in Auto mode, could read all stored positions in the order and jog again to those exact spots and consequently repeat the exact same operation (5.5.6).

4.6 Summary

The main purpose of this chapter was to give a top-down view towards the work which is about to be presented in following chapters. Due to the com-

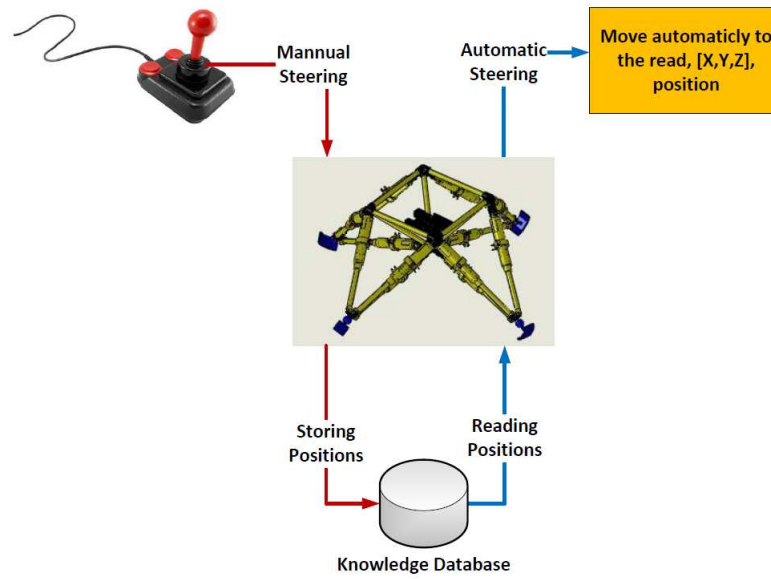


Figure 4.3: *Top-Down view of autonomy level of developed DNP*

plicated relations between different developed parts, this chapter was added to give a brief overview of the whole project. The descriptions and figures were presented without detailed information. Detailed information about the solutions are addressed during the development process. The following table tries to show a brief summary of the process in this chapter:

Table 4.1: Summary of top-down perspective to practical work done in project

Specification Name	Type
Analysis of area of application	Analysis of offshore platform was done by going through available academic literature and governmental/industrial reports about the subject.
Climbing Operation	The Walloid project misses end effector design and climbing gaits for the already ordered specification of arms and grippers.
Control Hardware	A Distributed Embedded System is designed and implemented which contains 5 micro-controllers and distributes the tasks between different micro-controllers, trying to shape a semi real-time system.
Control Algorithm	Based on specifications of the DES, a Distributed Navigation Program was designed and implemented in RSC architecture. In addition, self-developed simulation programs for graphical representation and testing purposes were planned due to the absence of a physical robot.

Chapter 5

Development Process

Make things as simple as possible, but not simpler.
- Albert Einstein

This chapter is dedicated to the development processes of earlier mentioned missing areas of Walloid project in 4.1. Development process here consists of going through ideas in early stages, maturity process and the reasons around denials, changes and approving usage of technologies. This section is divided into four main parts, regarding Climbing Operation (section 5.1), Control Hardware (sections 5.2, 5.3), Control Algorithm (sections 5.4, 5.5) and Simulations(section 5.6).

5.1 Climbing Strategy and Design

It was discussed in 4.1 that adhesion force and locomotive power, grasping arms, were already decided by Walloid project. Therefore no development was done in this area. The points for this development phase were decided to be end effectors that would be able to grasp and hang on bolts and a strategy for climbing (climbing gait).

5.1.1 End Effectors

The process of developing an end effector relies on the type of adhesion force (grippers in Walloid) and the manipulators that the end effector is going to be mounted on (arms in Walloid). This could easily be shown in a way the design of the end effector has evolved during the progress of this project (figure 5.2). The first six designs, belong to the X2 robot (initial robot for this project, figure 5.2), and the last two were designed for Walloid robot. None of these designs

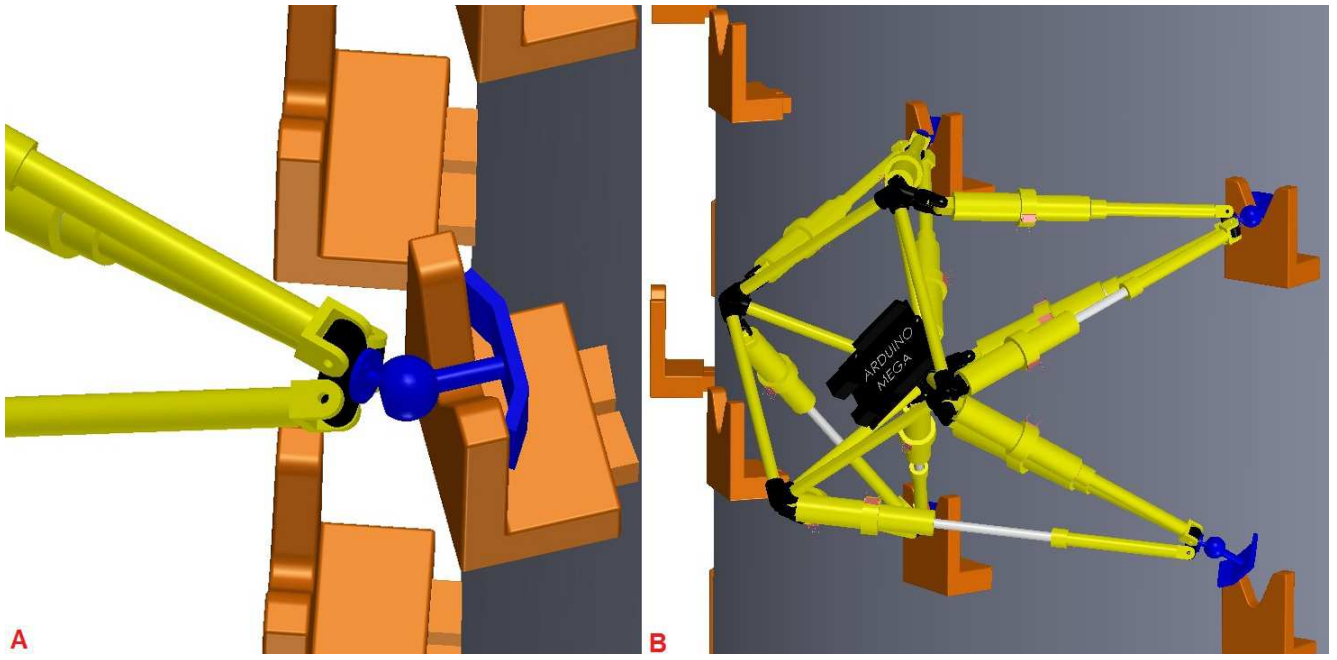


Figure 5.1:
Assembly of the designed end effector with the rest of Walloid chassis design in Solidworks

were tested in practice and were only designed in Solidworks (used CAD design tool). The final design (last end effector on the right) was assembled with the rest of Walloid design in Solidworks (figure 5.1). Specifications that were expected from an end effector for Walloid is presented in following list:

- Flexibility
- Eliminate the fear for fall in case of power interruption (redundancy)
- Tolerance against limited errors in positioning - Offset angle (redundancy)
- Auto charge in docking area

In addition to the end effector, the bolts have to be designed in a way that together they would satisfy the specifications. Figure 5.3 shows bolt types and also the corresponding end effector for such gripper solution.

This design was an interesting model, but failed in some key factors. According to figure 5.3 and 5.4, the design did satisfy the power independence requirement for staying on the wall (still hangs on the bolt without motors having power). However, it failed to be robust enough to tolerate errors in positioning (offset angle).

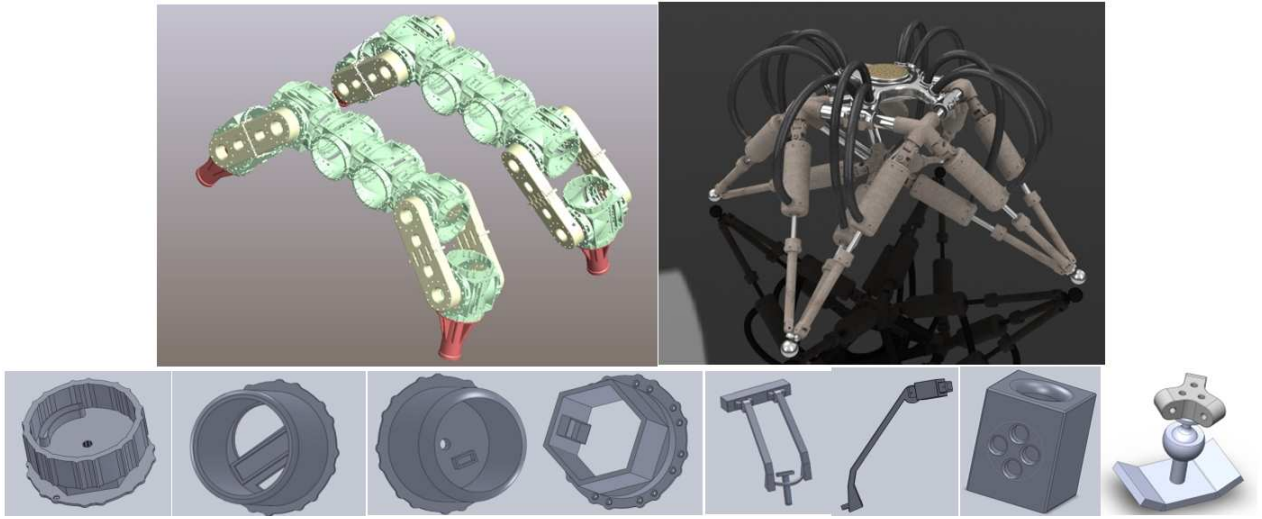


Figure 5.2:

Left to right: 1-6: Early designs for X2 prototype | 7-8: Walloid end effector designs

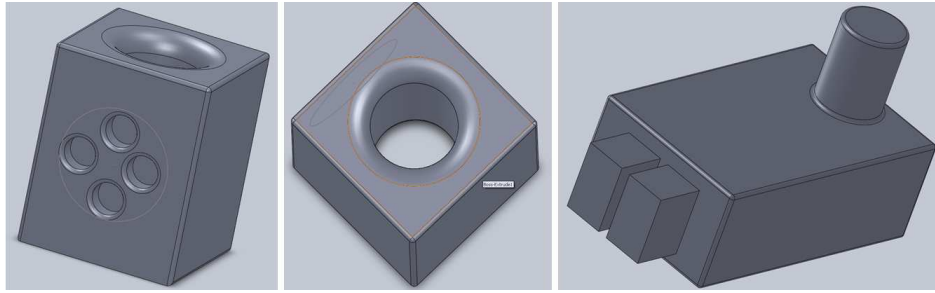


Figure 5.3:

Left to right: 1-2: semi-final End Effector design | 3-4: semi-final bolt design

Possible vibrations (especially in floating platforms) and lack of precision in the system can always lead into errors in positioning of the end effector on the bolt. However, if the bolt and end effector *were designed in a way to give some tolerance for error in positioning, less precision was required*. This error tolerance feature *increases the success chance* in positioning and could *save time and extra power usage and need for manual positioning in case of too many errors* (figure 5.4). The *error tolerance feature* gave the robot a room to have an *offset angle* (figure 5.4). The final end effector design was also equipped with a spherical wrist right on the top. This added design redundancy to correct errors in positioning during grasping. The spherical wrist would *bend according to the resistance from the surface* of the bolt. It's much *easier to correct such errors with smarter mechanical designs, rather than having several sen-*

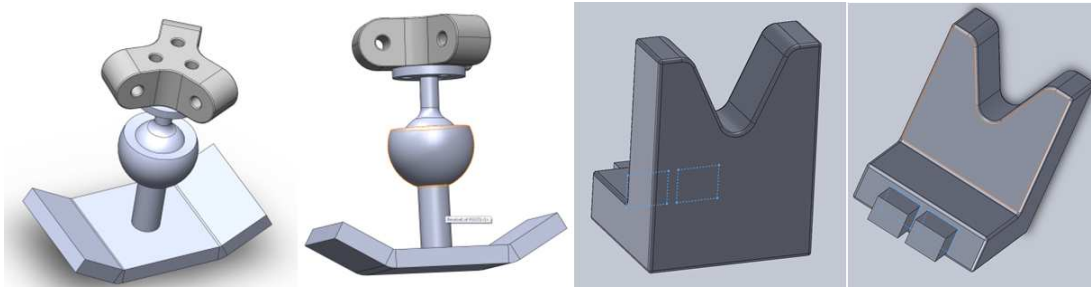


Figure 5.4: Left to right: 1-2: Final end effector design | 3-4: Final bolt design

sors gathering around the end effector and try to position the motors to reach 100% precision. The spherical wrist could be surrounded with rubber materials to be able to come back to the zero position of the end effector in case of absence of resistance from other materials (elastic properties of rubber). Such solution is easy to implement and very easy to maintain.

5.1.2 Climbing Gaits

Further development of Walloid to be an inspection / monitoring climbing robot would contain 2 main challenges:

- Redundancy and safety
- Speed

Climbing robots should be designed redundant. Fall of climbing robots from height could be fatal for both the robot and human workers in the environment. Therefore, to reach maximum redundancy, it was decided to develop simple and robust pre-programmed climbing gaits to reduce any chance of accidental behaviors (unplanned) from the robot. On the other hand low speed was an issue for Walloid robot (3.4.2) and inspection robots do need to move fast as they want to cover as much ground as possible and report discovered problems. Therefore increasing the climbing speed would be another area of focus here. This section represents four different climbing gaits, which three of them were functional and stable enough for implementation. However, due to the lack of physical presence of Walloid robot (and broken received parts), these gaits could not be tested in reality but only in a self-developed representation application (simulation). The presentation of these climbing gaits are discussed in 5.6.1. Later this virtual illustration of climbing process was connected to the control system to confirm reliability and functionality of the system.

Rotational Climbing Gait

One of the early ideas was a 3-phase climbing gait by adding the ability of rotating 180 degrees around one end effector (figure 5.5). This solution requires a powerful enough motor, in order to be able to rotate the whole robot chassis from 0-180 degrees around a single point. This would allow fast climbing speed and also the flexibility in choosing direction of moving (0-180, done 2 times would cover 360 degrees, meaning all directions).

$$\text{Torque} = (m.g).l$$

Although this gait enjoyed fast speed, it contained serious problems. Rotational movement around a single point had its own issues. The first problem was *the variable point of gravity which made torque force and inertia against this movement*. The length of the object is also important here and *the shorter the length is, the less energy is required* for rotation (Length is shown by L in figure 5.5). The force needed to rotate the robot chassis against these forces should be more than the sum of *both inertia and resistance torque*.

Rotating Force: $F > \text{Torque} + \text{inertia}$

Such large force produced only by a single motor, could result in *overheat, extra power usage and exhaustion of the sprocket wheels and other mechanical parts* involved in rotation process. These four extra motors would also impose *extra weight* on the chassis which was not desired. However, all these

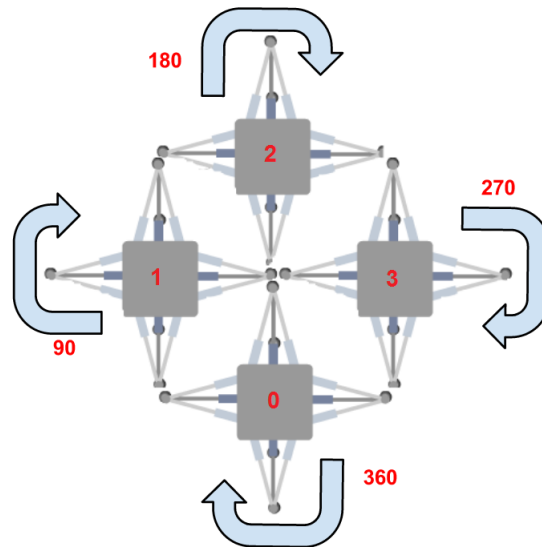


Figure 5.5:

Phase 1: Initial position | Phase 2: Rotated 90 degrees from initial position | Phase 3: Rotated 180 degrees from initial position (final phase)

challenges could be solved by choosing the right components and building materials. The *major issue* that made this solution unacceptable was the problem of having *three out of four arms to stay loose (not locked to the surface)* during the whole rotation process. It means *less stability and redundancy* which could even result in *instability* in case of wind and storm which are usual on offshore platforms. *Such big risk with all other maintenance and power consumption issues, in front of the only advantage of extra speed, show clearly that this solution could not be approved.*

Simple Pull-Up Climbing Gait

This easy climbing model consists of 5 phases. The cycle is completed after the fifth phase and the robot is completely climbed one step and is ready to iterate this operation again for further climbing.

- **Phase #1:** Left front arm releases, extends and grasps the next bolt on the line and locks.
- **Phase #2:** Right front arm releases, extends and grasps the next bolt on the line and locks.
- **Phase #3:** All 4 arms together would withdraw / extend while locked. Front arms would withdraw and rear arms would extend.
- **Phase #4:** Left rear arm releases, withdraws and grasps the next bolt on the line and fasten.
- **Phase #5:** Right rear arm releases, withdraws and grasps the next bolt on the line and fasten.

One of the **advantages** of this simple climbing pattern was that it would conquer the gravity only in one direction (opposite to the rotational climbing). It also enjoyed very good stability with having the minimum number of 3 arms locked at a time in the whole process. The load is also distributed equally between three fastened arms which prevented overheating of motors and extra load on one part of chassis. In addition, extra stability was reached in lifting operation by division of the needed force to lift up the whole chassis weight between all 4 motors on-board (4 arms). This would result in *perfect balance and would prevent overheating*. The only **advantage** of this climbing gait is the low speed and time consuming climbing operation. In following calculation, the time that each extend / withdraw operation takes is shown by **T_d** and the time that each grasp consumes is shown by **T_g**. Then the whole simple pull-up operation stride time would be:

$$T(\text{total}) = 8T_d + 4T_g$$

Assuming $T_d = T_g$, then:

$$T(\text{total}) = 8T_d + 4T_g = 12T_d$$

Optimized Pull-Up Climbing Gait

This simple vertical motion can be modeled partly to two different natural human motions. People doing pull-up training at gym, use their arms to lift their body up (figure 5.6), and the same does the robot in phase 3. However, regarded rear arms (feet in human training), things become different and the same motion could not be the source anymore. Pull-up training's aim is not climbing, but rather lifting the body up (the aim in phase 3). Therefore, one would not need to fasten the feet anywhere, but this is critical for the robot to maintain stability. On the other hand in rock climbing (dynamic climbing), feet are critical in holding stability (supportive feet / rear arms carrying the load). Both rock climber feet and robot rear arms are moved up one by one to join the rest of the body in climbing and lifting operation. One should bear in mind that due to joint limitations, the goal here is static climbing, while rock climbing is a dynamic climbing method. One might wonder why bother finding similarities and modeling an already developed pattern to nature. The answer would be that such modeling and discovering similarities could be inspiring and lead into lessons from natural evolution which has happened over many years.

Through modeling and re-thinking about the process, finally a new type of 5 phase climbing gait was developed, shown in figure 5.8. Based on different

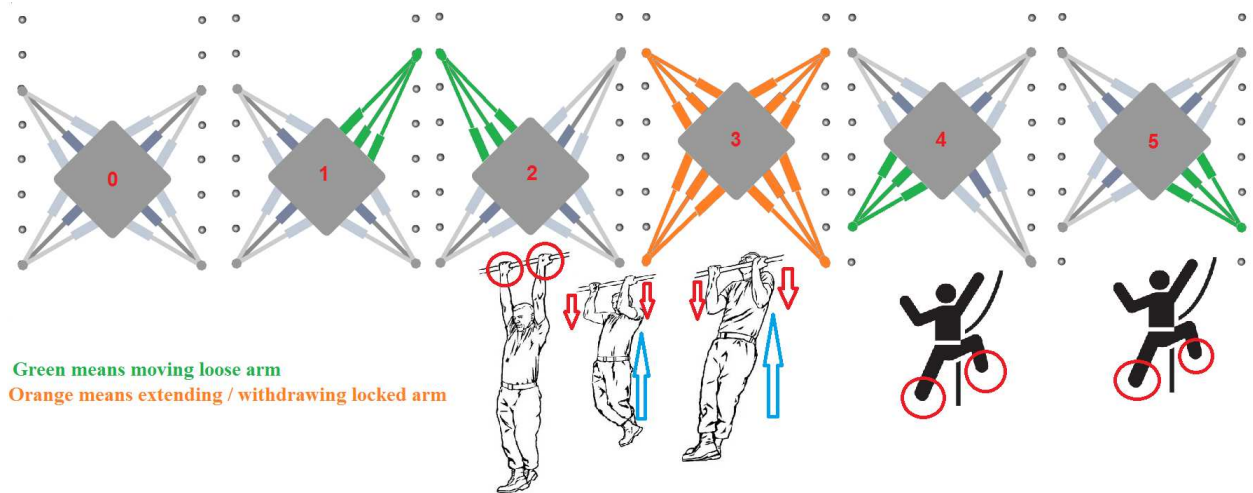


Figure 5.6: *Slow climbing and its similarity with pull-up exercise*

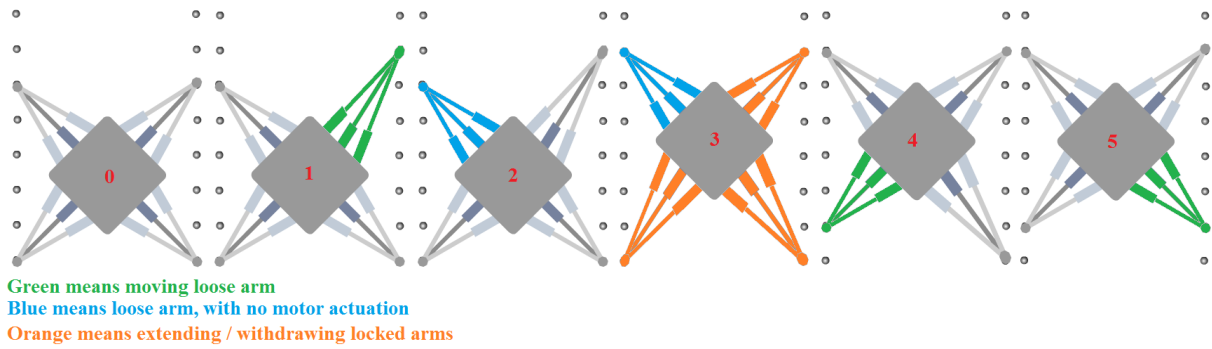


Figure 5.7: *Optimized slow climbing*

diagonal climbing models (rock climbing, ladder climbing, etc), having joint limitations in mind, a new gait was developed, trying *to save one extend operation* by *lifting up the robot one step earlier than previous pattern (phase 2 instead of 3)* and *accordingly skipping one extend operation* by the right front arm. This climbing gait *saved one grasp and hold operation* at each cycle, which for a robot with the speed issues was a big improvement. This gait is called for *optimized pull-up* gait due to this fact. Here all phases in figure 5.8 are listed in 5 steps:

- **Phase #1:** Left front arm releases, extends and grasps the next bolt on the line and fasten.
- **Phase #2:** Right front arm releases, and right after it all other three arms would start withdrawing (right front) and extending (both rear arms)
- **Phase #3:** Right front arm which was already released from phase 2, would extend and skip the next bolt, grasping the second bolt in the line.
- **Phase #4:** Left rear arm releases, withdraws and grasps the next bolt on the line and fasten.
- **Phase #5:** Right rear arm releases, withdraws and grasps the next bolt on the line and fasten.

The cycles here were different from previous pull-up model and at the end of the cycle, one stride (as previous one) was taken, but hopped over one bolt. For comparing the speed optimized climbing compared with simple climbing, the same distance should have been passed. Therefore it is assumed that phase number three does not belong to this cycle, but to the next cycle (so the exact same distance climbed as simple pull-up would be taken). Showing stretch / withdraw time by T_d and grasping by T_g . Then it would be:

$$T(\text{total}) = 6T_d + 4T_g$$

Assuming $T_d = T_g$ we have:

$$T(\text{total}) = 6T_d + 4T_d = 10T_d$$

Compared to simple pull-up algorithm, one could have a speed of **16.66% = 17% faster** and still **stable** (3 arms are fastened to the surface at a time).

This climbing gait, as the previous one is **conquering the gravity in one direction** (easiest), **stable with 3 arms fastened** to the surface at a time. The **only drawback** of this gait compared to previous gate (simple pull-up) could be **reducing the number of fastened arms during the lift operation from 4 to 3** which reduces force distribution affectivity and puts more pressure on remaining motors and make it less stable than simple pull-up. This is not a major problem as three arms (12 motors) should still be enough for such operation.

Dragging Climbing Gait

The last climbing gait discussed in this section is another method of climbing vertical surfaces. This pattern of climbing is not as fast as the rotating model, but faster than the other two pull-up methods. The robot initial configuration was different on the wall (figure 5.9) and during each stride in this model *only two arms* were involved and *the other two arms* were only for *supporting purposes*. Based on calculations done down here, this strategy would *speed up* climbing time by 33% during each stride.

Through modeling and re-thinking about the process, finally a new type of 5 phase climbing gait was developed, shown in figure 5.8. Based on different diagonal climbing models (rock climbing, ladder climbing, etc), having joint limitations in mind, a new gait was developed, trying *to save one extend operation* by *lifting up the robot one step earlier than previous pattern (phase 2*

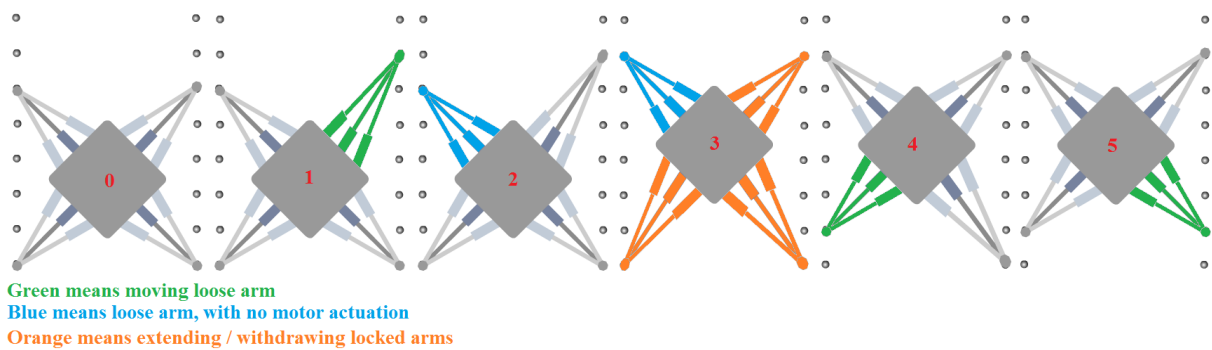


Figure 5.8: Optimized slow climbing

instead of 3) and accordingly skipping one extend operation by the right front arm. This climbing gait *saved one grasp and hold operation* at each cycle, which for a robot with the speed issues was a big improvement. This gait is called for *optimized pull-up* gait due to this fact. Here all phases in figure 5.8 are listed in 5 steps:

- **Phase #1:** Left front arm releases, extends and grasps the next bolt on the line and fasten.
- **Phase #2:** Right front arm releases, and right after it all other three arms would start withdrawing (right front) and extending (both rear arms)
- **Phase #3:** Right front arm which was already released from phase 2, would extend and skip the next bolt, grasping the second bolt in the line.
- **Phase #4:** Left rear arm releases, withdraws and grasps the next bolt on the line and fasten.
- **Phase #5:** Right rear arm releases, withdraws and grasps the next bolt on the line and fasten.

The cycles here were different from previous pull-up model and at the end of the cycle, one stride (as previous one) was taken, but hopped over one bolt. For comparing the speed optimized climbing compared with simple climbing, the same distance should have been passed. Therefore it is assumed that phase number three does not belong to this cycle, but to the next cycle (so the exact same distance climbed as simple pull-up would be taken). Showing stretch / withdraw time by T_d and grasping by T_g . Then it would be:

$$T(\text{total}) = 6T_d + 4T_g$$

Assuming $T_d = T_g$ we have:

$$T(\text{total}) = 6T_d + 4T_d = 10T_d$$

Compared to simple pull-up algorithm, one could have a speed of **16.66% = 17% faster** and still **stable** (3 arms are fastened to the surface at a time).

This climbing gait, as the previous one is **conquering the gravity in one direction** (easiest), **stable with 3 arms fastened** to the surface at a time. The **only drawback** of this gait compared to previous gate (simple pull-up) could be **reducing the number of fastened arms during the lift operation from 4 to 3** which **reduces force distribution affectivity and puts more pressure on remaining motors and make it less stable than simple pull-up**. This is **not a major problem** as three arms (12 motors) should still be enough for such operation.

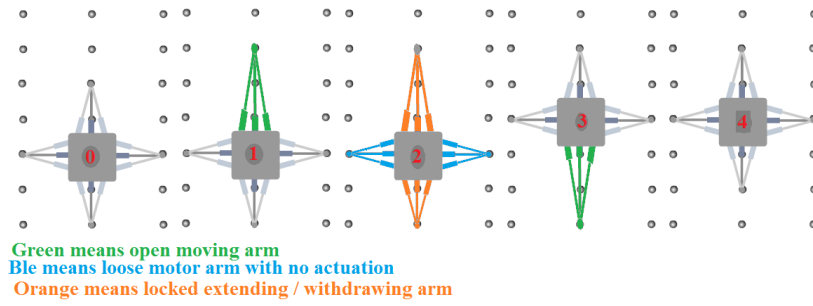


Figure 5.9: *Faster variation of climbing gait*

Dragging Climbing Gait

The last climbing gait discussed in this section is another method of climbing vertical surfaces. This pattern of climbing is not as fast as the rotating model, but faster than the other two pull-up methods. The robot initial configuration was different on the wall (figure 5.9) and during each stride in this model *only two arms* were involved and *the other two arms* were only for *supporting purposes*. Based on calculations done down here, this strategy would *speed up* climbing time by 33% during each stride.

- **Phase #1:** Front arm releases, extends and grasps the next bolt on the line and fasten.
- **Phase #2:** Both side arms release.
- **Phase #3:** Rear arm start extending and front arm starts withdrawing. This ends with side arms fastening to the bolts.
- **Phase #4:** Rear arm releases, withdraws and grasps the next bolt on the line and fasten.

The *advantage* of this gait is the *fast speed*, but at the same time it *reduces redundancy by having only 2 arms fastened* to the surface during lift-up. Moreover another important issue about this type of climbing model was the fact of *having a tail* which increases the stability and prevents the body from pitching back [43,44]. Previous pull-up models did not enjoy such characteristics. The importance of the tail in stability of climbing is an issue that was discovered by engineers and later was confirmed by biologists in animals like gecko [44]. Showing extend / withdraw time by T_d and grasping by T_g , one could write:

$$T(\text{total}) = 4T_d + 4T_g$$

Assuming $T_d = T_g$ so that:

$$T(\text{total}) = 4T_d + 4T_g = 8T_d$$

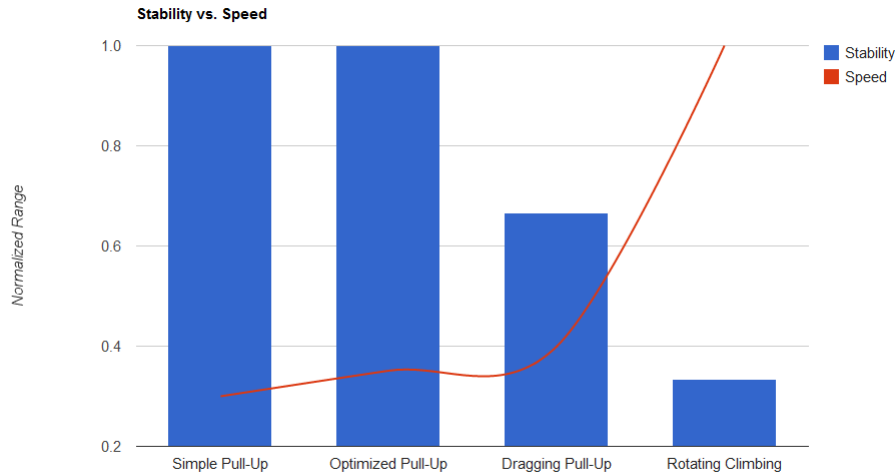


Figure 5.10:

Speed and Stability have opposite relation to each other. The factor of stability is based on minimum number of locked arms to the wall during operation

Compared to simple pull-up algorithm, dragging gait is **33.34% = 33% faster**, but **less stable** than previous pull-up gaits regarding the number of the locked arms, but one should not forget the issue of tail support. Finally it is important to mention that charts, virtual representations and analysis of the theory here, are presented at 6.2. Beside saving time and accordingly better speed, the optimized gaits save some motor actuation which results in lowering power consumption (figure 6.3).

5.2 Control Hardware, the Distributed Embedded System

5.2.1 On-board Motherboard

The year of 2010 was booming time for notebook pc also known as Eee PC's. *Having a notebook on-board the robot* would have huge advantages on processing power, so the idea of a robot that includes a striped notebook (just motherboard and a Solid State Disk) sounded very tempting. These machines have exceptionally low power consumption (7- 10 hours, depends on the battery) and weight usually under 1 kg (including chassis, screen and etc) [45]. Removing all the extra parts would increase battery life and reduce the weight. Alternatively

Table 5.1: Advantages and disadvantages of using a notebook motherboard in a climbing robot

Advantages	Disadvantages
Higher processing power	Not Embedded / Real-Time System
Extra Processing Applications (Matlab, etc)	Heavier than alternative micro-processors
Light battery with low power consumption	Less computation power than servers (direct connection by micro-controllers)
USB host providing power for other parts	Extra battery usage because of unused extra components that cannot be removed
Built-in networking components	Tolerance issues as these components are not built for frequent shock loads and vibration
Light weight	Not designed for extreme weather condition / salt
	Not designed for operating in high humidity
	Redundancy issues in case of system failure (SW and HW issues)

one could also think of using an extra ordinary alternative computer developed lately. The brand new Linux based, key ring sized *Raspberry Pi laptops with ARM11 processor (700MHz) and 1W power consumption* [46]. This inexpensive machine costs around 25-35\$ per today and are being developed for providing IT infrastructure for undeveloped countries. Table 5.1 shows the advantages and disadvantages around usage of such hardware as a control system.

In comparison with only micro-controller driven systems, systems with such offline computational power on board are very valuable choices. However the concerns about redundancy, non-deterministic (unreliable) basis of notebook OS (fair basis policy in running codes) and tolerance issues drive one to set aside such solutions. Finally it should be mentioned that this solution, if developed exclusively for offshore with required specifications, can increase the local processing power (however, servers are far more powerful) and result in smarter systems that could operate perfectly offline. **Figure 5.11** shows the photos of the notebook motherboard, Raspberry Pi motherboard and the idea of setting up the system with such motherboards.

5.2.2 Micro-controllers, the embedded system

On the other hand, the alternative solution for the initial notebook motherboard idea, was to focus on using micro-controllers as an embedded system and wirelessly forwarding the raw data stream for further processing (if necessary) to servers with high processing power (figure 5.12). The *main advantage* of such system is *deterministic* properties of an embedded system (the chance to know which code and when it is running). Table 5.2 shows the advantages and disadvantages of this choice. As the systems was divided into smaller parts with standard input and output, the complexity of the development work dropped and trouble shooting of the system became much easier. E.g. each embedded system could have been tested separately. However this approach made the system dependent on server connection and resulted in lower processing power. Regarding such disadvantages, it was decided to increase the number of micro-controllers onboard the system to increase computing power. The issues around

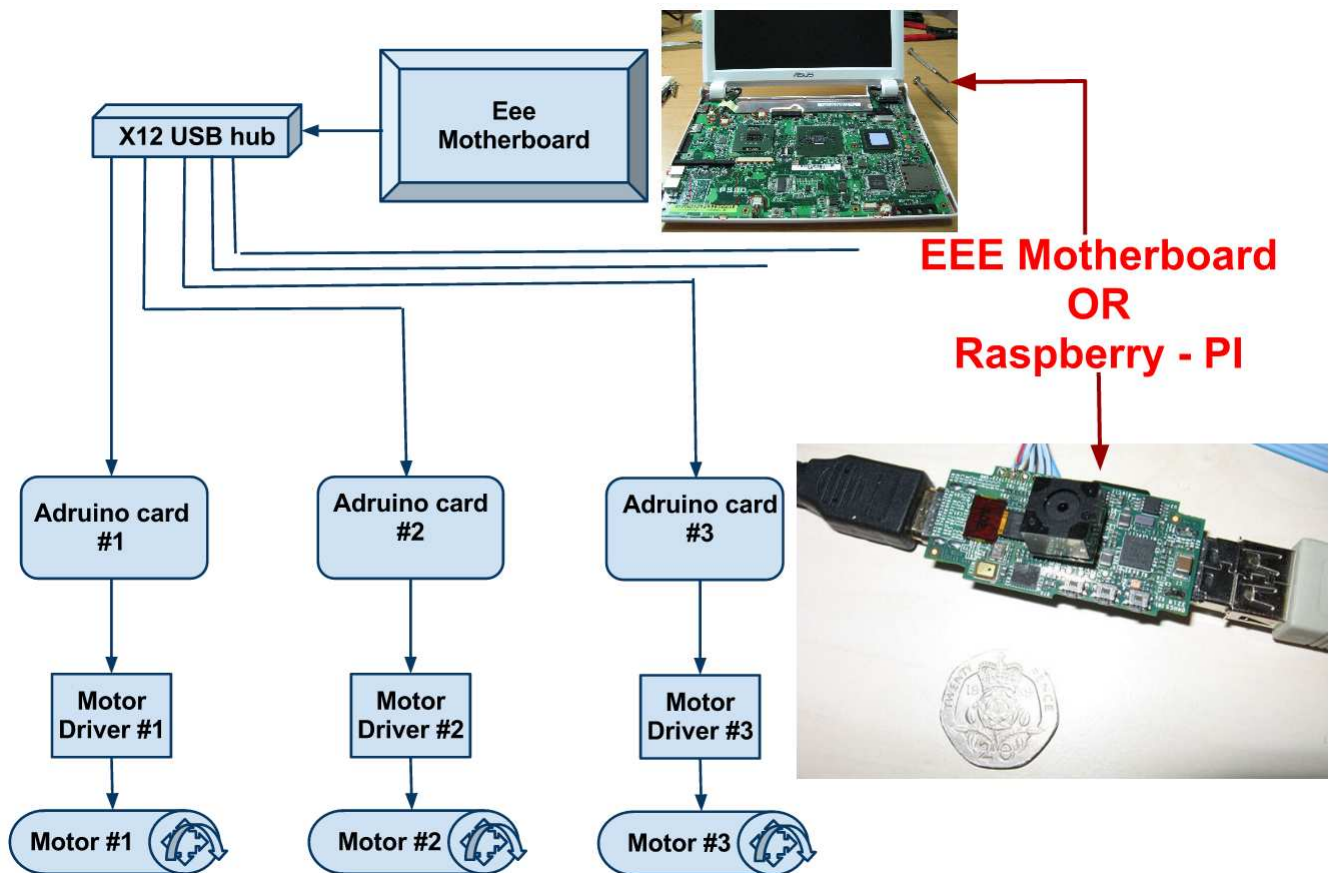


Figure 5.11: Left - Asus Eee 900 - Right, Raspberry Pi motherboard

Table 5.2: Advantages and disadvantages of using micro-controllers connected to server as the control hardware [26,27]

Advantages	Disadvantages
Low power consumption	Lower local processing power than alternatives
Determinism	Dependency on server connection
Light and small	Networking Components, not built-in
Easier Development	
Easier Troubleshooting	
Higher processing power combined with servers	
Easier proofing due to dimensions	
Inexpensive variant	

this and inter-communication between different micro-controllers in the robot is addressed later in 5.2.3. The issue of offline performance is also taken up in 7.7. One might wonder why it was not decided to have a more powerful micro-controller instead of going for a multi-micro-controller approach. The answer to this question is the *availability of components in the lab* and the *potentials that lie inside the multi-microcontroller approach* 5.2.3. The best available Arduino board, Mega, did not have enough ports for all motors and encoders on robot. Twelve motors would need four ports for their motor driver, two ports for each motor's encoder and in addition the sensors would come. The next section is dedicated to justifying the decision made here to move towards a multi-processing node system.

5.2.3 Centralized vs. Distributed Embedded Systems

A single embedded system, assumed having enough processing power, in charge of the whole system would have its own advantages and disadvantages. Centralized processing enjoys having all data gathered in one place, while in distributed systems; the raw data is always being exchanged between nodes, which require transaction time and processing power. On the other hand, planned distribution and division of tasks among inter-connected embedded systems could reduce the required necessary processing power on each node. Table 5.3 tries to show these points.

The long polling and interrupt queues could affect the system functional-

ity dramatically. This was the case in Walloid as number of encoder readings during lift operation was between 18 - 24 continuous simultaneous readings (depends on the chosen gait). This number only belongs to the mechanical operation and extra necessary sensors are not included. Such issues would decrease the determinism of the system, as the system would start being pressed from too many readings, while some of these could stop other processes and would run first (interrupts) [47]. Interrupts are valuable features on micro-controllers, but in case of frequent occurrence could make the system very slow.

Moreover, *redundancy* in DES is a very sensitive point. In a centralized system, everything relies on one single component which could break down. In such cases the whole system would be totally unreachable. However, in case of having multiple processing units on-board in a distributed system, even in case of one component failure, other processors (at least) could transmit the location to the service personnel, or even localize the error. This would result in less time used for recovering the robot and more efficient system design.

Due to all these above, the solutions of choosing multi-embedded system was preferred and chosen. However as table 5.3 shows, there are some challenges in distributed system design which should be closely looked into. Solutions are presented during the work to these disadvantages to make their effect

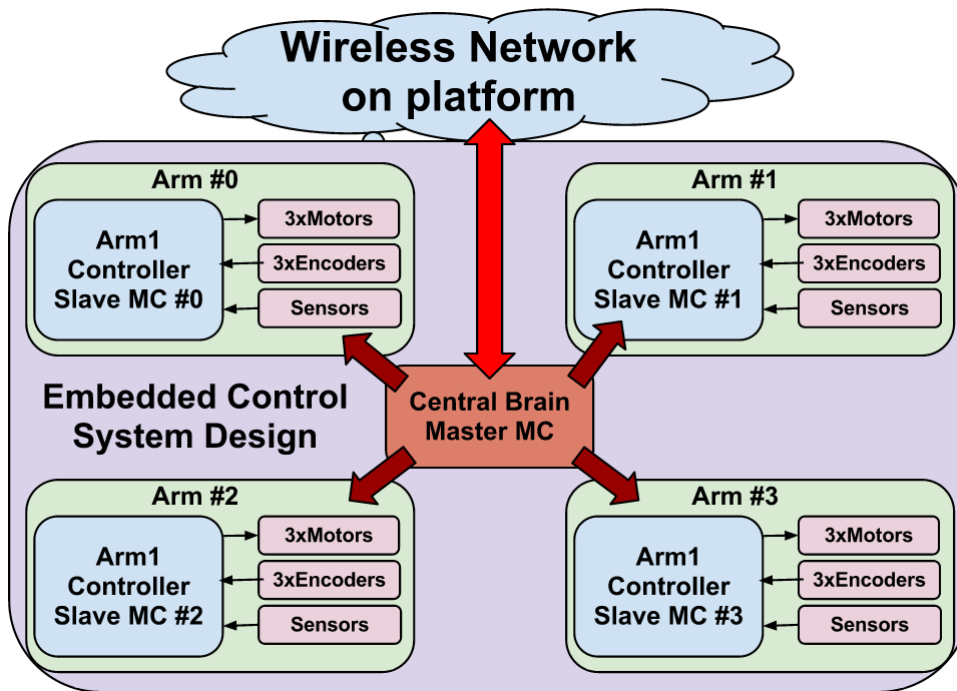


Figure 5.12: Embedded system / Server communication

Table 5.3: Centralized vs. Distributed Embedded System

Type	Advantages	Disadvantages
Centralized Processing	+Simpler Electronic scheme +Simpler programming issues +Centralized decision making	*Long interrupt traffic *Long polling traffic *Less redundancy (everything relies on one component)
Distributed Processing	*Increased Processing Power *Short interrupt traffic *Short polling traffic *Redundancy (several processing units on-board)	*Continuous need for exchange of data *Higher Design Complexity *Inter-Connection Challenges *Higher Power Consumption

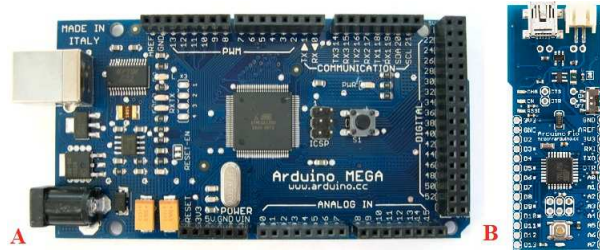


Figure 5.13:

A: Arduino Mega with AVR Atmega 1280 B: Arduino Fio with AVR Atmega 328P

as small as possible. Section 5.3 is dedicated to the choice of inter-connection medium and an attempt was made there to simplify the design as much as possible. The amount of exchanged data was also tried to be minimized by developing a communication protocol (section 5.5.5). Finally higher processing power issue was tried to be solved by utilizing sleep policies in navigation program (section 5.5.3). After deciding about a distributed design, it is now time to choose what kind of components would be proper on each node. The next section of this report would go through the choice of used micro-controller in the distributed embedded system design (DES).

5.2.4 Arduino Boards, the chosen embedded system

Micro-controllers can be divided into two categories, *Complex Instruction Set Computer (CISC)* and *Reduced Instruction Set Computer (RISC)*. The focus here is mostly on RISC models which benefits *simple design, low power consumption and small dimensions*. CISC models are more powerful, specialized and MACRO-like in their programming concept which is out of the focus area [26].

Having micro-controllers as the logical decision making units, it is useful to take a look at internal parts of a micro controller. One micro-controller usually consists of CPU, RAM (Random Access Memory), ROM (Read Only Memory), I/O ports, Serial and parallel ports timers, analog to Digital (AD) / Digital to Analog (DA) converters and nonvolatile memories (like EEPROM) [27]. Components such as serial ports (section 5.3) and EEPROM (section C.3 in Appendix) were directly used during implementation of the control hardware and software. There are many micro-controllers and platforms for micro-controlling in the market. Arduino, Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many other alternatives that do the same job [48]. Among all these choices Arduino was chosen as it is the right choice for prototyping purposes and contains the features that are important in the process of prototyping a solution [48]. Arduino boards with AVR micro-controllers enjoy following features [49]:

- Fair and reliable performance for prototyping
- Simplicity (HW designs and programming)
- Availability (in terms of SW, HW and learning materials such as a helpful community, tutorials, forums and local experiences at ROBIN group)
- Cross-Platform (Ability to run on Linux, Windows, Mac support without any modification)
- Open-Source (in terms of SW, HW. The chance for re-design and re-use in industrial applications. All plans for the modules are published under a Creative Commons license)
- Inexpensive HW

Due to all these points Arduino boards with AVR micro-controllers are the final choice for implementation of the distributed embedded design shown in figure 5.12. Going into details of choices made here, Arduino Mega (with AVR Atmega 1280) was used as the central brain (master controller) of the control hardware, while Arduino Fio (with AVR Atmega 328P) was used as the slave micro-controllers in the design. As the time did not allow to enter all details, it was assumed that connecting to higher level wireless networks could be easily done using an extra WiFi modem (WiFly shield) for Arduino boards [50]. On the other hand, the issue of inter-connecting several micro-controllers is critical in building the control hardware in the distributed embedded design in practice. This critical issue is explored in details in 5.3 and the top level plan (figure 5.12) is implemented with mentioned Arduino boards and proper inter-communication technologies.

5.2.5 Alternatives

Arduino has its own limitations as well. Here is a list of limitations using Arduino would impose to a project.

1. Arduino cannot be a USB host, meaning you cannot connect and power up USB devices to it.
2. Single serial port on standard Arduino boards, makes it possible to be connected to one device at a time (only Arduino Mega has 4 serial ports)
3. Arduino is designed for prototyping and educational purposes.
4. Arduino does not fit industrial applications directly, as Arduino requires 5V voltage, but mainly industrial projects require 24V.
5. Lack of Controller Area Network (CAN) support which this need is lately answered by a CAN shield for Arduino boards [51]

Regarding to alternative solutions, as discussed earlier, a notebook motherboard with local DAQ cards is a suitable choice (5.2.1) but for mobile systems such as Walloid robot, *Field Programmable Gate Array (FPGA) or Digital Signal Processors* (Specialized microprocessor with optimized architecture regarding mathematical operations) could also be considered. When discussing the specifications for the industrial vision, FPGA is a very tempting choice as it benefits all the following features.

- High reliability
- High determinism
- High performance
- True parallelism
- Reconfigurable

FPGA's are fast and reliable processing units and benefits from directly programmable logic gates. They are mainly used for high speed control systems, intelligent DAQ, digital communication protocols, sensor simulation and co-processing. The properties and area of application can easily show that FPGA could be one of the right alternatives for further development of a smart climbing robot.

5.3 Distributed Embedded System Design

Choice of inter-connection of Arduino boards is the topic of this section.

As showed before in figure 6.1, a DES is designed to contain *five micro-controllers*. The *master micro-controller, Arduino Mega with ATmega 2560 AVR (called master controller here after)*. Master controller is the central logic of the system and the master algorithm of navigation program would run on it. This unit is connected to *four other slave micro-controllers* (slave-controllers here after). Each slave-controller is just in charge of one and only one arm and the slave algorithm (positioning) would run on it. These limited intelligent controllers are in charge of three prismatic joints and the sensors around them (e.g. passive control sensor). *In another word, the master controller is in charge of strategy planning and the slave controllers are responsible for positioning the arms.*

The DES design allowed the master controller to be in direct contact with the server (figure 6.1) and also specified direct communication between all embedded systems (micro-controllers) in the design. One can name several cable-based and wireless solutions for the inter-communication of micro-controllers. The next two sections would try to evaluate such solutions, finding the right protocol or standard to implement this system.

5.3.1 Cable Based Distribution

Inter-Integrated Circuit, I2C

I2C or Inter-Integrated Circuit is a communication protocol to connect an embedded system (micro-controller) to other devices (E2PROM memory). The communication technology in this protocol is based on two bidirectional lines that are pulled up with resistors [52].

1. SDA: Serial Data Line
2. SCL: Serial Clock

I2C is a multi-master serial bus, which only allows one master at a time. The practical distance of communication is limited to few meters and the longer the cables are, the lower the speed would get (3.4 Mbit/s at high-speed mode, but 100 Kbit/s is common) [26,52]. The advantage of using I2C is the simplicity of connection design and its availability on Atmega AVR micro-controllers (analog pin 4 is SDA, analog pin5 SCL) [53]. Referring to the *disadvantages*, one can mention the *one master at a time* properties and the practical problem of writing back to master at any time faced during implementation. The second issue

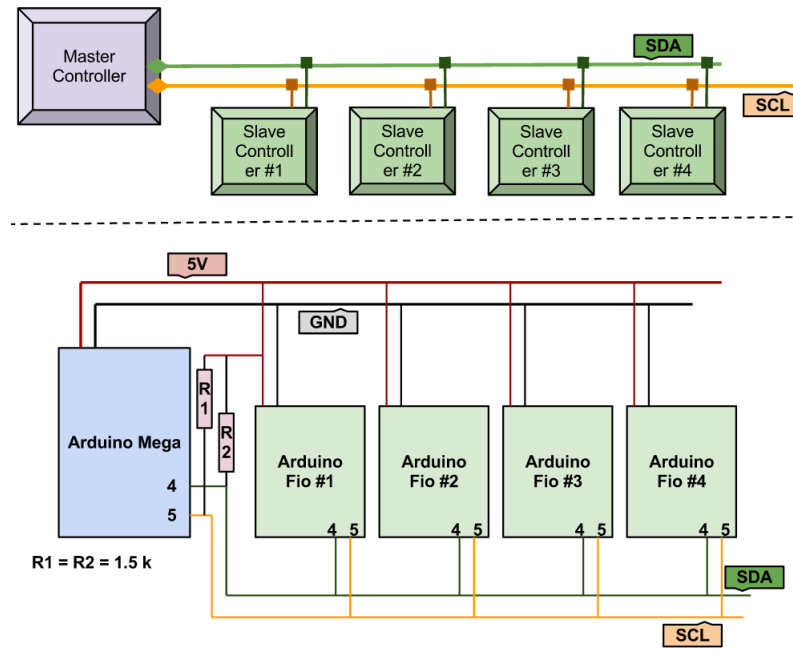


Figure 5.14: *I2C schematic for inter-connection of Arduino boards*

was discovered while implementing this inter-connection method (figure 5.14) and no solution was found around it. I2C works quite well, when it writes to a device and then expecting a feedback. This resulted in slave micro-controllers not being able to write to the master at any time (in case of emergency) and this was not acceptable for the design. What DES design needed was a chat (2 way communication) connection so the master and slaves could interact in a fast and stable way. Based on the experiment done with I2C in this project, it could not be qualified for this type of connection.

Ethernet, IEEE 802.3

Utilizing Ethernet technology is another approach in building a network of inter-connected embedded systems. Ethernet is a family of networking technologies (standardized under IEEE 802.3) for shaping Local Area Networks (LAN) [54, 55]. Ethernet is *mature, stable, fast* and in addition the most popular method of transmitting data in large scales [55]. The idea here is to prototype a LAN consisting of several micro-controllers with additional Ethernet adapter [56]. Ethernet star model is an interesting implementation for DES design. It consists a router (central node) in the middle and all the other surrounding nodes which are connected to it and at the same time two specific nodes can

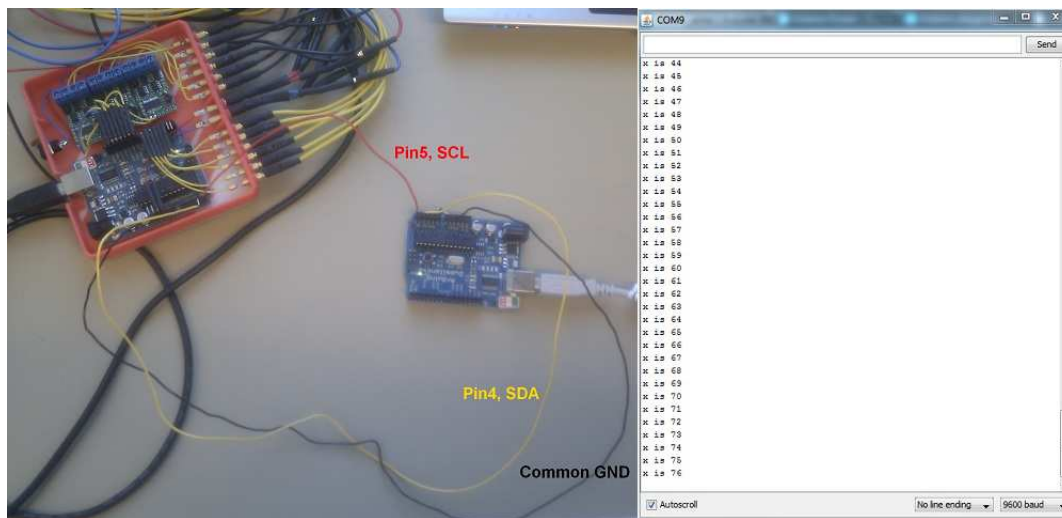


Figure 5.15:
Implementation of I2C connection of micro-controllers | 2: Program results of testing the I2C connection

communicate directly together (static IP). This reduces unnecessary flow of information.

The main disadvantage of this method is *high energy consumption for powering the central router* compared to other cable based method (sections 5.3.1, 5.3.1). Therefore, this *major disadvantage* imposed to a mobile robot (dependent on battery) was not acceptable and Ethernet was set aside. Ethernet is used in some of the tests during this project and showed very useful (5.5.4). Regarded discussions here and other applications that Ethernet is vastly used in, it can be concluded that Ethernet is more suitable for fixed robots (e.g. industrial manipulators) than mobile robots due to unlimited access to power consumption and their fixed place allowing Ethernet cables to be mounted.

UART/USART, Serial Communication

UART/USART (universal synchronous/asynchronous receiver/transmitter) is an integrated circuit allowing serial communication between different devices [26, 57]. This circuit equips the board with a receiver (Rx) and Transmit (Tx) ports which would be the physical layer for communications standards (RS-232 at Atmega 1280). Differences between this method and I2C is the lack of common clock which makes the whole system dependent on starting bit (internal clocks sampling Rx) [57]. Therefore it's very critical that transmission bit rate agrees on both sides (easily forgettable when changing the value on one

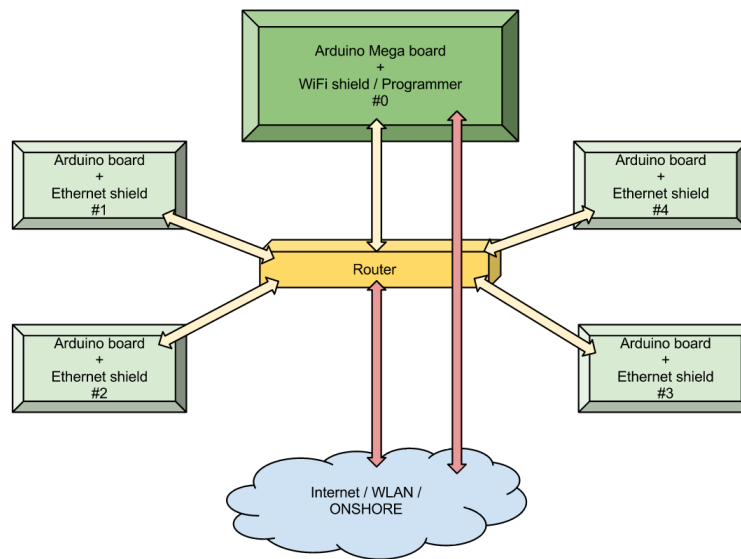


Figure 5.16: *Arduino star network*

side, but not the other). Setting up this kind of communication between two micro-controllers is *extremely simple*. The only thing had to be done was to connect the Rx pin of the first micro-controller to the Tx pin of the other one and vice versa. Together RxTx ports allow the micro-controllers to communicate together exactly in the same way that they communicate with the PC through a USB cable (Serial Communication). After connecting the cables, it is easy to setup and manage the connection by using *existing Serial interrupts and protocol on AVR micro-controllers*. This interrupt can even be used in waking-up the micro-controller from sleep mode (waking the sleeping nodes if necessary to talk to them). This feature is used in power optimizer algorithm, discussed later in section 5.5.3. Another *useful feature* of Serial communication on AVR micro-controllers is the availability of 4 different Serial channels on Arduino Mega boards (Atmega 1280) [47]. Current DES design only needs five micro-controllers inter-connected and this is enough. Another *very practical and important* properties of Serial communication in Arduino Mega (discovered through implementation), was that it would allow *communication with each slave micro-controller to go through a separate dedicated channel*. This means sending unnecessary data would *decrease by around 75%* and *accordingly bandwidth and processing power* needed for handling unnecessary data would be spared.

Due to availability, simplicity, reduction in exchanged data rate and the stability of this type of connection (based on experimenting in implementa-

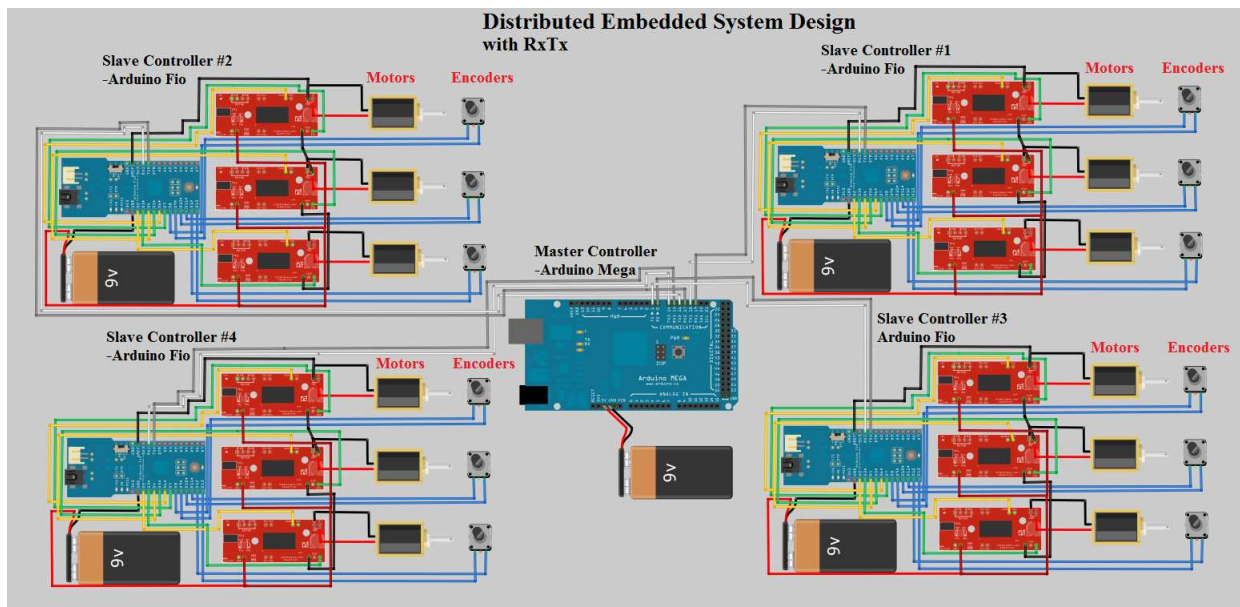


Figure 5.17: Implementation plan for RxTx system configuration

tion), *Serial connection was chosen as the ideal choice for cable based inter-connection of micro-controllers. This approach was implemented and the results were very satisfying due to all properties mentioned earlier which were fully implemented and used.* Nex section covers possibilities of wireless inter-communication between different units.

5.3.2 Wireless Distribution

Although *cable connection is a very secure connection (much easier to sniff wireless signals)*, due to *flexibility provided by wireless design, easier maintenance, troubleshooting, simpler assembly design and easier proofing issues (water, rust, salt, etc)*, wireless communication was a very tempting alternative. The technologies that could be used here are classified in four main aspects:

1. Infrared
2. WiFi
3. Bluetooth
4. ZigBee

Next sections would go into details of such technologies and evaluate their gains and losses when using them as implementing the DES final design.

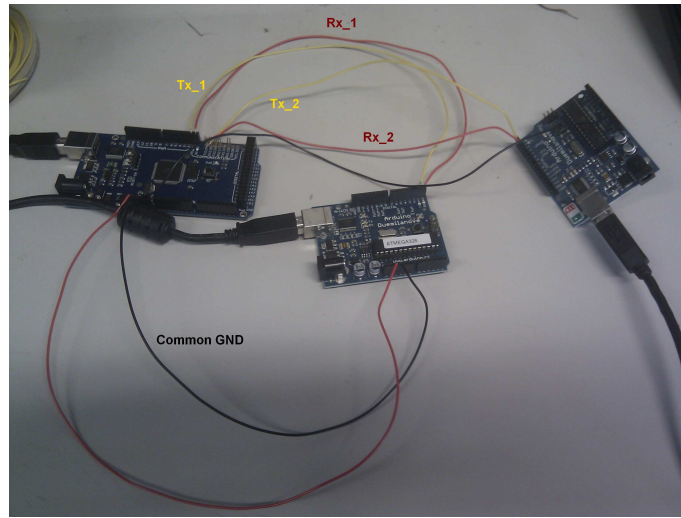


Figure 5.18:
Connecting three Arduino boards with Atmega AVR micro-controllers together with RxTx ports

Infrared

Some of the infrared properties such as very low power usage, safe due to short range (hard to sniff) and inexpensive [59,60]. On the other hand, there are some other properties that make it totally an unacceptable choice. E.g. transmission at eye sight (any obstacle can interrupt the whole connection) and instability

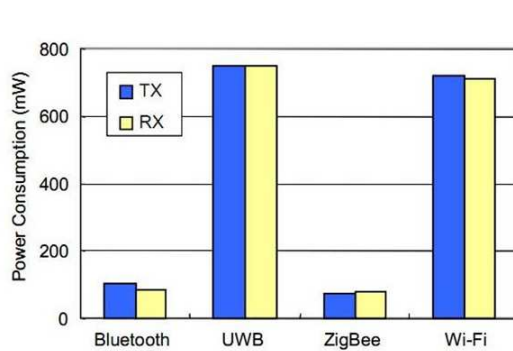


Fig. 5. Comparison of the power consumption for each protocol.

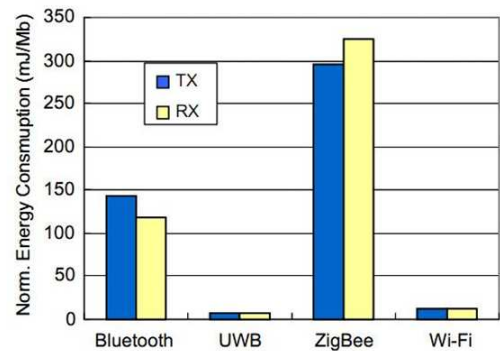


Fig. 6. Comparison of the normalized energy consumption for each protocol.

Figure 5.19:
Left: Power consumption for Bluetooth(BT), Ultra-wideband(UWB), ZigBee and WiFi
| Right: Normalized energy consumption for each method [58]

under harsh weather condition [60]. Offshore project usually suffers from harsh weather condition and *instability in harsh weather (sunlight, rain, pollution, etc)* was a *major disadvantage* that crossed off Infrared from the list of available choices.

WiFi, IEEE 802.11

Ethernet discussions more or less apply here, except the fact that every slave-controller here must have a WiFi adapter instead of an Ethernet adapter and a wireless router in center. The WiFi adapters are connected to a wireless LAN and moreover to a higher level system (platform network). Due to vast implementation, stability and maturity of WiFi technology, using such technology in the design can simplify the design process. However it may also arise some *disadvantages, such as high power consumption* [58] (Figure 5.19), *fear of signal drop outs, signal interference, jamming and intrusion*. All in all WiFi doesn't seem to be the right choice for internal communication between micro-controllers (especially due to high power usage), but right choice for the connection between robot and the network providing access to the server.

Bluetooth

High power consumption is a concern in designing mobile systems and therefore Bluetooth is an interesting case to discuss due to low power usage [58] (Figure 5.19). *Bluetooth or Zigbee* can be a very suitable candidates as both have very little power consumption [58]. Figure 5.19 clearly illustrates the differences in power consumption between different wireless protocols.

Earlier experiences with Bluetooth technology at ROBIN group did not turn out to be very satisfying (lack of stability, sudden drops outs, short range of support), but latest upgrades in this technology turned it to be an unavoidable choice. The early versions of Bluetooth (1.0 and 1.0b) were unstable. Problems with interoperability caused difficulties in connecting devices from different manufacturers. Later versions (1.1 and 1.2) have resolved many of these problems [61]. Stable communications between cellphones, car kits and hand frees, and the ability to support connection between several (up to 8 in active mode) devices at the same time and its short range in comparison with WiFi (harder sniffing) are advantages of this protocol. Planning to implement a network's several nodes together, Bluetooth Personal Area Network (Bluetooth PAN) can be used to make a network between all communicating nodes.

Disadvantages such as *slow device discovery* (extra time and energy used to discover a new device), *higher power consumption to keep a connection open*

compared to other protocols, limitation on number of active devices (8) for further development of the system [62, 63] plus *unavailability* of the hardware components in the lab, *price compared to other choices* (79\$ Bluetooth board instead of 25\$ ZigBee boards) crossed Bluetooth off the list as well. This resulted in ZigBee to be the last choice on wireless based EDS list of protocols. Next section would be covering the topic of ZigBee in DES design.

ZigBee, IEEE 802.15.4

ZigBee is a communication protocol based on IEEE 802 standards for Personal Area Networks (PAN). ZigBee(Figure 5.20) intends to be the protocol with *in-expensive, low power consuming* [58, 64] standard for wireless mesh networks in compare with competing protocols such as Bluetooth [64, 65]. They are active in various radio band fields such as industry, scientific and medical (2.4 GHz is most jurisdictions worldwide) [64, 66]. Figure 5.20 shows some products built based on ZigBee technology by Digi called XBee. Series one of XBee components does not support mesh networking. Implementation of XBee inter-communication in this project was done by XBee series 2.

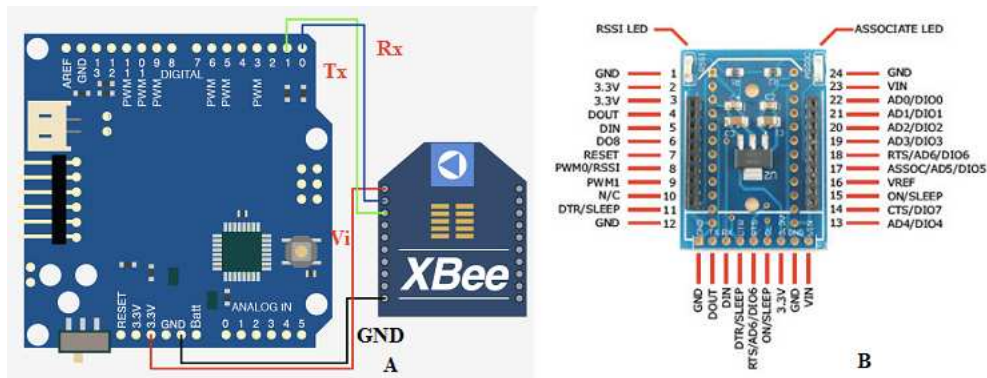


Figure 5.20:

A: Xbee connecting with Serial protocol to Arduino board [67] | B: Xbee pins [68]

Due to the *potentials of this protocol* and attempts to lower the power consumption even more, this technology sounds *promising to use as the inter-connection method between micro-controllers* in the DES design. In addition it is interesting to mention that some of the current components in the market allow remote wireless re-programming of micro-controllers (navigation program). This allows remote upgrading of navigation program, easier support, maintenance and troubleshooting. However, upgrading the navigation program in the middle of operation (could be sensitive), or in the field, is strongly

not advised. This also brings up some challenges in case of security compromise (being hacked). Meaning the machines could be re-programed by intruders to work against their initial goals. Such threats are addressed in the *docking station* topic in Appendix (section C.1).

At the end one might wonder if the *low data throughput* illustrated in figure 5.20, right side [58, 64], can be a *major disadvantage* for this technology. This is true that WiFi and UWB have far more data throughput than ZigBee and Bluetooth, but the question is where it is needed to have such high data throughput and what is the price to be paid for it (very high power consumption) *Robot-Server-Client design sends all communication through the master controller and forwarding of commands to slave controllers is filtered.* This justifies the choice of having WiFi as a communication protocol between the server and master controller, but *due to low traffic on slave controllers* the same reason does not apply for the slave controllers (exchanged data is supposed to be filtered and only sent to destination, the same way discussed in 5.3.1). Slave controllers have much less traffic and therefore *could do their job with a component that has lower data throughput and much less power consumption, meaning ZigBee.*

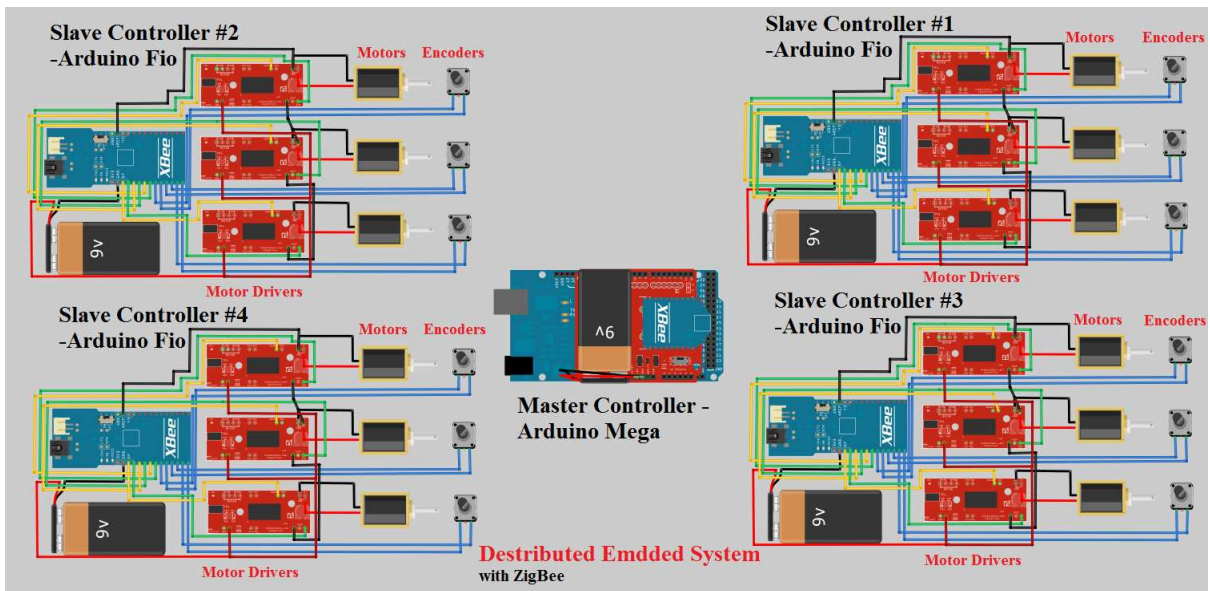


Figure 5.21: Implementation plan of ZigBee network

For testing this connection method, 1 Arduino Mega and 4 Arduino Fio and 5 XBee were used (figure 5.21). Arduino Fio is designed to directly dock XBee components, but one should use XBee shield to connect XBee to Arduino Mega. The connection between XBee components and Arduino board happens over

RxTx ports (UART/USART) 5.3.1. Bit rate of all serial communications (UART) were set to 19200. For simplifying the test and only focusing on XBee communication, the information during this test was sent through serial cable to master controller (Arduino Mega) through serial port (USB cable) and then distributed to the slave controllers (Arduino Fio) later. Something which is possible, but was not implemented in practice during this project was filtering of data only to the specified slave controller as it was done in 5.3.1. In theory it is possible to contact each single XBee at a time, but this means re-programming the master controller to only go into pair mode with only that specific XBee node on that specific slave controller. This is time (re-programming time and network search time) and power consuming (searching for new network) and not the most stable way to implement a navigation program with. Therefore a ring network with 5 XBee components were designed and implemented, where XBee on the master controller was the coordinator (one network, must have only one coordinator) and the XBee components on slave controllers were routers. The information flow went very smoothly through the whole connection and the implementation of ZigBee method was a success. The information from the coordinator (Arduino Mega) was on broadcast mode, but routers (Arduino Fio) sent only the feedbacks back to the coordinator (Arduino Mega).

5.4 Control Algorithm

This part is dedicated to the prototyping phase of the navigation program. The text in this part covers *development process of the control algorithm and decision making center* for implementing already planned climbing operation, with regards to the DES, in Walloid robot. Later details of *prototyping a distributed navigation algorithm with power optimization, remote controlling and semi-autonomous features* are addressed. Finally this trend would finish with presenting self-developed simulation application which are used for presentation and testing (system functionality) purposes.

5.4.1 Robot-Server-Client (RSC) Architecture and Development Tools

Earlier in 4.5, it was defined that the control algorithm would be developed in three layers such as Robot, Server and Client. An important issue about these modules is about the server part. Server in this project is an abstract issue and was not implemented, but a simple workstation was used instead and the connection to the hardware was either RS-232 by USB or Ethernet. It was assumed

that the robot in an ideal case would be in contact with the robot with wireless connection. On the other hand, it is assumed that in an already developed industrial ICT designs, *servers already exist*. Therefore, the controller algorithms were developed in a way to be able to work with different systems, as long as the server is compatible with the input standard of the robot and the server is compatible with the output standard from the robot. This is done by choosing development tools (e.g. Java, Processing, Arduino) that are cross-platform compatible with different platforms (Linux, Windows, Mac, etc).

To start shaping a system, we do need to have an overview of the system, especially the key controllers and indicators involved in that system. Figure 5.22 tries to define these key concepts and implement "divide and conquer" strategy in practice. This figure also tries to show possible stakeholders of the system (operators, task developers and simple clients).

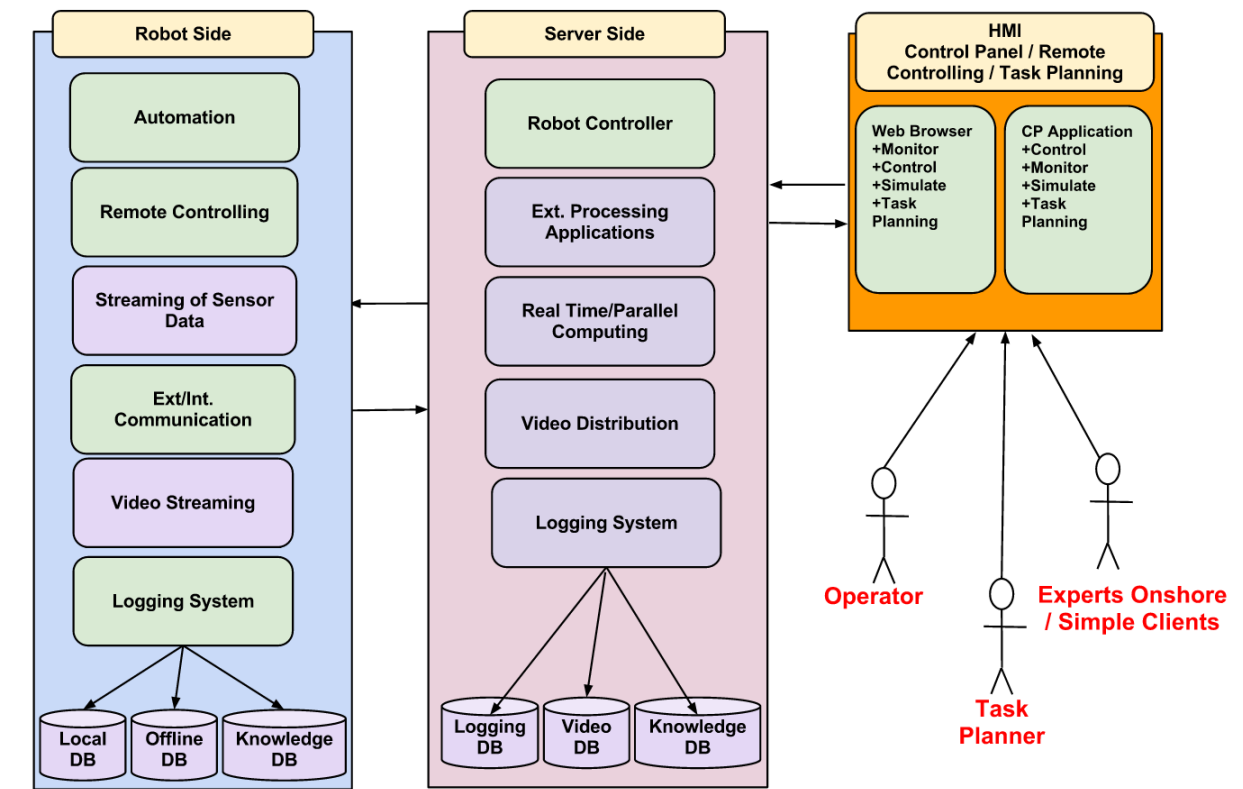


Figure 5.22:
Robot-Server-Client System Specifications | Green boxes were implemented, while purple boxes are abstract definitions

Moreover development tools should be chosen. As stability, redundancy, integration and further development potentials are key concepts in this project,

it is critical to choose right set of tools Therefore, Java Standard Edition (SE) was chosen as the main development tool on the server (workstation) side. Java is stable, mature, cross-platform, rich in number of libraries, redundant and with free development tools (e.g. Eclipse and Netbeans) [69]. In addition, Java Enterprise Edition (Java EE) as the industrial version of Java SE could be used for further development of the project [70]. Java might not be the fastest tool as C is, but it is stable, redundant and safe (Banks usually implement their system using Java) [69].

On the micro-controller side, by choosing Arduino in 5.2.2, the choice of programming language is limited to Arduino C and AVR C. Arduino C is a simplified micro-controller level C which is suitable for prototyping purposes, as it allows the developer to use their time on actually implementing their idea and not on complicated programming issues. Arduino C and its Integrated Development Environment (IDE) are *easy to use, cross-platform, free and licensed under Creative Commons Attribution-Share Alike 3.0 License and the code samples are published in public domain* [49].

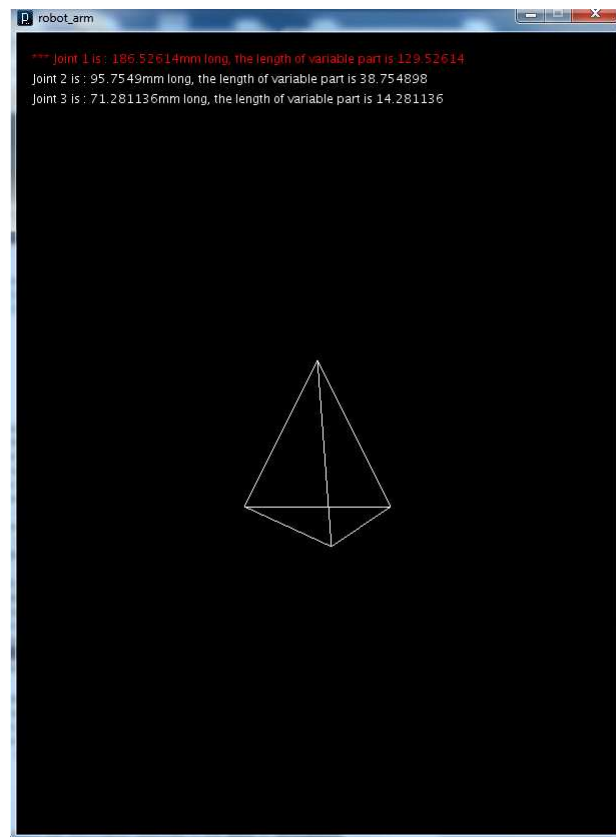


Figure 5.23: *Simulation*

Processing programming language was another tool used during this project. Processing is an Open-Source development tool suitable for *simple interaction and visual arts* [71]. Since the start in 2001 up to now, tens of thousands of students, artists, designers, researchers, and hobbyists enjoy using Processing in a diverse range of projects [72]. Processing can also *cooperate closely with Java* and can be integrated directly in Java classes and on *web pages (as a Java applet) and even on mobile platforms*. This specially helped in simulation and interface design as *Java has its own difficulties in graphical environments*.

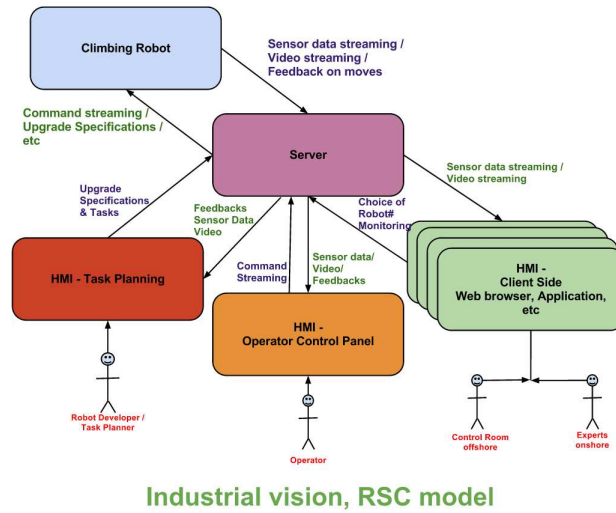


Figure 5.24: The industrial vision of a Robot - Server - Client

Figures 5.22 and 5.24,5.25, all together underline stakeholders of RSC system. This would need a little bit more clear definition. This separation of stakeholders was mostly done to give each layer authenticated access to the robot (Run mode, Teach mode and Monitoring mode). Following list would describe each stakeholder and their access level better.

The system stakeholders defined in figure 5.22 were:

1. **Robot Operators** steer robot manually from onshore facilities (Auto and Manual Mode).
2. **Task Developers** develops new functionalities on robot system and teaches robots new tasks (Auto, Manual and Teach mode).
3. **Offshore Experts / Simple Clients**, are those experts whose experience is needed in different cases. Such experts are not in control of the robot, but only monitor the sensor readings and processed raw data from the robot on their screens (no control mode access).

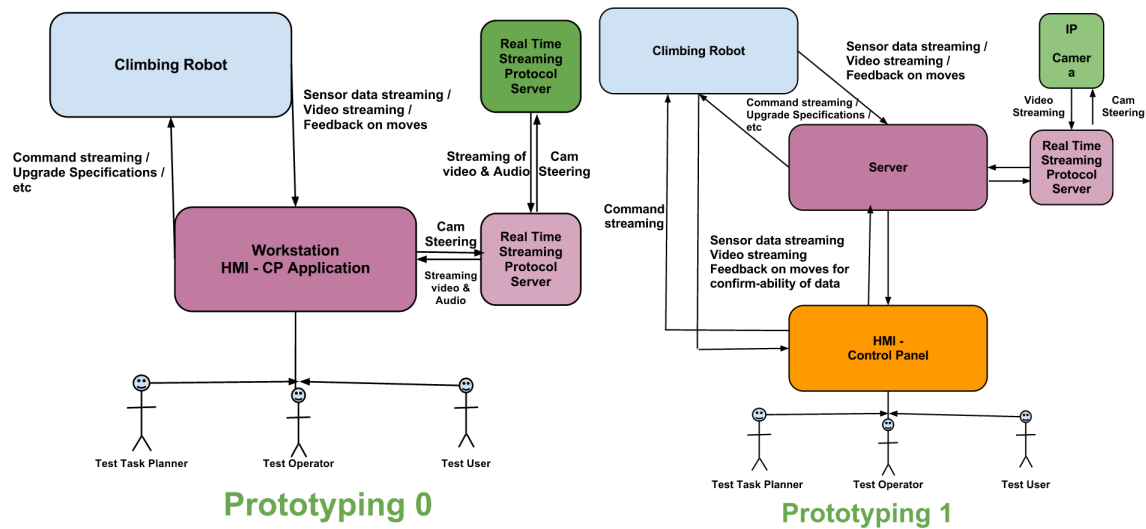


Figure 5.25:
 Combination of several network DAQ in rapid prototyping 0 | 2: Combination of Several Local DAQ in rapid prototyping 1 | 3: The industrial vision

Each of these stakeholders needs their own control panel (CP) to have access to the robot. To this some imaginary web based control panels were designed and were implemented later during the development phase (5.5.4).

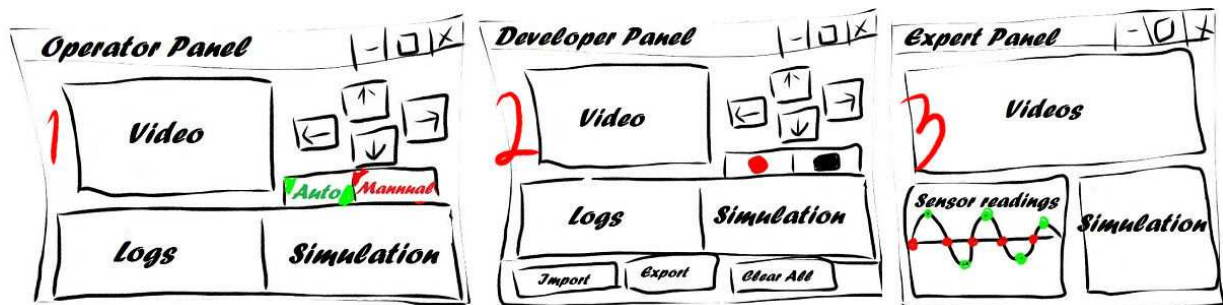


Figure 5.26:
 Early stage hand drawn GUI design for different stakeholders in a RSC model

5.4.2 Hardware - Software Interaction

To reach the goal in robotic systems, it is *vital to provide stable interaction between the hardware components and the software* running on them. Sometimes parts of the control software could be running on a totally different plat-

form (server) far away from the robot. In such occasions the commands are sent to the robot through different types of communication protocols. These sequences of commands could be as simple as motor moves on a remotely controlled robot, or more complex AI decision making commands. No matter how simple or complex these commands would be, everything would be sent as simple characters in a predefined protocol (Figure 5.37).

As assumed earlier in section 4.4, the external connection medium was *Wi-Fi* (access to server) and *RS232 / XBee for inter communication of micro-controllers*. As both XBee and RS-232 use RxTx ports on the Arduino board, it would not matter in coding phase which approach would be the final choice of implementation on DES. *Built-in Serial library in Arduino C* and *external RxTx library on Java* side [73] had to be imported to establish the serial communication. On both sides, reading of transmitted information was done using *serial interrupt* functions. Later this received information was sent to other functions (later other Arduino boards) in the navigation program to be processed.

5.4.3 Development Process

The first prototype developed during the process of this project was a simple connection between a control panel (CP) application on the workstation side (developed in Java SE) and the Arduino C code on a centralized micro-controller (single) of the Walloid arm. This is a stage that a centralized embedded system was being tested. Development in this step was a very good exercise, understanding the concept of serial communication and reaching a stable connection. The CP developed in Java SE was a single thread program that used the RxTx library [73] to communicate with the serial port with the micro-controller. Through this serial communication commands were sent to the micro-controller and feedbacks were received.

During this early stage a *manual navigation program, ability to connect to the serial port on different frequencies, logging system and a basic simulation* was developed (figure 5.27).

Receiving stream of data, processing it, logging it and simulating the process in Processing was too much for a single thread code to handle. Therefore, information loss was experienced from time to time. To solve this problem, two different threads were created that were run simultaneously, one reading and writing information to and from control hardware and the other would take care of logging and simulation (developed in next phase). Threads communication was done through having a shared *FIFO queue, where received data were stored to*. This data was read by the second thread and processed afterwards into logging system and simulation part. This solution had a very good effect

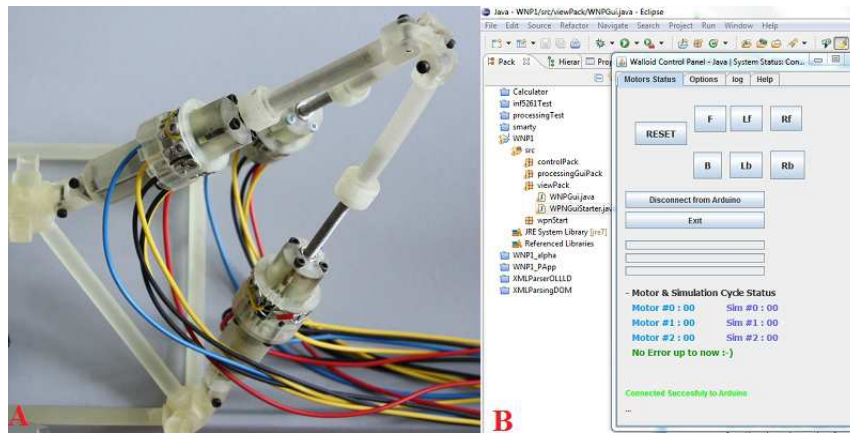


Figure 5.27: A: Walloïd arm | B: Walloïd Java based standalone control panel

on program reaction time. Threads communication was done through having a shared *FIFO queue, where received data were stored to*. This data was read by the second thread and processed afterwards into logging system and simulation part. This solution had a very good effect on program reaction time.

Java graphic difficulties made it hard to develop better simulation with Java SE. Therefore, it was decided to add a new development tools called Processing programming language to the project later. Processing is fully compatible with Java and can be totally integrated. This allowed continuing using the early developed CP and integrating the simulations developed in Processing with it. The *result of this stage* of development was a *simulation of the arm in 3D space that would follow the physical arm, based on feedbacks* from control algorithm on arm control hardware. The simulation *would warn the user while reaching boundaries of the workspace* (chance for the shaft to drop out). It is interesting to mention that problems of passive joint control occurred during this stage, where due to situation explained in 7.4, the arm stopped moving while the encoder kept sending readings to the CP and based on these feedbacks, the simulation were also showing a moving arm (whole system malfunctioning). However, in reality only the motor was rotating, but the faulty joint was not under control of the CP at all as the connection was broken. This was an interesting incident as through studying literature it was discovered that this is a major issue to handle in robots [18].

The issue of simulation became very important as the Walloïd received arm broke down (3.4.3) after a while and everything could only be tested on simulations. Early simulations were basic presentation of the single arm (figure 5.29) which was based on the received Walloïd arm. However, this changed totally later as simulation was the only tool for testing the results of the work.

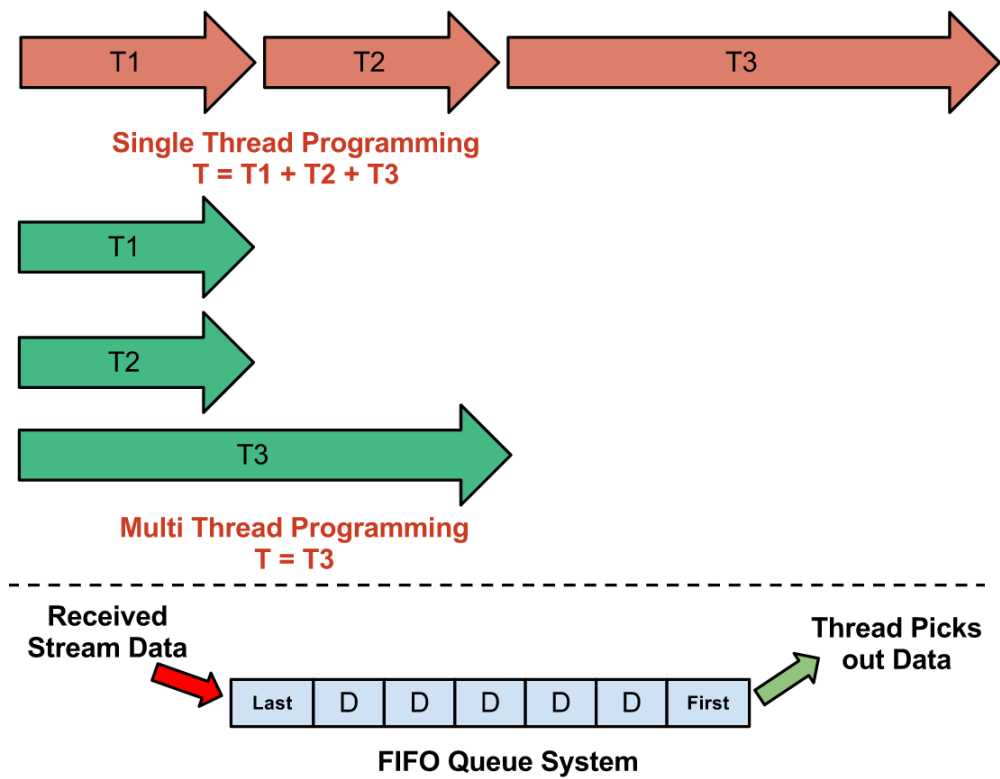


Figure 5.28: Multi thread programming, FIFO list

Therefore more focus was put on simulations and better functionality and results were achieved. The detailed process of simulation process is discussed in 5.6.

The development process of the navigation program continued and it was soon discovered that using a single Arduino card was not enough 5.2.2. Therefore, a decision was made to move from a centralized embedded system (single micro-controller) toward a distributed embedded system including 5 micro-controllers (section 5.2, 5.3). This affected the whole controller algorithm on the micro-controller side. According to the new situation on the hardware side, the code should be divided in two parts (figure 5.30):

1. Master controller: This part of the code is the main brain of the robot and all the communication to the outside world and strategic decisions are made in this unit (Arduino Mega).
2. Slave Controllers: These micro-controllers that are only in charge of one arm would run an identical program. This program is in direct contact with the master controller and follows commands accordingly.

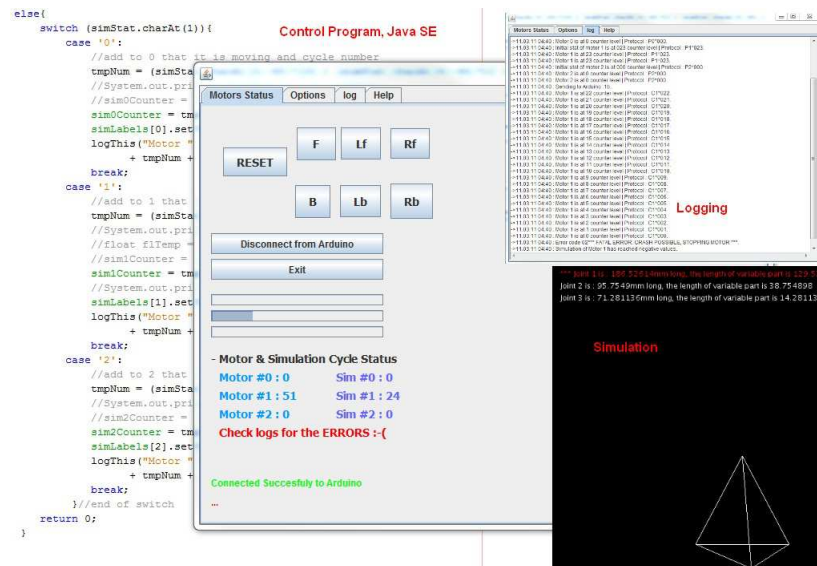


Figure 5.29: Control panel, simulated data flow, logging and simulation

This radical change in architecture resulted in new types of inter-communication methods to be tested and lots of experience in ring and mesh networking with ZigBee components was gained. The distributed system was implemented with three different methods, such as I2C, Serial RxTx and ZigBee (5.3). Libraries used for implementing these protocols were Serial library (for both Serial RxTx and ZigBee) and Wire library (for I2C) [53].

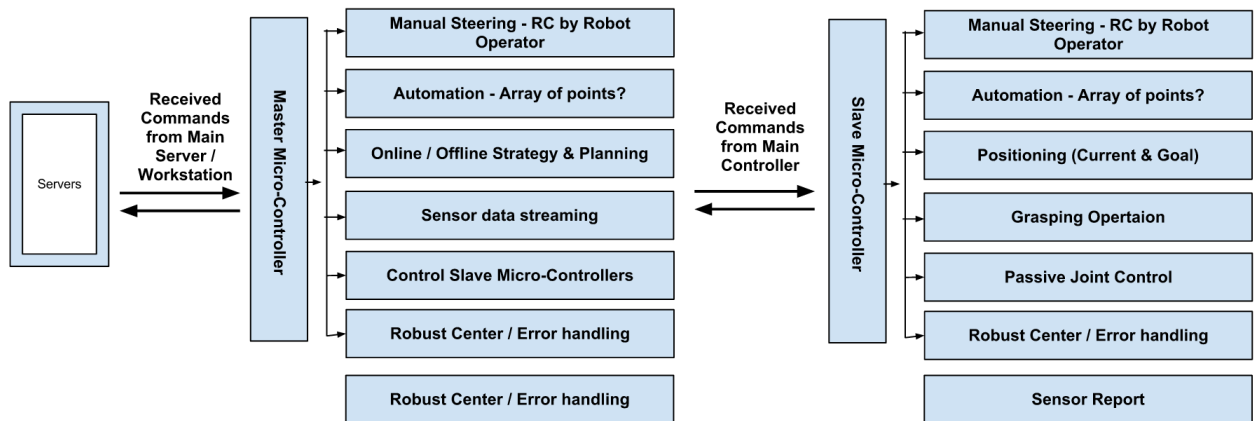


Figure 5.30: Master & Slave Micro-Controller tasks

5.5 Distributed Navigation Program (DNP) and Features

DNP Algorithm

Previous section 5.4.3 covered the development process and how it has affected the programming phase of control system. Based on the algorithm designed for DES, an Arduino C code was developed and was divided between all Arduino Boards. According to figure 5.31, the main controller (Arduino Mega) receives the commands from server and then distributes it further to slave controllers (Arduino Fio). These commands are coded in a communication protocol to minimize the bandwidth usage (read 5.5.5 for detailed information). Based on this protocol, the master control would know which part of the system should receive this message (number 0 represent masters, and 1-4 represent each slave). The conditions here for master controller is to either act as the command orders (the commands is for master controller), or redirects the message to the right slave controller. Navigation program on the master controller operates in three mode, Auto, Manual and Teach mode which were described earlier in section 5.4.1. The algorithm shown in figure 5.31 clearly shows that with regards to different modes the DNP would have, different action would be taken. E.g. teach mode results in all movements to be logged. This part is continued by the positioning logic in DNP in next section.

5.5.1 DNP Positioning Logics

The process in 3.3 shows how the kinematics calculation was simplified to one single rule saying that for reaching one point, the variable distance of all three prismatic joints in one arm are needed. This variable distance has a direct connection with encoder readings (section 3.3). Assuming there is one counter for each prismatic joint, then each encoder readings, stating forward movement would increment the counter value by one and each reading stating backward move would decrement the counter value. Based on Walloid received arm specification each prismatic joints counter could vary between 0 - 217. This was equal with 54.4 mm, meaning each counter unit was equal with almost 0.25 mm (figure 3.6).

5.5.2 Reading Sensor Data

Reading sensor data is done by polling as there is not enough ports with interrupt features on AVR micro-controllers on Arduino boards (2 on most of Ar-

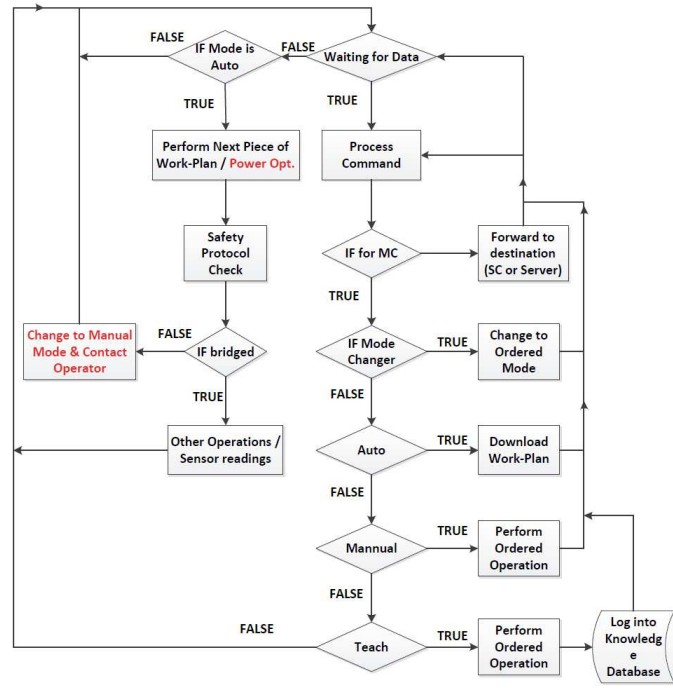


Figure 5.31: Distributed Navigation Program Flowchart

duino boards and 4 on Arduino Mega). Such ports should be used in monitoring the very sensitive sensor readings. However, one should be aware that hooking signals with continuous updates to interrupt ports can be more time consuming than polling. Therefore, in DNP design, encoders values, which are frequently updated are read by polling method and interrupt ports are reserved for critical issues such as wake-up interrupts, collision detectors or passive joint control sensors 7.4, etc. Other issues such as encoder readings and storing sensor data are discussed in details in Appendix and therefore, are ignored here C.3.

5.5.3 Power Optimizer Feature

Advantages and disadvantages of such system was discussed in sections 5.3 and 5.2.2. This strategy contains advantages that one cannot look away, but *also imposes extra challenges, such as delays (5.3) and high power consumption* (5 micro-controllers use more power than one). Back to the discussion in climbing gaits, an arm hanging on a bolt according to dynamic physic rules does not use any energy for only hanging there. This means no electric energy from batteries is used at all. On the other hand each arm has one embedded

system dedicated to control only that arm. Another fact is that only one arm is not fastened to the bolt at a time, doing the grasp operation and the rest are in idle mode (beside lift mode).

Doubt questions might then be raised that *as long as only one arm moves at a time and the operation speed of the arms are slow (3.4.2) and therefore time gaps between operations can be long, then why not having a sleep mode policy* for such idle embedded systems ? To solve this challenge a *power optimizer algorithm* was developed which tries to *manage and optimize* the system power consumption from master control and save energy as much as possible by *putting the whole idle arm's embedded system (control hardware and motors) to sleep mode*. These embedded systems would be *woken up* later, when they are needed in operation.

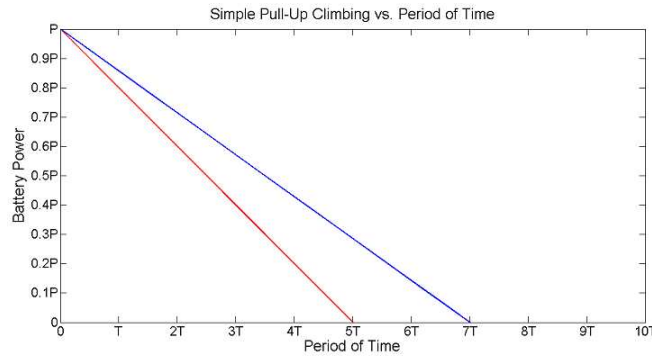


Figure 5.32:

Simple pull-up climbing gait Power consumption before and after implementation of Power Optimizer Algorithm (40% difference)

Practically this is *implemented by using AVR sleep routines* which allows one to put the micro-controller to sleep mode and turn off parts that are not in use. To this we have used AVR microcontroller features and *AVR C directly [74] [75] and not Arduino C*. Here one of the very useful features of Arduino was used which allowed direct usage of AVR C in Arduino C code. For this *avr/power* and *avr/sleep* of AVR libraries were imported and used. The idea here is based on *sending sleep or wake-up command from master controller to slave controllers*. The wake-up operation is based on interruptions (USART interrupt) and the micro-controller will continue running the code from where it went to sleep mode. One can also use other interrupt ports to wake-up the micro-controller (in case of having sensors, monitoring sensitive readings).

This strategy *reduced the battery consumption up to 40%* in a simple pull-up climbing gate (figure 5.6). Calculations in figure 5.33 shows the mathematical proof around this topic (the amount of energy in sleep mode is very low and

can be ignored). Figure 5.32 shows different climbing gaits power consumption before and after implementation of Power Optimizer Algorithm. The source code is available in Appnedix.

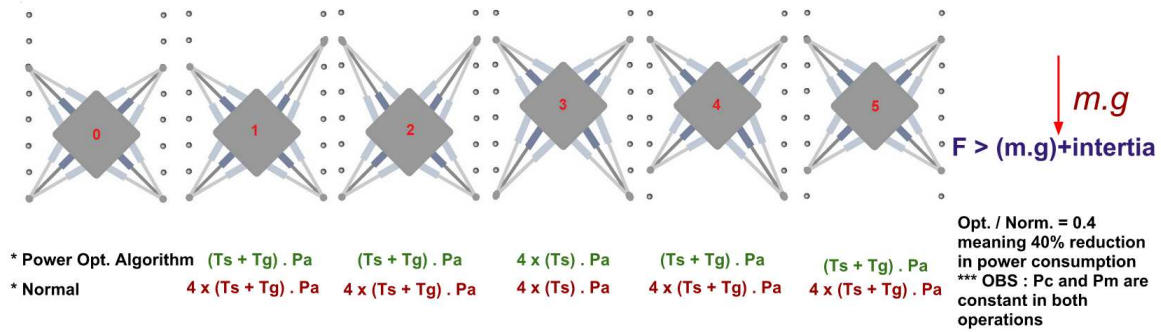


Figure 5.33:

Optimized Algorithm can reduce power consumption in one robot stride by 40%. Ts: Arms stride time, Tg: Grasp time, Pa: Arm Power Consumption, Pm: Motor Power Consumption, Pc: Master Controller (Core Micro-controller) Power Consumption

5.5.4 Remote Control System

Today AI with presenting commercial products (autonomous cleaning robots) to the market has shown new potentials. This proves that the AI technology has reached the level of mass production in general scale. *However, the chance for human mistakes always exists* (developers) and this also applies to AI systems created by mankind. *This means that based on the sensitivity of issue, such autonomous solutions cannot be fully trusted without a backup plan* A.1. That's why it is critically important that any autonomous systems that are in charge of sensitive tasks, such as those in petroleum industry, are equipped with overriding methods (remote control).

On the other hand, it is sometimes hard to define job scenarios as the problem is just too complicated for a robot to handle A.1. Another challenge in making job scenarios for robots in offshore platforms is the big number of unplanned situations that can occur during daily operation on a platformA.1. *Such limitations and difficulties in task planning make remote controlling and its surrounding technologies popular for robotic automation in offshore platforms*A.1.

We had several types of remote controlling methods implemented during this project. The first remote controlling prototype was the initial CP developed in Java which was developed on early stages of the work. This application was a

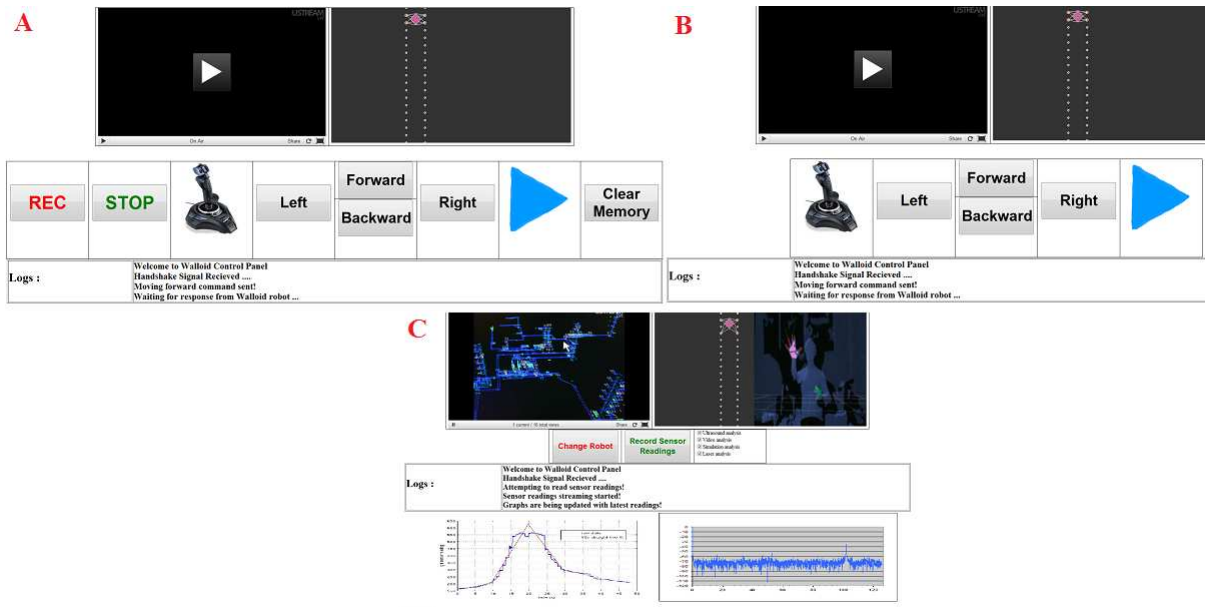


Figure 5.34: Web based Remote Control Panel, A:Developer | B:Operator | C:Expert

stand-alone application on a workstation which was connected to the robot through a USB cable (Serial RxTx). Here *simply the arm could be controlled by keyboard arrow keys or buttons* (figure 6.7) (5.4.3). Later this was also achieved over the Ethernet and ZigBee connection *in the distributed embedded system design and the system manual mode was developed into a Teach mode where all actions were logged for automating the recording process* (more about Teach mode in 5.5.5). The development type changed as it was decided to go for a RSC approach which brought servers into picture. Therefore it was much more redundant, flexible and safer to have the control and modeling part of this program on the server side, while having the view on the client side 5.4.1.

To this, *web based control panels were designed and implemented*(figure 6.7), which could send commands to the robot through network. This was *implemented and tested with Ethernet shield* due to availability issue (only networking component in the lab). During these tests, the control hardware was connected to a router where it could receive commands from the server and act accordingly.

5.5.5 Logging System

Logging is one of the most important operations (for later autonomy) in the navigation program both on micro-controller side and on server / workstation side.

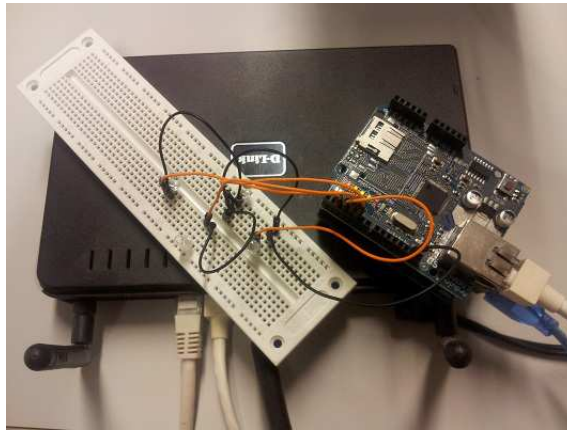


Figure 5.35: *Ethernet shield used in web based CP test*

The logging issue contains storing all controlling events happening on the robot. Logs here could be used for *troubleshooting, maintenance and automation purposes*. As the *automation level of Walloid robot is to repeat the already taught tasks again and again*, the more detailed logging is, the more precise it would be to re-create a task. Logs are supposed to be stored on both server / workstation (task-planning and automation purpose) and micro-controller (offline automation, 7.7) side, using external memory such as SD card (figure 5.37). The limited resources did not allow this was done only in theory on micro-controller side. However, this was completely implemented on server / workstation side. It is important to mention that the code to write the logs to SD cards on the micro-controller was also implemented on DNP program, but was never tested with an SD card connected to Arduino board.

On the other hand, *logging information requires both processing power and bandwidth*. This means there should be a balance between "**what**", "**how detailed**" and "**how often**" things should be logged. The way the logging system is designed is important as well. E.g. *one can use short unique combinations of letters and numbers (communication protocol), which are machine-friendly, instead of human-friendly long sentences (figure)*. Such sentences can be transferred back to the fully human friendly sentences when they are being stored on the server (with higher processing power). This solution and similar approaches can reduce the amount of exchanged data and processing power required to send them.

Finally before moving to the topic of semi-autonomy, it should be mentioned that for better autonomy it was decided to add the feature of tagging logs (combining paths and tasks for creating dynamic work-plans). This resulted in the developers in teach mode would be able to tag an operation with a specific

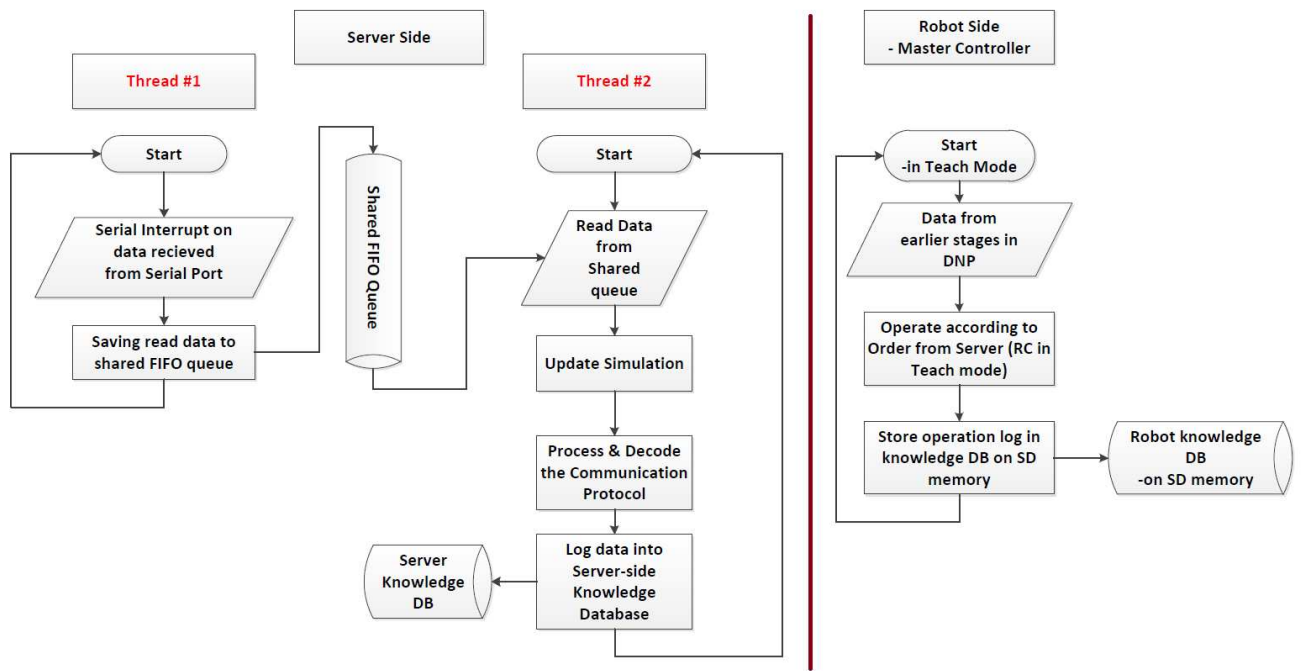


Figure 5.36: Logging flowchart on server side and robot side

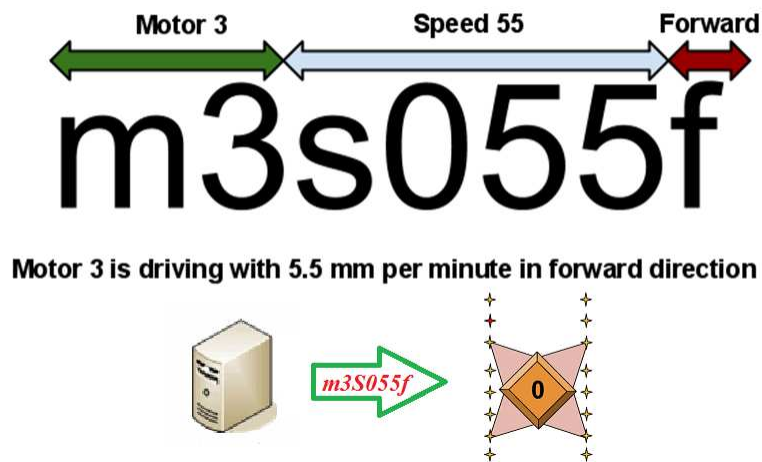


Figure 5.37:
Communication protocol example between Server and Robot in RSC model

name, e.g. reading gas valve position.

Table 5.4: Examples of protocol nodes

Communication Protocol Node	Tag	Description
m#s###f / m#s###b	tag_taskTest1	Motor # is driving with ### mm/s speed forward/backward
m#c###	tag_taskReadGasValve	Motor #'s counter value is ###
m#ON / m#OF	tag_pathE10	Motor # ON / OFF

5.5.6 Semi-Autonomous System

The **level of autonomous** in DNP is as simple as **re-doing an already ran operation**. As mentioned in 5.5.5, this is *simply done by logging everything during the operation in Teach mode and storing it on the knowledge database (a file on SD card here) and simply re-doing each single operation from logs database afterwards*. This is the easiest and safest way to implement automation. The knowledge database could be a SD disk on the robot and a database on Server side. The process of teaching a robot should be done by a task developer who is **familiar with the robot limitations and the task**. Technically a "task" only contains a series of certain operations (arm positions, paths, lift, grasp and release), which if done in correct order would re-create the initial operation.

Through development process when the Walloid arm broke down and simulation process started, new horizons were seen in simulation world. One of these possibilities was to auto-create data needed for climbing a path and combining with already developed tasks by robot developers to create work-plans. Having the path plan with necessary bolts dimensions plus the initial position of the robot, a code could easily calculate the necessary information for the robot to climb and hang on the bolts on the path. Moreover, in case of having a database of tasks, this information combined with already developed tasks (e.g. how to move the camera to detect the gas valve position) would make a route that the robot climbs all the paths to the monitoring spots, performs the task and returns back to the docking station. This could even be optimized more by using the salesman algorithm for finding the shortest way to go through all the spots. This would mean shortest way from docking station, visiting all the monitoring spots, and then come back again to docking station. **The initial idea for this code was done, but was not implemented fully**. Based on these discussions learning processes can be done in 2 different ways:

1. **Manual way** : The easy and safe method is to manually steer the robot

through the whole process in *teach mode*, while everything logs and is getting stored in knowledge base. Redoing these logs exactly repeats the same task.

2. **Automated way:** An autonomous task and route generator prototype program could be developed which by having *initial position of the robot and location of monitoring spots* in the field could generate the work-plan of the robot accordingly (using traveling salesman algorithm to find shortest way). It is assumed that the path necessary information (e.g. bolt positions and dimensions) and initial starting point of the robot (e.g. in docking station) would be already provided for the task and route generator program (future works).

The *disadvantage* of such automated methods compared to the manual way could be the *bugs and mistakes* that such programs could contain and *differences between a real-world and virtual simulation*. These kind of errors could be solved by monitoring the process by task developers. Such approaches are also used in industry in similar system. E.g. ABB RobotStudio tries to semi-automate programming of industrial manipulators (partly automated, mostly manual). This programming tool allows the program to run a simulation the offline program developed in virtual 3D environment to be approved by the programmer before running on the real world robot [38]. It is obvious that the simulations and offline programming of dynamic environments such as oil and gas platforms are much more challenging than a fixed industrial manipulator.

5.6 Simulation and conformability of data

Simulation is to imitate a real phenomenon or object by using something else (e.g. learning to get on a horse on a wooden horse). From engineers point of view, this means recreating the behavior and characteristics of a real system in a computer model [76]. Simulations can vary from a total text based program that generate and evaluate data, to the simulated object in a 2D or 3D system, trying to present a full understanding (or verifying) of the system behavior in real world. Simulation for development and verifying system behavior is popular for industrial manipulators (e.g. ABB RobotStudio, MotoSim, Delphi, etc) [38, 39, 77].

5.6.1 Implemented Simulations

As the previous table defines four simulations that were implemented during this project, where one of them lacks GUI (text based) and the other two are im-

Table 5.5: List of implemented and conceptual simulations developed

Implemented Simulations	*Data Flow Generation *2D/3D Simulation of climbing gaits
Conceptual Simulations	*Training Tool (Training new operators) *Simulation of robot working environment in 3D world (based on sensor readings)

plemented in 2D and 3D environment using Processing. *The results and logics from text-based simulations were used later in graphical simulations for connecting to the control hardware and represent the climbing operation based on feedbacks from the robot hardware.*

Data Flow Simulation

The data flow generator did not have any graphical output, but was designed to test the system functionality. This was based on generating real-time data flow to test different parts of the system.

The simulation here tried to generate commands on the server and micro-controller side to test the flow of commands in the whole DES hardware components and server. On the micro-controller side, re-creation of reality was implemented by waiting (delay function) as long as the real arm would take to do that operation (based on experiences from Walloid arm). Afterwards feedback is generated and sent to the server.

Climbing Gait Simulation

This program was started from a small workspace presentation which was a simple simulation trying to *exactly follow the robot movements* in reality *based on control hardware feedbacks* and represents the arm situation in the virtual environment. This was developed in Processing, was a more advanced version of earlier joints presentations in Java with basic graphics. The experience gained from updating simulations based on hardware feedbacks was later used in a more advanced simulation representing robot climbing operation.

The climbing gait presentation was developed in Processing and later combined with Java. *This 2D environment goal was to represent the different climbing gaits graphically (only time factor was implemented in simulation).* This was a need that was felt after the Walloid arm broke down as lack of physical representation of the developed work was observed. This code includes a two dimensional illustrations of the climbing robot on the wall (figure 5.40). The simulation at this stage could *illustrate the amount of time being saved*

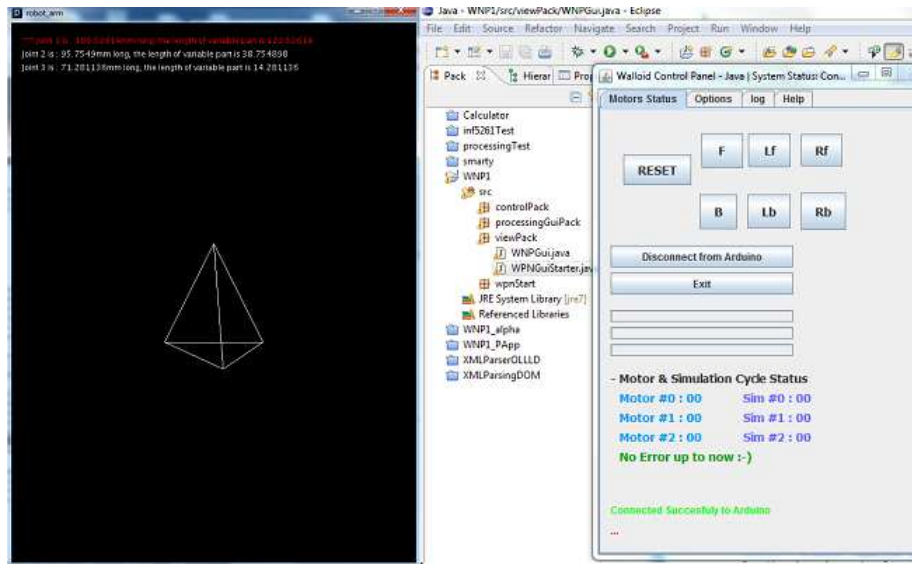


Figure 5.38: Workspace simulation with the Java CP and the arm

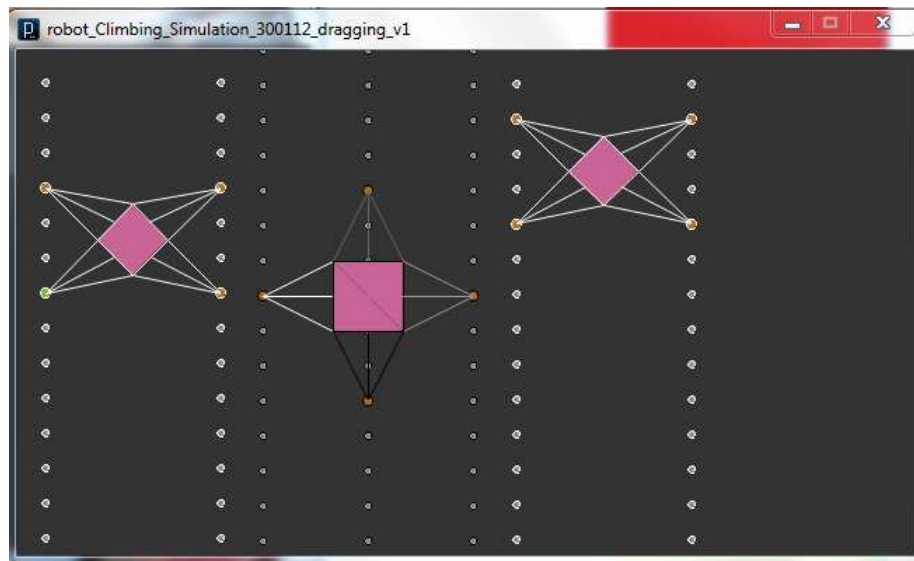


Figure 5.39: Different climbing gaits simulation

between different climbing gaits. Later this simulation was *equipped with the logic developed in data flow generator and workspace simulator to connect to control hardware and to be able to show the real time position of the robot based on feedbacks from the robot.* Therefore, this simulation can be called as a next version of the development that was started in 5.6.1 and later continued in

the workspace presentation.

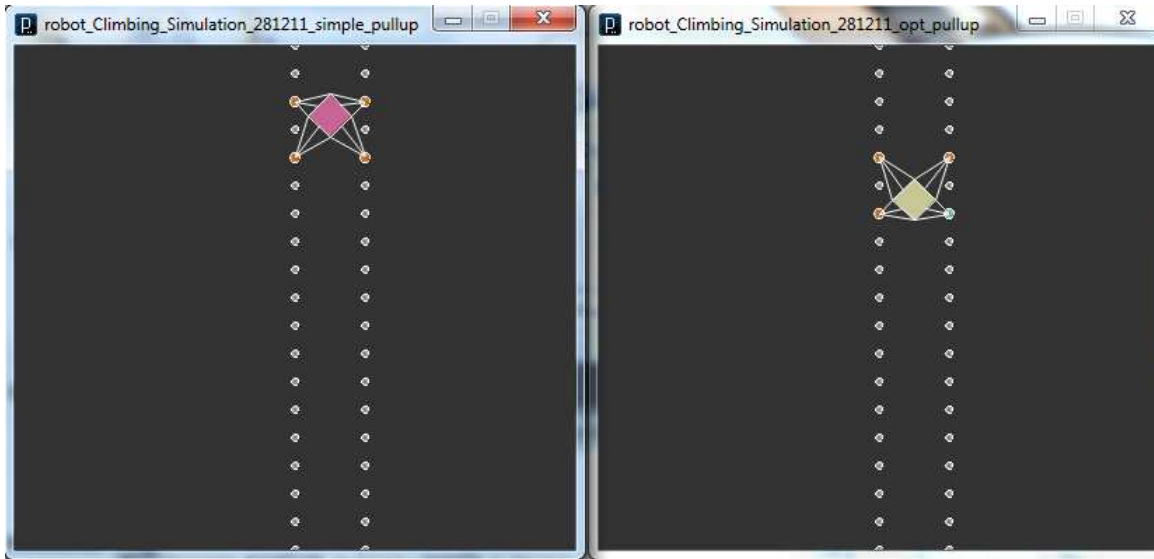


Figure 5.40: *Optimized pull-up gait vs. simple pull-up gait*

5.6.2 Conceptual Simulations

The 3D technology and virtual reality has developed rapidly in last decade and relevant products are all over the market. In the last few years a big raise in number of online games has happened which consist servers and millions of real users controlling virtual characters in these 3D worlds (World of War craft, Counterstrike and etc). The fact that these games have been performing so well on networks under such press (millions of users) is a proof for stability of such technologies. *This is an opportunity for the industry to step in and use this valuable experiences and expertise developed (processing power, networking and 3D modeling) in gaming field to shape a system to let all active automated nodes on a platform to work together.* Some industrial applications, such as ABB RobotStudio, use these experiences in 3D modeling and simulation for offline programming of their industrial manipulators [38]. As already implemented in many other applications (military, plane industry and etc), simulations can also be used for other purposes such as training new operators [78–80]. This results in better trained operators and developers when it comes to task planning and robot control in sensitive operations. The closer the simulation is to reality the better the results of such trainings are.

The simulation is helping today in integrated operations at StatoilHydro with building a virtual reality based on real time sensors readings received from

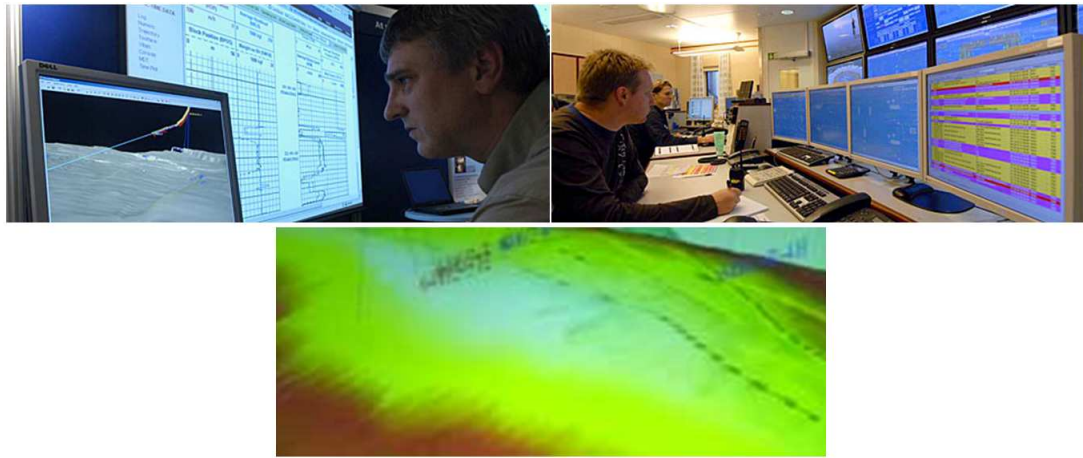


Figure 5.41:
StatoilHydro experts looking at a virtual reality based on sensor readings from bottom of the sea

sensors under the water [19,29]. The same solution could be applied to the top-side automation. Monitoring and service robots carry cameras and can send back live feeds of their environment to the control room. From time to time due to practical problems the use of camera might not be possible. Problems such as camera failures, slow connection links and environmental issues like smoke and fog. Such incidents in hostile environments like platforms are inevitable and a well-designed industrial robot is supposed to be prepared for such incidents. Especially if they are assigned to sensitive / complex tasks that can put the safety of the platform and in worst case the whole region in jeopardy. Here the simulation can come to aid by re-construction of the environment from sensor readings and earlier simulation of the whole environment (NASA working on same issue for pilots [81]). *This ability can help the robot operator to guide the robot through these situations on the platform to a safe place, or continue the operation and help in catastrophic situations where human workers must evacuate the area and cannot get near to the problem area.*

5.7 Summary

This was the longest and most challenging chapter of the report. The aim here was to cover almost all practical work. As planned in the previous chapter, the process started with climbing operation. The kinematics and workspace needed for such operation was already developed in 3.3. During this chapter, the work was continued by designing a series of candidates and a final end

effector design was presented at the end (5.1.1). The climbing operation is finalized by developing four pre-programmed climbing gaits, where three of them were functional (5.1.2). The graphical presentation of these gaits was presented in a self-developed simulation program.

- **Simple pull-up** - (*figure 5.6*)
- **Optimized pull-up (17% faster than 1)** - (*figure 5.8*)
- **Dragging (33% faster than 1)** - *figure 5.9*

Phase two was devoted to the development of control hardware (sections 5.2, 5.3). Here the process was described from an early stage (using an Eee motherboard) to a final mature design. Different approaches for developing a hardware controller in this part were discussed, compared and the final choice was made. Fruit of discussions and practical work in this section was a *distributed embedded system* consisting of *5 inter-connected micro-controllers* (Arudino boards). The master node was an Arduino Mega board and four other Arduino Fio slave nodes. The boards were inter-connected in two manners (cable and wireless), using *RS-232 and ZigBee* (sections 5.3.1, 5.3.2). The third phase of the development process (5.4) included developing a control algorithm for Walloid robot in a Robot-Server-Client model. Again the process was described in detail and the result, a *Distributed Navigation Program*, was presented (5.5). The DNP consists a *master and slave algorithm* which would be running on master and slave boards in DES design. The DNP has following features:

- **Remote Control** - Remote control through web based control panels (5.5.4).
- **Semi-Autonomous** - Ability to recreate operations which are logged in knowledge database (5.5.6).
- **Optimized Power Consumption** - done by master algorithm utilizing AVR sleep policy (5.5.3).

Finally simulations as a tool for the presentation of climbing operation, was presented. This could test the systems functionality by connecting to control hardware and software. This was followed by some conceptual simulation ideas which can be used if Walloid or similar projects are implemented in industrial scale.

Table 5.7 summarizes the activities done in this chapter.

Specification Name	Type
Robot Type	Inspection Climbing Robot
Locomotive Power	4xArms
Adhesion Solution	Grasping Arms
End effector	4 End Effectors capable of grasping bolts (figure 5.4).
Climbing Gaits	3 climbing gaits 1 - Simple pull-up 2 - Optimized pull-up 3 - Dragging
Micro-controller	Arduino boards with AVR Atmega 8bit chipset family (Arduino Mega for master and Arduino Fio for slave)
Distributed Embedded Systems	Network of 5 inter-connected Arduino boards were connected together by using RS-232 / ZigBee technology (Arduino Mega, Arduino Fio)
HW - SW interfacing	RS-232 (cable) / XBee (wireless), both connected through Serial port
Navigation System	Distributed Navigation Program with autonomy, remote control and power optimization features
Semi-Autonomy	+ Able to re-run pre-defined tasks
Remote Control	+ Remote Controlled through web-based CP and standalone Java SE application
Simulation	+ Data Flow Generator + Taks and Route Generator + Climbing Gaits Graphical Representation + Graphical Programming / Task Planning Tool + etc
High Level Programming Language	Java SE, Processing, HTML and Matlab
Low Level Programming Language	Arduino C, AVR C

Chapter 6

Implemented Control Systems and Results

Make things as simple as possible, but not simpler Albert Einstein

In a top level perspective of the system(chapter 4), some missing parts in Walloid project was defined and a work-plan was presented to develop these needed parts. The results of development process in chapter 5 is presented here. For easier understanding, these missing parts are presented again, in detail.

- Offshore Industry Point Of View
- End Effectors
- Climbing Gaits
- Control Hardware
- Control Algorithm
- Simulation

6.1 Offshore Industry Point Of View

Due to deadly **incidents(HSE concerns)**, **high wages (finanical benefits)** and **difficult (and costly) transportation of experts** to current and future platforms (section 2.4.1), offshore industry is interested in robotics automation (section A.1) [5, 11, 18]. Therefore it is important to discover potentials in this vast wealthy market of robotics automation. Section 2.4 in chapter 2 presented an analysis of offshore environment, possibilities and challenges for robotic projects

to enter that area of application. This was done with going through the academic literature, governmental and industrial reports about this field. Through this it was concluded that mobile robots have a good potential area of application. Such application could be **unmanned, shallow water, deep water and floating platforms**. Specially unmanned platforms with 2 weeks interval maintenance could be a tempting market [11]. Possible potential assignments for a mobile robot (such as Walloid) in offshore environment could be [11]:

- Monitoring overall situation
- Live video feed of environment
- Gauge readings
- Valve and lever position readings
- Monitoring gas level
- Acoustic anomalies
- Surface condition
- Gas leakage
- Fire detection and locating

Finally the analysis ends with presenting challenges such as harsh condition of offshore platforms are addressed. Conditions such as humidity, salt, rust, standards, cultural factors and last but not least HSE requirements (high redundancy).

6.2 Climbing Operation Results

Attempts in developing climbing process started by calculating kinematics and workspace of Walloid in 3.3 (figure 6.1). This was continued in the areas of focus and *resulted in developing a final solution for end effector and hanging bolts* (details in 5.1.1) and *three functional climbing gaits* (details in table table C.2 and 5.1.2).

Figure 6.2 shows the final end effector, assembled with the rest of the robot parts from Walloid project. This figure shows the robot assembly with the final end effector, hanging on the specially designed bolts. Regarding to development process of end effector, the bolts and the end effector were designed in a

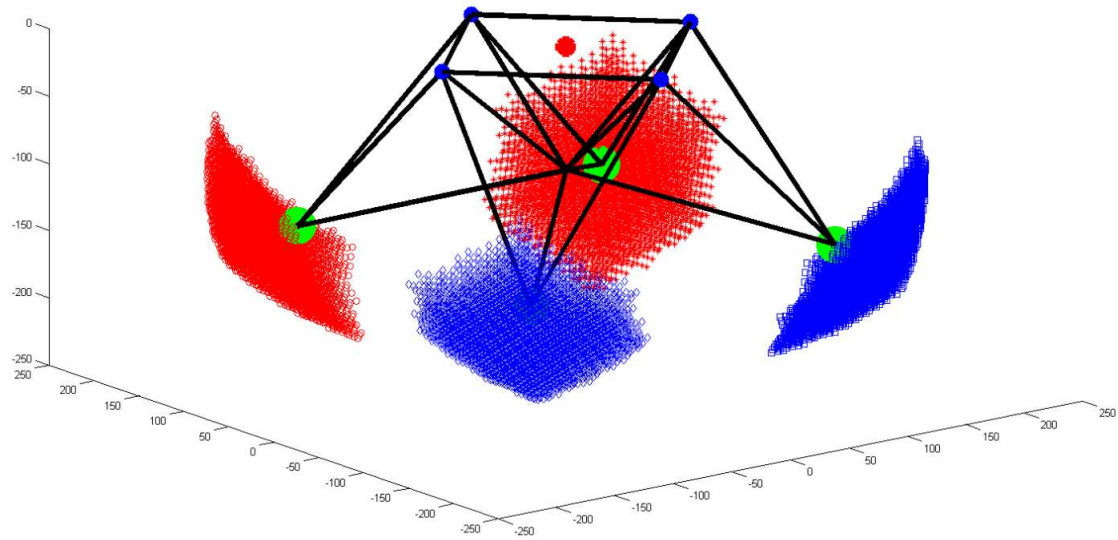


Figure 6.1: *Walloid robot workspace with 5mm precision.*

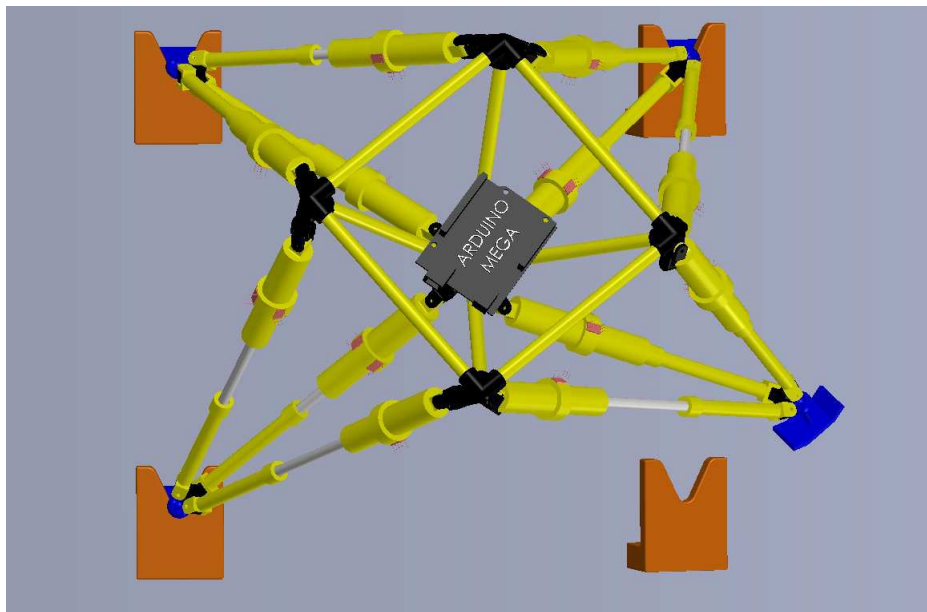


Figure 6.2:
Final end effector design (blue) assembled with the rest of Walloid robot parts.

way to reduce necessary precision in grasping operation and increase the error tolerance feature of the system.

Table 6.1: Table of climbing gaits features

#	Gait Name	Speed	Min No of Fastened Arms
1	Simple Pull-Up	V	3
2	Optimized Pull-Up	$V + (21/100)V$ (17% faster)	3
3	Dragging	$V + (33/100)V$ (33% faster)	2

Next phase contained the development of climbing gaits, where 4 pre-programmed gaits were developed. The rotational gait which also was the fastest developed gait turned out to be too unstable and was set aside. The other three developed strategies were:

All these three gaits enjoyed acceptable stability (minimum 2 arms gripping the bolts), but they were different in speed. Figure 5.32 (left) shows how increasing speed by skipping some phases, reduces power consumption (section 5.1.2). The same figure on the right, tries to show the opposite relation between speed (distance / time) and stability in Walloid design. The figure obviously shows that as the stability grows, the speed reduces. The factor of stability in this graph is the minimum number of locked arms to the wall during operation.

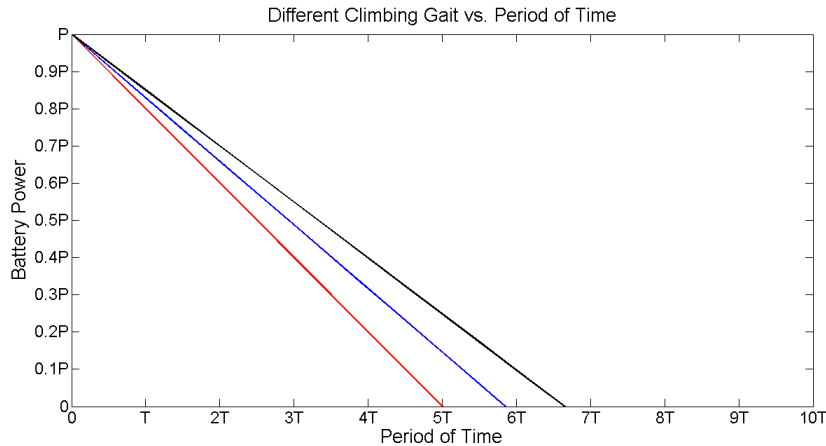


Figure 6.3:

Red line stands for simple pull-up(assumed as base of consumption), blue for optimized pull-up and black for dragging. Optimized climbing and dragging save steps (motor actuation), therefore beside time and consequently speed they would be saving power as well.

An application (simulation) was developed to represent these climbing gaits based on Time delays assumptions made in 5.1.2 (gravity and other factors were

not included). The interesting point about this virtual representation was to get a virtual illustration of the fact that how fast each of these gaits were compared to the others (figure 6.4). This application was made in Processing and was later further developed and connected to the DES and DNP to illustrate the gaits with feedbacks from the control system. Therefore it also could confirm the system functionality (2.4).

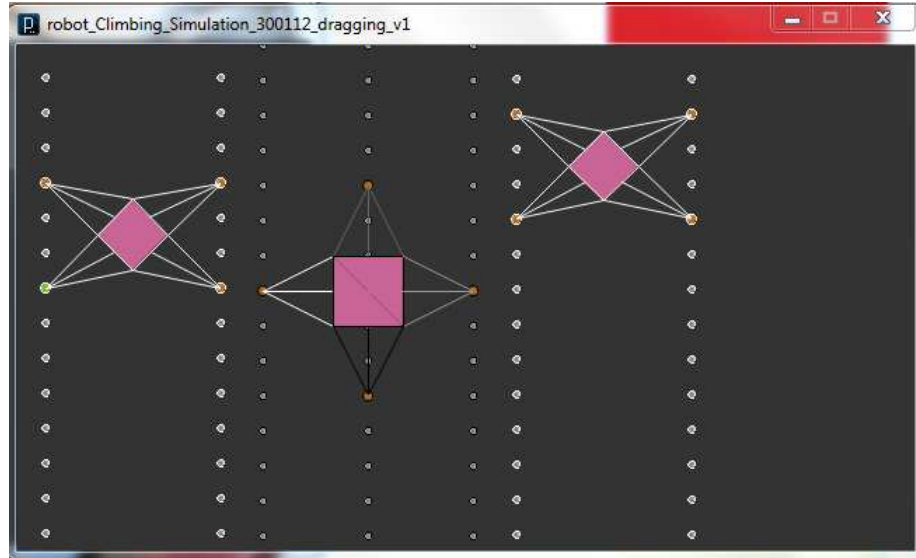


Figure 6.4: *Virtual presentation of climbing Gaits differences of speed*

6.3 Control Systems

The control system development process was divided into two parts, the control hardware and algorithm. The hardware controller was set of electronics components that were necessary to control the robot arms in operation, while the control algorithm is the software which is run on the hardware and takes care of logic at the system.

6.3.1 Distributed Embedded System

Beside experiments with different components during this project, the *fruit of all practical work with hardware components* was the Distributed Embedded System design. This system was implemented by using 5 Arduino boards (1 Mega and 4 Fio). This inter-connection of boards was implemented by using

I2C, RS-232(Serial) and ZigBee (wireless). The implementation of I2C protocol was stopped in the middle due to serious real time communication issues between master and slave node (in details in 5.3.1). The second implemented protocol was the serial connection using UART in hardware layer and RS-232 standard 5.3.1. This type of communication proved itself to be simple to install (cross connection of Rx and Tx), reliable and fast under the implementation and later tests (table C.3 in appendix). Serial communication based on RS-232 is highly recommended due to the experiments in this project **** 2.4 and discussion in 5.3.1).

On the other hand, if industrial circumstances would prefer wireless connection, then one could benefit ZigBee technology. XBee components figure 5.20 which were used in the wireless implementation of this project proved to be trustable under small tests done with data flow generator 5.6.1. Beside some time delays due to unknown reasons (possibly configuration issues), all the messages under test reached their destinations (Arduino Fio) and feedbacks were received on the other side (Arudino Mega). Using ZigBee in the design brings flexibility to the system and simplifies the whole maintenance and repairing process. This means in assembly and repair process, the technicians do not need to care about wiring between different nodes (Arduino boards here) in the system (almost plug and play concept) as all the communications were done wirelessly by already configured XBee components (right networking configurations). In addition, XBee components allow automatic upgrade of navigation program wirelessly. Cable based system does not have such advantages but benefits from being fast, secure and more redundant. It is up to the developer team and strategy makers of the project to choose in such cases based on existing circumstances.

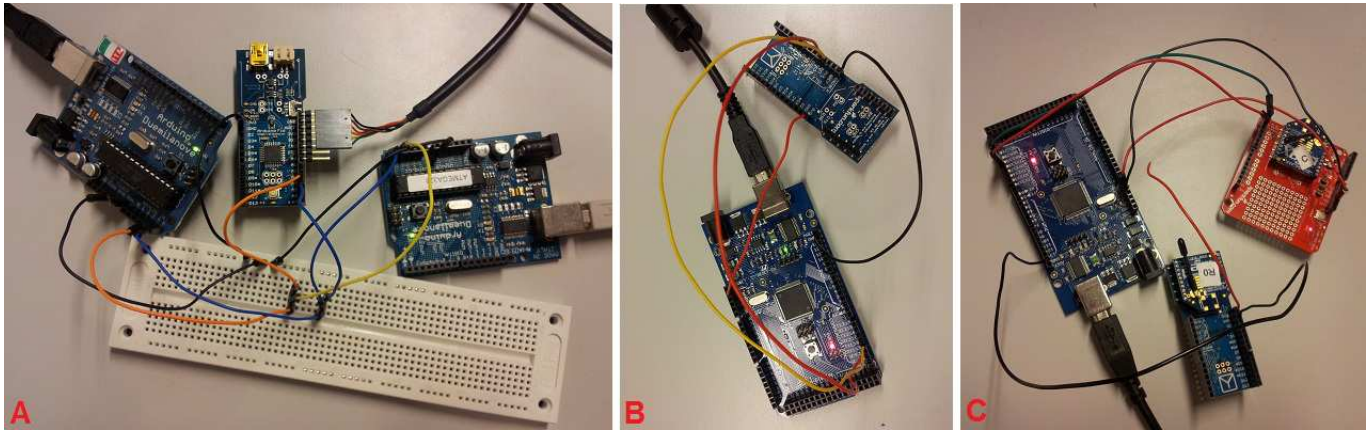


Figure 6.5: A: I2C - B: RS-232 - C:ZigBee

One of the most important issues that distinguished RS-232 (Serial) between all implemented models, was the fact that due to availability of the 4 RxTx ports on Arduino Mega, it was possible to filter the data traffic only to the interested destinations, specified in that message. E.g. a message to move right front arm, would not be sent to all other three arms but only to the right front arm. Such approach was difficult to apply in XBee and would cause time delays. Figure 6.6 shows the implementation of the DES with XBee components. The RxTx figure was decided to move to Appendix due to lack of space.

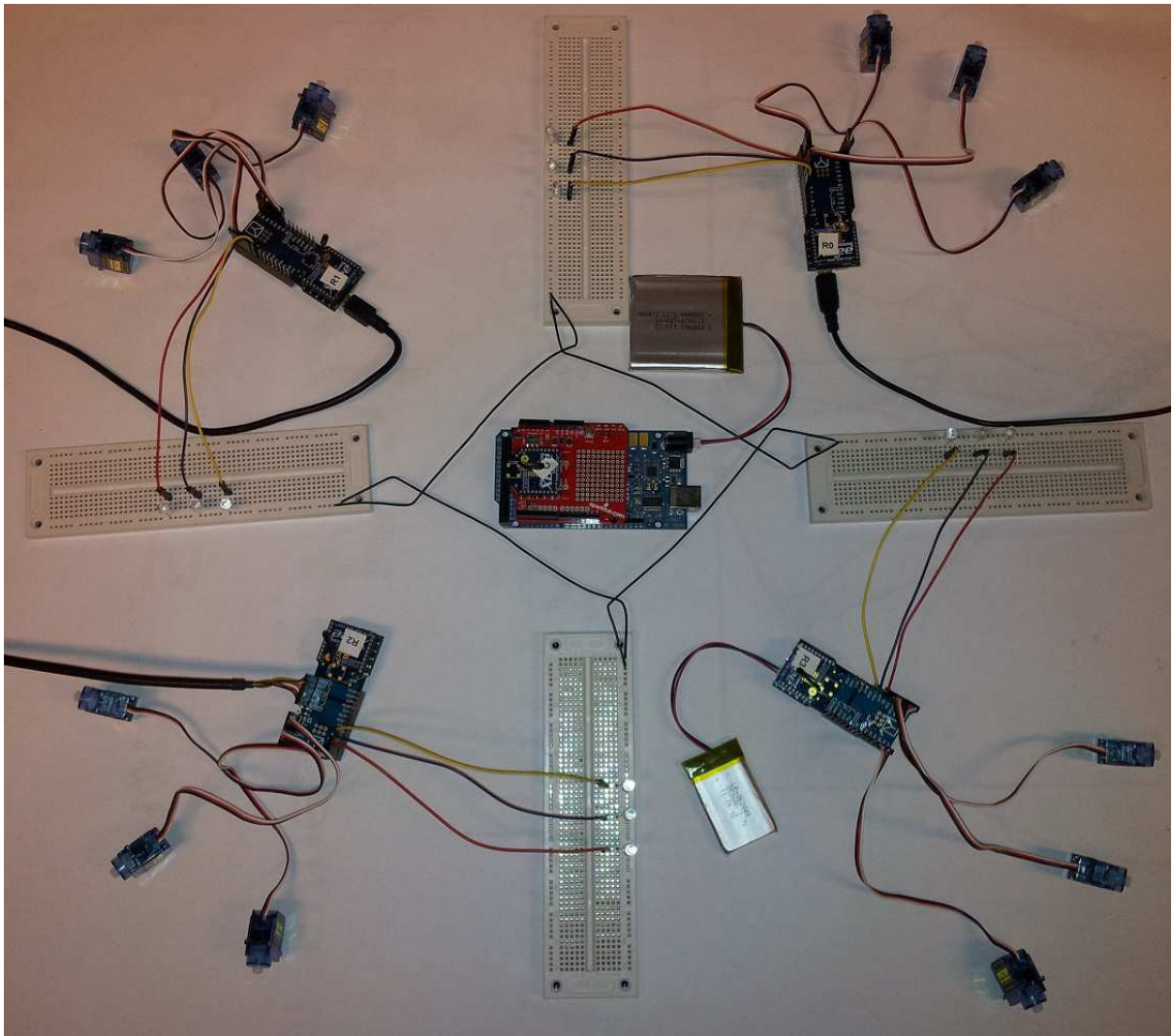


Figure 6.6:
Distributed Embedded System implemented by using ZigBee and servo motors

6.3.2 Distributed Navigation Program

The control algorithm is in charge of moving mechanical parts, with the help of control hardware, to implement the climbing gaits. This is done here through a distributed algorithm which is divided between different Arduino boards in D.E.S. design. This Distributed Navigation Program (DNP) developed had following features:

- Manual Remote Control
- Semi-Autonomy
- Optimized Power Consumption of Electronic Components

Manual remote control and overriding of the robot is one of the key specifications that must have been implemented in the concept due to HSE and safety issues. To this, three web-based remote control web pages, for developers, operators and offshore experts, were designed in a way that could be used as a portal to the robot system. (Figure 6.7) shows these control panels which are used for different purposes (steering, teaching the robot and monitoring received signals).

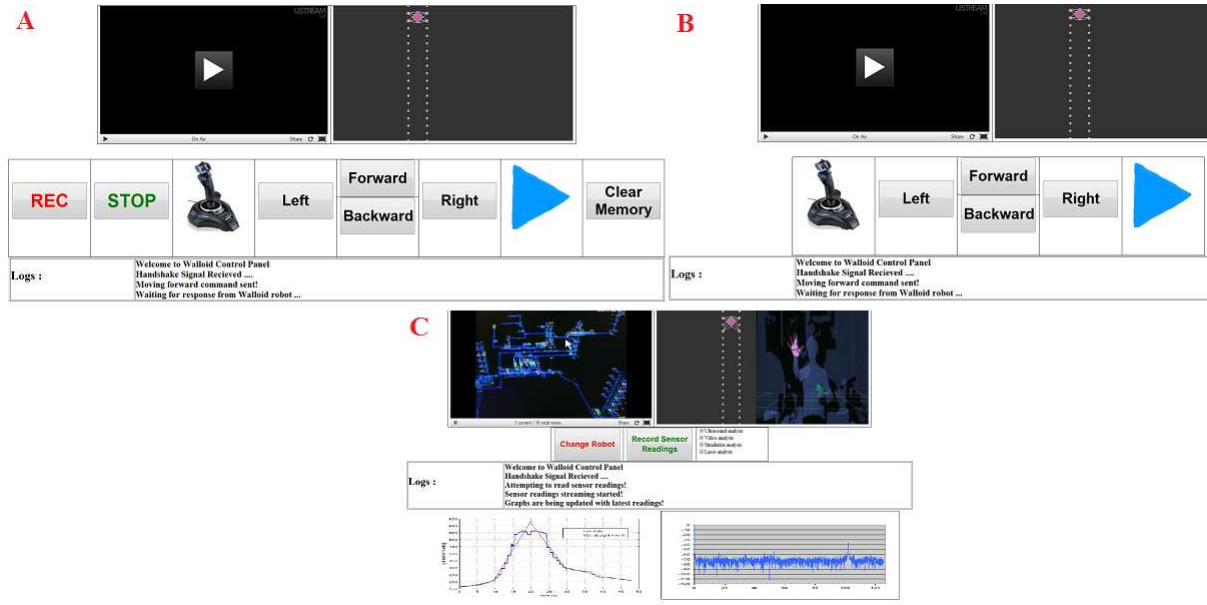


Figure 6.7: Web based Remote Control Panel, A:Developer | B:Operator | C:Expert

Manual steering was only the beginning of the development phase, although very critical. Walloid equipped with current DNP, developed through this project,

would become a semi-autonomy robot that can be taught doing tasks. The learning / teaching process is a simple approach and it is just to log whatever that happens under manual steering and re-create exact same behaviors later.

One of the disadvantage of DES design was extra power consumption. This is a serious disadvantage. Therefore a sleeping policy was added by using AVR sleeping policy. This means with benefiting AVR sleep mode, master controller (Arudio Mega) orders the idle slave controllers to enter sleep mode and later wakes them up by a serial interrupt. Based to discussions in section 5.5.3, this policy resulted in 40% reduction in power usage by micro-controllers while climbing by simple pull-up climbing gait. One might reason that compared to motor consumptions (current small motors are almost equal with Arduino boards), the Arduino boards would consume very little power. However, this power consumption in addition to the savings made during climbing gaits (faster gaits by skipping actuations), table C.2, development would have enough effect for the system and this could be continued in future.

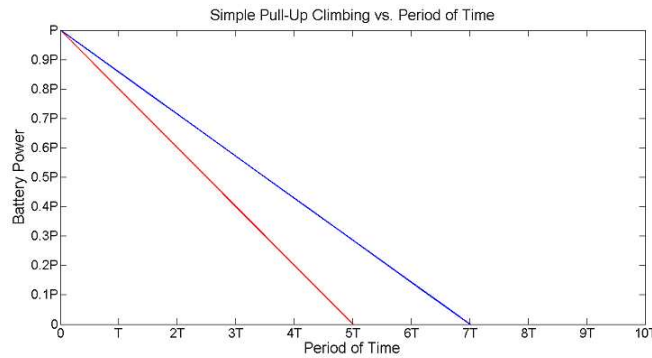


Figure 6.8:
Simple pull-up climbing gait Power consumption before and after implementation of Power Optimizer Algorithm (40% difference)

6.4 Simulation

The self-developed simulations were used during the process, since the Walloid arm broke down. Then self-developed simulations became the only way to have a physical presentation of the work done in virtual reality.

- Data Flow Generator: This was used for testing purposes in absence of a physical robot, motors, encoders and etc.

- Climbing Presentation and Confirmability Center: This simulation was developed by Processing and its aim at the beginning was only to illustrate climbing gaits in 3D / 2D environment, but later this was connected to the control hardware and worked as a confirmation tool for what the DNP and DES together were trying to achieve (climbing in right direction).

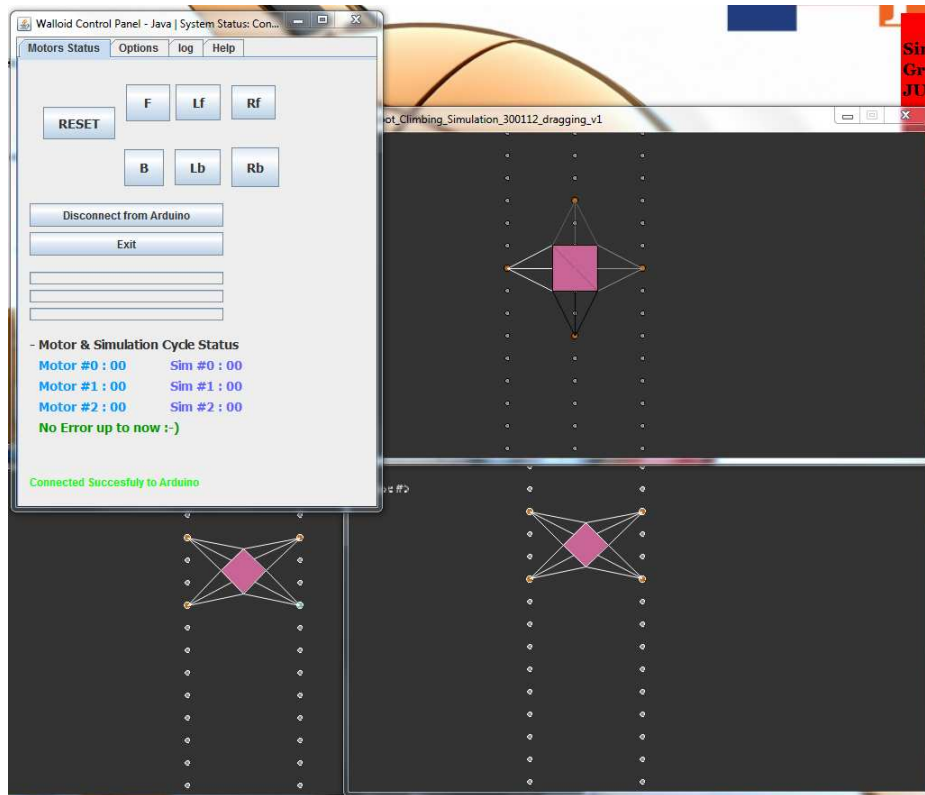


Figure 6.9:
Processing simulations and Java CP were cooperating to connect to the hardware.

6.5 Summary

This section was dedicated to go through implemented systems and results obtained from such attempts. The chapter starts by mentioning the initial plan in top-down view (chapter 4) and present the detailed results accordingly. Analysis of area of application (offshore platforms) was already addressed in chapter 2 and was therefore ignored.

Climbing Operation:

- Workspace calculations from Matlab (figure 6.1)
- Assembly of end effector final design with the rest of Walloid parts in Solidworks (figure 6.2)
- Three developed pre-programmed climbing gaits with graphical presentation of their speed were presented. The effects of these gaits on power consumption are also presented.

Control Hardware, the Distributed Embedded System:

- Implementation of DES with RS-232 and ZigBee is discussed and shown with figures (figure 6.6, 6.5)
- Time delays in using different technologies is measured and presented (table 6.5)

Control Algorithm, the Distributed Navigation Program:

- Web based control panels (figure 6.7)
- Semi-autonomous work-plan generator results are presented (2.4)
- Power Optimizer Algorithms effect on different climbing patterns were shown in charts (figure 5.32)

Specification Name	Type
Robot Type	Climbing Robot
Locomotive Power	4xArms
Adhesion Solution	Grasping Arms
End effector	4 End Effectors capable of grasping bolts with error tolerance in positioning (figures 5.4, 6.2).
Climbing Gaits	3 climbing gaits 1 - Simple pull-up (<i>Figure 5.6</i>) 2 - Optimized pull-up (<i>Figure 5.8</i>) 3 - Dragging <i>Figure 5.9</i>
Remote Accessible Robots	WiFi accssesed units
Micro-controller	Arduino boards with AVR Atmega 8bit chipset family (Arduino Mega for master and Arduino Fio for slave)
Distributed Embedded Systems	Network of 5 inter-connected Arduino boards connected together using RS-232 / ZigBee technology (Arduino Mega, Arduino Fio)
HW - SW interfacing	RS-232 (cable) / XBee (wireless), both connected through Serial port
Navigation System	Distributed Navigation Program with autonomy, remote control and power optimization features
Semi-Autonomy	+ Able to re-run pre-defined tasks
Remote Control	+ Remote Controlled through web-based CP and stand-alone Java SE application
High Level Programming Language	Java SE, Processing and Matlab
Low Level Programming Language	Arduino C, AVR C
Simulation	+Data Flow Generator +Climbing Gaits Presentation

Chapter 7

Robustness Issues

7.1 List of Issues

Robustness or error handling is the ability of the whole system to cope with the errors that occur during the operation. Errors could happen in any embedded systems. Therefore, the error handling should be expanded to monitor them in different processes, in all layers. Thereafter, the system should try to handle the situation and recover from the critical state to the normal state. The issue of recovery is very important, as it is almost impossible to fully stop errors from happening. The main areas of concern about error occurrences in a climbing robot are categorized down here at Table 7.1. The marked solution orientation defines which chapter includes the discussion of those challenges.

Table 7.1: Robustness issues and robot design, hardware and software oriented solutions

Robustness issue	Design Oriented	Hardware Oriented	Software Oriented
Passive joint control		X	X
Loss of contact of end effectors	X	X	X
Instability / Current angle of the system		X	X
Positioning Issues	X	X	X
Loss of vision during manual steering		X	X
Path blocking by broken robots		X	X
Power Loss		X	
Offline Modus, Network-less Operation			X

7.2 Blocked Paths

The design of Walloid has made it path dependent. This means that the robot should always have a free path to be able to climb on and inspect and monitor the environment. What would happen if a robot due to some technical problems gets stuck? E.g. a passive joint that does not respond, loss of power without the reserve battery to be able to come online, faulty bolts, etc. In these cases the other robots need to know that the path is blocked. The first and easiest solution to this would be allowing the server to know about this issue and broadcasting it to all active robots in the field. The other approach could be having critical situation signals on the robot that in case of troubles would trigger and broadcast "blocked path signal". This would both let other robots to know the path is blocked and also let the technicians to be able to locate an unresponsive robot.

Table 7.2: Risk analysis of blocked paths

Name	Description
Risk Name	Blocked Path
Reasons	*Passive Joint *Permanent power loss *Faulty bolts
Consequences	*Blocked paths which would prevent other robots to continue their work
Solutions	*Informing the server about situation to broadcast it to others *Broadcast local signals

Another approach which best fits offline robots is broadcasting a critical signal with current position to both warn the robots nearby to avoid this path and ask for help from the operators. Having the current position contained in the critical signal let the robots on the other paths to know about which path to avoid and re_calculate their way to the target and also lets the operators to be able to locate and fix the robot which is sending the emergency signal.

7.3 Positioning after Improper Shutdowns

The positioning issues are discussed already in kinematics and workspace sections (3.3). This section would try to address, discuss and solve the criteria of positioning in a robust control system in abnormal situations. Situations like sudden loss of power, unscheduled / improper shutdowns, etc. To be able to continue operation in such cases, it is critical to be aware of the current situation of the arms and the task that the robot was performing right before the sudden or improper shutdowns.

Table 7.3: Risk analysis positioning after improper shutdowns

Name	Description
Risk Name	Positioning after improper shutdowns
Reasons	*Sudden loss of power *Power interruption and restart
Consequences	*Loosing the overview over location of joints
Solutions	*Having zero position sensors *Storing current position into a non-volatile memory

7.3.1 Zero Positioning

Zero positioning and the ability to save the current position in case of losing power are very important features of a control system. This could be achieved in different ways:

- Hardware Oriented
- Software Oriented

The hardware oriented solution usually consists of one or several physical sensors which could detect either the current position of the actuators in the environment, or being in the zero position. The ability to detect both the zero position and the current position of the actuator depends on the number of sensors, their types and the design of the system. E.g. RepRap Darwin, an

Table 7.4: Advantages and disadvantages of using current position system

Pros	Cons
More control over Robot joints	Limited lifespan for E2PROM
Monitoring both zero and current positions	I/O operations are slow
Logging and troubleshooting opportunity	Interrupts/Polling requires CPU power

Open Source 3D printer uses beam light forks sensors in order to know if the printer actuator is in zero position.

However, this approach was not preferred for Walloid, as for a mobile robot being aware of current position is critical (not only the zero position). The next solution for this challenge would be the software oriented approach and the focus would be on current position of the joints. This information could be reached by keeping track of the information that was received from the encoders. This approach is discussed in detail in next section.

7.3.2 Current Positioning, the software oriented approach

The location of joint (variable length of the joint) and the direction of the movement are already known through reading encoders. To have this stored somewhere for abnormal situation (power loss), one could easily save this already computed resource to a non-volatile memory. This information could be used when the robot is restarted, turned off and last but not least in case of sudden loss of power.

In order to implement this idea, this information should be saved for every joint move on a non-volatile memory (E2PROM memory available on the micro-controller or an external SD card). This solution was implemented and turned out very well. Atmega micro-controllers benefits from E2PROM memory (built-in) which is non-volatile and is usually used to store small amount of data. This fits the needs as each prismatic joint requires one single integer number (a counter). This solution had several advantages in comparison with the previous hardware oriented approach for only zero positioning. The constant update of current position on non-volatile memory gave the opportunity to keep track of *not only the zero position, but the current position of the robot arm*. In addition to precise arm positioning, this feature can give the navigation program a very useful logging and troubleshooting opportunity.

7.4 Passive Joint Control

Passive joint means a joint which fails to function. As the motors would not be in control of the joint any more, it would follow natural rules like gravity and inertia and could even collapse [18]. Focusing on offshore platforms where HSE concern would be important, this issue could become very serious. Passive joint / joint failure can result in robot joints not to follow the received instructions. A Robust system would be able to detect this problem and warn the decision making center / robot operator, but lack of this ability could result in navigation system to believe everything works fine, while it is not. Back to Walloid robot, one could think of this as an arm that did not grasp the bolt due to joint failure and this could result in robot fall as the second arm would start the unlocking phase.

Table 7.5: Risk analysis of passive joint control

Name	Description
Risk Name	Passive Joint Control
Reasons	*Joint Failure *Broken Parts *,etc.
Consequences	*Losing control of the joint *Fear for crash / collapse
Solutions	*General: Design joints in a way to discover joint problems and warn the technicians *Walloid: Distance meter sensor should be placed in each joint jacket

During testing the received arm, broken rotor (figure 7.1) resulted in the motor shaft to turn the rotor around, while due to the half broken part, the force was not transferred to the screwing shaft. This error generated a very unique situation where the encoder was misleading the micro-controller about the joint working well, while the prismatic joint actually did not move at all. In such cases on-site human observer could easily find out this problem (as it was done under the test), but a navigation system or in case of remote controlling from onshore, it would not always be very easy to discover such errors. Therefore it is very critical to consider such possible problems before designing a robotic joint. Some of these situations can only be discovered during testing or from previous experiences with passive joint challenges (like in this case for Walloid robot).

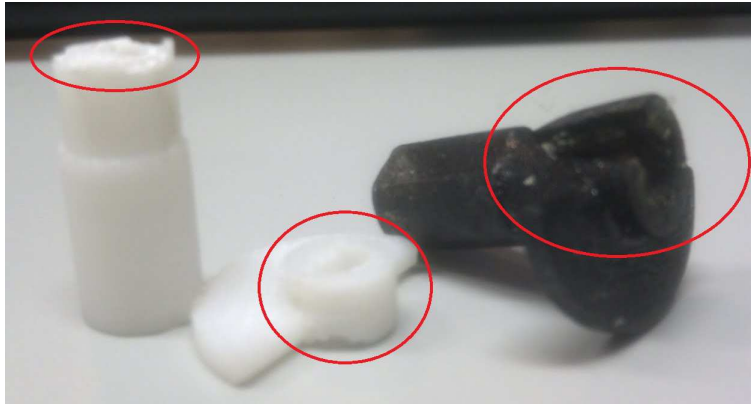


Figure 7.1:

The rotor on the right (black) was broken in a way that still could forward the motor rotation to the rotary part, but no to the screw shaft

The passive joint problem could be fixed by adding one sensor to each joint. This sensor could be of various types and should be placed either inside the prismatic joint jacket or on the end effector. This sensor measures the distance of the end effector to the surface of the wall and its various values confirms the movement of the end effector, which itself would be the result of movement of all three prismatic joints of an arm. The advantage of this solution is the few numbers of necessary sensors (only four sensors for all 12 joints). However, this solution would not be able to detect which joint suffers from passive joint control issue and would only confirm if one arm was in motion or not.

One might think this is enough for discovering arm fault, but it is important to pay attention that the values would still change (indicating no problem) even if one of the joints would be working fine. Therefore, one should go for the second choice which was placing sensors in each joint jacket to measure the distance from screw shaft to the sensors location (figure 7.2). To prevent extra load on the system this could be done in very low frequencies (beginning, middle and end of the move).

7.5 Power Interruption

Power interruption issue is the situation when the system loses the power and gets it back again. In case of sudden power loss, there are choices like back-up batteries to continue the operation. Such solutions are critical for a robust system due to the importance of reliability for industrial applications. Imagining a powerless hanging robot on a platform wall in the middle of the sea can illus-

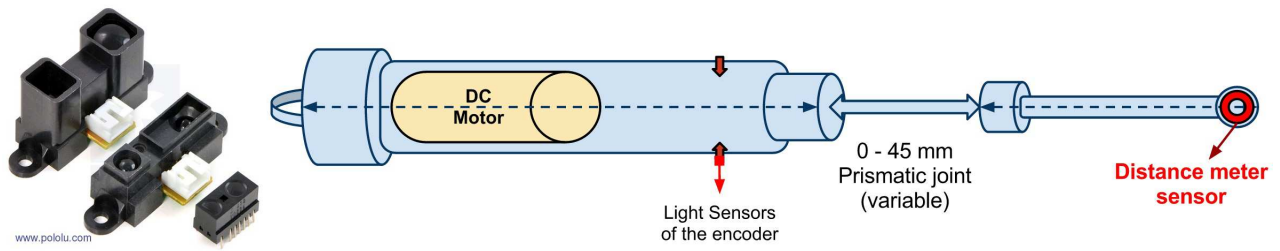


Figure 7.2:
Pololu Digital Distance Sensor - 2: Prismatic joint with distance meter sensor - 3:
Sensor - Micro-controller schematic

Table 7.6: Risk analysis of power interruption

Name	Description
Risk Name	Power Interruption
Reasons	*Battery Failure *Bad contacts *,etc.
Consequences	*Losing power in the middle of operation
Solutions	*Off the shelf product called Automatic Battery Back-Up Switch (ICL7673) equipped with a proper capacitor

trate the importance of such solutions. Therefore, the priority is to develop a way to let the system come back on after sudden power interruption. One can place a backup battery onboard the robot to allow the system to recover power loss. In order to make this happen automatically, one need to design a logical circuit that can discover the loss of power from the main source and bring the reserved source online. (Figure 7.3)shows such system.

First the plan was to design a circuit which could take care of the power loss issues, but later products were found which were being produced in industrial scale for such purposes. This *off the shelf inexpensive IC (Automatic Battery Back-Up Switch) called ICL7673* [82] (around 5\$) was used to discover the power interruption and automatically bring the backup battery online. However, this does not eliminate the shock imposed to the system. Moreover to prevent the restart of the system, the circuit can be modified (Figure 7.3) by

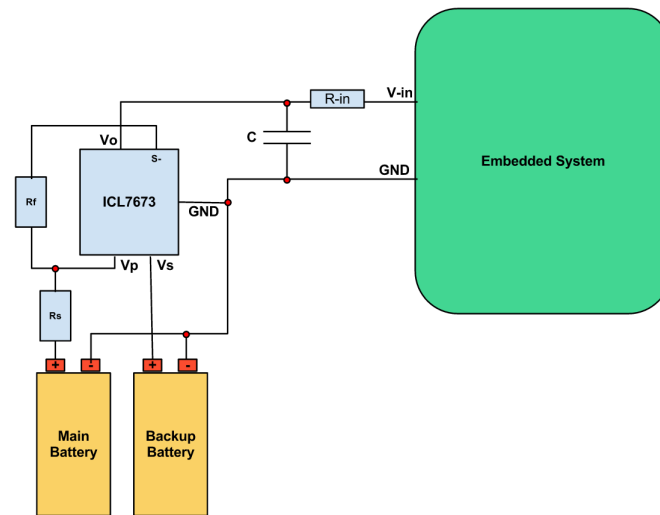


Figure 7.3:
Left schematic, switching to spare battery with power loss - Right schematic, switching to spare battery without power loss

adding one proper capacitor to feed the system with stored power, during the time it takes to bring the backup battery online (feeding the system with power from the back-up battery again).

7.6 Instability / Current Orientation

The awareness of the control system over the current angle (orientation) of the system is useful information for navigation program. Current angle of the system refers to the angle that the robot makes with horizon (**figure 7.4**). This angle would be zero when the robot is walking on the floor or 90 degrees on a straight wall. The orientation angle would be varying during strides and over adjoining surfaces. One might ask how this angle might help in tackling the instability problem. The idea here is to let the control system be aware of the current angle and use this in decision making and safety protocol. One could also think about using this input as an indication about successful grasping. E.g. *sudden varying values, after the grasping operation is finished successfully according to the control software, could indicate possible problems in grasping operation.*

The easiest way to find out the current orientation of the robot chassis is to mount a Gyro sensor (gyroscope) [83] on-board the robot to monitor current angle. It is possible to detect the rotations in x-y-z orientations (figure 7.4) due to the latest upgrades in gyroscope hardware. Another *important issue is to*

Table 7.7: Risk analysis of instability

Name	Description
Risk Name	Instability
Reasons	*Bad contacts to the surface *Vibrations *,etc.
Consequences	*Loosing balance and possible fall
Solutions	*Adding a Gyroscope to the robot to find out the current orientation

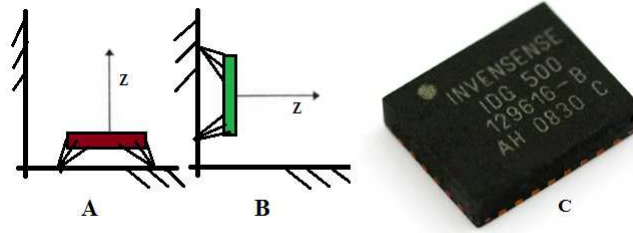


Figure 7.4:

Left to right: zero and ninety degrees angle with horizontal line | 2: Gyro sensor [83]

ignore the small changes as these can be possible noises due to the vibration from the surface. This approach was not implemented due to lack of availability of the sensor.

7.7 Offline Modus, Network-less Operation

The assumptions have been always based on being connected to the server and enjoy the high processing power from the server side, but there are occasions that the robot might fail to go online. The reasons for such a problem could be various failures from environmental issues to hardware problems or server being down. Table 7.8 tries to do a risk analysis based on loosing connection in the system which results in the robot to go to offline modus.

Table 7.8 goes through the possible risks that results in system going offline.

Table 7.8: Risk analysis of offline robots

Name	Description
Risk Name	Offline Robot
Reasons	Hardware failure Environmental interruption Server being down Miscommunication between server and HMI
Consequences	Considerable loss of processing possibilities (processing power and external application processing on server side) Loss of server guided decision making Loss of communication with other robots in the field Logging gap Gap between received information which results in errors in the system (motor counter is 30 and suddenly after connection established again it is 70). This is discussed in details in next section. <u>Possible</u> loss of video streaming (independent hardware in sending streaming data, can happen if the problem is on network stations or server down)
Solutions	Alternative connections (3G, 4G, GPRS Satellite network connections) Preparing for an offline modus for processing and decision making Local knowledge and goal database Redundant server configuration (several backup servers to switch to) Enough local sensors to be able to avoid crashing into obstacles and other robot with offline processing On-site communication between robots (Xbee) Local buffers and queues on the robot (SD card) to store information in offline modus The system should be able to cope with losing connection and recovering the system failures after reestablishment of the connection

Consequences and solutions to such risks are also presented. This table also shows how paying enough attention to redundancy can affect the whole system. E.g. *special attentions in independent video streaming from micro-controllers*

operation allows the video report to continue, although the connection of the robot system with the servers is interrupted. System malfunction and wireless adapter are situations that this independent video system still allows the technicians to evaluate robot situation based on visual report and call for right actions (critical or normal recovery plan by operators).

As the first solution in the table 7.8 specifies, an alternative backup network connection for both embedded systems and camera can be provided. Such *alternatives* can be *mobile broadband with 3G or 4G technology*.

7.8 Security Concerns

Table 7.9: Risk analysis of security concerns

Name	Description
Risk Name	Security Concerns
Reasons	*Hacks *Sabotages *Sniffing
Consequences	*System out of control
Solutions	*More security in all levels *Filtered Network

The wireless networked operation on platforms not only reduces the implementation and maintenance prices A.1, but also brings up security concerns. These concerns can be sniffing, hacks, sabotages and etc. Due to these threats, it is best that wireless networks shape an isolated and protected Intranet for only internal communication between users, servers and the robots in that union. Another approach can be filtering the users with their physical MAC addresses (white list). However this does not fully protect the network, but increases the security level. Due to the fact that the already existing network infrastructure ought to be used from the Internet (for the users to be able to connect to the system and work with it), this network cannot be fully isolated, but the data exchange should be limited in jointing nodes and specially pay attention to security in all levels (VPN, firewalls, anti-virus programs, anti-sniffing programs, etc).

For the Robot-Server side, one might suggest encryption of the exchanged messages. This is a great idea, which has its own downsides. Based on the complexity of the encryption keys, such operations would slow down the process of data exchange between the robot and the servers would not be real time

anymore. This is a very important issue that the security is a vital part of this design, but at the same time should respect the real-time data communication.

7.9 Summary

Robustness chapter tried to cover the abnormal situations that could occur for Walloid and presented solutions to recover the system and bring it back to normal mode. Table 7.9 shows such abnormal situations.

Issue Name	Solution
Blocked Paths	Informing Server / Broadcasting emergency signal
Positioning after Shutdown	Storing current position on non-volatile memories like EEPROM / SD card
Passive Joint Control	Adding a distance meter sensor inside joint jacket
Power Interruption	Automatic Battery Back-Up Switch(ICL7673) equipped with a proper capacitor
Instability	Adding a gyroscope sensor could help in monitoring the orientation
Offline Mode	<ul style="list-style-type: none"> *Alternative connections (3G, 4G, GPRS Satellite network connections) *Preparing for an offline modus for processing and decision making *Local knowledge and goal database *Redundant server configuration (several backup servers to switch to) <p>Enough local sensors to be able to avoid crashing into obstacles and other robot with offline processing</p> <p>On-site communication between robots (Xbee)</p> <p>Local buffers and queues on the robot (SD card) to store information in offline modus</p> <p>The system should be able to cope with losing connection and recovering from the system failures after reestablishing the connection</p>
Security Concerns	Balance between tight security and not interrupting or slowing the real-time communication

Chapter 8

Conclusion

Questions provide the key to unlocking our unlimited potential.
- Anthony Robbins

8.1 Conclusion

This project was an attempt in further development of an ongoing project at University of Oslo, called Walloid robot. Walloid is a prototype climbing robot for offshore platforms. **Offshore issues, climbing operation and control systems** were topics of interest in this work. As Walloid robot is not finished yet due to practical challenges, the work here was evaluated by simulation results (self-developed simulation applications).

With regards to the remaining parts of Walloid project (chapter 3) and the analysis of offshore platforms as the area of application (section 2.4), practical work was started and driven in three directions:

- **Climbing Operation**
- **Control Hardware**
- **Control Algorithm**

Climbing Operation:

The climbing gaits were developed with focus on speed issue. In these climbing gaits, names were simple pull-up, optimized pull-up and dragging gait, which the second and third gait were respectively 17% and 33% faster than the initial simple pull-up. However, increasing the speed resulted in less stability (minimum number of locked arms), but still acceptable. These gaits also resulted in reduction in power consumption as they improved the speed by skipping some operations (section 5.1.2).

Control Hardware

:

During development of hardware controller, a distributed approach was chosen due to need for increased processing power, real-time responses and availability (lack of enough number of ports on the boards and absence of more powerful processors in the lab). The Distributed Embedded System was implemented by 5 Arduino boards (one master and four slave boards) and the inter-connection was made possible by both cabled (RS-232, UART) and wireless (ZigBee) method.

Control Algorithm

:

The control algorithm consisted of a master algorithm and a slave algorithm running respectively on DES master board and slave boards. Due to the slow speed of climbing phases, by utilizing AVR sleep policy, 40

At the end, the work done during the project was evaluated by self-developed simulation application. These simulations were developed for testing, presentation and later automation purposes.

8.2 My Contribution

Despite the extensive amount of topics to attend to during this project, I have tried to stay focused and looked at key concepts which are critical in the concept of turning Walloid into an inspection climbing robot. Among all contributions I've had in this project, three following points are highlighted:

- **Three pre-programmed climbing gaits were developed, with focusing on optimizing the speed. Second gait was 17% and third gait was 33% faster than first one.**
- **Distributed Embedded System was designed with 5 Arduino boards and two approaches of it with RS-232(UART) and ZigBee was implemented.**
- **Distributed Navigation Program developed based on hardware specifications (DES), with remote control (web-based), semi-autonomy (recreating already stored/logged job in knowledge database) and power optimizing (40% reduction in processors power consumption) features.**

8.3 Further Works

As in any other academic project, no project is ever finished and there is always room for improvements. Following concepts are recommended if one would like to further develop Walloid project or my contributions to it.

- **Building the actual robot as it is currently on hold**
- **Testing and verifying the current project with actual robot**
- **Expand the kinematics calculation of the robot to satisfy different types of movements, corresponding to those found in industrial manipulators, linear movement (MoveL) and Joint movement (MoveJ)**
- **Further development of distributed embedded system for other purposes and applications**
- **Further development of distributed navigation system to a fully automated navigation system that could use the idle extra processing power to compute overloaded information**
- **Working on Walloid design for being able to move between adjoining surfaces**
- **Further development of the simulation programs with gravity force and other important factors.**
- **Finishing the started process of Route and Task Generator program and expand it to accept several active robots in the field with number of tasks which should be done in the field and get optimized work-plans for each robot.**
- **Working on speed issues by changing the motors to faster motors (e.g. hydraulic motors instead of screw shaft)**

Appendix A

Interviews

A.1 Anders Røyørø

**Principal Researcher Efficient Production
TPD RD New Development Solutions
Statoil ASA**

Q: Is there any offshore requirements exactly pointing at robotic automation in offshore application?

A: No. Per today there are just general requirements for offshore application that applies to robotic automation of oil and gas platforms. Safety is the main requirement in such cases and one of the main concerns is the chance of starting a spark which could kindle a fire or explosion.

Q: Per today what kind of robots are active on platforms?

A: Right now there are different kinds of robots such as industrial manipulators active on the platforms. This also applies to wheeled mobile robots, but up to now we do not have any climbing, nor flying robot on our platforms and this is because of our special attention for the safety protocols and lack of stable AI technologies in this field. We think this would change in near future as per today there are AI algorithms that can control a plane with passengers automatically and offshore requirements are not higher than aerospace industry.

Q: what kind of field of research are you doing for offshore?

A: Inspection, maintenance, semi-automatic and manually guided tasks are areas which robots are active per today. Mostly we use robots to check, inform or confirm some special tasks. We do not see for ourselves the chance to get welding robots approved for usage in north sea at least because of NORSOK requirements for such tasks are very high. This that a robot can make sparks and therefore starts a fire on the platform is a very sensitive issue.

Q: What does semi-automatic tasks mean?

A: By semi-automatic I mean tasks that are planned by us for the robot, and later the robot can do the work automatically based on the input scenario (work plan) from us. It's important that we do have control access on robots as not all the tasks can easily be planned for a robot and most important thing is that we do not know all the aspects of occurrences on a platform and therefore the need for manual steering can suddenly be required. Another limitation is the physical shape of the platforms that are not designed for operational robots.

Q: Do future platform designs consider a robot friendly environment design?

A: Yes. The future platforms design do consider the need for robotic automation in future.

Q: After TAIL IO, is there any new robotic automation projects ongoing at StatoilHydro ?

A: Yes, but I am afraid I can not name the projects, nor the activities. I can only say they cover maintenance and repair tasks.

Appendix B

Visual Reports

B.1 3D Designs

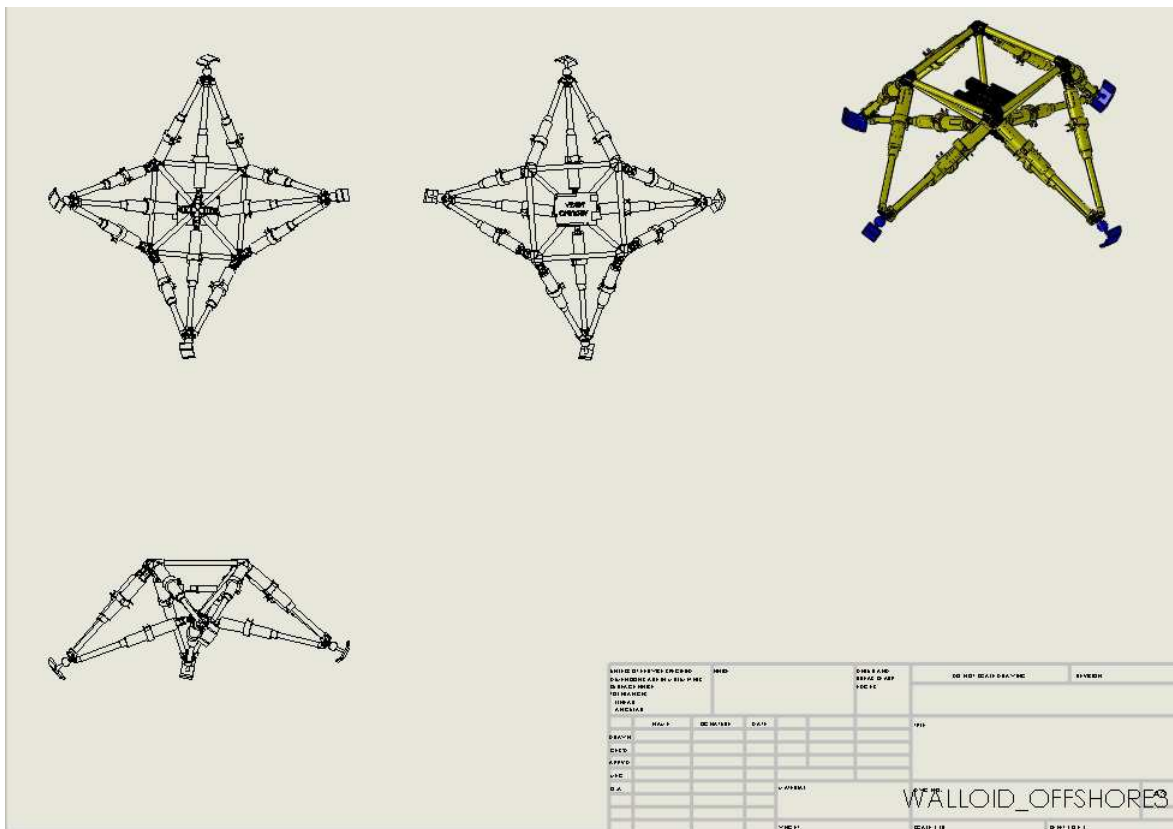


Figure B.1: Walloid robot designs

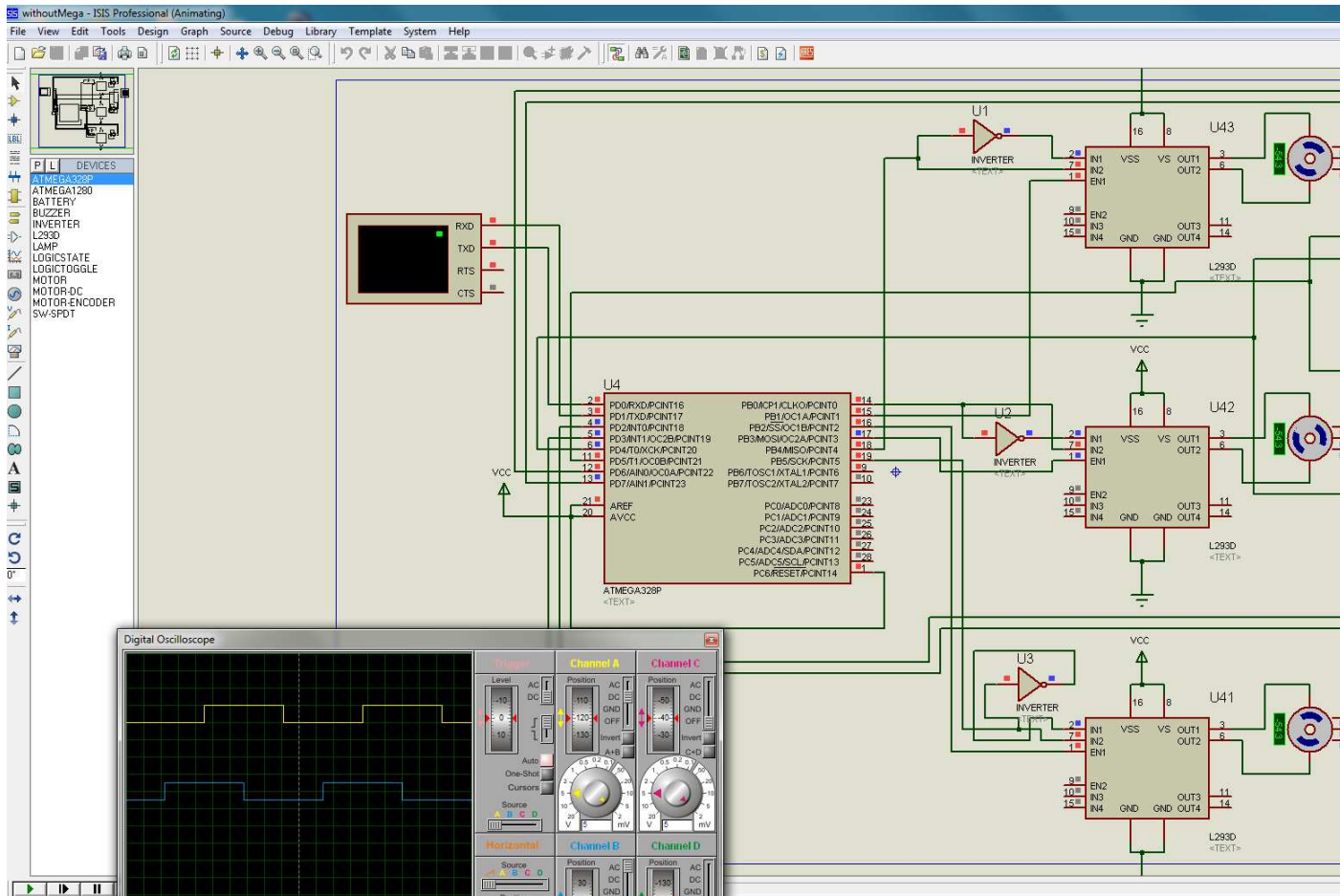


Figure B.5:
Proteus simulation of the DES, which wasn't so successful due to the complexity of the simulation process in Proteus

Appendix C

Remainings

C.1 Climbing Operation

Adhesion Solutions

Climbing robots use different adhesion methods as an anti-gravity force. Climbing operation can be divided into two main parts. First, the adhesions to the surfaces of the walls which keeps the robot from falling and second the locomotive power that actually moves the robot forward on the wall [20]. It was discussed in ?? that adhesion force and locomotive power, grasping arms, was already decided by Walloid project. Therefore no development was done in this area, but this type of force is compared with other alternatives C.2.

Docking Station

Docking station means somewhere like a nest for the robots to charge the batteries, getting fixed and even being cleaned up. Developing a robot specialized for a platform in the middle of the sea, a docking station needs to be somewhere safe from all the extreme conditions. Somewhere safe for the hard shell of a robot that is water proof, salt proof or any other super characteristic, to be opened up and fixed. The specifications of such shelter are presented down here.

- Re_charging spot
- Salt / Water / humidity / ice proof
- Washing spot for robots with high pressure water
- Repair spot
- Software upgrade spot

Table C.1: Industrial climbing robots active in different fields [84–90]

Task	Area of Application	Name	Climbing	Control System	Cons	Pros
Cleaning / Inspection	Ships / Underwater	Hull Bugg [84]	Wheels, Negative Pressure	Auto / RC	Fits only even surfaces	Autonomous, does what it supposed to do perfectly
Inspection	Oil / Water Tanks	[85]	Tracked / Magnet	RC	Only metal surfaces, Operator should be present	Stable, fully controlled by OP
Welding	Welding on metal platforms	MRWS / MRWS mini [91]	Tracked / Magnet	RC / Only on magnetic surfaces, Operator should be on site	Stable, fully controlled by operator	
Glass cleaning / Inspection and Skyscrapers / Buildings	GEKKO Junior G1 [86]	Tracked / Suction cup	Auto / RC	Fits only smooth surfaces	Stable, Auto, remote control option	
Solar panel cleaning / Solar power plans	GEKKO Junior G3 [87]	Tracked / Suction cup	Auto / RC	Fits only smooth surfaces	Stable, Auto, remote control option	
Abrasive Blast cleaning / Ships/Oil tanks/Nuclear plants	UA [88]	Wheels / Suction cup	RC	-	-	
Cleaning / Polishing / Inspection	Ships/Oil tanks/Nuclear plants	UD [89]	Tracked / Suction cup	RC	-	-

Table C.2: Adhesion techniques advantages and disadvantages

Solution	Pros	Cons	Examples
Grip, Ring Bolt	*Low coast (path) *Extra Stable *Redundant	*Path needed *High precision required	Monkey, Human, Apes
Van der Waals force	*Path free *Flexible in terrains *Relatively stable	*High cost of producing the material *Not Redundant (Would fall in case of loss of power) * Not good for rough surfaces *Question of good functionality in extreme cold and wet conditions and also on rough terrains	Gecko, Lizard
Lock(IKEA type) itself to the wall	*Relatively coast(path) *Extra stable *Redundant	Low *Path needed *Very high precision required	-
Suction cup	*Easy implementation *Path free *Stable on smooth surfaces	*Clear/Smooth terrain required	-
Magnet	*Very stable *Path independent (on metal walls)	*Path dependent (on non-metal walls) *Extra power for electric magnets required *Unstable under special conditions (thick ice coverage) *Not Redundant	-
Pressing to the inner surface	*Stable *Redundant	* Destructive for the terrain	Cats, Koalas, Goanna

One of the early specifications of our climbing robot design was the ability to charge itself while climbing the path. The contact between the end effector and the bolt which is on the wall can be a perfect way to conduct electricity to the battery charger. We know that in each move, there are at least two and mostly three end effectors fastened to the surface (discussed in climbing gaits). This means there is always at least one contact with the bolt and in case of having bolts that conduct electricity, we can charge our batteries and never have the problem of shortage of energy. Such specification could be even turned into a safety protocol which simply checks if the contact is made and the electricity stream is flowing through the arm toward the battery. This is done by a simple voltage detector. Such properties could also be used to detect the location of the robot in the platform.

Although this idea sounds very ideal, but in implementation of this idea, there are challenging issues. To have electricity on the bolts without any protection, *even with low voltage*, is definitely not advised, especially when this system is supposed to be implemented in the middle of the sea. High level of humidity, rain, ice and the usual metal surface of the platform walls are most common things that can conduct out electricity from the bolts. This in best case can cause short circuits when the robot is trying to grab the bolts, or in worst case can be dangerous for human operators in the environment (although there is a low voltage). *Therefore it is not advised to implement such specification on the whole path, but just inside the docking station.* This station should be a shelter with low humidity, far from ice, rain and sea water. A place where robots can dock there while they are done with their daily tasks, get re-charged, washed up with high pressure water (probably by another robot), and even fixed by human operators (??).

One can choose to implement such specification, low voltage electricity for charging batteries on bolts, either on the whole path inside the docking area, or only on the exact 4 bolts where the robot hangs on and rests / gets maintained and repaired. Another issue which ties to the topic of docking area is the issue of upgrading the navigation software. Having a connection to the server, being able to receive packages wirelessly, gives us the chance to upgrade the navigation program automatically. However this might not sound that wise, regarding incidents such as hacking and taking over the control of one of the U.S. military drones [?, drone]hich means such machines can be re-programmed and used for other purposes. This security issue can be solved by letting upgrades of the navigation program to happen if and only if the robot is located inside the docking area. This means optimized usage of maintaining time (e.g. upgrading the navigation program while the robot is being washed with high pressure water).

Vibration Mechanical oscillations about an equilibrium point are called vi-

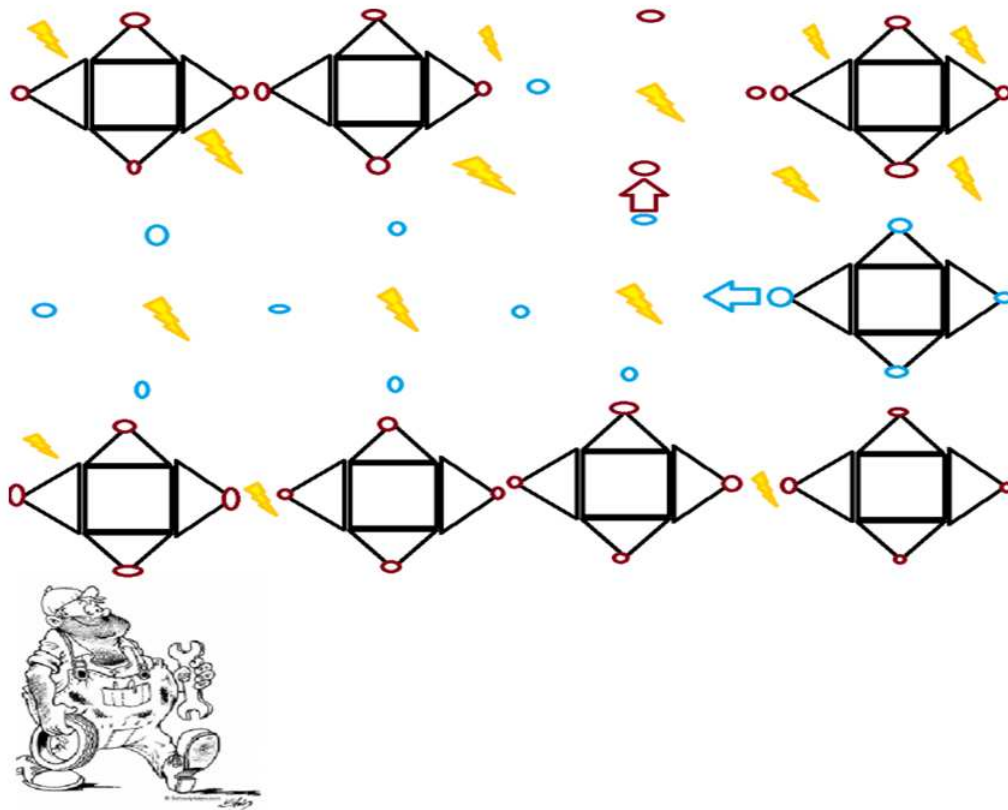


Figure C.1:
Robots moving in docking station to their place, being charged, maintained and software upgraded

bration. Vibration could be either periodic (pendulum) or random (a tire on an uneven road). Back to our topic, on-board engines and electronic devices encounter mechanical vibration and shock loads in most marine vessels including oil and gas platforms. The waves force which crashes to the structure in an irregular pattern from all sides, form a type of vibration that could affect the operation on the platform. It's obvious that these shock loads are more considerable when there is a storm which would result in more powerful waves that respectively results in higher rates of vibration in the structure. This vibration rate also depends on the type of platform which can be fixed, floating, Spar, etc. Floating platforms vibrates almost like a ship and the vibration effect is high on them.

Beside structural concerns, vibrations could damage a robot which is a machine made of both electronics and mechanical parts. Mechanically everything that vibrates will eventually fail. This would differ from case to case based on

the vibration rate and the type of mechanical activity. E.g. a non-mobile robot that is mounted on the wall or to the ground with screws could get unstable over time due to the worn out screws. The same could also happen to a mobile robot which does have parts which are in close contact with each other. Vibrations makes their contact even more and adds to the friction which results in worn out edges that would need frequent maintenance part changes (e.g. screw shafts).

The problem of vibration and the electromagnetic devices can be divided in two categories. The physical problems at joining points of different components and the problem of the components with piezoelectric characteristics. The first problem would be local as shocks and vibrations would affect soldering joints, socketed parts, non-locking connectors and hanging heavy components over-time. Disconnection or bad connection can be a typical problem that might rise up for machines that operate in such conditions. On the other hand the second problem would create trouble for other devices that are sensitive to noise. Components that exhibit piezoelectric characteristics (ceramic capacitors, usually based on barium titanate), when vibrating, give out electrical noise or undesired signals on undesired bandwidth (REFERENCE???). These bandwidths could have been reserved for special operations on platforms which can result into critical situations in case of interfering with important signals such as control room signals.

Solutions for tackling these challenges should be focused on reducing transmitted shock loads to sensitive equipment or using equipment that are built to use in such environments (e.g. CompactPCI than PCI cards). There are several suppliers that specializes their products and services to tackle this specific problem [?, gmtg] Such products usually are more expensive and are certificated under standards that test vibration and shock resistance capabilities. Standards like US military spec, MIL-STD-810 [36] or other offshore equivalents. Also solutions such as mounting electronics on rubber bushes or even gel (flash memories of black box in planes) may be considered (REFERENCE). It's recommended to test the target environment with an average vibration rate over a period of time. This would show if there is any need for such concerns, before choosing any of the earlier mentioned solutions. Field test with an accelerometer with logging abilities in a long enough period of time can give us a very good overview over amount of vibration on site.

C.2 Hardware Issues

Connection Delay Tests, ZigBee vs. RS-232 :

The test were done under equal conditions, where the code was kept equal

Table C.3: Test results from Distributed Embedded System implemented for Walloid robot

Protocol	Frequency	Accuracy	Average Time	Max Time	Max Delay	No. of tests	No. of sample pr. tests
RS-232 (Serial)	19200Hz	100%	1.12s	1.00s	***Slowest-Avg***		30,000
ZigBee (Zig-Bee)	19200Hz	99.9%	196.00s	210.75s	***Slowest-Avg***		716

(except for I2C, where the reading port was not Serial). ZigBee and RS-232 both use 19200 hz data rate and the test was repeated to reach stable results (average values) and would not be misguided by small changes. It was expected that ZigBee results would take more at time, as XBee components that was used in this test used RxTx ports (RS-232) to send data from XBee component to the micro-controller. However the numbers recorded showed that same communication with same code, ran on the same hardware, would take around 210 times more time to be sendt, recieved on the other side and then recieve the feedback.

C.3 Software Issues

Remote - Server - Client

Robot : The robot specifications and hardware design is discussed in details in previous chapters. The navigation program in control of the climbing robot was developed through a development process (5.4.3) and ended with a distributed navigation program written in Arduino C and AVR C??, divided between different inter-connected micro-controllers. The navigation system is implemented to operate in three modes, *Auto*, *RC (Remotely Controlled)* and *Teach* (??). As defined in 5.2.2 and 5.3, the robot is supposed to be connected to WiFi network and through that to server.

Server :As stated before as an assumption, in previous chapter 5.2.2, the robots are online with help of WiFi technology and therefore could be easily connected to server computers. servers are powerful machines capable of high speed data processing with additional processing software / libraries (e.g. Matlab, Java EE, OpenCV, Control Program with several parallel threads, data logging, etc). The server could also be in control the multicasting/broadcasting of streamed video from the climbing robot to the clients. Future video processing and advanced decision making for automatic steering can easily be implemented on server-side. *An attempt for automating climbing operation, task planning and work-plan production for several robots on-board the platform*

is presented in 5.5.6.

Clients : Although the control system developed during this work, once taught something, can work independently, having a teaching tool, redundancy and HSE concerns call for manual overrides and control. Robot operator could steer the robot remotely through CP in the *RC mode*. In this case the operator could be anywhere and the only requirement is a stable network connection. On the other hand concept of client in an offshore climbing robot is not only the operator. As figure 5.22 showed, the possible stakeholders of such system could be task developers and experts observing sensor readings (continued in ??). During this work both a standalone CP (developed in Java SE) and web based CP's were developed (5.5.4).

Flexibility 5.5.4, redundancy, stability ??, easier development and troubleshooting (single problem at a time) in later phases are the result of this approach. It is critical to have *standard input - output model* designed in advance to ensure success in later integration. During this work manual remote control 5.5.4, offline and server side automation 2.4 are implemented and at the same time for extra safety manual overrides are always possible during whole process.

Another issue about these modules is about the server and HMI. It is assumed that in an already developed industrial ICT designs, *these parts will be replaced by already existing systems*. Therefore the controller algorithms are designed to be able to work with different systems, as long as the server is compatible with the input standard of the robot and the server is compatible with the output standard from the robot. This is done by choosing development tools (e.g. Java, Processing, Arduino) that are compatible with different platforms (Linux, Windows, Mac, etc) ??.

stakeholder5 The user case diagram in figure 5.22 clearly stated three kinds of stakeholders for a RSC system. The separation of stakeholders was done to assign different authorizations to each group. The authorizations could limit specific group from running robot in one special mode (Auto, RC, Teach). E.g. experts have no control authorizations at all, while operators can run the robot in Auto and RC mode. Finally task developers have access to all levels of the navigation system and can take the robot to Teach mode and teach the robot new tasks.

1. **Robot Operator :** This user could either be onshore or in the control room on the platform running robot in RC or Auto mode. The CP for this stakeholder can read commands from the control column and send it to the server for forwarding along to the climbing robot (Figure ??, 1).
2. **Robot Developer / Task Planer CP :** Control Panel for the task planer to take the robot to the **Teach mode** and either teach the robot by running

it manually and recording the moves, or by loading new tasks and paths to robot memory (developed in simulator environment). The limitation for Teach mode results in **more security and stability** as only authorized experts could change the configurations on the robot (Figure 5.26, 2).

3. **Simple Clients** : Simple clients are those users who do not have any authorization to control the robot, but only receive the information streamed out from the robot environment, e.g. sensor data, video, etc. Such users could be the experts onshore trying to monitor and help a robot operator / offshore staff in sensitive tasks (Figure 5.26, 3).

————— **Current Positioning Reading sensor readings** The sensors from the encoder are connected to the micro-controller pins. Based on which pins they are connected to we would have the luxury of choosing between two methods of reading data from these pins, interrupts and polling. Different AVR Atmega micro-controllers have various numbers of pins with ability to run interrupt codes based on external hardware signals. Number of these pins can vary from two to four on different types of micro-controllers. The problem arises when discussing three DC motors; each encoder monitoring the motor shaft requires 2 pins; on each arm, which means anyway we would have too few pins to work on. This limitation in further works could be resolved by using more advanced processing units with higher capabilities. The interrupts save the CPU resources. When an interrupt signal is generated by the hardware, everything would halt and the CPU would run the feedback code for that specific interrupt. Afterwards the CPU would continue the normal execution of tasks once again. Such approach guarantees running the feedback code, but in case of simultaneous interrupts would act based on a priority table in AVR Atmega datasheets [47].

The other approach to read the input data is polling, which simply means periodically reading the status of pins to determine whether received signal should be read. From programming point of view the functions in charge of polling are simple function that are placed in the infinitive loop of micro-controller. This is opposite of interrupts functions that are marked by the specific interrupt code. Although interrupts are slow, but they are accurate (determinism). In case of CPU being too busy on other functions, we might risk losing reading some input data. The algorithm for implementation of reading input data by both interrupts and polling is archived under Appendix.

————— **Storing sensor readings** In order to program the discussed solution about current positioning in practice, there are needs for nonvolatile memories combined in an embedded system. The first and easiest choice for this kind of technology is EEPROM (E2PROM). EEPROM stands for Electrically

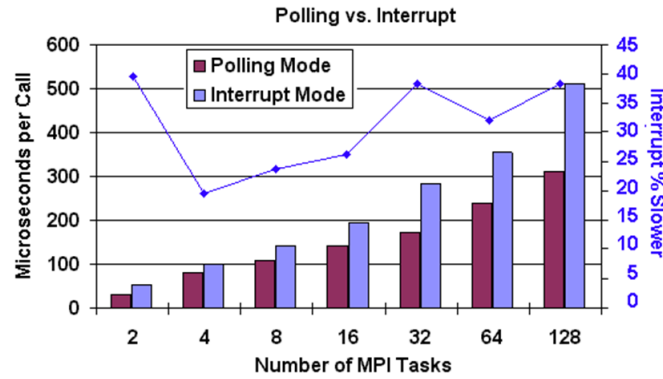


Figure C.2:

In case of continuous receiving signals polling can be faster, halting, running interrupt functions, and retrieving system to the previous mode is time consuming [92]

Erasable Programmable Read-Only Memory. This type of memory is usually used for saving small amounts of information that needs to be saved safely in case of losing power/switching off the device. One of the most known usage of EEPROM is in Personal Computers (PC) BIOS system which stores the startup configuration of your PC. The advantage of EEPROM technology is their availability on AVR Atmega micro-controllers. They are already there and can easily be used.

The only advantages of using currently available EEPROM technology are the limitations that follow this technology. Although these problem are mostly fixed in modern EEPROMS (slow single byte operation which is now changed to multi-byte page operations), but the limited lifespan is the biggest challenge in using this technology. Limited lifespan means that the EEPROM memory is not usable after a certain number of erase/write cycles. You can see the detailed number of these limitations for different Arduino boards / micro-controllers in **table C.4**.

As our goal is for offshore platforms environment, it's important to mention that according to some experiments the number of these cycles in room temperature can go up to more than 12 times bigger than the guaranteed number from Atmel [93]. It is also important to bear in mind that the datasheet value applies to all operating temperature varying from -55°C to +125°C and this is over our functioning range. It would be nice to see same test results for one and other extreme temperatures (Offshore environment can get extremley cold and windy).

If EEPROM is frequently re-written the lifespan can be an important de-

Table C.4: EEPROM memories limitations

Processor	Arduino board	Write/Erase	EEPROM Memory
Atmel ATmega168	*LilypadOld *Nano *Diecimila	100,000	512 Bytes
Atmel ATmega328	*Duemilanove *Fio *Uno *SMD Lilypad	100,000	1 Kilobyte
Atmel ATmega1280 or 2560	Mega series	100,000	4 Kilobyte

sign consideration, as end of EEPROM lifespan also means the end of micro-controller life. Using external EEPROM memories(C.3) could fix this problem and can be an easy maintenance choice. The disadvantage of not using the internal memory is losing some pins which are occupied by the external memory. The current positioning system with EEPROM is implemented and the code can be found in **Appnedix**.

Considering the problem of lifespan in EEPROM memories, one might think of other solutions for storing current position. The other solutions could be connecting an external microSD card shield, or any type of nonvolatile memory that does not have the short lifespan problem (Flash memory and etc). The disadvantage of these solutions is also losing extra pins which would be occupied by the microSD card shield or similar solutions (**figure C.3**).



Figure C.3:

Left to right : 1:Extrenal EEPROM memory | 2: MicroSD shield for Arduino boards | 3: microSD card

Appendix D

Source Code

D.1 Control program, Java

```
// ***** Java

package wpnStart;

import viewPack.*;
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author shahabfm_adm
 */
public class WPNStarter{
    //Variable
    WPNGuiStarter startView;

    public WPNStarter() {
        startView = new WPNGuiStarter();
    }
    /**
     * @param args the command line arguments
     */

    public static void main(String[] args) {
        // TODO code application logic here
        new WPNStarter();
    }
}

package controlPack;

/*
```

```

* To change this template, choose Tools | Templates
* and open the template in the editor.
*/

/**
 *
 * @author Shahab F.
 */
//This class:
// - Starts up the communication with the Arduino.
// - Reads the data coming in from the Arduino and
//   converts that data in to a useful form.
// - Closes communication with the Arduino.

//Load Libraries
import java.io.*;
import java.util.TooManyListenersException;
//Load RXTX Library
import gnu.io.*;
import java.awt.Color;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import javax.swing.JLabel;
import java.util.Enumeraation;
import javax.swing.JProgressBar;
import javax.swing.Timer;

//***** WALLOIDARDUINO *****
public class WPNArduino implements Runnable, SerialPortEventListener, ActionListener{

    //Used to in the process of converting the read in characters -
    //first in to a string and then into a number.
    String rawStr="";
    String tempStr = "";
    //Declare serial port variable
    SerialPort mySerialPort;
    private JLabel[] motorLabels;
    private JLabel[] simLabels;
    private JProgressBar[] progressBar;
    private JLabel errorLabel;
    private TextArea logText;
    private boolean problemsWhileConnecting = false;
    //time & date for logging purpose
    Calendar cal = Calendar.getInstance();
    SimpleDateFormat sdf = new SimpleDateFormat();
    private boolean stop = false;
    //Declare input steam
    InputStream in;
    OutputStream out;
    //Motor Status
    private int mStatFromArduino[] = new int[3];
    private int m0stat = -1, m1stat = -1, m2stat = -1; // -1 = stopped motors | 0 = forward | 1 = backward
    //Simulator Status
    private int sim0Counter = 0, sim1Counter = 0, sim2Counter = 0;

    Timer timer;

    public WPNArduino(JLabel[] importedLabels, JLabel[] simL, JProgressBar[] progBar, JLabel errorL, TextArea

```

```

        motorLabels = importedLabels;
        simLabels = simL;
        progressBar = progBar;
        errorLabel = errorL;
        logText = logT;
        problemsWhileConnecting = false;
        for (int i=0; i<3; i++)
            mStatFromArduino[i] = 0;

        /*      MOVE THIS TO RUN METHOD
        timer = new Timer(347, this);
        timer.setInitialDelay(0);
        timer.start();

        logThis("Walloid C1S0 O.S. started ! ");
        */
    }

    // necessary method for Runnable classes
    public void run(){
        timer = new Timer(347, this);
        timer.setInitialDelay(0);
        timer.start();

        logThis("Walloid C1S0 O.S. started ! ");
    }

    //Reading all the active serial ports on PC and adding them to one
    //jCombo box at the GUI
    public String[] readAllSerialPorts(){
        int counter = 0;
        String[] tempStr = new String[10];
        try
        {
            //Finds and opens the port
            Enumeration portIdList = CommPortIdentifier.getPortIdentifiers();
            while (portIdList.hasMoreElements()) {
                CommPortIdentifier cpi = (CommPortIdentifier) portIdList.nextElement();
                if (cpi.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                    tempStr[counter] = cpi.getName();
                    counter++;
                }
            }
        } //end of while
    } //end of try
    catch (Exception e)
    {
        System.out.println("Port in Use: "+e);
        logThis("Port in Use: " + e );
        problemsWhileConnecting = true;
    }
    String[] returnStr = new String[counter+1];
    for (int i=0; i<counter; i++){
        returnStr[i] = tempStr[i];
    } //end of for
    //returning an array of Strings
    return returnStr;
} //end of readAllSerialPorts

//This open's the communications port with the arduino

```

```

public void start(String portName,int baudRate)
{
    stop = false;
    try
    {
        //Finds and opens the port
        CommPortIdentifier portId = CommPortIdentifier.getPortIdentifier(portName);
        mySerialPort = (SerialPort)portId.open("my_java_serial" + portName, 2000);
        //2000 is the timeout value that is given to the system to release the port (2000 ms = 2 seconds), R
        if (mySerialPort!=null){
            System.out.println("Serial port found and opened");
            logThis("Serial port found and opened");
        }
        //configure the port
        try
        {
            mySerialPort.setSerialPortParams(baudRate,
            mySerialPort.DATABITS_8,
            mySerialPort.STOPBITS_1,
            mySerialPort.PARITY_NONE);
            System.out.println("Serial port params set: " + baudRate);
            logThis("Serial port params set: " + baudRate);
        }
        catch (UnsupportedCommOperationException e)
        {
            System.out.println("Probably an unsupported Speed");
            logThis("Probably unsupported speed !!!");
            problemsWhileConnecting = true;
        }

        //establish stream for reading from the port
        try
        {
            in = mySerialPort.getInputStream();
            out = mySerialPort.getOutputStream();
        }
        catch (IOException e)
        {
            System.out.println("couldn't get streams");
            logThis("Couldn't get streams.");
            problemsWhileConnecting = true;
        }

        // we could read from "in" in a separate thread, but the API gives us events
        try
        {
            mySerialPort.addEventListener(this);
            mySerialPort.notifyOnDataAvailable(true);
            System.out.println("Event listener added");
            logThis("Event listener added.");
        }
        catch (TooManyListenersException e)
        {
            System.out.println("couldn't add listener");
            logThis("Couldn't add listener");
        }
    }
    catch (Exception e)
    {
        System.out.println("Port in Use: " + e);
    }
}

```

```

        logThis("Port in use : " + e);
        problemsWhileConnecting = true;
    }
}

//Used to close the serial port
public int closeSerialPort(){
    try{
        in.close();
        stop=true;
        mySerialPort.close();
        System.out.println("Serial port closed");
        logThis("\n" + "Serial port closed.");
        return 0;
    }
    catch (Exception e){
        System.out.println(e);
        logThis("Exception occured closing port : " + e);
        return -1;
    }
}

//end of closSerialPort()

//Method sends a character
public int sendData(String chArray)
{
    try{
        //System.out.print("\n*** Sending command");
        logThis("Sending to Arduino :" + chArray + ".");
        for (int i=0; i<chArray.length(); i++){
            out.write(chArray.charAt(i));
            //System.out.print(" " + chArray.charAt(i) + " ");
        }
        System.out.println("");
        return 0;
        //for (int i=0; i<3; i++)
        //out.write(telorange.charAt(i));
        //System.out.println("*** Sent Data");
    }
    catch (Exception e)
    {
        System.out.println(e);
        logThis("Exception occured in sending data(" + chArray + ") - " + e);
        return -1;
    }
}

//HANDSHAKE METHOD
public boolean handshakeChecker(){
    int handshakeCounter = 0;

    //System.out.println("Sending Ack");
    try{
        //out.write('h');
        char handShakeChar;
        while(true){
            out.write('h');
            handShakeChar = (char)in.read();
            if (handShakeChar == 'h'){
                //System.out.println("*** ACK received from Arduino ***");
                logThis("ACK received from Arduino" + handshakeCounter);
                return true;
            }
        }
    }
}

```

```

        if (handshakeCounter > 3){
            logThis("*** Error, HANDSHAKE protocol was not received ***");
            return false;
        }
    } catch (Exception e){
        System.out.println(e);
        logThis("Exception occurred in HANDSHAKE protocol " + e);
        problemsWhileConnecting = true;
        return false;
    }
} //end of handshakeChecker

public boolean problemWhileConnecting(){
    return problemsWhileConnecting;
}

public int resetAllMotors(){
    try
    {
        //System.out.println("*** Trying to RESET ***");
        out.write('R');
    }
    catch (Exception e)
    {
        System.out.println(e);
        logThis("Exception occurred in RESETTING all motors " + e);
        return -1;
    }
    return 0;
} //end of resetAllMotors

/*
This method reads the characters sent by the arduino
The Arduino output looks like this:

128.43A153.25B0.84C242.62D0.63E128.43A153.25B0.84C242.62D0.63E128.43A153.25B0.84C242.62D0.63E

The method below divides this continuous stream of characters send by the Arduino into values
The letter after the numbers plus decimal point identifies the numbers and decimal point that
preceded it.

So:

128.43A: value = 128.43 with identifier A (A in this case is real power)

Lets go through this in further detail:

Program recieves characters one after the other like this:
ch = 1 – isDigit(ch) = true and so it is added to the string rawStr
ch = 2 – isDigit(ch) = true and so it is added to the string rawStr
ch = 8 – isDigit(ch) = true and so it is added to the string rawStr
ch = . – isDigit(ch) = false but we check if ch='.' and if so it is added to the string rawStr
ch = 4 – isDigit(ch) = true and so it is added to the string rawStr
ch = 3 – isDigit(ch) = true and so it is added to the string rawStr
ch = A – isDigit(ch) = false but – isLetter(ch) = true and so we then convert the rawStr that is now "128
double with double value = Double.parseDouble(rawStr);
Now we have a value with an identifier and we then print the value to terminal but we could just as well
if (ch=='A') realPower = value;

```

The last thing we do is reset rawStr="" and then the above process is repeated.

```

*/
public void serialEvent(SerialPortEvent event)
{
    //Reads in data while data is available
    while (event.getEventType()== SerialPortEvent.DATA_AVAILABLE){
        //System.out.println("beginning of while : " + tempStr);
        //System.out.print("data available : ");
        try
        {
            //-----
            //Read in the available character
            //System.out.println("Inside SerialPortEventn");
            char tempC;
            char ch = (char)in.read();
            //tempStr += ch;
            //System.out.print(ch);
            if (safeToPrint(ch)){
                tempStr = tempStr + ch;
                //System.out.print(ch);
            }
            /* The protocol for receiving data from Arduino :
            * h : Handshake protocol
            *
            * C### : C for Counter, First # stands for motor number, * a separating sign
            * and the last two digits stands for counter number from Arduino card
            *
            * E### : Error code ###
            *
            * IF the buffer length is more than 5, then a critical error in reading has happened
            */
            /* if (tempStr.equals("h")){
                System.out.print("Handshake received");
                logThis("*** Handshake received from Arduino ***");
                tempStr = "";
            }
            */
            if ((tempStr.length() == 6) && (tempStr.charAt(0) == 'C') ){
                updateMotorStatLabels(tempStr);
                //System.out.println("tempStr is now : " + tempStr);
                tempStr = "";
            } else if ((tempStr.length() == 3) && (tempStr.charAt(0) == 'E') ){
                //translateErrorMsg(tempStr);
                logThis("Error code " + tempStr.charAt(1) + tempStr.charAt(2) + translateErrorMsg(tempStr) );
                //System.out.println("tempStr is now : " + tempStr);
                tempStr = "";
            } else if ((tempStr.length() == 1) && (tempStr.charAt(0) == 'R') ){
                logThis("*** RESETTING ***");
                //System.out.println("tempStr is now : " + tempStr);
                tempStr = "";
            } else if ((tempStr.length() == 6) && (tempStr.charAt(0) == 'P') ){
                setCurrentPosition(tempStr);
                //System.out.println("tempStr is now : " + tempStr);
                tempStr = "";
            } else if (tempStr.length() > 6){
                logThis("Too many characters in Serial Buffer, Emptying the buffer now. Current character se
                System.out.println("LOGGED bigger than 6 char : " + tempStr.length() + " -" + tempStr+"!");
                tempStr = "";
            }
            else{
                //System.out.println("ELSE : " + tempStr);

```



```

    }
    //making sure NULL characters are not printed
    //tempStr = "" + ch;

    /*if (ch == 'C'){
        switch ((char)in.read()){
            case '0':
                //add to 0 that it is moving and cycle number
                tempC = (char)in.read();
                if (safeToPrint(tempC))
                    System.out.print(tempC);
                if (safeToPrint(tempC))
                    motorLabels[0].setText("#" + tempC);
                break;
            case '1':
                //add to 1 that it is moving and cycle number
                tempC = (char)in.read();
                if (safeToPrint(tempC))
                    System.out.print(tempC);
                if (safeToPrint(tempC))
                    motorLabels[1].setText("#" + tempC);
                break;
            case '2':
                //add to 2 that it is moving and cycle number
                tempC = (char)in.read();
                if (safeToPrint(tempC))
                    System.out.print(tempC);
                if (safeToPrint(tempC))
                    motorLabels[2].setText("#" + tempC);
                break;
        } //end of switch
        ch = (char)in.read();
        tempStr += ch;
        System.out.print(ch);
        ch = (char)in.read();
        tempStr += ch;
        System.out.print(ch);
        //sensorLabel.setText(tempStr);
        //signalInterrupt(tempStr);
        //System.out.println("strTemp is : " + tempStr);
    } else if (safeToPrint(ch)){
        System.out.print(ch);
    } //end of else-if

    //break;
    *
    */
}
catch (IOException e)
{
    logThis("IOException occured " + e);
}
//System.out.println("End of while : " + tempStr);
} //end of while
} //end of serialEvent()

private void printOutSafe(char ch){
    if (ch >= 1 && ch <= 126)
        System.out.print(ch);
} //end of printOutSafe

private boolean safeToPrint(char ch){

```

```

        if (ch >= 32 && ch <= 126)
            return true;
        else
            return false;
    } //end of safeToPrint()

    private int logThis(String logMsg){
        if (logText != null){
            logText.setText(logText.getText() + "->" + sdf.format(cal.getTime()) + " : " + logMsg + ".\n");
            return 0;
        } //end of if
        else{
            return -1;
        }
    } //end of logThis

    private int updateSimStatLabels(String simStat){
        int tmpNum = 0;
        //System.out.println("IN UPDATE MOTOR" + motorStat);
        if (simStat == null || simStat.length() != 6){
            return -1;
        }
        else{
            switch (simStat.charAt(1)){
                case '0':
                    //add to 0 that it is moving and cycle number
                    tmpNum = (simStat.charAt(3)-48)*100 + (simStat.charAt(4)-48)*10 + (simStat.charAt(5)-48) ;
                    //System.out.println("tmpNum is : " + tmpNum);
                    //sim0Counter = (int)(tmpNum * (100/213));
                    sim0Counter = tmpNum;
                    simLabels[0].setText("Sim #0 : " + sim0Counter);
                    logThis("Motor " + simStat.charAt(1) + " is at "
                        + tmpNum + " counter level | Protocol : " + simStat);
                    break;
                case '1':
                    //add to 1 that it is moving and cycle number
                    tmpNum = (simStat.charAt(3)-48)*100 + (simStat.charAt(4)-48)*10 + (simStat.charAt(5)-48) ;
                    //System.out.println("tmpNum is : " + tmpNum);
                    //float flTemp = (100/213) * (float)tmpNum;
                    //sim1Counter = (int)(tmpNum * (100/213));
                    sim1Counter = tmpNum;
                    //System.out.println("mSfromArduino is : " + flTemp);
                    simLabels[1].setText("Sim #1 : " + sim1Counter);
                    logThis("Motor " + simStat.charAt(1) + " is at "
                        + tmpNum + " counter level | Protocol : " + simStat);
                    break;
                case '2':
                    //add to 2 that it is moving and cycle number
                    tmpNum = (simStat.charAt(3)-48)*100 + (simStat.charAt(4)-48)*10 + (simStat.charAt(5)-48);
                    //System.out.println("tmpNum is : " + tmpNum);
                    //sim2Counter = (int)(tmpNum * (100/213));
                    sim2Counter = tmpNum;
                    simLabels[2].setText("Sim #2 : " + sim2Counter);
                    logThis("Motor " + simStat.charAt(1) + " is at "
                        + tmpNum + " counter level | Protocol : " + simStat);
                    break;
            } //end of switch
            return 0;
        }
    }

    private int updateMotorStatLabels(String motorStat){
        int tmpNum = 0;

```

```

//System.out.println("IN UPDATE MOTOR" + motorStat);
if (motorStat == null || motorStat.length() != 6){
    return -1;
}
else{
    switch (motorStat.charAt(1)){
        case '0':
            //add to 0 that it is moving and cycle number
            tmpNum = (motorStat.charAt(3)-48)*100 + (motorStat.charAt(4)-48)*10 + (motorStat.charAt(5)-48);
            //System.out.println("tmpNum is : " + tmpNum);
            mStatFromArduino[0] = tmpNum;
            motorLabels[0].setText("Motor #0 : " + mStatFromArduino[0]);
            logThis("Motor " + motorStat.charAt(1) + " is at "
                    + tmpNum + " counter level | Protocol : " + motorStat);
            break;
        case '1':
            //add to 1 that it is moving and cycle number
            tmpNum = (motorStat.charAt(3)-48)*100 + (motorStat.charAt(4)-48)*10 + (motorStat.charAt(5)-48);
            //System.out.println("tmpNum is : " + tmpNum);
            mStatFromArduino[1] = tmpNum;
            //System.out.println("mStatFromArduino is : " + mStatFromArduino[1]);
            motorLabels[1].setText("Motor #1 : " + mStatFromArduino[1]);
            logThis("Motor " + motorStat.charAt(1) + " is at "
                    + tmpNum + " counter level | Protocol : " + motorStat);
            break;
        case '2':
            //add to 2 that it is moving and cycle number
            tmpNum = (motorStat.charAt(3)-48)*100 + (motorStat.charAt(4)-48)*10 + (motorStat.charAt(5)-48);
            //System.out.println("tmpNum is : " + tmpNum);
            mStatFromArduino[2] = tmpNum;
            motorLabels[2].setText("Motor #2 : " + mStatFromArduino[2]);
            logThis("Motor " + motorStat.charAt(1) + " is at "
                    + tmpNum + " counter level | Protocol : " + motorStat);
            break;
    } //end of switch
    return 0;
}

} //end of logThis()

private String translateErrorMsg(String temp){
    int tempInt = Integer.parseInt("" + temp.charAt(1) + temp.charAt(2));
    errorLabel.setForeground(Color.red);
    errorLabel.setText("Check logs for the ERRORS : -(");
    switch (tempInt){
        case 0 :
            return "*** General ERROR ***";
        case 1 :
            return "*** ERROR reading from ENCODER ***";
        case 2 :
            return "*** FATAL ERROR, CRASH POSSIBLE, STOPPING MOTOR ***";
        case 3:
            return "*** FATAL ERROR, ROBOT ARM MIGHT GO LOOSE, STOPPING MOTOR ***";
        default :
            return "*** Unknow ERROR ***";
    } //end of switch
} //end of translateErrorMsg()

public void readCurrentPosition(){
    try{
        out.write('P');
    } catch (Exception e){

```

```

        System.out.println(e);
        logThis("Exception occured in reading Current Position protocol " + e);
        problemsWhileConnecting = true;
    }
} //end of setCurrentPosition()

//SIMULATION UPDATE DOESNT WORK !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
public void setCurrentPosition(String temp){
    logThis("Initial stat of motor " + temp.charAt(1) + " is at "
            + temp.charAt(3) + temp.charAt(4) + temp.charAt(5)
            + " counter level | Protocol : " + temp);
    updateMotorStatLabels(temp);
    updateSimStatLabels(temp);
    //simCnt[temp.charAt(1)-48] = (temp.charAt(3)-48)*100 + (temp.charAt(4)-48)*10 + (temp.charAt(5)-48);
    //System.out.println("simCnt is : " + simCnt[temp.charAt(1)-48] + "|" + ((temp.charAt(3)-48)*10 + (temp
} //end of setCurrentPosition()

public void actionPerformed(ActionEvent e) {
    if (m0stat == 0){
        if ((sim0Counter + 1) <= 100){
            sim0Counter++;
            progressBar[0].setValue(sim0Counter);
            simLabels[0].setText("Sim #0 : "+sim0Counter);
        } else{
            //notificationLabel.setText("Simulation of Motor 0 has reached MAX values");
            logThis("Simulation of Motor 0 has reached MAX values");
            m0stat = -1;
        }
    }
    else if(m0stat == 1){
        if ((sim0Counter - 1) > 0){
            sim0Counter--;
            progressBar[0].setValue(sim0Counter);
            simLabels[0].setText("Sim #0 : "+sim0Counter);
        }
        else{
            //notificationLabel.setText("Simulation of Motor 0 has reached negative values");
            logThis("Simulation of Motor 0 has reached negative values");
            m0stat = -1;
        }
    }
}
if (m1stat == 0){
    if ((sim1Counter + 1) <= 100){
        sim1Counter++;
        progressBar[1].setValue(sim1Counter);
        simLabels[1].setText("Sim #1 : "+sim1Counter);
    } else{
        //notificationLabel.setText("Simulation of Motor #1 has reached MAX values");
        logThis("Simulation of Motor #1 has reached MAX values");
        m1stat = -1;
    }
}
else if(m1stat == 1){
    if (sim1Counter - 1 > 0){
        sim1Counter--;
        progressBar[1].setValue(sim1Counter);
        simLabels[1].setText("Sim #1 : "+sim1Counter);
    }
    else{
        //notificationLabel.setText("Simulation of Motor 1 has reached negative values");
        logThis("Simulation of Motor 1 has reached negative values");
    }
}

```



```

/*
 * walloidGUI.java
 *
 * Created on 06.jan.2011, 19:23:19
 */
import controlPack.*; // Arduino Communication
import java.awt.event.*;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import java.awt.*;
import java.awt.geom.Line2D;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Calendar;
//import javax.swing.Timer;
import javax.swing.JProgressBar;
import javax.swing.UIManager;
import processingGuiPack.*;

/**
 *
 * @author shahabfm_adm
 */
public class WNPGui extends javax.swing.JFrame implements KeyListener {

    //Variables
    WPNArduino arduino;
    String[] COMList;
    boolean connectedToArduino;
    //int sim0Counter = 0, sim1Counter = 0, sim2Counter = 0;
    //int m0stat = -1, m1stat = -1, m2stat = -1; // -1 = stopped motors | 0 = forward | 1 = backward
    //Timer timer;
    Line2D lin;
    URL url;
    //time & date for logging purpose
    Calendar cal = Calendar.getInstance();
    SimpleDateFormat sdf = new SimpleDateFormat();

    /** Creates new form walloidGUI */
    public WNPGui(){
        super("Walloid Control Panel - Java | System Status: Connected");
        initComponents();
        this.setLocation(200, 200);
        setVisible(true);
        if (this.isFocusable() == false){
            this.setFocusable(true);
        }
        else{
            System.out.println("Could not set JFrame Focusable");
        }
        smartyTabbedPanel.addKeyListener(this);
        JLabel[] motorLabels = {motor0StatusLabel, motor1StatusLabel, motor2StatusLabel};
        JLabel[] simLabels = {sim0StatusLabel, sim1StatusLabel, sim2StatusLabel};
        JProgressBar[] progressBar = {progressBar0, progressBar1, progressBar2};
        arduino = new WPNArduino(motorLabels, simLabels, progressBar, errorLabel, logText);
        COMList = arduino.readAllSerialPorts();
        for (int i=0; i<COMList.length; i++){
            comComboBox.insertItemAt(COMList[i], i);
        }
        int tempNum = comComboBox.getItemCount();
        //System.out.println(tempNum);
    }

```

```

        comComboBox.setSelectedIndex(tempNum-2);//selecting the last COM
        arduinoAckLabel.setForeground(Color.red);
        arduinoAckLabel.setText("NOT Connected to Arduino");
        setupHelpPage();
        connectedToArduino = false;
        //timer = new Timer(347, this);
        //timer.setInitialDelay(0);
        //timer.start();
        notificationLabel.setForeground(Color.red);
        //Constructing HelpPane

    }//end of constructor

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
    private void initComponents() {

        smartyTabbedPanel = new javax.swing.JTabbedPane();
        motorStatTabbedPanel = new javax.swing.JPanel();
        motor0StatusLabel = new javax.swing.JLabel();
        motor1StatusLabel = new javax.swing.JLabel();
        motor2StatusLabel = new javax.swing.JLabel();
        exitBtn = new javax.swing.JButton();
        startBtn = new javax.swing.JButton();
        motor0Label1 = new javax.swing.JLabel();
        leftBtnForward = new javax.swing.JButton();
        forwardBtn = new javax.swing.JButton();
        backBtn = new javax.swing.JButton();
        rightBtnForward = new javax.swing.JButton();
        resetBtn = new javax.swing.JButton();
        arduinoAckLabel = new javax.swing.JLabel();
        progressBar0 = new javax.swing.JProgressBar();
        progressBar1 = new javax.swing.JProgressBar();
        progressBar2 = new javax.swing.JProgressBar();
        leftBtnBack = new javax.swing.JButton();
        rightBtnBack = new javax.swing.JButton();
        notificationLabel = new java.awt.Label();
        sim0StatusLabel = new javax.swing.JLabel();
        sim1StatusLabel = new javax.swing.JLabel();
        sim2StatusLabel = new javax.swing.JLabel();
        errorLabel = new javax.swing.JLabel();
        optionsTabbedPanel = new javax.swing.JPanel();
        comComboBox = new javax.swing.JComboBox();
        comLabel = new javax.swing.JLabel();
        baudrateLabel = new javax.swing.JLabel();
        baudrateComboBox = new javax.swing.JComboBox();
        logTabbedPanel = new javax.swing.JPanel();
        logText = new java.awt.TextArea();
        helpTabbedPanel = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        htmlDisplay = new javax.swing.JEditorPane();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent evt) {
                formKeyPressed(evt);
            }
        })
    }

```



```

});

motorStatTabbedPanel.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N

motor0StatusLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
motor0StatusLabel.setForeground(new java.awt.Color(0, 153, 255));
motor0StatusLabel.setText("Motor #0 : 00");

motor1StatusLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
motor1StatusLabel.setForeground(new java.awt.Color(0, 153, 255));
motor1StatusLabel.setText("Motor #1 : 00");

motor2StatusLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
motor2StatusLabel.setForeground(new java.awt.Color(0, 153, 255));
motor2StatusLabel.setText("Motor #2 : 00");

exitBtn.setText("Exit");
exitBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        exitBtnActionPerformed(evt);
    }
});

startBtn.setText("Connect to Arduino");
startBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        startBtnActionPerformed(evt);
    }
});

motor0Label1.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
motor0Label1.setText("— Motor & Simulation Cycle Status");

leftBtnForward.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
leftBtnForward.setText("Lf");
leftBtnForward.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        leftBtnForwardActionPerformed(evt);
    }
});

forwardBtn.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
forwardBtn.setText("F");
forwardBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        forwardBtnActionPerformed(evt);
    }
});

backBtn.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
backBtn.setText("B");
backBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        backBtnActionPerformed(evt);
    }
});

rightBtnForward.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
rightBtnForward.setText("Rf");
rightBtnForward.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        rightBtnForwardActionPerformed(evt);
    }
});

```

```

});
resetBtn.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
resetBtn.setText("RESET");
resetBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        resetBtnActionPerformed(evt);
    }
});

arduinoAckLabel.setForeground(new java.awt.Color(0, 153, 0));

leftBtnBack.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
leftBtnBack.setText("Lb");
leftBtnBack.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        leftBtnBackActionPerformed(evt);
    }
});

rightBtnBack.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
rightBtnBack.setText("Rb");
rightBtnBack.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        rightBtnBackActionPerformed(evt);
    }
});

notificationLabel.setText("...");

sim0StatusLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
sim0StatusLabel.setForeground(new java.awt.Color(102, 102, 255));
sim0StatusLabel.setText("Sim #0 : 00");

sim1StatusLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
sim1StatusLabel.setForeground(new java.awt.Color(102, 102, 255));
sim1StatusLabel.setText("Sim #1 : 00");

sim2StatusLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
sim2StatusLabel.setForeground(new java.awt.Color(102, 102, 255));
sim2StatusLabel.setText("Sim #2 : 00");

errorLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
errorLabel.setForeground(new java.awt.Color(0, 153, 0));
errorLabel.setText("No Error up to now : -)");

javax.swing.GroupLayout motorStatTabbedPanelLayout = new javax.swing.GroupLayout(motorStatTabbedPanel);
motorStatTabbedPanel.setLayout(motorStatTabbedPanelLayout);
motorStatTabbedPanelLayout.setHorizontalGroup(
    motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(motorStatTabbedPanelLayout.createSequentialGroup()
            .add(motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .add(arduinoAckLabel)
                .addComponent(new javax.swing.JLabel("Sim #0 : 00"))
                .addComponent(new javax.swing.JLabel("Sim #1 : 00"))
                .addComponent(new javax.swing.JLabel("Sim #2 : 00"))
                .addComponent(new javax.swing.JLabel("No Error up to now : -)"))
            )
            .addGap(10, 10, 10)
            .add(motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .add(sim0StatusLabel)
                .add(sim1StatusLabel)
                .add(sim2StatusLabel)
            )
            .addGap(10, 10, 10)
            .add(motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .add(leftBtnBack)
                .add(rightBtnBack)
            )
        )
        .add(motorStatTabbedPanelLayout.createSequentialGroup()
            .add(notificationLabel)
            .addGap(10, 10, 10)
            .add(errorLabel)
        )
);

```

```

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(rightBtnBack))
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addGap(27, 27, 27)
        .addComponent(resetBtn)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(forwardBtn, javax.swing.GroupLayout.PREFERRED_SIZE, 49, javax.swing.GroupLayout.Alignment.LEADING)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(leftBtnForward, javax.swing.GroupLayout.PREFERRED_SIZE, 50, javax.swing.GroupLayout.Alignment.LEADING)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(rightBtnForward)))
    .addContainerGap(197, Short.MAX_VALUE))
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addContainerGap()
        .addComponent(startBtn, javax.swing.GroupLayout.DEFAULT_SIZE, 397, Short.MAX_VALUE)
        .addGap(477, 477, 477))
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addContainerGap()
        .addComponent(exitBtn, javax.swing.GroupLayout.DEFAULT_SIZE, 397, Short.MAX_VALUE)
        .addGap(477, 477, 477))
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addContainerGap()
        .addComponent(progressBar0, javax.swing.GroupLayout.DEFAULT_SIZE, 397, Short.MAX_VALUE)
        .addGap(477, 477, 477))
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addContainerGap()
        .addComponent(progressBar1, javax.swing.GroupLayout.DEFAULT_SIZE, 397, Short.MAX_VALUE)
        .addGap(477, 477, 477))
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addContainerGap()
        .addComponent(progressBar2, javax.swing.GroupLayout.DEFAULT_SIZE, 397, Short.MAX_VALUE)
        .addGap(477, 477, 477))
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addContainerGap()
        .addGroup(motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING))
            .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
                .addGap(10, 10, 10)
                .addGroup(motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING))
                    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
                        .addComponent(motor2StatusLabel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(motor0StatusLabel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(motor1StatusLabel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.Alignment.LEADING)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING))
                            .addComponent(sim0StatusLabel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(sim1StatusLabel, javax.swing.GroupLayout.DEFAULT_SIZE, 128, javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(sim2StatusLabel)))
                    .addComponent(errorLabel)))
            .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
                .addComponent(motor0Label1)
                .addGap(138, 138, 138)))
        .addGap(322, 322, 322))
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addContainerGap()
        .addComponent(notificationLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 363, javax.swing.GroupLayout.Alignment.LEADING)
        .addContainerGap(104, Short.MAX_VALUE))
);
motorStatTabbedPanelLayout.setVerticalGroup(
    motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(motorStatTabbedPanelLayout.createSequentialGroup())
        .addGroup(motorStatTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING))

```

```

        .addGroup(motorStatTabbedPanelLayout.createSequentialGroup ())
        .addGap(27, 27, 27)
        .addGroup(motorStatTabbedPanelLayout.createParallelGroup (javax.swing.GroupLayout.Alignment.BASELINE))
        .addComponent(forwardBtn, javax.swing.GroupLayout.PREFERRED_SIZE, 41, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(leftBtnForward, javax.swing.GroupLayout.PREFERRED_SIZE, 40, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(rightBtnForward, javax.swing.GroupLayout.PREFERRED_SIZE, 38, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGroup(motorStatTabbedPanelLayout.createSequentialGroup ())
        .addGap(52, 52, 52)
        .addComponent(resetBtn, javax.swing.GroupLayout.PREFERRED_SIZE, 36, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(9, 9, 9)
        .addGroup(motorStatTabbedPanelLayout.createParallelGroup (javax.swing.GroupLayout.Alignment.BASELINE))
        .addComponent(backBtn, javax.swing.GroupLayout.PREFERRED_SIZE, 41, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(leftBtnBack, javax.swing.GroupLayout.PREFERRED_SIZE, 40, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(rightBtnBack, javax.swing.GroupLayout.PREFERRED_SIZE, 44, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(startBtn)
        .addPreferredGap (javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(exitBtn)
        .addGap(18, 18, 18)
        .addComponent(progressBar0, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap (javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(progressBar1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap (javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(progressBar2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(motor0Label1)
        .addPreferredGap (javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(motorStatTabbedPanelLayout.createParallelGroup (javax.swing.GroupLayout.Alignment.BASELINE))
        .addComponent(motor0StatusLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 17, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(sim0StatusLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 17, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap (javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(motorStatTabbedPanelLayout.createParallelGroup (javax.swing.GroupLayout.Alignment.BASELINE))
        .addComponent(motor1StatusLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 17, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(sim1StatusLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 17, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap (javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(motorStatTabbedPanelLayout.createParallelGroup (javax.swing.GroupLayout.Alignment.BASELINE))
        .addComponent(motor2StatusLabel)
        .addComponent(sim2StatusLabel)
        .addPreferredGap (javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(motorStatTabbedPanelLayout.createParallelGroup (javax.swing.GroupLayout.Alignment.LEADING))
        .addGroup(motorStatTabbedPanelLayout.createSequentialGroup ())
        .addGap(54, 54, 54)
        .addComponent(arduinoAckLabel, javax.swing.GroupLayout.DEFAULT_SIZE, 29, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(32, 32, 32))
        .addGroup(motorStatTabbedPanelLayout.createSequentialGroup ())
        .addComponent(errorLabel)
        .addGap(68, 68, 68)
        .addComponent(notificationLabel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addContainerGap ()))
);

smartyTabbedPanel.addTab("Motors Status", motorStatTabbedPanel);

comLabel.setFont(new java.awt.Font("Tahoma", 1, 14));
comLabel.setText("COM Ports :");

baudrateLabel.setFont(new java.awt.Font("Tahoma", 1, 14));
baudrateLabel.setText("Baud Rate");

baudrateComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "4800", "9600", "14400" }));
baudrateComboBox.setSelectedIndex(7);

```

```

javax.swing.GroupLayout optionsTabbedPanelLayout = new javax.swing.GroupLayout(optionsTabbedPanel);
optionsTabbedPanel.setLayout(optionsTabbedPanelLayout);
optionsTabbedPanelLayout.setHorizontalGroup(
    optionsTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(optionsTabbedPanelLayout.createSequentialGroup()
            .addGap(0)
            .addGroup(optionsTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(comComboBox, javax.swing.GroupLayout.PREFERRED_SIZE, 155, javax.swing.GroupLayout.DEFAULT_SIZE)
                .addComponent(comLabel)
                .addComponent(baudrateLabel)
                .addComponent(baudrateComboBox, javax.swing.GroupLayout.PREFERRED_SIZE, 155, javax.swing.GroupLayout.DEFAULT_SIZE)
            )
            .addGap(312, Short.MAX_VALUE)
        )
);
optionsTabbedPanelLayout.setVerticalGroup(
    optionsTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(optionsTabbedPanelLayout.createSequentialGroup()
            .addGap(0)
            .addComponent(comLabel)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(comComboBox, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(baudrateLabel)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(baudrateComboBox, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE)
            .addGap(405, Short.MAX_VALUE)
        )
);

smartyTabbedPanel.addTab("Options", optionsTabbedPanel);

javax.swing.GroupLayout logTabbedPanelLayout = new javax.swing.GroupLayout(logTabbedPanel);
logTabbedPanel.setLayout(logTabbedPanelLayout);
logTabbedPanelLayout.setHorizontalGroup(
    logTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(logText, javax.swing.GroupLayout.DEFAULT_SIZE, 477, Short.MAX_VALUE)
);
logTabbedPanelLayout.setVerticalGroup(
    logTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(logText, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
);

smartyTabbedPanel.addTab("log", logTabbedPanel);

jScrollPane1.setViewportView(htmlDisplayer);

javax.swing.GroupLayout helpTabbedPanelLayout = new javax.swing.GroupLayout(helpTabbedPanel);
helpTabbedPanel.setLayout(helpTabbedPanelLayout);
helpTabbedPanelLayout.setHorizontalGroup(
    helpTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(helpTabbedPanelLayout.createSequentialGroup()
            .addGap(0)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 457, Short.MAX_VALUE)
            .addGap(0)
        )
);
helpTabbedPanelLayout.setVerticalGroup(
    helpTabbedPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(helpTabbedPanelLayout.createSequentialGroup()
            .addGap(0)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 486, Short.MAX_VALUE)
            .addGap(0)
        )
);

smartyTabbedPanel.addTab("Help", helpTabbedPanel);

```

```

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(smartlyTabbedPanel, javax.swing.GroupLayout.DEFAULT_SIZE, 482, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(smartlyTabbedPanel)
        );

        pack();
    } // </editor-fold> // GEN-END: initComponents

    private void exitBtnActionPerformed(java.awt.event.ActionEvent evt) { // GEN-FIRST: event_exitBtnActionPerformed
        // TODO add your handling code here:
        System.exit(0);
    } // GEN-LAST: event_exitBtnActionPerformed

    private void startBtnActionPerformed(java.awt.event.ActionEvent evt) { // GEN-FIRST: event_startBtnActionPerformed
        // TODO add your handling code here:
        if (connectedToArduino != true) {
            arduino.start(comboBox.getSelectedItemAt().toString(),
                Integer.parseInt(baudrateComboBox.getSelectedItemAt().toString()));
            //arduino.handshakeChecker();
            if (!arduino.problemWhileConnecting()) {
                arduinoAckLabel.setForeground(Color.green);
                arduinoAckLabel.setText("Connected Succesfully to Arduino");
                connectedToArduino = true;
                startBtn.setText("Disconnect from Arduino");
            }
        } else {
            arduinoAckLabel.setForeground(Color.red);
            arduinoAckLabel.setText("Problem connecting to Arduino");
        }
        arduino.handshakeChecker();
        arduino.readCurrentPosition();
        /* if (arduino.handshakeChecker()) {
            //System.out.println("Ack received");
            arduinoAckLabel.setForeground(Color.green);
            arduinoAckLabel.setText("Connected Succesfully to Arduino");
            connectedToArduino = true;
            startBtn.setText("Disconnect from Arduino");
        } */
    } //end of if connected
    else {
        //disconnect from Arduino
        if (arduino.closeSerialPort() == 0) {
            startBtn.setText("Connect to Arduino");
        } else {
            JOptionPane.showMessageDialog(null, "Could not disconnect from Arduino", "Input Error", JOptionPane.ERROR_MESSAGE);
        } //end of else closeSerialPort
    } //end of else
} // GEN-LAST: event_startBtnActionPerformed

@Override
public void paint(Graphics g) {
    //g.setColor(Color.black);
    //g.drawLine(10, 500, 100, 470);
    super.paintComponents(g);
}

```

```

Graphics2D g2 = (Graphics2D) g;
lin = new Line2D.Float(0, 0, 0, 0);
g2.draw(lin);
}

private void forwardBtnActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_forwardBtnActionPerf
    // TODO add your handling code here:
    if (connectedToArduino){
        arduino.sendData("0f");
        arduino.setM0stat(0);
        //ProcSimGui.moveMotorIndx(0);
    }
    else{
        JOptionPane.showMessageDialog(null, "You need to first connect to the Arduino card", "Input Error
    }
    lin = new Line2D.Float(0, 0, 100, 100);
    repaint();
}
//GEN-LAST:event_forwardBtnActionPerformed

private void backBtnActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_backBtnActionPerfor
    // TODO add your handling code here:
    if (connectedToArduino){
        arduino.sendData("0b");
        arduino.setM0stat(1);
    }
    else{
        JOptionPane.showMessageDialog(null, "You need to first connect to the Arduino card", "Input Error
    }
}
//GEN-LAST:event_backBtnActionPerformed

private void leftBtnForwardActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_leftBtnForwa
    // TODO add your handling code here:
    if (connectedToArduino){
        arduino.sendData("1f");
        arduino.setM1stat(0);
    }
    else{
        JOptionPane.showMessageDialog(null, "You need to first connect to the Arduino card", "Input Error
    }
}
//GEN-LAST:event_leftBtnForwardActionPerformed

private void rightBtnForwardActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_rightBtnFor
    // TODO add your handling code here:
    if (connectedToArduino){
        arduino.sendData("2f");
        arduino.setM2stat(0);
    }
    else{
        JOptionPane.showMessageDialog(null, "You need to first connect to the Arduino card", "Input Error
    }
}
//GEN-LAST:event_rightBtnForwardActionPerformed

private void resetBtnActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_resetBtnActionPerf
    // TODO add your handling code here:
    if (connectedToArduino){
        arduino.resetAllMotors();
        arduino.setM0stat(-1);
        arduino.setM1stat(-1);
    }
}

```



```

        arduino.setM2stat(-1);
    }
    else
        JOptionPane.showMessageDialog(null, "Not connected to Arduino card to be able to reset the data",
    }//GEN-LAST:event_resetBtnActionPerformed

    private void formKeyPressed(java.awt.event.KeyEvent evt) {//GEN-FIRST:event_formKeyPressed
        // TODO add your handling code here:
    }//GEN-LAST:event_formKeyPressed

    private void leftBtnBackActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_leftBtnBackActi
        // TODO add your handling code here:
        // TODO add your handling code here:
        if (connectedToArduino){
            arduino.sendData("1b");
            arduino.setM1stat(1);
        }
        else{
            JOptionPane.showMessageDialog(null, "You need to first connect to the Arduino card", "Input Error
        }
    }//GEN-LAST:event_leftBtnBackActionPerformed

    private void rightBtnBackActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_rightBtnBackA
        // TODO add your handling code here:
        if (connectedToArduino){
            arduino.sendData("2b");
            arduino.setM2stat(1);
        }
        else{
            JOptionPane.showMessageDialog(null, "You need to first connect to the Arduino card", "Input Error
        }
    }//GEN-LAST:event_rightBtnBackActionPerformed

    //KeyListener implementatiton
    public void keyTyped(KeyEvent e){
        //displayInfo(e, "Key Typed");
    }

    public void keyPressed(KeyEvent e){
        switch (e.getKeyCode()){
            case (38)://UP
                System.out.println("KEY PRESSED : " + e.getKeyText(e.getKeyCode()) + " and key code is : " + e
                //arduino.sendData('0', 'f');
                break;
            case (37)://RIGHT
                //arduino.sendData('2', 'f');
                System.out.println("KEY PRESSED : " + e.getKeyText(e.getKeyCode()) + " and key code is : " + e
                break;
            case (39)://LEFT
                //arduino.sendData('1', 'f');
                System.out.println("KEY PRESSED : " + e.getKeyText(e.getKeyCode()) + " and key code is : " + e
                break;
            case (40)://DOWN
                //arduino.sendData('0', 'b');
                System.out.println("KEY PRESSED : " + e.getKeyText(e.getKeyCode()) + " and key code is : " + e
                break;
        }
        //end of switch
        //displayInfo(e, "Key Pressed");
    }

    public void keyReleased(KeyEvent e){
        //displayInfo(e, "Key Released");
    }

```

```

}

private void displayInfo(KeyEvent e, String keyStatus){

    //You should only rely on the key char if the event
    //is a key typed event.
    int id = e.getID();
    String keyString;
    if (id == KeyEvent.KEY_TYPED) {
        char c = e.getKeyChar();
        keyString = "key character = '" + c + "'";
    } else {
        int keyCode = e.getKeyCode();
        keyString = "key code = " + keyCode
            + " ("
            + KeyEvent.getKeyText(keyCode)
            + ")";
    }
    int modifiersEx = e.getModifiersEx();
    String modString = "extended modifiers = " + modifiersEx;
    String tmpString = KeyEvent.getModifiersExText(modifiersEx);
    if (tmpString.length() > 0) {
        modString += " (" + tmpString + ")";
    } else {
        modString += " (no extended modifiers)";
    }

    String actionString = "action key? ";
    if (e.isActionKey()) {
        actionString += "YES";
    } else {
        actionString += "NO";
    }

    String locationString = "key location: ";
    int location = e.getKeyLocation();
    if (location == KeyEvent.KEY_LOCATION_STANDARD) {
        locationString += "standard";
    } else if (location == KeyEvent.KEY_LOCATION_LEFT) {
        locationString += "left";
    } else if (location == KeyEvent.KEY_LOCATION_RIGHT) {
        locationString += "right";
    } else if (location == KeyEvent.KEY_LOCATION_NUMPAD) {
        locationString += "numpad";
    } else { // (location == KeyEvent.KEY_LOCATION_UNKNOWN)
        locationString += "unknown";
    }

    //Display information about the KeyEvent...
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    new WNPGui().setVisible(true);
}

// Variables declaration – do not modify//GEN-BEGIN:variables
private javax.swing.JLabel arduinoAckLabel;

```

```

private javax.swing.JButton backBtn;
private javax.swing.JComboBox baudrateComboBox;
private javax.swing.JLabel baudrateLabel;
private javax.swing.JComboBox comComboBox;
private javax.swing.JLabel comLabel;
private javax.swing.JLabel errorLabel;
private javax.swing.JButton exitBtn;
private javax.swing.JButton forwardBtn;
private javax.swing.JPanel helpTabbedPanel;
private javax.swing.JEditorPane htmlDisplayer;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JButton leftBtnBack;
private javax.swing.JButton leftBtnForward;
private javax.swing.JPanel logTabbedPanel;
private java.awt.TextArea logText;
private javax.swing.JLabel motor0Label1;
private javax.swing.JLabel motor0StatusLabel;
private javax.swing.JLabel motor1StatusLabel;
private javax.swing.JLabel motor2StatusLabel;
private javax.swing.JPanel motorStatTabbedPanel;
private java.awt.Label notificationLabel;
private javax.swing.JPanel optionsTabbedPanel;
private javax.swing.JProgressBar progressBar0;
private javax.swing.JProgressBar progressBar1;
private javax.swing.JProgressBar progressBar2;
private javax.swing.JButton resetBtn;
private javax.swing.JButton rightBtnBack;
private javax.swing.JButton rightBtnForward;
private javax.swing.JLabel sim0StatusLabel;
private javax.swing.JLabel sim1StatusLabel;
private javax.swing.JLabel sim2StatusLabel;
private javax.swing.JTabbedPane smartyTabbedPanel;
private javax.swing.JButton startBtn;
// End of variables declaration//GEN-END: variables

//Timer based eventlistener which takes care of the simulation at 347 nSecond
/*public void actionPerformed(ActionEvent e) {
    if (m0stat == 0){
        if ((sim0Counter + 1) <= 100){
            sim0Counter++;
            progressBar0.setValue(sim0Counter);
            sim0StatusLabel.setText("Sim #0 : "+sim0Counter);
        } else{
            notificationLabel.setText("Simulation of Motor 0 has reached MAX values");
            logThis("Simulation of Motor 0 has reached MAX values");
            m0stat = -1;
        }
    }
    else if(m0stat == 1){
        if ((sim0Counter - 1) > 0){
            sim0Counter--;
            progressBar0.setValue(sim0Counter);
            sim0StatusLabel.setText("Sim #0 : "+sim0Counter);
        }
        else{
            notificationLabel.setText("Simulation of Motor 0 has reached negative values");
            logThis("Simulation of Motor 0 has reached negative values");
            m0stat = -1;
        }
    }
}
if (m1stat == 0){
    if ((sim1Counter + 1) <= 100){

```

```

        sim1Counter++;
        progressBar1.setValue(sim1Counter);
        sim1StatusLabel.setText("Sim #1 : "+sim1Counter);
    } else {
        notificationLabel.setText("Simulation of Motor #1 has reached MAX values");
        logThis("Simulation of Motor #1 has reached MAX values");
        m1stat = -1;
    }
}
else if(m1stat == 1){
    if (sim1Counter - 1 > 0){
        sim1Counter--;
        progressBar1.setValue(sim1Counter);
        sim1StatusLabel.setText("Sim #1 : "+sim1Counter);
    }
    else{
        notificationLabel.setText("Simulation of Motor 1 has reached negative values");
        logThis("Simulation of Motor 1 has reached negative values");
        m1stat = -1;
    }
}
if (m2stat == 0){
    if ((sim2Counter + 1) <= 100){
        sim2Counter++;
        progressBar2.setValue(sim2Counter);
        sim2StatusLabel.setText("Sim #2 : "+sim2Counter);
    } else {
        notificationLabel.setText("Simulation of Motor #2 has reached MAX values");
        logThis("Simulation of Motor #2 has reached MAX values");
        m0stat = -1;
    }
}
else if(m2stat == 1){
    if (sim2Counter - 1 > 0){
        sim2Counter--;
        progressBar2.setValue(sim2Counter);
        sim2StatusLabel.setText("Sim #2 : " + sim2Counter);
    }
    else{
        notificationLabel.setText("Simulation of Motor 2 has reached negative values");
        logThis("Simulation of Motor 2 has reached negative values");
        m2stat = -1;
    }
}
}
}*/

private int logThis(String logMsg){
    if (logText != null){
        logText.setText(logText.getText() + "->" + sdf.format(cal.getTime()) + " : " + logMsg + ".\n");
        return 0;
    } //end of if
    else{
        return -1;
    }
}

private int setupHelpPage(){
    try{
        UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        url = new URL("http://walloid.blogspot.com/p/about-walloid.html");
        htmlDisplayer.setEditable(false);
        htmlDisplayer.setPage(url);
    }
}

```



```

        { 0, 0, 0 },
        { 0, 0, 0 }
    };
    float[] destPos = { 250, 250, 250 };

    public ProcSimGui(PApplet p){
        prSimApplet = p;
    } //end of constructor

    int motorIndex = 0;

    public void initialize(){
//        prApplet.stroke(255, 0, 0);
//        prApplet.line(basePos[0][0], basePos[0][1], basePos[0][2], destPos[0], destPos[1], destPos[2]);
//        prApplet.stroke(0, 255, 0);
//        prApplet.line(basePos[1][0], basePos[1][1], basePos[1][2], destPos[0], destPos[1], destPos[2]);
//        prApplet.stroke(0, 0, 255);
//        prApplet.line(basePos[2][0], basePos[2][1], basePos[2][2], destPos[0], destPos[1], destPos[2]);
//        prApplet.fill(255, 0, 0);
//        prApplet.ellipse(250,250, 15, 15);
        drawLines();
        d1 = prSimApplet.dist(basePos[0][0], basePos[0][1], basePos[0][2], destPos[0], destPos[1], destPos[2]);
        d2 = prSimApplet.dist(basePos[1][0], basePos[1][1], basePos[1][2], destPos[0], destPos[1], destPos[2]);
        d3 = prSimApplet.dist(basePos[2][0], basePos[2][1], basePos[2][2], destPos[0], destPos[1], destPos[2]);
        System.out.println("d1 = " + d1);
        System.out.println("d2 = " + d2);
        System.out.println("d3 = " + d3);
        //System.out.println("d3 = " + prSimApplet.dist(basePos[2][0], basePos[2][1], basePos[2][2], destPos[0], destPos[1], destPos[2]));
        float[] temp = {255,250,260};
        //prApplet.ellipse(150, 30, 150, 10, 10);
        drawMotors();
    } //end of initialize

    public void display(){
        drawMotors();
    } //end of display

    public int simulateJoints(){

        return 0;
    } //end of simulateJoints

    private void drawMotors(){
//MOTOR 0
        prSimApplet.fill(250,250,150);//Yellow
        prSimApplet.stroke(1);
        prSimApplet.ellipse(mXVal, mYVal, 20, 20);
        prSimApplet.fill(100,200,100);//Green
        prSimApplet.rect(mXVal, mYVal-25, 100, 50);
        prSimApplet.fill(100,200,100);//Green
        prSimApplet.rect(mXVal+100, mYVal-10, 50, 20);
// *****
//Here comes the prismatic joint
        prSimApplet.fill(255,230,0); //Strong YELLOW
        prSimApplet.rect(mXVal+150, mYVal-3, motorIndex, 8);
// *****
        prSimApplet.fill(250,250,150);//Yellow
        prSimApplet.ellipse(mXVal+200, mYVal, 20, 20);
        prSimApplet.fill(100,200,100);//Green
        prSimApplet.rect(mXVal+150, mYVal-15, 40, 30);
//MOTOR 1
        prSimApplet.fill(250,250,150);//Yellow

```

```

        prSimApplet.stroke(1);
        prSimApplet.ellipse(mXVal, mYVal + 100, 20, 20);
        prSimApplet.fill(44,67,129); //Dark Blue
        prSimApplet.rect(mXVal, mYVal+75, 100, 50);
        prSimApplet.fill(44,67,129); //Dark Blue
        prSimApplet.rect(mXVal+100, mYVal+90, 50, 20);
        // *****
        //Here comes the prismatic joint
        prSimApplet.fill(255,230,0); //Strong YELLOW
        prSimApplet.rect(mXVal+150, mYVal+97, motorIndex, 8);
        // *****
        prSimApplet.fill(250,250,150); //Yellow
        prSimApplet.ellipse(mXVal+200, 2*mYVal, 20, 20);
        prSimApplet.fill(44,67,129); //Dark Blue
        prSimApplet.rect(mXVal+150, mYVal+85, 40, 30);
        //MOTOR 2
        prSimApplet.fill(250,250,150); //Yellow
        prSimApplet.stroke(1);
        prSimApplet.ellipse(mXVal, mYVal+200, 20, 20);
        prSimApplet.fill(206,41,0); //RED
        prSimApplet.rect(mXVal, mYVal+175, 100, 50);
        prSimApplet.fill(206,41,0); //RED
        prSimApplet.rect(mXVal+100, mYVal+190, 50, 20);
        // *****
        //Here comes the prismatic joint
        prSimApplet.fill(255,230,0); //Strong YELLOW
        prSimApplet.rect(260, 297, motorIndex, 8);
        // *****
        prSimApplet.fill(250,250,150); //light Yellow
        prSimApplet.ellipse(mXVal+200, mYVal+200, 20, 20);
        prSimApplet.fill(206,41,0); //RED
        prSimApplet.rect(mXVal+150, mYVal+185, 40, 30);
    } //end of drawMotors

    private void drawLines(){
        prSimApplet.background(0);
        prSimApplet.stroke(255, 0, 150);
        for (int i = 0; i < 3; i++){
            prSimApplet.line(basePos[i][0], basePos[i][1], basePos[i][2], destPos[0], destPos[1],
            destPos[2]);
        }
    } //end of drawLines

    private void changeTargetPoint(float[] target){
        destPos = target;
    }

    public boolean moveMotorIndx(int i){
        float tempDist = prSimApplet.dist(basePos[i][0], basePos[i][1], basePos[i][2], destPos[0]+1,
        destPos[1], destPos[2]);
        if ( (tempDist > 350) && (tempDist < 540) ){
            prSimApplet.line(basePos[i][0], basePos[i][1], basePos[i][2], ++destPos[0], ++destPos[1],
            ++destPos[2]);
            return true;
        }
        return false;
    } //end of movetoXYZ

    /*private int ifOkayToMove(float[] target){
        float[] dist = {0, 0, 0};
        dist[0] = prSimApplet.dist(basePos[0][0], basePos[0][1], basePos[0][2], target[0], target[1],
        target[2]);
        dist[1] = prSimApplet.dist(basePos[1][0], basePos[1][1], basePos[1][2], target[0], target[1],
        target[2]);
        dist[2] = prSimApplet.dist(basePos[2][0], basePos[2][1], basePos[2][2], target[0], target[1],
        target[2]);
        for (int i=0; i < 3; i++){
            if ( (dist[i] > 350) && (dist[i] < 540) ){

```



```

        changeTargetPoint(target);
        drawLines ();
        return 0;
    }
}
return -1;
} //end of ifOkayToMove

public boolean moveToXYZ(float[] target){
    if (ifOkayToMove(target) == 0){
        return true;
    } else {
        return false;
    }
} //end of movetoXYZ

public boolean moveMotorIndx(int i){
    float tempDist = prSimApplet.dist(basePos[i][0], basePos[i][1], basePos[i][2], destPos[0]+1, destPos[1]+1, destPos[2]+1);
    if ( (tempDist > 350) && (tempDist < 540) ){
        float tempTarget[] = {destPos[0]+1, destPos[1]+1, destPos[2]+1};
        changeTargetPoint(tempTarget);
        drawLines ();
        return true;
    }
    return false;
} //end of movetoXYZ
*/
} //end of class ProcSimGui

package processingGuiPack;

import processing.core.PApplet;

public class ProcMonGui extends PApplet {

    //Variables
    //PApplet prMonApplet;

    public ProcMonGui(){
        //prMonApplet = p;
        //PApplet.main(new String[] { "--present", "MyProcessingSketch" });
        System.out.println("HELLOOOOOO PROCESSING");
        this.setup();
    } //end of constructor

    public void setup(){
        this.size(100, 100);
        this.background(0);
    }

}

```

D.2 Control algorithm, Arduino C

```

// ****
// ****
// ****
// ****
// **** ARDUINO
****

```

```
//**** Master Controller
#include <EEPROM.h>
#include <MATH.h>
#include <avr/power.h>
#include <avr/sleep.h>
#include <SD.h>

//Variables
boolean enORdis;
int readValue = 0;
String readValueTemp = "";
boolean manual;
boolean auto_;
boolean teach;
boolean findCurrentPosition = false;
boolean sleepPolicy = false;
boolean feedbackRecieved = true;
File logFile , restoreFile;
String automationFromRestoreLog[40] = {"@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!", "@1123123123!"};
int automationCounter = 0;
int automationRunningCounter = 0;

struct COOR_POINTS {
    byte cnt0;
    byte cnt1;
    byte cnt2;
    COOR_POINTS *nextPoint;
};

typedef COOR_POINTS coordinates;

void setup(){
    Serial.begin(19200);
    Serial1.begin(19200);
    Serial2.begin(19200);
    Serial3.begin(19200);
    Serial.println("SETUP MEGA");
    //Serial1.write("@11!");
    //Serial1.print("@11!");
    //Serial1.write("@11!");
    pinMode(8, OUTPUT);
    for (int i=0; i<400; i++){
        //Serial.print("\n"@1123123123!\n",");
    }

}

//end of setup

void loop(){
    checkSerial();
    checkSensors();
    //Serial.println("Processing");
    //delay(1000);
```

```

} //end of loop

//@(boardNumber(0-4)-1 digits )(CounterNumbers(0-207)-9 digits )(PWM_Speed(0-255)3 digits )!

void checkSerial(){
  //variables
  String tempStr0 = "";
  String tempStr1 = "";
  String tempStr2 = "";
  String tempStr3 = "";
  // Serial.println("TEST1");
  if ( Serial.available() > 0 ){
    readValue = Serial.read();
    //Serial.println(".");
    if ((char)readValue == '@'){
      while (readValue != '!'){
        readValue = Serial.read();
        digitalWrite(13, HIGH);
        if ((char)readValue != '!' && readValue > 32 && readValue < 125){
          tempStr0 = tempStr0 + (char) readValue;
        }
      } //end of if !
    } //end of while
    //Serial.println("it is");
    //Serial.println(tempStr0);
    digitalWrite(13, LOW);
  } //end of if
} //end of if Serial0

// *****

if ( Serial1.available() > 0 ){
  readValue = Serial1.read();
  if ((char)readValue == '@'){
    while (readValue != '!'){
      readValue = Serial1.read();
      digitalWrite(13, HIGH);
      if ((char)readValue != '!' && readValue > 32 && readValue < 125){
        tempStr1 = tempStr1 + (char) readValue;
      }
    } //end of if !
  } //end of while
  digitalWrite(13, LOW);
} //end of if
} //end of if Serial1

// *****

if ( Serial2.available() > 0 ){
  readValue = Serial2.read();
  if ((char)readValue == '@'){
    while (readValue != '!'){
      readValue = Serial2.read();
      digitalWrite(13, HIGH);
      if ((char)readValue != '!' && readValue > 32 && readValue < 125){
        tempStr2 = tempStr2 + (char) readValue;
      }
    } //end of if !
  } //end of while
  digitalWrite(13, LOW);
} //end of if
} //end of if Serial2

// *****

```

```

    if ( Serial3.available() > 0 ){
        readValue = Serial3.read();
        if ((char)readValue == '@'){
            while (readValue != '!'){
                readValue = Serial3.read();
                digitalWrite(13, HIGH);
                if ((char)readValue != '!' && readValue > 32 && readValue <125){
                    tempStr3 = tempStr3 + (char) readValue;
                }
            }
            digitalWrite(13, LOW);
        }
    }
}

// *****

//Serial.print("read");
//if(tempStr0 != "") Serial.print(tempStr0);

//processing the incoming
if (tempStr0 != ""){
    processCenter(tempStr0);
    if (teach == true)
        logThis(tempStr0);
}
if (tempStr1 != ""){
    processCenter(tempStr1);
    if (teach == true)
        logThis(tempStr1);
}
if (tempStr2 != ""){
    processCenter(tempStr2);
    if (teach == true)
        logThis(tempStr2);
}
if (tempStr3 != ""){
    processCenter(tempStr3);
    if (teach == true)
        logThis(tempStr3);
}

if (auto_ == true){
    automationProcess();
}

//restarting the queue
tempStr0 = "";
tempStr1 = "";
tempStr2 = "";
tempStr3 = "";

}

// *****

//**** set to sleep

void sendSleepSigna(int i){
    if (i == 1)
        Serial1.write('S');
}

```

```

        else if (i==2)
            Serial2.write('S');
        else if (i==3)
            Serial3.write('S');
    }

void PSO_SleepController(){
    /* The 5 sleeping choices:
    *     SLEEP_MODE_IDLE           - LEAST power savings | HIGHEST availability
    *     SLEEP_MODE_ADC
    *     SLEEP_MODE_PWR_SAVE
    *     SLEEP_MODE_STANDBY
    *     SLEEP_MODE_PWR_DOWN      - HIGHEST power savings | LEAST availability
    */

    // Setting to sleep mode, although this one save least energy but allows us to easily bring the board back
    set_sleep_mode(SLEEP_MODE_IDLE);
    sleep_enable();
    // Disabling all other parts to save extra energy
    power_adc_disable();
    power_spi_disable();
    power_timer0_disable();
    power_timer1_disable();
    power_timer2_disable();
    power_twi_disable();
    sleep_mode();
    //DEVICE GOES TO SLEEP NOW!
    // Zzzzzzzzz
    sleep_disable(); // Waking up, everything back online and now and continue running the code from this exact
    power_all_enable();
}

void automationProcess(){
    //automationFromRestoreLog[automationRunningCounter];
    if (automationRunningCounter < 40 && feedbackRecieved){
        Serial.print(automationFromRestoreLog[automationRunningCounter]);
        automationRunningCounter++;
        sleepPolicy = true;
        feedbackRecieved = false;
    }
    else{
        Serial.print("Automation is over - ");
        Serial.println(automationRunningCounter);
    }
}

void logThis(String tempStr){
    String tempTag = "";

    if (SD.begin(8)){
        for (int i = 0; i < tempStr.length(); i++){// more than 10 char would be tags
            tempTag = tempTag + tempStr[i];
        } //end of for
        String fileName = "local_knowledge_DB"+tempTag+".txt";
        logFile = SD.open("test.txt", FILE_WRITE); // APPENDS

        if (logFile) {//writing to log file
            logFile.println(tempStr);
            logFile.close();
        } else {

```

```

        Serial.print("E05");
    } //end of if
    } else {
        Serial.print("E06");
        return;
    } //end of else
} //end of logthis

// *****

void restoreKnowledge(String tempTag){
    automationCounter = 0;
    automationRunningCounter = 0;
    if (SD.begin(8)) {
        String fileName = "local_knowledge_DB"+tempTag+".txt ";
        restoreFile = SD.open("test.txt");
        if (restoreFile){
            while (restoreFile.available()){
                if (automationCounter < 4600){
                    automationFromRestoreLog[automationCounter] = (String)restoreFile.read();
                    automationCounter++;
                } //end of 10000 limitation
            } //end of while
            restoreFile.close();
        } else {
            Serial.print("E07");
            return;
        }
    } else {
        Serial.print("E07");
        return;
    }
}

// *****

void checkSensors(){
    // Conceptual based on available sensors onboard the robot
    // ANALOGS
    // int value0 = analogRead(Gyroscope);
    // int value1 = analogRead(distanceMeter);
    // int value2 = analogRead(thermoMeter);
    // int value3 = analogRead(Rfeed);
    // DIGITAL
    // int value2 = digitalRead(encoder);
    // int value3 = analogRead(lightForkSensor);
    // ProcessSensor(int, int, double, double, tag);
} //end of checkSensors

void processCenter(String tempStr){
    //Serial.println("TEST2");
    if (tempStr[0] != '0'){
        readValueTemp = "@" + readValueTemp + "!";
        switch (readValueTemp[2]){
            case '1':
                Serial1.println(readValueTemp);
                sendSleepSigna(2);
                sendSleepSigna(3);
            }
    }
}

```

```

        break;
    case '2':
        Serial2.println(readValueTemp);
        sendSleepSigna(1);
        sendSleepSigna(3);
        break;
    case '3':
        Serial3.println(readValueTemp);
        sendSleepSigna(1);
        sendSleepSigna(2);
        break;
    //case '4':
    //Serial4.println(readValueTemp);
    //break;
} //end of switch
}
else{
    switch (tempStr[1]){
        case 'm' ://Manual mode
            manual = true;
            auto_ = false;
            Serial.print("Manual");
            break;
        case 'a'://Auto mode
            auto_ = true;
            manual = false;
            Serial.print("Auto");
            break;
        case 't'://Teach mode
            auto_ = false;
            manual = true;
            teach = true;
            Serial.print("Teach");
            break;
        case 'h' : //handshake
            Serial.print("h");
            delay(50);
            break;
        //READING CURRENT STORED POSITION
        case 'P' : //find the current position
            findCurrentPosition = true;
            delay(50);
            break;
        case 'f':
            //feedback from Slave controllers, send it back to server
            feedbackRecieved = true;
            Serial.println(tempStr);
            feedbackProcessing(tempStr);
        case 'l':
            //Upload log to the server
            break;
        case 'o':
            //go offline
            break;
        default :
            Serial.println("\n***E00 - Wrong input***");
            //Serial.print("tempStr is :");
            //Serial.print(tempStr);
            //Serial.println(" ***");
            break;
    } //end of switch
    tempStr = "";
}

```



```

        } //end of if related to Mega
    } //end of processCenter

    void feedbackProcessing (String tempFeed){
    } //end of feedbackProcessing

    void feedbackMoveJ(String tempStr){
        //Serial.println("feedbackMoveJ");
        Serial.println(tempStr);
    }

    void feedbackBoardAct(String tempStr){
        //Serial.println("feedbackBoardAct");
        Serial.println(tempStr);
    }

    void feedbackError(String tempStr){
        //Serial.println("feedbackError");
        Serial.println(tempStr);
    }

    void processMoveOrder(String tempStr){
        int boardNumber = -1;
        int motorSpeed = 0;
        int counter[3] = {
            0,0,0 };
        //Serial.println("MOVEJ");
        if (tempStr.length() == 13){
            //boardNumber = atoi(tempChar);
            boardNumber = tempStr[0] - 48;
            //Serial.print("Board Number is : ");
            //Serial.println(boardNumber);
            // *tempChar = tempStr[1] + tempStr[2] + tempStr[3];
            counter[0] = 100*(tempStr[1] - 48) + 10*(tempStr[2] - 48) + (tempStr[3] - 48);
            //Serial.print("Counter 0 is : ");
            //Serial.println(counter[0]);
            counter[1] = 100*(tempStr[4] - 48) + 10*(tempStr[5] - 48) + (tempStr[6] - 48);
            //Serial.print("Counter 1 is : ");
            //Serial.println(counter[1]);
            counter[2] = 100*(tempStr[7] - 48) + 10*(tempStr[8] - 48) + (tempStr[9] - 48);
            //Serial.print("Counter 2 is : ");
            //Serial.println(counter[2]);
            motorSpeed = 100*(tempStr[10] - 48) + 10*(tempStr[11] - 48) + (tempStr[12] - 48);
            //Serial.print("Motor Speed : ");
            //Serial.println(motorSpeed);
            //Serial1.write("@1234567899999!");
            MoveJ(motorSpeed, counter);
        } //end of if
        else Serial.println("E01 - Wrong length of tempStr");
    } //end of processMoveOrder

    void MoveJ(int _mSpeed, int _counter[]){
        //digitalWrite(pinDir[0], HIGH);
        //digitalWrite(pinPwm[0], HIGH);
        turnOnLED();
        delay(2000);
        turnOffLED();
        delay(2000);
        turnOnLED();
    }

```

```

    delay(2000);
    turnOffLED();
    delay(2000);
    turnOnLED();
    delay(2000);
    turnOffLED();
} //end of Move]

void processEnDisOrder(String tempStr){
    int boardNumber = -1;
    if (tempStr.length() == 2){
        boardNumber = tempStr[0] - 48;
        //Serial.print("Board Number is : ");
        //Serial.println(boardNumber);
        if ((tempStr[1] - 48) == 0){
            enORdis = false;
            //Serial.println("DISABLED");
            enableDisable(false);
            //Serial1.write("@10!");
        }
        else if ((tempStr[1] - 48) == 1){
            enORdis = true;
            //Serial.println("ENABLED");
            enableDisable(true);
            //Serial1.write("@11!");
        } //END OF ELSE-IF
    } //end of if
    else Serial.println("E01 - Wrong length of tempStr");
} //end of processEnDisOrder

void enableDisable(boolean _flag){
    if (_flag){
        analogWrite(52, 0); //R
        analogWrite(50, 255); //B
        analogWrite(48, 255); //G
    }
    else {
        analogWrite(52, 255); //R
        analogWrite(50, 255); //B
        analogWrite(48, 255); //G
    }
} //end of enableDisable

void turnOnLED(){
    analogWrite(46, 0); //R
    analogWrite(44, 0); //B
    analogWrite(42, 255); //G
} //end of turnOnLED

void turnOffLED(){
    analogWrite(46, 255); //R
    analogWrite(44, 255); //B
    analogWrite(42, 255); //G
} //end of turnOnLED

//**** Slave Controller

#include <EEPROM.h>
#include <MATH.h>

```

```

#include <avr/power.h>
#include <avr/sleep.h>

byte pinEncA[] = {
  3,5,7}; // digital inputs
byte pinEncB[] = {
  2,4,6}; // digital inputs

// MOTOR OUTPUTS
byte pinDir[] = {
  13,8,12}; // digital output, controls direction for motor nr. [x]
byte pinPwm[] = {
  10,11,9}; // analog output, controls motor speed, nr. [x]

  safetyAllow = false;

struct COOR_POINTS {
  byte cnt0;
  byte cnt1;
  byte cnt2;
  COOR_POINTS *nextPoint;
};
typedef COOR_POINTS coordinates;

//Global variables
int value;
int readValue;
int pwmValue;
int motorDir;
boolean inMotion = false;
//int channelA = 2;
//int channelB = 3;
int cycle[] = {0, 0, 0};
int goal[] = {0, 0, 0};
double a0 = 150.00; // measure system is millimeter(mm) and
//Max opening of one actuator 45 mm -> 45/217 -> 0.20737327188940092165898617511521 = 0.2073 mm for every cou
double b0 = 140.00;

double dist0 = 190.00;
double origo[] = {0.00, 0.00, 0.00};

int cnt0goal=0,cnt1goal=0,cnt2goal=0;

double motor[3][3] = {
  {((-3.00/8.00)*a0), ((-1*sqrt(3)/8.00)*a0), 0.00},
  {(sqrt(3)/4.00)*a0, 0.00, 0.0},
  {(3.00/8.00)*a0), ((-1*sqrt(3)/8.00)*a0), 0.00}
};
//double motor0[] = {(-1*(3/8)*a0), (-1*(sqrt(3)/8)*a0), 0.00}; //Mi
//double motor1[] = {(sqrt(4)/4)*a0, 0.00, 0.00}; //Mj
//double motor2[] = {(3/8)*a0, -1*(sqrt(3)/8)*a0, 0.00}; //Mk

boolean f = false, b = false, r = false, l = false, manulaSteering = false;
int cyclecounter[] = {
  -1, -1, -1};
int cyclecounterold[] = {
  -1, -1, -1};
//13 , 10 right, down
//8 , 11 middle, top
//9, 12 left, down

void setup(){

```

```

Serial.begin(19200);
//Serial.print("WELCOME TO C1S0");
for (int i=0; i < 3; i++){
  pinMode(pinDir[i], OUTPUT);
  pinMode(pinPwm[i], OUTPUT);
  //in case, just to be sure ...
  digitalWrite(pinPwm[i], LOW);
  // just in case to be sure
  digitalWrite(pinDir[i], LOW);
  pinMode(pinEncA[i], INPUT);
  pinMode(pinEncB[i], INPUT);
  digitalWrite(pinEncA[i], LOW);
  digitalWrite(pinEncB[i], LOW);
  //cyclecounter[i] = -1;
  //cyclecounterold[i] = -1;
  //safetyAllow = true;
} //end of for
findCurrentPosition(false);
} //end of setup()

void loop(){
  commandCenter();
  checkEncoder();
  if (inMotion == true){
    runMotors();
  }
} //end of loop

void commandCenter(){
  //variables
  int temp = 0;
  String tempStr = "";

  if ( Serial.available() > 0 ){
    readValue = Serial.read();
    //delay(50);
    if (readValue == '@'){
      while (readValue != '!'){
        readValue = Serial.read();
        digitalWrite(13, HIGH);
        if ((char)readValue != '!' && readValue > 32 && readValue < 125){
          tempStr = tempStr + (char) readValue;
        } //end of if !
      } //end of while
      switch (tempStr[1]){
        case 'h' : //handshake
          Serial.print("@0h!");
          delay(50);
          break;
        case 'P' : //find the current position
          findCurrentPosition(true);
          delay(50);
          break;
        case 'g' :
          if (moveToXYZ(48, 55, 47)){
            Serial.println("\nTRUE returned from moveToXYZ");
          } else{
            Serial.println("\nFALSE returned from moveToXYZ");
          }
          break;
      }
    }
  }
}

```

```

case 'a': //1a123123123

    cnt0goal = (tempStr[2]-48) * 100 + (tempStr[3]-48) * 10 + (tempStr[4]-48);
    cnt1goal = (tempStr[5]-48) * 100 + (tempStr[6]-48) * 10 + (tempStr[7]-48);
    cnt2goal = (tempStr[8]-48) * 100 + (tempStr[9]-48 * 10) + (tempStr[10]-48);
    inMotion = true;
    Serial.println("-----");
    Serial.println(tempStr);
    runMotors();
    break;

case 'S':
    if (safetyAllow == true)
        PSO_SleepController();
    break;
case '0' ://motor #0
    int temp__;
    temp__ = 0;
    if (Serial.available() < 0)
        ;
    delay(500);
    temp__ = Serial.read();
    if (temp__ == 102){// -f -> forward
        //Serial.print("*** Right forward ***");
        digitalWrite(pinDir[0], LOW);
        digitalWrite(pinPwm[0], HIGH);
    }//end of if
    else if (temp__ == 98){// -b -> backward
        //Serial.print("*** Right backward ***");
        digitalWrite(pinDir[0], HIGH);
        digitalWrite(pinPwm[0], HIGH);
    }
    else{
        Serial.println("E00");
        //Serial.print(temp__);
    }
    f = false;
    b = false;
    l = true;
    r = false;
    break;
case '1' ://motor #1
    int temp_;
    if (Serial.available() < 1)
        ;
    delay(500);
    temp_ = Serial.read();
    if (temp_ == 102){// -f
        //Serial.print("*1FW*");
        digitalWrite(pinDir[1], LOW);
        digitalWrite(pinPwm[1], HIGH);
    }//end of if
    else if (temp_ == 98){// -b
        //Serial.print("*1BW*");
        digitalWrite(pinDir[1], HIGH);
        digitalWrite(pinPwm[1], HIGH);
    }
    else{
        Serial.println("E00");
        //Serial.print(temp_);
    }
}

```

```

        f = true;
        b = false;
        l = false;
        r = false;
        break;
    case '2' : //l -> left
        temp_ = 0;
        if (Serial.available() < 1)
            ;
        delay(500);
        temp_ = Serial.read();
        if (temp_ == 102){ //motor #2
            //Serial.print("*2FW*");
            digitalWrite(pinDir[2], LOW);
            digitalWrite(pinPwm[2], HIGH);
        } //end of if
        else if (temp_ == 98){ // -b
            //Serial.print("*2BW*");
            digitalWrite(pinDir[2], HIGH);
            digitalWrite(pinPwm[2], HIGH);
        }
        else{
            Serial.println("E00");
            //Serial.print(temp_);
        }
        f = false;
        b = false;
        l = false;
        r = true;
        break;
    case 109 : //m->turning on and off manual steering
        manulaSteering = !manulaSteering; //turning manual steering off and on by pressing m
        sendFeedbackManualSteering();
        break;
    case 82 : //R->RESET
        Serial.println("R");
        resetAll();
        if (manulaSteering){ //if manual steering is on, a Reset will set everything to ZERO
            EEPROM.write(0, 0);
            EEPROM.write(1, 0);
            EEPROM.write(2, 0);
            findCurrentPosition(false);
        }
        f = false;
        b = false;
        l = false;
        r = false;
        break;
    } //end of switch
}
/*
if (f || b || r || l){
    Serial.print("cycle[i]: ");
    convertToThreeDigits(cycle[i]);
} //end of if
*/
}
} //end of commandCenter

//int angleMaalt;

void runMotors(){

```

```

/*Serial.print("cnt0goal");
Serial.println(cnt0goal);
Serial.print("cycle");
Serial.println(cycle[0]);*/

if (cnt0goal != cycle[0]){
    if (cnt0goal > cycle[0]){ //motor #2
        //Serial.print("*2FW*");
        digitalWrite(pinDir[0], LOW);
        digitalWrite(pinPwm[0], HIGH);
    } //end of if
    else if (cnt0goal < cycle[0]){ // -b
        //Serial.print("*2BW*");
        digitalWrite(pinDir[0], HIGH);
        digitalWrite(pinPwm[0], HIGH);
    }
}

if (cnt1goal != cycle[1]){
    if (cnt1goal > cycle[1]){ //motor #2
        //Serial.print("*2FW*");
        digitalWrite(pinDir[1], LOW);
        digitalWrite(pinPwm[1], HIGH);
    } //end of if
    else if (cnt1goal < cycle[1]){ // -b
        //Serial.print("*2BW*");
        digitalWrite(pinDir[1], HIGH);
        digitalWrite(pinPwm[1], HIGH);
    }
}

if (cnt2goal != cycle[2]){
    if (cnt2goal > cycle[2]){ //motor #2
        //Serial.print("*2FW*");
        digitalWrite(pinDir[2], LOW);
        digitalWrite(pinPwm[2], HIGH);
    } //end of if
    else if (cnt2goal < cycle[2]){ // -b
        //Serial.print("*2BW*");
        digitalWrite(pinDir[2], HIGH);
        digitalWrite(pinPwm[2], HIGH);
    }
}

if (cnt0goal == cycle[0] && cnt1goal == cycle[1] && cnt2goal == cycle[2] ){
    inMotion = false;
    Serial.print("@0fsu!");
}

}

void PSO_SleepController(){
    /* The 5 sleeping choices:
    * SLEEP_MODE_IDLE          - LEAST power savings | HIGHEST availability
    * SLEEP_MODE_ADC
    * SLEEP_MODE_PWR_SAVE
    * SLEEP_MODE_STANDBY
    * SLEEP_MODE_PWR_DOWN     - HIGHEST power savings | LEAST availability
    */

    // Setting to sleep mode, although this one save least energy but allows us to easily bring the board back
    set_sleep_mode(SLEEP_MODE_IDLE);
    sleep_enable();

```



```

// Disabling all other parts to save extra energy
power_adc_disable();
power_spi_disable();
power_timer0_disable();
power_timer1_disable();
power_timer2_disable();
power_twi_disable();
sleep_mode();
//DEVICE GOES TO SLEEP NOW!
// Zzzzzzzzz
sleep_disable(); // Waking up, everything back online and now and continue running the code from this exact
power_all_enable();
}

void checkEncoder(){
  int A;
  int B;
  int i;
  for (i = 0; i < 3; i++){
    cyclecounterold[i] = cyclecounter[i];
    A = digitalRead(pinEncA[i]);
    B = digitalRead(pinEncB[i]);

    if ( (A==LOW) && (B==LOW) )
      cyclecounter[i] = 0;
    else if ( (A==LOW) && (B==HIGH) )
      cyclecounter[i] = 1;
    else if ( (A==HIGH) && (B==HIGH) )
      cyclecounter[i] = 2;
    else if ( (A==HIGH) && (B==LOW) )
      cyclecounter[i] = 3;

    switch (cyclecounter[i]){
      case (0) :
      {
        if ( cyclecounterold[i] == 0){
          //Do nothing
        }
        else if ( cyclecounterold[i] == 1){
          if (ifReachedHighBorder(cycle[i])){
            criticalSituation(i, 1);
          }
          else{
            cycle[i]++;
            saveCurrentPosition(i);
            Serial.print("C");
            Serial.print(i);
            Serial.print("*");
            convertToThreeDigits(cycle[i]);
          }
        }
        else if ( cyclecounterold[i] == 2){
          Serial.println("E01");// Error reading from Encoder-
        }
        else if ( cyclecounterold[i] == 3){
          if (ifReachedLowBorder(cycle[i])){
            criticalSituation(i, 0);
          }
          else{

```

```

        cycle[i]--;
        saveCurrentPosition(i);
        Serial.print("C");
        Serial.print(i);
        Serial.print("*");
        convertToThreeDigits ( cycle[i] );
    }
}
else
    ;// Serial.print("ERROR");
    break;
} //end of case -0
case (1) :
{
    if ( cyclecounterold[i] == 0){
        if (ifReachedLowBorder(cycle[i])){
            criticalSituation(i, 0);
        }
        else{
            cycle[i]--;
            saveCurrentPosition(i);
            Serial.print("C");
            Serial.print(i);
            Serial.print("*");
            convertToThreeDigits ( cycle[i] );
        }
    }
    if ( cyclecounterold[i] == 2){
        if (ifReachedHighBorder(cycle[i])){
            criticalSituation(i, 1);
        }
        else{
            cycle[i]++;
            saveCurrentPosition(i);
            Serial.print("C");
            Serial.print(i);
            Serial.print("*");
            convertToThreeDigits ( cycle[i] );
        }
    }
    else if ( cyclecounterold[i] == 3){
        Serial.println("E01");// Error reading from Encoder-
    }
    else
        ;// Serial.print("ERROR");
        break;
}
case (2) :
{
    if ( cyclecounterold[i] == 1){
        if (ifReachedLowBorder(cycle[i])){
            criticalSituation(i, 0);
        }
        else{
            cycle[i]--;
            saveCurrentPosition(i);
            Serial.print("C");
            Serial.print(i);
            Serial.print("*");
            convertToThreeDigits ( cycle[i] );
        }
    }
}
}

```

```

else if ( cyclecounterold[i] == 2)
;
else if ( cyclecounterold[i] == 3){
    if (ifReachedHighBorder(cycle[i])){
        criticalSituation(i, 1);
    }
    else{
        cycle[i]++;
        saveCurrentPosition(i);
        Serial.print("C");
        Serial.print(i);
        Serial.print("*");
        convertToThreeDigits ( cycle[i] );
    }
}
else if ( cyclecounterold[i] == 0){
    Serial.println("E01");// Error reading from Encoder-
}
else
    ;// Serial.print("ERROR");
break;
} //end of case-2
case(3) :
{
    if (cyclecounterold[i] == 0){
        if (ifReachedHighBorder(cycle[i])){
            criticalSituation(i, 1);
        }
        else{
            cycle[i]++;
            saveCurrentPosition(i);
            Serial.print("C");
            Serial.print(i);
            Serial.print("*");
            convertToThreeDigits ( cycle[i] );
        }
    }
    else if (cyclecounterold[i] == 1){
        Serial.println("E01");// Error reading from Encoder-
    }
    else if ( cyclecounterold[i] == 2){
        if (ifReachedLowBorder(cycle[i])){
            criticalSituation(i, 0);
        }
        else{
            cycle[i]--;
            saveCurrentPosition(i);
            Serial.print("C");
            Serial.print(i);
            Serial.print("*");
            convertToThreeDigits ( cycle[i] );
        }
    }
    else if ( cyclecounterold[i] == 3)
;
    else
        ;// Serial.print("ERROR");
        break;
} //end of case-3
}
} //end of for
} //end of checkEncoder()

```

```

int criticalSituation(int motorNumber, int type){
    digitalWrite (pinPwm[motorNumber], LOW); // Setting all PWM's to LOW
    if (type == 0){
        Serial.println("E02"); // *** FATAL ERROR, CRASH POSSIBLE, STOPPING MOTOR ***
        Serial.println(motorNumber); // *** FATAL ERROR, CRASH POSSIBLE, STOPPING MOTOR ***
    }
    else if (type == 1){
        Serial.println("E03"); // *** FATAL ERROR, ROBOT ARM MIGHT GO LOOSE, STOPPING MOTOR ***
        Serial.println(motorNumber); // *** FATAL ERROR, CRASH POSSIBLE, STOPPING MOTOR ***
    }
    //Serial.print(motorNumber);
    return 0;
} //end of criticalSituation

int resetAll(){
    for (int i=0; i < 3; i++){
        digitalWrite (pinPwm[i], LOW); // Setting all PWM's to LOW
        Serial.print("C");
        Serial.print(i);
        Serial.print("=");
        convertToThreeDigits(cycle[i]);
    } //end of for
    return 0;
} //end of resetAll()

void convertToThreeDigits(int input){
    if (input >= 0 && input < 10){
        /*switch (input){
            case 0 :
                Serial.print("00");
                break;
            case 1 :
                Serial.print("01");
                break;
            case 2 :
                Serial.print("02");
                break;
            case 3 :
                Serial.print("03");
                break;
            case 4 :
                Serial.print("04");
                break;
            case 5 :
                Serial.print("05");
                break;
            case 6 :
                Serial.print("06");
                break;
            case 7 :
                Serial.print("07");
                break;
            case 8 :
                Serial.print("08");
                break;
            case 9 :
                Serial.print("09");
                break;
            default :
                Serial.print("00");
        */
    }
}

```

```

        break;
    } //end of switch
    */
    Serial.print("00");
    Serial.print(input);
}
else if ( input >= 10 && input < 100){
    Serial.print("0");
    Serial.print(input);
}
else{
    Serial.print(input);
}
} //end of convertToThreeDigits()

void saveCurrentPosition(int index){
    if (!manulaSteering){
        EEPROM.write(index, cycle[index]);
    }
} //end of saveCurrentPosition

void findCurrentPosition(boolean printInfo){
    for (int i = 0; i < 3; i++){
        value = EEPROM.read(i);
        cycle[i] = value;
        if (printInfo){
            Serial.print("P");
            Serial.print(i);
            Serial.print("*");
            convertToThreeDigits(value);
        }
    }
} //end of findCurrentPosition

boolean ifReachedHighBorder(int index){
    if ((!manulaSteering) && (index + 1 > 217)){
        return true;
    }
    else{
        return false;
    }
}

boolean ifReachedLowBorder(int index){
    if ((!manulaSteering) && (index - 1 < 0)){
        return true;
    }
    else{
        return false;
    }
}

void sendFeedbackManualSteering(){
    if (manulaSteering){
        Serial.print("m00");
    }
    else{
        Serial.print("m01");
    }
}

void handleMotor(byte index, byte dir, byte pwm){

```

```

    if (dir == 0){
        digitalWrite (pinDir[index], LOW);
    }
    else{
        digitalWrite (pinDir[index], HIGH);
    }
    if (pwm == 0){
        digitalWrite (pinPwm[index], LOW);
    }
    else{
        digitalWrite (pinPwm[index], HIGH);
    }
}
} //end of handleMotor

boolean moveToXYZ(int x, int y, int z){
    for (int i = 0; i<3; i++){
        int tempCnt = howManyCounterRound(calcDist(i, x, y, z));
        if (tempCnt == -1){ //Not reachable point
            resetAll();
            Serial.println("ERROR NOT REACHABLE POINT, RESETTING");// Error in xyz control
            return false;
        }
        // Serial.println("\nCounter 1 ?");
        // Serial.println(tempCnt);
        // Serial.println("\n");
        startMotor(i, tempCnt);
        if (i == 2 ){
            return true;
        }
    }
    Serial.println("E04");// Error in xyz control
    return false;
} //end of moveToXYZ()

double calcDist(byte i, double x, double y, double z){

    Serial.println("\nCalculated distance is : \n");
    Serial.println(( sqrt( squareOfInput(x - motor[i][0]) + squareOfInput(y - motor[i][1]) + squareOfInput(z - motor[i][2]) ) ));
    return ( sqrt( squareOfInput(x - motor[i][0]) + squareOfInput(y - motor[i][1]) + squareOfInput(z - motor[i][2]) ) );
}

int howManyCounterRound(double dist){
    double tempDis = dist - dist0;
    Serial.println("\nTempdis is :");
    Serial.println(tempDis);
    Serial.println("\nHow many counter round is left ?");
    int tempCnt = (int)(tempDis/0.2073);
    Serial.println(tempCnt);
    Serial.println("\n");
    if (abs(tempCnt) > 217){
        Serial.println("\n*** UNREACHABLE POINT ***\n");
        return -1;
    }
    return tempCnt;
}

int startMotor(byte index, int cnt){
    // Serial.println("\nCounter 2 ?");
    // Serial.println(cnt);
    // Serial.println("\n");
    goal[index] = abs(cnt);
    if (cnt > 0 && goal[index] != cycle[index] ){

```

```

        handleMotor(index, 1, 1); //needs to go forward
    }
    else if ( cnt < 0 && goal[index] != cycle[index] ) {
        handleMotor(index, 0, 1); //needs to go backward
    }
}

int checkIfReachedGoal(){
    for (int i = 0; i++; i<3){
        if (goal[i] == cycle[i]){
            stopMotor(i);
        }
    }
}

int stopMotor(byte index){
    digitalWrite(pinPwm[index], LOW); // Setting all PWM's to LOW
}

double squareOfInput(double input){
    return (input*input);
} //end of squareOfInput

```

D.3 Simulation, Processing

```

//*****
//***** PROCESSING CODE, SIMULA-
TIONS //*****

```

```

//Robot Simulation program 23.11.11
//by Shahab
//*****
//*****
//Robot Simulation program 23.11.11
//by Shahab
//***** DRAGGING
int x0 = 175, y0 = 0;
int x1 = 250, y1 = 0;
int x2 = 325, y2 = 0;
int counter = 0;

int x1D = 200, y1D = 70;
int x2D = 250, y2D = 70;
int x3D = 200, y3D = 20;
int x4D = 250, y4D = 20;
int x5D = 250, y5D = 20;
int x6D = 250, y6D = 20;
int x7D = 250, y7D = 20;
int x8D = 250, y8D = 20;

int x1R = 200, y1R = 70;
int x2R = 250, y2R = 70;
int x3R = 200, y3R = 20;
int x4R = 250, y4R = 20;

int direction = 1;

```



```

int scale = 25;
int robotScale = 50;

boolean moveDown = false , moveUp = false;

int[] arm1Color = {200,100,0}, arm2Color = {200,100,0}, arm3Color = {200,100,0}, arm4Color = {200,100,0}, robotColor = {200,100,0};

void setup() {
    size(640, 360, P3D);
    smooth();
    noStroke();
    initialize();
}

void draw() {
    background(51);
    fill(204);
    for (int i=0; i<500; i=i+scale){
        ellipse(x0, y0+i, 5, 5);
        ellipse(x1, y1+i, 5, 5);
        ellipse(x2, y2+i, 5, 5);
    } //end of for
    moveRobot();
    drawRobot();
}

void initialize(){
    x1D = 200;
    y1D = 100;
    x2D = 300;
    y2D = 100;
    x3D = 200;
    y3D = 50;
    x4D = 300;
    y4D = 50;
    x5D = 250;
    y5D = 0;
    x6D = 250;
    y6D = 150;
    x7D = 325;
    y7D = 75;
    x8D = 175;
    y8D = 75;
    x1R = (x7D+x8D)/2-(robotScale/2);
    y1R = (y5D+y6D)/2-(robotScale/2);
    x2R = (x7D+x8D)/2+(robotScale/2);
    y2R = (y5D+y6D)/2-(robotScale/2);
    x3R = (x7D+x8D)/2-(robotScale/2);
    y3R = (y5D+y6D)/2+(robotScale/2);
    x4R = (x7D+x8D)/2+(robotScale/2);
    y4R = (y5D+y6D)/2+(robotScale/2);
} //end of initialize

void robotLocation(){
    if (moveDown){
        y1R = y1R + scale;
        y2R = y2R + scale;
        y3R = y3R + scale;
        y4R = y4R + scale;
    }
    else if (moveUp){
        y1R = y1R - scale;
    }
}

```

```

        y2R = y2R - scale;
        y3R = y3R - scale;
        y4R = y4R - scale;
    }
    x1R = (x7D+x8D)/2-(robotScale/2);
    x2R = (x7D+x8D)/2+(robotScale/2);
    x3R = (x7D+x8D)/2-(robotScale/2);
    x4R = (x3D+x4D)/2+(robotScale/2);

} //end of robotLocation

void drawRobot(){
    //END EFFECTORS
    fill(arm1Color[0],arm1Color[1],arm1Color[2]);
    ellipse(x5D, y5D, 7, 7);
    fill(arm2Color[0],arm2Color[1],arm2Color[2]);
    ellipse(x6D, y6D, 7, 7);
    fill(arm3Color[0],arm3Color[1],arm3Color[2]);
    ellipse(x7D, y7D, 7, 7);
    fill(arm4Color[0],arm4Color[1],arm4Color[2]);
    ellipse(x8D, y8D, 7, 7);
    //ROBOT
    fill(robotColor[0],robotColor[1],robotColor[2]);
    //quad(x1R, y1R, x2R, y2R, x3R, y3R, x4R, y4R);
    rect(x1R, y1R, robotScale, robotScale);
    //arm1
    stroke(100);
    line(x5D, y5D, 0, x1R, y1R, 0);
    line(x5D, y5D, 0, x2R, y2R, 0);
    line(x5D, y5D, 0, x1R + (robotScale/2), y1R, 0);
    //arm2
    stroke(150);
    line(x7D, y7D, 0, x2R, y2R, 0);
    line(x7D, y7D, 0, x4R, y4R, 0);
    line(x7D, y7D, 0, x2R, y2R + (robotScale/2), 0);
    //arm3
    stroke(255);
    line(x8D, y8D, 0, x1R, y1R, 0);
    line(x8D, y8D, 0, x3R, y3R, 0);
    line(x8D, y8D, 0, x1R, y1R + (robotScale/2), 0);
    //arm4
    stroke(0);
    line(x6D, y6D, 0, x3R, y3R, 0);
    line(x6D, y6D, 0, x4R, y4R, 0);
    line(x6D, y6D, 0, x3R + (robotScale/2), y3R, 0);
    if (moveUp || moveDown)
        delay(1000);
    else if (cycleCounter == 5) {
        delay(1000);
        cycleCounter = 0;
    }
}
int cycleCounter = 0;
void moveRobot(){
    if (moveDown){
        switch (cycleCounter%5){
            case 1 :
                y6D = y6D + scale;
                cycleCounter++;
                break;
            case 2 :
                y7D = y7D + scale;

```

```

        y8D = y8D + scale;
        robotLocation();
        cycleCounter = cycleCounter + 3;
        break;
    case 0 :
        y5D = y5D + scale;
        cycleCounter++;
        moveDown = false;
        break;
    }
} //end of if
else if (moveUp){
    switch (cycleCounter%5){
        case 1 :
            y5D = y5D - scale;
            cycleCounter++;
            break;
        case 2 :
            y7D = y7D - scale;
            y8D = y8D - scale;
            robotLocation();
            cycleCounter = cycleCounter + 3;
            //robotLocation();
            break;
        case 0 :
            y6D = y6D - scale;
            cycleCounter++;
            moveUp = false;
            break;
    } //end of switch
} //end of else-if
} //end of moveRobot

void keyPressed(){
    if (keyCode == DOWN){
        //direction = 1;
        counter++;
        moveDown = true;
        cycleCounter = 1;
        println("Moving robot DOWN");
    } else if (keyCode == UP){
        //direction = -1;
        counter--;
        moveUp = true;
        cycleCounter = 1;
        println("Moving robot UP");
    }
} //end of keyPressed

// ***** Opt. Pull-Up
// *****
//Robot Simulation program 23.11.11
//by Shahab
// *****
int x0 = 200, y0 = 0;
int x1 = 325, y1 = 0;
int counter = 0;

int x1D = 200, y1D = 70;
int x2D = 250, y2D = 70;

```

```

int x3D = 200, y3D = 20;
int x4D = 250, y4D = 20;

int x1R = 200, y1R = 70;
int x2R = 250, y2R = 70;
int x3R = 200, y3R = 20;
int x4R = 250, y4R = 20;

int[] arm1Color = {200,100,0}, arm2Color = {200,100,0}, arm3Color = {200,100,0}, arm4Color = {200,100,0}, robotColor = {200,100,0};

int direction = 1;

int scale = 25;
int robotScale = 50;

boolean moveDown = false, moveUp = false, beginningFlag = true;

void setup() {
  size(640, 360, P3D);
  smooth();
  noStroke();
  initialize();
}

void draw() {
  background(51);
  fill(204);
  for (int i=0; i<500; i=i+scale){
    ellipse(x0, y0+i, 5, 5);
    ellipse(x1, y1+i, 5, 5);
  } //end of for
  moveRobot();
  drawRobot();
}

void initialize(){
  x1D = 200;
  y1D = 75;
  x2D = 325;
  y2D = 75;
  x3D = 200;
  y3D = 0;
  x4D = 325;
  y4D = 0;
  x1R = (x3D+x4D)/2-(robotScale/2);
  y1R = (y1D+y3D)/2;
  x2R = (x3D+x4D)/2;
  y2R = (y1D+y3D)/2-(robotScale/2);
  x3R = (x3D+x4D)/2+(robotScale/2);
  y3R = (y1D+y3D)/2;
  x4R = (x3D+x4D)/2;
  y4R = (y1D+y3D)/2+(robotScale/2);
} //end of initialize

void robotLocation(){
  if (moveDown){
    y1R = y1R + scale;
    y2R = y2R + scale;
    y3R = y3R + scale;
    y4R = y4R + scale;
  }
  else if (moveUp){

```

```

        y1R = y1R - scale;
        y2R = y2R - scale;
        y3R = y3R - scale;
        y4R = y4R - scale;
    }
    x1R = (x3D+x4D)/2-(robotScale /2);
    x2R = (x3D+x4D)/2;
    x3R = (x3D+x4D)/2+(robotScale /2);
    x4R = (x3D+x4D)/2;

} //end of robotLocation

void drawRobot(){
    //END EFFECTORS
    fill(arm1Color[0],arm1Color[1],arm1Color[2]);
    ellipse(x1D, y1D, 7, 7);
    fill(arm2Color[0],arm2Color[1],arm2Color[2]);
    ellipse(x2D, y2D, 7, 7);
    fill(arm3Color[0],arm3Color[1],arm3Color[2]);
    ellipse(x3D, y3D, 7, 7);
    fill(arm4Color[0],arm4Color[1],arm4Color[2]);
    ellipse(x4D, y4D, 7, 7);
    //ROBOT
    fill(robotColor[0],robotColor[1],robotColor[2]);
    quad(x1R, y1R, x2R, y2R, x3R, y3R, x4R, y4R);
    //arm1
    stroke(255);
    line(x3D, y3D, 0, x1R, y1R, 0);
    line(x3D, y3D, 0, x2R, y2R, 0);
    line(x3D, y3D, 0, x1R + (robotScale /4) , y1R - (robotScale /4), 0);
    //arm2
    stroke(255);
    line(x4D, y4D, 0, x2R, y2R, 0);
    line(x4D, y4D, 0, x3R, y3R, 0);
    line(x4D, y4D, 0, x2R + (robotScale /4), y2R + (robotScale /4), 0);
    //arm3
    stroke(255);
    line(x1D, y1D, 0, x1R, y1R, 0);
    line(x1D, y1D, 0, x4R, y4R, 0);
    line(x1D, y1D, 0, x1R + (robotScale /4), y1R + (robotScale /4), 0);
    //arm4
    stroke(255);
    line(x2D, y2D, 0, x3R, y3R, 0);
    line(x2D, y2D, 0, x4R, y4R, 0);
    line(x2D, y2D, 0, x3R - (robotScale /4), y3R + (robotScale /4), 0);

    if ( (moveUp || moveDown) && (cycleCounter > 2) ){//during climbing
        delay(1000);
    }
    else if (cycleCounter == 1 && optFlag == false) {
        delay(1000);
    }
    else if (cycleCounter == 2 && optFlag == true){
        delay(1000);
    }
    else if (cycleCounter == 5) {//ending phase
        delay(1000);
        cycleCounter = 0;
    }
    resetColors();
}

```

```

void resetColors(){
    arm1Color[0] = 200;
    arm1Color[1] = 100;
    arm1Color[2] = 0;
    arm2Color[0] = 200;
    arm2Color[1] = 100;
    arm2Color[2] = 0;
    arm3Color[0] = 200;
    arm3Color[1] = 100;
    arm3Color[2] = 0;
    arm4Color[0] = 200;
    arm4Color[1] = 100;
    arm4Color[2] = 0;
    robotColor[0] = 200;
    robotColor[1] = 100;
    robotColor[2] = 150;
    //optimized color
    if (!optFlag && !beginningFlag && optDone){
        arm1Color[0] = 110;
        arm1Color[1] = 200;
        arm1Color[2] = 0;
    } //end of if
    else if (optFlag && !beginningFlag && optDone){
        arm2Color[0] = 110;
        arm2Color[1] = 200;
        arm2Color[2] = 150;
    } //end of else
} //end of resetColors

int cycleCounter = 0;
boolean optFlag = false , optChecker = false , optDone = false;

void moveRobot(){
    if (moveDown){
        switch (cycleCounter%5){
            case 1 :
                //WHEN FLAG IS FALSE, LEG 2 IS THE OPTIMIZED ARM
                cycleCounter++;
                if (!optFlag){ // Simple
                    y1D = y1D + scale;
                    arm1Color[0] = 200;
                    arm1Color[1] = 200;
                    arm1Color[2] = 100;
                }
                break;
            case 2 :
                //WHEN FLAG IS TRUE, LEG 1 IS THE OPTIMIZED ARM
                cycleCounter++;
                if (optFlag){ // Simple
                    y2D = y2D + scale;
                    arm2Color[0] = 200;
                    arm2Color[1] = 200;
                    arm2Color[2] = 100;
                }
                break;
            case 3 :
                robotLocation();
                robotColor[0] = 200;
                robotColor[1] = 200;
                robotColor[2] = 150;
                cycleCounter++;
                if (optFlag){ //Arm1 is moving optimized

```

```

        y1D = y1D + scale;
    } //end of if
    else{
        y2D = y2D + scale;
    } //end of else
    optFlag = !optFlag;
    optDone = true;
    break;
case 4 :
    y3D = y3D + scale;
    cycleCounter++;
    arm3Color[0] = 200;
    arm3Color[1] = 200;
    arm3Color[2] = 100;
    break;
case 0 :
    y4D = y4D + scale;
    arm4Color[0] = 200;
    arm4Color[1] = 200;
    arm4Color[2] = 100;
    moveDown = false;
    break;
}
} //end of if
else if (moveUp){
    switch (cycleCounter%5){
        case 1 :
            y4D = y4D - scale;
            cycleCounter++;
            break;
        case 2 :
            y3D = y3D - scale;
            cycleCounter++;
            //robotLocation();
            break;
        case 3 :
            robotLocation();
            cycleCounter++;
            break;
        case 4 :
            y2D = y2D - scale;
            cycleCounter++;
            break;
        case 0 :
            y1D = y1D - scale;
            cycleCounter++;
    } //end of switch
} //end of else-if
} //end of moveRobot

void keyPressed(){
    if (keyCode == DOWN){
        //direction = 1;
        counter++;
        moveDown = true;
        cycleCounter = 1;
        println("Moving robot DOWN");
        beginningFlag = false;
    } else if (keyCode == UP){

```



```

        //direction = -1;
        counter--;
        moveUp = true;
        cycleCounter = 1;
        println("Moving robot UP");
        beginningFlag = false;
    }
} //end of keyPressed

// ***** simple pull-up

// *****
//Robot Simulation program 23.11.11
//by Shahab
// *****

int x0 = 200, y0 = 0;
int x1 = 325, y1 = 0;
int counter = 0;

int x1D = 200, y1D = 70;
int x2D = 250, y2D = 70;
int x3D = 200, y3D = 20;
int x4D = 250, y4D = 20;

int x1R = 200, y1R = 70;
int x2R = 250, y2R = 70;
int x3R = 200, y3R = 20;
int x4R = 250, y4R = 20;

int direction = 1;

int scale = 25;
int robotScale = 50;

boolean moveDown = false , moveUp = false;

String phase = "";

void setup() {
    size(640, 360, P3D);
    smooth();
    noStroke();
    initialize();
}

void draw() {
    background(51);
    fill(204);
    for (int i=0; i<500; i=i+scale){
        ellipse(x0, y0+i, 5, 5);
        ellipse(x1, y1+i, 5, 5);
    } //end of for
    moveRobot();
    drawRobot();
    fill(255);
    text(phase, 15, 20, 70, 70);
    println(phase);
}

void initialize(){
    x1D = 200;
    y1D = 75;

```

```

x2D = 325;
y2D = 75;
x3D = 200;
y3D = 0;
x4D = 325;
y4D = 0;
x1R = (x3D+x4D)/2-(robotScale /2);
y1R = (y1D+y3D)/2;
x2R = (x3D+x4D)/2;
y2R = (y1D+y3D)/2-(robotScale /2);
x3R = (x3D+x4D)/2+(robotScale /2);
y3R = (y1D+y3D)/2;
x4R = (x3D+x4D)/2;
y4R = (y1D+y3D)/2+(robotScale /2);
} //end of initialize

void robotLocation(){
    if (moveDown){
        y1R = y1R + scale;
        y2R = y2R + scale;
        y3R = y3R + scale;
        y4R = y4R + scale;
    }
    else if (moveUp){
        y1R = y1R - scale;
        y2R = y2R - scale;
        y3R = y3R - scale;
        y4R = y4R - scale;
    }
    x1R = (x3D+x4D)/2-(robotScale /2);
    x2R = (x3D+x4D)/2;
    x3R = (x3D+x4D)/2+(robotScale /2);
    x4R = (x3D+x4D)/2;
} //end of robotLocation

void drawRobot(){
    fill(200,100,0);
    ellipse(x1D, y1D, 7, 7);
    ellipse(x2D, y2D, 7, 7);
    ellipse(x3D, y3D, 7, 7);
    ellipse(x4D, y4D, 7, 7);
    fill(200,100,150);
    // ellipse((x1D+x2D)/2, y1R, 30, 30);
    // Robot
    // rect(x1R, (y1D+y3D)/2-(robotScale /2), 30, 30);
    // quad(x1R, y1R, (x3D+x4D)/2, (y1D+y3D)/2-(robotScale /2), (x3D+x4D)/2+(robotScale /2), (y1D+y3D)/2, (x3D+x4D)/2, (y1D+y3D)/2+(robotScale /2));
    // robotLocation();
    quad(x1R, y1R, x2R, y2R, x3R, y3R, x4R, y4R);
    // arm1
    stroke(255);
    line(x3D, y3D, 0, x1R, y1R, 0);
    line(x3D, y3D, 0, x2R, y2R, 0);
    line(x3D, y3D, 0, x1R + (robotScale /4), y1R - (robotScale /4), 0);
    // arm2
    stroke(255);
    line(x4D, y4D, 0, x2R, y2R, 0);
    line(x4D, y4D, 0, x3R, y3R, 0);
    line(x4D, y4D, 0, x2R + (robotScale /4), y2R + (robotScale /4), 0);
    // arm3
    stroke(255);
    line(x1D, y1D, 0, x1R, y1R, 0);

```

```

line(x1D, y1D, 0, x4R, y4R, 0);
line(x1D, y1D, 0, x1R + (robotScale / 4), y1R + (robotScale / 4), 0);
//arm4
stroke(255);
line(x2D, y2D, 0, x3R, y3R, 0);
line(x2D, y2D, 0, x4R, y4R, 0);
line(x2D, y2D, 0, x3R - (robotScale / 4), y3R + (robotScale / 4), 0);
if (moveUp || moveDown)
    delay(1000);
else if (cycleCounter == 5) {
    delay(1000);
    cycleCounter = 0;
}
}
int cycleCounter = 0;
void moveRobot(){
    if (moveDown){
        switch (cycleCounter%5){
            case 1 :
                y1D = y1D + scale;
                cycleCounter++;
                phase = "Phase #1";
                break;
            case 2 :
                y2D = y2D + scale;
                cycleCounter++;
                phase = "Phase #2";
                break;
            case 3 :
                robotLocation();
                cycleCounter++;
                phase = "Phase #3";
                break;
            case 4 :
                y3D = y3D + scale;
                cycleCounter++;
                phase = "Phase #4";
                break;
            case 0 :
                y4D = y4D + scale;
                cycleCounter++;
                phase = "Phase #5";
                moveDown = false;
                break;
        }
    }
    //end of if
    else if (moveUp){
        switch (cycleCounter%5){
            case 1 :
                y4D = y4D - scale;
                cycleCounter++;
                phase = "Phase #1";
                break;
            case 2 :
                y3D = y3D - scale;
                cycleCounter++;
                phase = "Phase #2";
                //robotLocation();
                break;
            case 3 :
                robotLocation();
                cycleCounter++;

```

```

        phase = "Phase #3";
        break;
    case 4 :
        y2D = y2D - scale;
        cycleCounter++;
        phase = "Phase #4";
        break;
    case 0 :
        y1D = y1D - scale;
//        cycleCounter++;
        phase = "Phase #5";
        moveUp = false;
        break;

    } //end of switch
} //end of else-if
} //end of moveRobot

void keyPressed(){
    if (keyCode == DOWN){
        //direction = 1;
        counter++;
        moveDown = true;
        cycleCounter = 1;
        println("Moving robot DOWN");
    } else if (keyCode == UP){
        //direction = -1;
        counter--;
        moveUp = true;
        cycleCounter = 1;
        println("Moving robot UP");
    }
} //end of keyPressed

```

D.4 Matlab, Workspace

```

//WORKSPACE ANIMATION
// *****
clear all;
close all;
clc;
cnt0 = 0;
cnt1 = 0;
cnt2 = 0;
%d = CONST(47mm + 10mm) + VARIABLE(45mm)
d0 = 190 + (cnt0*0.25);
d1 = 190 + (cnt1*0.25);
d2 = 190 + (cnt2*0.25);
%*****
a0 = 150;
b0 = 140;
%*****
x0 = (sqrt(2)*a0)/2;
y0 = 0;
z0 = 0;

%***

```

```

x1 = 0;
y1 = (sqrt(2)*a0)/2;
z1 = 0;

%***
x2 = 0;
y2 = 0;
z2 = -1*sqrt(b0^2-(a0^2/2));

XLine = [0,x0,x1,0,x2,x1,0,x0,x2]
YLine = [0,y0,y1,0,y2,y1,0,y0,y2]
ZLine = [0,z0,z1,0,z2,z1,0,z0,z2]

%*****

xMin = 1000;
xMax = -999;
yMin = 1000;
yMax = -999;
zMin = 1000;
zMax = -999;
counter = 0;

plot3(0,0,0,'or');
hold on
plot3(x0,y0,z0,'og');
hold on
plot3(x1,y1,z1,'ob');
hold on
plot3(x2,y2,z2,'ok');
hold on
% plot([0,0,0],[x0,y0,z0])
% hold on
% plot3([x0,y0,z0],[x1,y1,z1],[x2,y2,z2])

a = '*****';

% line([x1,y1,z1],[x0,y0,z0],'Color',[.8 .8 .8]);
% hold on
% line([x2,y2,z2],[x0,y0,z0],'Color',[.8 .8 .8]);
% hold on
% line([x1,y1,z1],[x2,y2,z2],'Color',[.8 .8 .8]);
% hold on
%

counter = 0;
% X range 260 - 400
% Y range 260 - 400
% Z range 20 - -150
for x=50:20:250
    for y=50:20:250
        for z=-220:20:-50
            r0 = sqrt((x-x0)^2 + (y-y0)^2 + (z-z0)^2);
            r1 = sqrt((x-x1)^2 + (y-y1)^2 + (z-z1)^2);
            r2 = sqrt((x-x2)^2 + (y-y2)^2 + (z-z2)^2);
%            r0 = sqrt((x-103.2376)^2 + y^2 + z^2);

```



```

counter
xMinVec
xMaxVec
yMinVec
yMaxVec
zMinVec
zMaxVec
a

numtimes=1;
fps=10;
movie(M,1,fps)

% Now save the movie as an mpeg file for use on the Web:
% map=colormap % Uses the previously defined colormap
% mpfwrite(M,map,'sstmovie.mpg')

// *****
// ***** WORKSPACE FULL

clear all;
close all;
clc;
cnt0 = 0;
cnt1 = 0;
cnt2 = 0;
%d = CONST(47mm + 10mm) + VARIABLE(45mm)
d0 = 190 + (cnt0*0.25);
d1 = 190 + (cnt1*0.25);
d2 = 190 + (cnt2*0.25);
%*****
a0 = 150;
b0 = 140;
%*****
x0 = (sqrt(2)*a0)/2;
y0 = 0;
z0 = 0;
%***
x1 = 0;
y1 = (sqrt(2)*a0)/2;
z1 = 0;
%***
x2 = 0;
y2 = 0;
z2 = -1*sqrt(b0^2-(a0^2/2));
%****
x3 = 0;
y3 = -1 * (sqrt(2)*a0)/2;
z3 = 0;
%***
x4 = -1 * (sqrt(2)*a0)/2;
y4 = 0;
z4 = 0;

%robot chassis
XLine = [x0,x1,x2,x0,x3,x4,x2,x3,x4,x1];
YLine = [y0,y1,y2,y0,y3,y4,y2,y3,y4,y1];
ZLine = [z0,z1,z2,z0,z3,z4,z2,z3,z4,z1];
plot3(XLine,YLine,ZLine,'color','k','LineWidth',5);
hold on
%*****
%robot arms

```

```

%arm 0
ix0 = 130;
iy0 = 130;
iz0 = -140;
XArm = [x0, ix0, x1, x2, ix0];
YArm = [y0, iy0, y1, y2, iy0];
ZArm = [z0, iz0, z1, z2, iz0];
plot3(XArm, YArm, ZArm, 'color', 'k', 'LineWidth', 5);
hold on

%arm 1
ix1 = -130;
iy1 = 130;
iz1 = -140;
XArm = [x1, ix1, x4, x2, ix1];
YArm = [y1, iy1, y4, y2, iy1];
ZArm = [z1, iz1, z4, z2, iz1];
plot3(XArm, YArm, ZArm, 'color', 'k', 'LineWidth', 5);
hold on

%arm 2
ix2 = -130;
iy2 = -130;
iz2 = -140;
XArm = [x3, ix2, x4, x2, ix2];
YArm = [y3, iy2, y4, y2, iy2];
ZArm = [z3, iz2, z4, z2, iz2];
plot3(XArm, YArm, ZArm, 'color', 'k', 'LineWidth', 5);
hold on

%arm 3
ix3 = 130;
iy3 = -130;
iz3 = -140;
XArm = [x3, ix3, x0, x2, ix3];
YArm = [y3, iy3, y0, y2, iy3];
ZArm = [z3, iz3, z0, z2, iz3];
plot3(XArm, YArm, ZArm, 'color', 'k', 'LineWidth', 5);
hold on

% Points
plot3(0, 0, 0, '*r', 'LineWidth', 20);
hold on
plot3(x0, y0, z0, 'ob', 'LineWidth', 10);
hold on
plot3(x1, y1, z1, 'ob', 'LineWidth', 10);
hold on
%plot3(x2, y2, z2, 'ob');
hold on
plot3(x3, y3, z3, 'ob', 'LineWidth', 10);
hold on
plot3(x4, y4, z4, 'ob', 'LineWidth', 10);
hold on
plot3(ix0, iy0, iz0, 'og', 'LineWidth', 30);
hold on
plot3(ix1, iy1, iz1, 'og', 'LineWidth', 30);
hold on
plot3(ix2, iy2, iz2, 'og', 'LineWidth', 30);
hold on
plot3(ix3, iy3, iz3, 'og', 'LineWidth', 30);
hold on

```



```

counter0 = 0;
counter1 = 0;
counter2 = 0;
counter3 = 0;

%Workspace loops
for x=50:5:250
    for y=50:5:250
        for z=-220:5:-50
            r0 = sqrt((x-x0)^2 + (y-y0)^2 + (z-z0)^2);
            r1 = sqrt((x-x1)^2 + (y-y1)^2 + (z-z1)^2);
            r2 = sqrt((x-x2)^2 + (y-y2)^2 + (z-z2)^2);
            cnt0 = (r0 - 190)/0.25;
            cnt1 = (r1 - 190)/0.25;
            cnt2 = (r2 - 190)/0.25;
            if ( cnt0>=0 && cnt0<= 217 && cnt1>=0 && cnt1<= 217 && cnt2>=0 && cnt2<= 217 )
                plot3(x,y,z,'r')
                hold on
                counter0 = counter0 + 1;

%                if (r0 < 195 && r1 < 195 && r2 < 195)
%                    v = [x,y,z]
%                    r0
%                    r1
%                    r2
%                end
            end
        end
    end
end

counter0

for x=-50:-5:-250
    for y=-50:-5:-250
        for z=-220:5:-50
            r3 = sqrt((x-x3)^2 + (y-y3)^2 + (z-z3)^2);
            r4 = sqrt((x-x4)^2 + (y-y4)^2 + (z-z4)^2);
            r2 = sqrt((x-x2)^2 + (y-y2)^2 + (z-z2)^2);
            cnt3 = (r3 - 190)/0.25;
            cnt4 = (r4 - 190)/0.25;
            cnt2 = (r2 - 190)/0.25;
            if ( cnt3>=0 && cnt3<= 217 && cnt4>=0 && cnt4<= 217 && cnt2>=0 && cnt2<= 217 )
                plot3(x,y,z,'db')
                hold on
                counter1 = counter1 + 1;
            end
        end
    end
end

counter1

for x=50:5:250
    for y=-50:-5:-250
        for z=-220:5:-50
            r3 = sqrt((x-x3)^2 + (y-y3)^2 + (z-z3)^2);
            r0 = sqrt((x-x0)^2 + (y-y0)^2 + (z-z0)^2);
            r2 = sqrt((x-x2)^2 + (y-y2)^2 + (z-z2)^2);
            cnt3 = (r3 - 190)/0.25;
            cnt0 = (r0 - 190)/0.25;

```

```

        cnt2 = (r2 - 190)/0.25;
        if ( cnt3>=0 && cnt3<= 217 && cnt0>=0 && cnt0<= 217 && cnt2>=0 && cnt2<= 217 )
            plot3(x,y,z,'sb')
            hold on
            counter2 = counter2 + 1;
        end
    end
end

counter2

for x=-50:-5:-250
    for y=50:5:250
        for z=-220:5:-50
            r1 = sqrt((x-x1)^2 + (y-y1)^2 + (z-z1)^2);
            r4 = sqrt((x-x4)^2 + (y-y4)^2 + (z-z4)^2);
            r2 = sqrt((x-x2)^2 + (y-y2)^2 + (z-z2)^2);
            cnt1 = (r1 - 190)/0.25;
            cnt4 = (r4 - 190)/0.25;
            cnt2 = (r2 - 190)/0.25;
            if ( cnt1>=0 && cnt1<= 217 && cnt4>=0 && cnt4<= 217 && cnt2>=0 && cnt2<= 217 )
                plot3(x,y,z,'or')
                hold on
                counter3 = counter3 + 1;
            end
        end
    end
end

counter3

%
% numtimes=1;
% fps=10;
% movie(M,1,fps)
%
% Now save the movie as an mpeg file for use on the Web:
% map=colormap % Uses the previously defined colormap
% mpgwrite(M,map,'sstmovie.mpg')

```


List of Figures

1.1	<i>Walloid Robot, an ongoing project at ROBIN group, University of Oslo</i>	3
2.1	<i>left to right Max V a chain-driven climber using vacuum cups, developed at University of Aalen - Rest six legged welding robot using magnetic force, Developed at CSIC Madrid - Roma grasping robot, specialized for inspection in steel bridge, developed at University of Madrid . .</i>	7
2.2	<i>Operator works side by side with a climbing welding robot (on-site user).</i>	9
2.3	<i>Autonomous cleaning climbing robot for glass and solar panels.</i>	10
2.4	<i>Artificial intelligence and smart agents.</i>	11
2.5	<i>The components of an embedded system [27]</i>	14
2.6	<i>Number of incidents in UK offshore from 2006 - 2010 [32]</i>	19
2.7	<i>A:Bad weather condition B:Shallow water platform</i>	21
2.8	<i>After being submerged in concentrated salt water for 5,000 hours, the unprotected iron T-Bolt on the left is totally corroded and unusable. . .</i>	23
2.9	<i>Washing salt off equipment with high pressure water</i>	23
3.1	<i>From left to right: Walloid robot received part, Walloid 3D design, in-house developed encoder (rotor and sensors), the implemented version of in-house designed version of the encoder, prototype</i>	28
3.2	<i>Left to right: 1: Control system hardware 2: Walloid arm</i>	29

3.3	<i>In-house built encoder, the rotor and the Sensors (A and B), together both read the rotation of the rotor and transfer motion to the motor axel</i>	30
3.4	<i>Walloid Prismatic joint</i>	31
3.5	<i>Walloid robot arm 2: Motor-rotor-encoder design</i>	32
3.6	<i>Left to right: Arrow model of Walloid arm Prismatic joint of the robotic arm</i>	32
3.7	<i>GPS positioning system</i>	33
3.8	<i>Robot workspace caluclated in Matlab by the logic discovered and developed here</i>	34
3.9	<i>A: Walloid arm B: Top view of Walloid chassis with one arm(as the received part was)</i>	35
3.10	<i>The longer the feet extends, the lower the height of the robot gets (A) and this means less place for overcoming obstacles B: Maximum height of Walloid</i>	36
3.11	<i>Adjoining surfaces is a challenge and possibility can be a candidate for future works</i>	37
3.12	<i>Left to right: 1: Part-1 is thinner than part-2 which imposes a weak point to the construction 2: Weak points of rotor in ROBIN developed encoder 3: Motor overheating and breakage results</i>	38
3.13	<i>Left to right: 1: Better construction as new rotor design with thicker walls, based on new jacket design 2: The redesigned jacket giving more space</i>	39
4.1	<i>Top-Down view of DES, designed for Control Hardware System</i>	43
4.2	<i>Top-Down view of DNP designed, for Control algorithm</i>	44
4.3	<i>Top-Down view of autonomy level of developed DNP</i>	45

5.1	<i>Assembly of the designed end effector with the rest of Walloid chassis design in Solidworks</i>	48
5.2	<i>Left to right: 1-6: Early designs for X2 prototype 7-8: Walloid end effector designs</i>	49
5.3	<i>Left to right: 1-2: semi-final End Effector design 3-4: semi-final bolt design</i>	49
5.4	<i>Left to right: 1-2: Final end effector design 3-4: Final bolt design . . .</i>	50
5.5	<i>Phase 1: Initial position Phase 2: Rotated 90 degrees from initial position Phase 3: Rotated 180 degrees from initial position (final phase)</i>	51
5.6	<i>Slow climbing and its similarity with pull-up exercise</i>	53
5.7	<i>Optimized slow climbing</i>	54
5.8	<i>Optimized slow climbing</i>	55
5.9	<i>Faster variation of climbing gait</i>	57
5.10	<i>Speed and Stability have opposite relation to eachother. The factor of stability is based on minimum number of locked arms to the wall during opeartion</i>	58
5.11	<i>Left - Asus Eee 900 - Right, Raspberry Pi motherboard</i>	60
5.12	<i>Embedded system / Server communication</i>	62
5.13	<i>A: Arduino Mega with AVR Atmega 1280 B: Arduino Fio with AVR Atmega 328P</i>	63
5.14	<i>I2C schematic for inter-connection of Arduino boards</i>	67
5.15	<i>Implementation of I2C connection of micro-controllers 2: Program results of testing the I2C connection</i>	68
5.16	<i>Arduino star network</i>	69

5.17	<i>Implementation plan for RxTx system configuration</i>	70
5.18	<i>Connecting three Arduino boards with Atemga AVR micro-controllers together with RxTx ports</i>	71
5.19	<i>Left: Power consumption for Bluetooth(BT), Ultra-wideband(UWB), ZigBee and WiFi Right: Normalized energy consumption for each method [58]</i>	71
5.20	<i>A: Xbee connecting with Serial protocol to Arduino board [67] B: Xbee pins [68]</i>	73
5.21	<i>Implementation plan of ZigBee network</i>	74
5.22	<i>Robot-Server-Client System Specifications Green boxes were implemented, while purple boxes are abstract definitions</i>	76
5.23	<i>Simulation</i>	77
5.24	<i>The industrial vision of a Robot - Server - Client</i>	78
5.25	<i>Combination of several network DAQ in rapid prototyping 0 2: Combination of Several Local DAQ in rapid prototyping 1 3: The industrial vision</i>	79
5.26	<i>Early stage hand drawn GUI design for different stakeholders in a RSC model</i>	79
5.27	<i>A: Walloid arm B: Walloid Java based standalone control panel</i>	81
5.28	<i>Multi thread programming, FIFO list</i>	82
5.29	<i>Control panel, simulated data flow, logging and simulation</i>	83
5.30	<i>Master & Slave Micro-Controller tasks</i>	83
5.31	<i>Distributed Navigation Program Flowchart</i>	85

5.32	<i>Simple pull-up climbing gait Power consumption before and after implementation of Power Optimizer Algorithm (40% difference)</i>	86
5.33	<i>Optimized Algorithm can reduce power consumption in one robot stride by 40%. Ts: Arms stride time, Tg: Grasp time, Pa: Arm Power Consumption, Pm: Motor Power Consumption, Pc: Master Controller (Core Micro-controller) Power Consumption</i>	87
5.34	<i>Web based Remote Control Panel, A:Developer B:Operator C:Expert</i>	88
5.35	<i>Ethernet shield used in web based CP test</i>	89
5.36	<i>Logging flowchart on server side and robot side</i>	90
5.37	<i>Communication protocol example between Server and Robot in RSC model</i>	90
5.38	<i>Workspace simulation with the Java CP and the arm</i>	94
5.39	<i>Different climbing gaits simulation</i>	94
5.40	<i>Optimized pull-up gait vs. simple pull-up gait</i>	95
5.41	<i>StatoilHydro experts looking at a virtual reality based on sensor readings from bottom of the sea</i>	96
6.1	<i>Walloid robot workspace with 5mm precision.</i>	101
6.2	<i>Final end effector design (blue) assembled with the rest of Walloid robot parts.</i>	101
6.3	<i>Red line stands for simple pull-up(assumed as base of consumption), blue for optimized pull-up and black for dragging. Optimized climbing and dragging save steps (motor actuation), therefore beside time and consequently speed they would be saving power as well.</i>	102
6.4	<i>Virtual presentation of climbing Gaits differences of speed</i>	103
6.5	<i>A: I2C - B: RS-232 - C:ZigBee</i>	104

6.6	<i>Distributed Embedded System implemented by using ZigBee and servo motors</i>	105
6.7	<i>Web based Remote Control Panel, A:Developer B:Operator C:Expert</i>	106
6.8	<i>Simple pull-up climbing gait Power consumption before and after implementation of Power Optimizer Algorithm (40% difference)</i>	107
6.9	<i>Processing simulations and Java CP were cooperting to connect to the hardware.</i>	108
7.1	<i>The rotor on the right (black) was broken in a way that still could forward the motor rotation to the rotary part, but no to the screw shaft . .</i>	116
7.2	<i>Polou Digital Distance Sensor - 2: Prismatic joint with distance meter sensor - 3: Sensor - Micro-controller schematic</i>	117
7.3	<i>Left schematic, switching to spare battery with power loss - Right schematic, switching to spare battery without power loss</i>	118
7.4	<i>Left to right: zero and ninety degrees angle with horizontal line 2: Gyro sensor [83]</i>	119
B.1	<i>Walldroid robot designs</i>	128
B.2	<i>Bolt Design</i>	129
B.3	<i>End effector design</i>	130
B.4	<i>Spherical wrist design</i>	131
B.5	<i>Proteus simulation of the DES, which wasn't so successful due to the complexity of the simulation process in Proteus</i>	132
C.1	<i>Robots moving in docking station to their place, being charged, maintained and software upgraded</i>	137

C.2	<i>In case of continuous receiving signals polling can be faster, halting, running interrupt functions, and retrieving system to the previous mode is time consuming [92]</i>	142
C.3	<i>Left to right : 1:Extrenal EEPROM memory 2: MicroSD shield for Arduino boards 3: microSD card</i>	143

Bibliography

Bibliography

- [1] Daily Mail. 50 dead as oil rig capsizes and sinks <http://www.dailymail.co.uk>, 2012 (accessed on 28.01.2012).
- [2] CNBC. Cnbc, mexican gulf, 2010 (accessed on 02.03.2011).
- [3] Dr. Thore Langeland. The norwegian oil industry association (2006) potential value of integrated operations on the norwegian shelf, 2011 (accessed 15.07.2011).
- [4] Svein Vatland Arne Ulrik Bindingsbø. The tail io project improves operations. *Touch Briefings*, 5:46 – 51, 2008.
- [5] Trond Michael Andersen Svein Vatland, Paula Doyle. Integrate operation - <http://library.abb.com/>. *Touch Briefings*, 5:46 – 51, 2007.
- [6] StatoilHydro. Case study: Io on the kristin platform, statoilhydro website - <http://www.statoil.com/>, 2011 (accessed 17.09.2011).
- [7] InTech. Oil companies of future - <http://www.isa.org>, 2012 (accessed on 29.01.2012).
- [8] Mats Høvin. Walloid / x2 robots - <http://www.robotikk.com/reseach.html>, 2009 (accessed 15.08.2011).
- [9] Walloid weblog - <http://walloid.blogspot.com>. 2010.
- [10] ABB. 10 good reasons to invest in robots - <http://www04.abb.com/global/seitp>, 2011 (accessed on 02.10.2011).
- [11] Kai Pfeiffer Birgit Graf. Mobile robotics for offshore automation. In *Proceedings of the EURON/IARP International Workshop on Robotics for Risky Interventions and Surveillance of the Environment*, Benicassim, Spain, January 2008.

- [12] New Scientist. Foxconn aims for a million robot workers by 2014 - <http://www.newscientist.com/>, 2011 (accessed on 04.08.2011).
- [13] European Robotics research Network. White paper - industrial robot automation - <http://www.euron.org/miscdocs/docs/euron2/year2/dr-14-1-industry.pdf>, 2012 (accessed on 23.01.2012).
- [14] International Federation of Robotics. Industrial robots statistics - www.ifr.org, 2012 (accessed on 23.01.2012).
- [15] European Commission. Work on preparatory studies for eco-design requirements of eups, lot 17 vacuum cleaners - <http://ec.europa.eu/energy/efficiency/studies>, 2012 (accessed on 23.01.2012).
- [16] S. Trujillo, B. Heyneman, and M. Cutkosky. Constrained convergent gait regulation for a climbing robot. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5243–5249, may 2010.
- [17] Thomas A. Potts George I. Matsumoto. Two ways of researching the underwater world. 31, 2010.
- [18] Paal Johan From. Off-shore robotics, robust and optimal solutions for autonomous operation. *PhD Thesis, Norwegian University of Science and Technology Press, 2010*, 312, 2010.
- [19] StatoilHydro. Statoilhydro website, integrated operation - <http://www.statoil.com/>, 2011 (accessed 16.09.2011).
- [20] K. Berns, C. Hillenbr, and T. Luksch. Climbing robots for commercial applications - a survey, 2003.
- [21] A. Steinfeld. Interface lessons for fully and semi-autonomous mobile robots. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2752 – 2757 Vol.3, april-1 may 2004.
- [22] S. Trujillo, B. Heyneman, and M. Cutkosky. Constrained convergent gait regulation for a climbing robot. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5243–5249, may 2010.
- [23] R.T. Pack, Jr. Christopher, J.L., and K. Kawamura. A rubbertuator-based structure-climbing inspection robot. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 1869–1874 vol.3, apr 1997.

- [24] George A. Bekey. *Autonomous Robots*, page 560. The MIT Press, 1 edition, 2005.
- [25] Peter Norvig Stuart Russell. *AI a mordern approach*, page 1152. Prentice Hall, 2 edition, 2009.
- [26] Ørjan G. Martinsen. *PC-basert instrumentering og mikrokontrollere*, page 232. Gyldendal akademisk, 1 edition, 2006.
- [27] Raj Kamal. *Embedded Systems: Architecture, Programming and Design*, page 704. McGraw-Hill Education (India), 2 edition, 2009.
- [28] British Petroleum. Remotely operated vehicles, bp report - <http://www.bp.com/>, 2011 (accessed on 02.10.2011).
- [29] Digital Energy journal. Integrate operations at kristin - <http://c183554.r54.cf1.rackcdn.com/novdec07web.pdf>. *Digital Energy journal*, November - December 2007, 44:2 – 3, 2007.
- [30] Reuters. Reuters offshore incident timeline, 2010 (accessed on 11.04.2011).
- [31] Bente E. Moen Tone Morken, Ingrid Sivesind Mehlum. Work-related musculoskeletal disorders in norway's offshore petroleum industry. *Oxford University Press on behalf of the Society of Occupational Medicine*, 6, 2007.
- [32] Health and UK Safety Executive. Offshore injury, ill health and incident statistics - <http://www.hse.gov.uk/>, 2010 (accessed on 11.04.2011).
- [33] R. Mayor G. Bright, D. Ferreira. Automated pipe inspection robot. *Industrial Robot*, 24(4):285–289, 1997.
- [34] Erik Kyrkjebø, Pål Liljebäck, and Aksel A. Transeth. A robotic concept for remote inspection and maintenance on oil platforms. *ASME Conference Proceedings*, 2009(43413):667–674, 2009.
- [35] M. Stratmann T. Kamimuraa. The influence of chromium on the atmospheric corrosion of steel. 1999.
- [36] GMT. Mil-std-810 - <http://www.dtc.army.mil/navigator/>, (accessed on 11.10.2011).
- [37] Magnus Lange. Study and development of manipulators in an academic environment. *Master Project, University of Oslo 2011*, 170, 2011.
- [38] Abb robotstudio <http://www.abb.com/product/>. visited 2012.01.22.

- [39] Motoman motosim <http://www.motoman.com/products/software/>. visited 2012.01.220.
- [40] UiO ROBIN Group, IFI. Robin group wiki page, <http://robin.wiki.ifi.uio.no>, (accessed on 01.04.2011).
- [41] Object Co. Object connex 500 - <http://www.objet.com>, 2011 (accessed on 01.04.2011).
- [42] Lars Skaret. A stewart platform based replicating rapid prototyping system with biologically inspired path-optimization. *Master Project, University of Oslo 2011*, 134, 2011.
- [43] O. Unver, A. Uneri, A. Aydemir, and M. Sitti. Geckobot: a gecko inspired climbing robot using elastomer adhesives. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2329 – 2335, may 2006.
- [44] Sangbae Kim, M. Spenko, S. Trujillo, B. Heyneman, D. Santos, and M.R. Cutkosky. Smooth vertical surface climbing with directional adhesion. *Robotics, IEEE Transactions on*, 24(1):65 –74, feb. 2008.
- [45] Asus Co. Asus eee - <http://uk.asus.com/eee/>, 2011 (accessed on 22.09.2011).
- [46] Raspberry Pi Foundation. Raspberry pi - <http://www.raspberrypi.org/sample-page>, 2011 (accessed on 22.09.2011).
- [47] Atmel. Atmel - <http://www.atmel.com>, 2011 (accessed on 18.08.2011).
- [48] Arduino. Arduino website - <http://www.arduino.cc/en/guide/introduction>, 2011 (accessed on 23.09.2011).
- [49] Arduino. Arduino website, reference page - <http://www.arduino.cc/en/reference/homepage>, 2011 (accessed on 01.07.2011).
- [50] Sparkfun. Wifly shield - <http://www.sparkfun.com/products/9954>, 2011 (accessed on 29.09.2011).
- [51] Sparkfun. Sparkfun can-bus shield - <http://www.sparkfun.com/products/10039>, 2011 (accessed on 29.09.2011).

- [52] Jonathan W. Valvano. *Embedded Microcomputer Systems: Real Time Interfacing*, page 816. CL-Engineering, 3 edition, 2011.
- [53] Arduino. I2c/wire library - <http://arduino.cc/>, 2012 (accessed on 20.01.2012).
- [54] Ieee 802.3 <http://www.ieee802.org/3/>. visited 2011.12.22.
- [55] Jan L. Harrington. *Ethernet Networking for the Small Office and Professional Home Office*, page 352. Morgan Kaufmann, 3 edition, 2007.
- [56] Arduino Ethernet Shield. Arduino ethernet shield - <http://www.arduino.cc/en/main/arduinoethernetshield>, 2011 (accessed on 15.07.2011).
- [57] David Russell and Mitchell Thornton. *Introduction to Embedded Systems: Using ANSI C and the Arduino Development Environment (Synthesis Lectures on Digital Circuits and Systems)*, page 276. Morgan and Claypool Publishers, 1 edition, 2010.
- [58] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen. A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 46–51, nov. 2007.
- [59] Monroe Schlessinger and Irving J. Spiro. *Infrared Technology Fundamentals (Optical Science and Engineering)*, page 480. CRC Press, 2 edition, 1994.
- [60] J.M. Kahn and J.R. Barry. Wireless infrared communications. In *Proceedings of the IEEE*, volume 85, pages 265 – 298, 1997.
- [61] Jens Eliasson, Per Lindgren, and Jerker Delsing. A bluetooth-based sensor node for low-power ad hoc networks. *Journal of Computers*, 3(5), 2008.
- [62] Philippe Bonnet, Allan Beaufour, Mads Bondo Dydensborg, and Martin Leopold. Bluetooth-based sensor networks. *SIGMOD Rec.*, 32:35–40, December 2003.
- [63] Jens Eliasson, Per Lindgren, and Jerker Delsing. A bluetooth-based sensor node for low-power ad hoc networks. *Journal of Computers*, 3(5), 2008.
- [64] C. Evans-Pughe. Bzzzz zzz [zigbee wireless standard]. *IEE Review*, 49(3):28 – 31, march 2003.
- [65] ZigBee Alliance. Zigbee technology features, 2011 (accessed on 16.10.2011).

- [66] ZigBee Alliance. Zigbee specifications, <http://www.zigbee.org/>, 2011 (accessed on 16.10.2011).
- [67] Bildr. Xbee, bildr.org, 2012 (accessed on 13.01.2012).
- [68] RobotShop. Xbee, [robotshop](http://www.robotshop.com) - <http://www.robotshop.com>, 2012 (accessed on 13.01.2012).
- [69] Java.com. Java programming language - <http://www.java.com/en/about/>, 2012 (accessed on 19.01.2012).
- [70] Oracle. Java ee - <http://www.oracle.com/technetwork/java/>, 2012 (accessed on 19.01.2012).
- [71] Processing Org. Processing language - <http://processing.org/>, 2011 (accessed on 18.08.2011).
- [72] Processing Org. Processing features - <http://processing.org/about/>, 2011 (accessed on 18.08.2011).
- [73] Rxtx library used in java - <http://rxtx.qbang.org>. 2011.
- [74] Arduino. Arduino-avr sleep mode, <http://arduino.cc/>, 2012 (accessed on 05.01.2012).
- [75] Atmel. Avr atmel sleep mode - <http://www.atmel.com>, 2012 (accessed on 10.01.2012).
- [76] Mikihiro Ohnari. *Simulation Engineering*, page 190. Distributor in the USA and Canada, IOS Press, 1 edition, 1998.
- [77] Delfoi simulator <http://www.delfoi.com/>. visited 2012.01.22.
- [78] Curtis Blais, Don Brutzman, Doug Horner, and Major Shane Nicklaus. Web-based 3d technology for scenario authoring and visualization: The savage project. In *The Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)*, volume 2001 - Conference Theme: Warfighter Readiness Through Innovative Training Technology, 2001.
- [79] Microsoft flight simulator <http://www.microsoft.com/games/fsinsider/>. visited 2012.01.22.
- [80] Flyit <http://www.flyit.com/>. visited 2012.01.22.

- [81] Nasa helping pilots see through 'soup' - <http://www.nasa.gov/centers/langley/news/factsheets/soup.html>. visited 2012.01.22.
- [82] InterSil. Icl7673 <http://www.intersil.com/>, 2012 (accessed on 12.01.2012).
- [83] Sparkfun. Dual axis gyro - idg500, sparkfun - <http://www.sparkfun.com/products/9070>, 2011 (accessed on 12.10.2011).
- [84] Office of Naval Research. Hull bug - <http://www.onr.navy.mil>, 2011 (accessed on 21.05.2011).
- [85] Robotic technologies of Tennessee. Tennessee climbing robot - <http://www.robotictechtn.com>, 2011 (accessed on 03.08.2011).
- [86] Serbot Swiss Innovation. Gekko junior g1 - <http://www.serbot.ch>, 2011 (accessed on 05.10.2011).
- [87] Serbot Swiss Innovation. Solar power plants - <http://www.serbot.ch>, 2011 (accessed on 05.10.2011).
- [88] URAKAMI Research and Development. Ua - abrasives blast cleaning - <http://www.urakami.co.jp>, 2011 (accessed on 05.08.2011).
- [89] URAKAMI Research and Development. Ud - cleaning / polishing / inspection - <http://www.urakami.co.jp>, 2011 (accessed on 05.08.2011).
- [90] URAKAMI Research and Development. Um - surface polishing - <http://www.urakami.co.jp>, 2011 (accessed on 05.08.2011).
- [91] Robotic technologies of Tennessee. Mrws robot - <http://www.nsrp.org>, 2011 (accessed on 03.08.2011).
- [92] Livermore Computing Center Super-Computers. Mpi performance - <https://computing.llnl.gov>, 2011 (accessed on 19.10.2011).
- [93] Tronixstuff weblog. Eeprom lifespan - <http://tronixstuff.wordpress.com/>, 2011 (accessed on 20.06.2011).