Technological Educational Institute of Crete
Department of Informatics Engineering

All-terrain autonomous legged robotic platform

Ioannis Deligiannis

Submitted in part fulfilment of the requirements for the degree of
Master of Science

Approved by: Dr. Giorgos Papadourakis

September 2016

Abstract

Developing an all-terrain legged robotic platform capable of maneuvering and navigating in a non-structured environment has been an area of research in recent years. This master thesis presents the analysis, design, and implementation of an indoor/outdoor robotic motion platform which is intended for use in intelligent autonomous robots. The platform will provide a stable motion system for robots equipped with various sensory systems, to perform tasks autonomously to reach a goal.

For the development of this all-terrain robotic platform, we use two ARM Cortex M4 microcontrollers using a Real-time Operating System (RTOS) as well as a simple sensory system. The communication between the microcontrollers was established using the Controller Area Network (CAN) protocol as well as a novel encryption/validation messaging protocol above it. Moreover, a novel memory allocation scheme was analyzed and used, providing efficient dynamic memory allocation and defragmentation for embedded systems lacking a Memory Management Unit (MMU). Using as main criteria the efficiency in handling both external and internal memory fragmentation, as well as the requirements of soft real-time applications in constraint-embedded systems, the proposed solution of memory management delivers a more precise memory allocation process. This scheme is evaluated providing encouraging results regarding performance and reliability compared to the default memory allocator.

The outcomes of these endeavors would apply to problems in harsh environments, for example, in space research, investigating the surface of a planet without endangering human lives, or in disaster areas to locate injured persons and to give rescue personnel a more detailed picture of the area. The work done includes the initial development of a five-legged robotic platform containing a discrete-event based motion controller as well as a communication system providing an interface for easy handling the motion system whereby providing a more realistic experience of teleoperation, remote sensing, and semi-autonomous robot behavior than what currently exists. – While it is too early to incorporate the current research results into a final product, this development is specially designed on a smaller scale which can be used in mock system trials instead of toy-based robots, thereby providing more accurate representations of the challenges encountered by full-sized robotic automation systems.

The work completed to date acts as a starting point from which improvements and extensions could be made and incorporated. In this regard, suggestions for future work are also presented.

Acknowledgements

I would like to express my gratitude to my thesis advisor Dr. Giorgos Papadourakis of the Department of Informatics Engineering at Technological Educational Institute of Crete. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it. I would also like to acknowledge Dr. George Kornaros for his guidance in the field of the Embedded Systems, as well as gratefully indebted to his contribution and comment for the Memory Management Scheme.

The research leading to the results of the Adaptive Memory Management Scheme has received funding from the European Union (EU) FP7 project SAVE under contract FP7-ICT-2013-10 No 610996, and Horizon 2020 project TAPPS (Trusted Apps for open CPSs) under RIA grant No 645119.

Finally, I must express my very profound gratitude to my family as well as to Adriana Theodorakopoulou for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Contents

List of Tables

2 Five Legged Robotic Platform

3 Software Design and Implementation

List of Figures

2 Five Legged Robotic Platform

3 Software Design and Implementation

Chapter 1

Introduction

In the last four decades, the field of robotics knows great flourish and acceptance, both at research level and application in everyday life. Nowadays, the rapid evolution appears from the applications and the deployment of almost all over the world and from the research results. [1, 2, 3] Countries such as Japan, US and Western and Eastern European countries, which have made significant progress to research and technology, constitute the basic core of support and development of robotics. The reasons have to do with the economic benefits for their national economies as a result of automation and the overall modernization of production, and with the expectations and prospects fueled by the results at a scientific level. [1, 2, 4] Another reason for the spread of robotics is the closely dependent and interactive relationship with cutting-edge technology such as IT, Electronics, Automatic Control and the Craftsman Intelligence Systems, Sensor Technology, the Communications and others, which operates multiplying and triggering developments.

In our days, robotics applications are popular, as many are the references to media about achievements in research laboratories and several research groups dealing with particularly demanding and promising projects Notified robots that perform tasks in space and controlled from Earth, operating underwater explorations and work involved in medical surgery, or even robots socialize with people.[1, 3] There have been presented humanoids, crawlers - mobile robots, robots that have vision and hearing, having a perception of changes in their environment, they have fingers and manage unknown objects with skill, moving in non-structured or in unfamiliar areas, setting targets and taking decisions. Current robotic systems serve different areas, not just research applications, but also industrial, exploratory and daily life applications. Nowadays, the purpose of Robotics is the development and integration of advanced sensor systems, adaptability, and understanding to various known and unknown environmental conditions, decision-making in these environments, intelligence and autonomy.[1] The robotics research and technology, is the resultant of the following main sciences [1, 2, 5, 6]:

- Engineering - Mechanical engineering: analysis and design mechanical components and mechanisms (couplings, joints, the transmission, axles, gears, action tools, etc.) and their functions.

- Mathematics, algorithms, algebra, tables, mathematical and numerical analysis, differential equations, optimization, Euclidean, analytic and computational geometry to solve mathematical accounting systems and the efficient implementation of complex calculations and control strategies, geometric representations, mathematical modeling and simulation, etc.

- Automatic control systems for the study and design of the arm control system, based on the pilot information feedback model of the internal sensors. Generally, the problem of robotic control is, to obtain dynamic models of the operator, mathematical relationships and the use of these models to the laws produce or control strategies to achieve the desired response and efficiency of the robotic system.

- Artificial Intelligence (and intelligent control) for integrating external sensory feedback, ie high level information to the robot control system, for achieving intelligent and 'autonomous' behavior in robot organized groups (social robotics).

- Software Technology and Programming, for the design, development, coding and implementation of the necessary algorithms and operation / management of the robotic system software point of view, both in programming application level (in integrated environments or programming languages) and operating system.

- Electronic, implementing the above functions with analog and digital circuits, microprocessors and sensors. Besides construction technologies of nano-electronics, also proclaimed critical transformation agent of Robotics and Mechatronics systems in general (Mechatronics) acquired status by using these.

- Systems and production management for analysis, design, optimization and other practical issues of implementation of industrial applications with robots and advanced automated production cells.

1.1 Autonomous Robots and Application Types

Robots are mechanical or virtual artificial agents, usually electro-mechanical machines which are controlled by a computer program or electronic circuitry. There are separated into different categories depending on their functionality. Most popular robots are placed in hazardous places because these robots perform the tasks that humans are not able to access. Some robots can operate by themselves and other robots always need guidance from humans to perform the tasks. They are used in various fields like medical, space communication, military applications, and so on. An automatic robot is a type of manipulated robotic system considered to be as one of the earliest robotic systems by the control system it possesses. They are divided into three main categories based on their characteristics and applications: Autonomous controlled robots, Remote controlled robots and Manually controlled robots. Out of three types of robotic manipulation system, the autonomous system is further classified into four types, which can be implemented over a wide range of applications: Programmable, Non-programmable, Adaptive, and Intelligent.[7, 8, 9]

Programmable Automatic Robots: are a first generation robot with an actuator facility on each joint. They can be reprogrammable based on the kind of task they are commissioned to. Robot kits like Lego mind storms, Bioloid from programmable Robotics can help the students to learn about its programming and working. The advanced mobile robot, robotic arms, and Gadgeteer are some of the most known examples of these robots type. However, the main drawback of this autonomous robot is that once is it programmed it cannot change its functionality even if there is an emergency. These robots can be used in different applications like mobile robotics, industrial controlling and spacecraft applications.

Non-Programmable Automatic Robots: are one of the primary types of robot, in fact, a non-programmable robot. It is not even considered as a robot but as an exploiter lacking reprogrammable controlling device. An example of these types of robots are the mechanical arms used in industries which are attached to a programmable device. These types of robots find applications in some of the devices including path guiders and medical products carriers and also some line follower robots.

Adaptive Robots: are also industrial robots that can be adapted independently to various ranges in the process. However, these robots are more complex than programmable robots. They be adapted up to a certain extent, and after evaluation, they can perform the action required in that adapted area. They are mostly equipped with a sophisticated sensory and control system. Sensors are used to sense environmental conditions, process variables and other parameters related to a particular task. Control system usually has access to these sensor signals, and depending on the implemented algorithm, they control the outputs. These robots can be used in different applications like aerospace, medical, consumer goods, house-hold applications and manufacturing industrial areas.

Intelligent Robots: as the name suggests, are the most intelligent of all the other types of robots with sensors and microprocessors for storing and processing the data. These robots performance is highly efficient due to their situation-based analyzing and task performing abilities. Intelligent robots can sense the senses like pain, smell and taste and are also capable of vision and hearing, and in accordance, perform the actions and expressions like emotions, thinking and learning. These robots find their applications in the fields like medical, military applications and home appliance control systems, etc.

1.2 Benefits of Using Robots

The benefits of robots have increased their flexibility with being capable of performing a variety of tasks and applications. They are more precise and consistent than human workers. Robots also allow for increased production and profit margin because they can complete tasks faster. Robots have the ability to work around the clock since they do not require vacations, sick days, or breaks. They also make fewer mistakes than humans, saving companies time.Other benefits of robotics are that they can work in any environment, adding to their flexibility. Robots are able to eliminate dangerous jobs for humans because they are capable of working in hazardous environments. They can handle lifting heavy loads, toxic substances, and repetitive tasks. Moreover, robotics can be beneficially utilized in the place of human labor in production units that requires to function for twenty-four hours and even on holidays to increase production and profitability. As human labor often does not feel comfortable working such odd hours, equal to the capacity of the machines, robots are an ideal working partner for such tense circumstances. [1, 8, 9]

The benefits of robots have opened the door for their use in many fields. Their ability to be customized provides companies the flexibility to use them for a variety of tasks.

1.3 Global Robotics Technology Market

Robotics technology is used in a wide range of industries including automotive, healthcare, aerospace, infrastructure and defense. They can be used for numerous activities The increasing need of automation solutions is a factor that drives the international market [10]. Manufacturers are increasingly opting for process automation, owing to the rise in labor costs and market competition. A dynamic rise in the number of smartphones and tablets supplements the process of robot development and contributes to the growth of the global market. Additionally, emerging technologies in robotics and cost reduction in products with the use of robots supplement the market growth. With the growth of robotics technology, there will be a significant increase in the number of jobs available for the human workforce. Lack of awareness about the advantages of robots in the market, along with the high installation cost of robotics technology are the major drawbacks for this market. [11, 12]

The growing demand for automation in mechanical, aerospace and defense industries, along with the growing adoption of new technologies like nano-robotics and cloud robotics, highlight the growth opportunities for players in the market. There has been an increase in the adoption of robotics by SMEs, owing to the development of small, compact and affordable robots by the key players in the market.

The vendors in the market offer a wide range of robotics systems to meet the requirement of the customers. The development of robots requires high-quality hardware and software to control the robots operations and integration along with its auxiliary systems in a working environment. The key players in the market are developing small, compact, energy efficient and low-cost robotics systems, which can be used in a wide range of industries, particularly by SMEs. Additionally, the vendors in the market are acquiring and collaborating with the top companies in the market, to enhance their product portfolio.[11]

The global Robotics Technology market is segmented on the basis of types of robots, components, applications and geography. The various types of robots used across different industries include industrial robots, service robots and mobile robots. Industrial robots account for a larger share among others. The component segment is further categorized into hardware, software, and services. The different hardware components include sensors, controllers, effectors, power supply and actuators. Robotic technology finds applications across a wide range of domains including healthcare, defense and security, automotive, aerospace, infrastructure, industrial and residential. The microscopic analysis of the market has been performed by examining various regions such as North America, Europe, Asia-Pacific and LAMEA.

Until 2020 the global robotics technology market is expected to reach 82.7 billion dollars and almost 1/3 of the total production will be mainly referred to the industrial robots.

1.4 Objective of the Thesis

The central objective of this thesis is the analysis and the developement of a five legged robotic platform to provide an all-terrain mean of transport for autonomous robots by applying the latest available knowledge in the field of robotics.

In addition, a novel memory management scheme as well as a simple yet effective encrypted communication mechanism using the CAN protocol are introduced and the performance and efficiency of the algorithms are presented.

1.5 Structure of the Thesis

The thesis is divided into 3 chapters covering the design and the construction of the robotic platform including the inverse kinematics, the software design and implementation including the memory management scheme, communication protocols, sensory system and the computer based controling application, summary and conclusion followed by references. A brief description of each chapter is provided in the following paragraph.

The Chapter of design and construction of the robotic platform provides a review relevant to the mechanical parts of the platform. An effort has been made to comprehensively cover the work of different researchers in the field robotics for study of various methods used by different authors for solving the Inverse Kinematic. There will be presented the kinematics algorithms that where used in this platform as well as some information about the Gait movements.

The Chapter of software design and implementation starts with a description of basic principles and requirements of the software design followed by and literature review of real-time operating systems and memory management schemes. Moreover, the will be a short description of some well known communication protocols used in the embedded systems which will be followed by an extended analysis of the encypted mossaging protocol over the CANBus. Finally there will be provided some information about the sensory system as well as the data noise reduction using a modified Kalman filter.

Finally the Chapter of summary and conclusion contains brief discussion about the topic which may require further study and investigation as well as future work and implementation of the robotic platform. The thesis is concluded in chapter 7 with references.

Chapter 2

Five Legged Robotic Platform

2.1 Introduction

On the most basic level, human beings are made up of five major components:

- A body structure
- A muscle system to move the body structure
- A sensory system that receives information about the body and the surrounding environment
- A power source to activate the muscles and sensors
- A brain system that processes sensory information and tells the muscles what to do

Robots are made up of the very same components. A typical robot has a movable mechanical structure, a motoring system, a sensory system, a power supply and a ”brain” that controls all of these elements. Essentially, robots are man-made versions of animal life – they are machines that replicate the human and animal behavior. Of course, humans also have intangible attributes, such as intelligence and morality, but on a physical level, the previously mentioned list covers it. In this chapter, we will discuss and analyze the structure of the robotic platform as well as the electronic components and circuit which drives the platform.

2.2 Locomotion mechanisms

Many of the unique abilities that robots possess come from the fact that they are often made mobile. Achieving the mobility required is however rarely done without difficulty. The terrain over which robots must be able to move is often uneven, slippery or muddy, which gives rise to many challenges, particularity stability. A mobile robot requires locomotion mechanisms that will enable it to move unbounded throughout its environment. However, there is a significant variation of possible movement ways, and so the selection of a robots approach to locomotion is an important aspect of mobile robot design. [13]

Their biological counterparts have inspired most of the locomotion mechanisms. Biological systems succeed in moving through a wide variety of harsh environments. Therefore, it can be desirable to copy their selection of locomotion mechanisms. However, the drawbacks in replicating nature in this are tough for several reasons beginning with the mechanical complexity. For example, insects achieve a level of robustness that human fabrication techniques are not able to achieve. Moreover, a major problem is the energy storage system and the muscular and hydraulic activation systems used by large animals and insects for reaching a certain amount of torque, response time and efficiency that is far exceed similarly man-made systems. These limitations lead us to develop mobile robots using wheeled mechanisms which are a well-known human technology for vehicles or articulated legs inspired by spiders locomotion mechanism as well as some mammals. In general, legged locomotion expects higher degrees of freedom and therefore greater mechanical complexity than wheeled systems. Spiders' locomotion mechanism inspired the structure of this experimental robot but adjusted accordingly for maximum functionality/performance and as minimum as possible power consumption. For that reason, the most suitable structure is a 5-legged robotic platform which combines the structure stability of a 6-legged robot and the lowered power consumption of a 4-legged robot (compared to the previous one). The distribution of the working space of each leg is equal thus the body construction is symmetrical.

The legs end-effector consists of two dissimilar plates which create a stable but also flexible enough to withstand the forces caused by the motion. The materials used to build the main frame is plexi-glass and PCB boards for feet. This selection of the materials produces a light-weighted thus strong and flexible structure.

Below there are presented the AutoCAD sketches for both the main frame (body) and the end-effector of the legs.

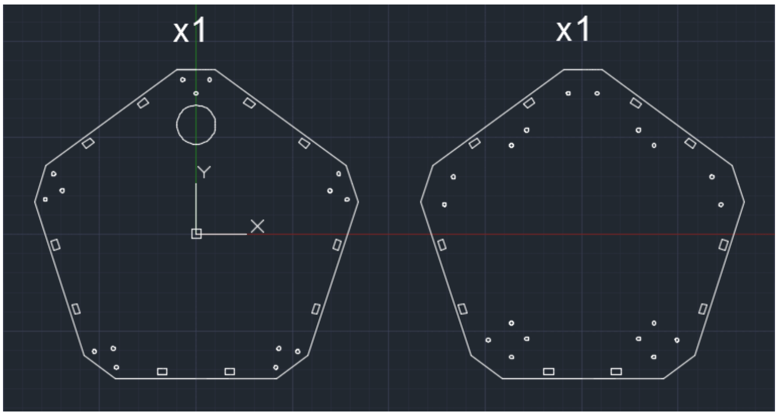


Figure 2.1: AutoCAD drawings of the main frame

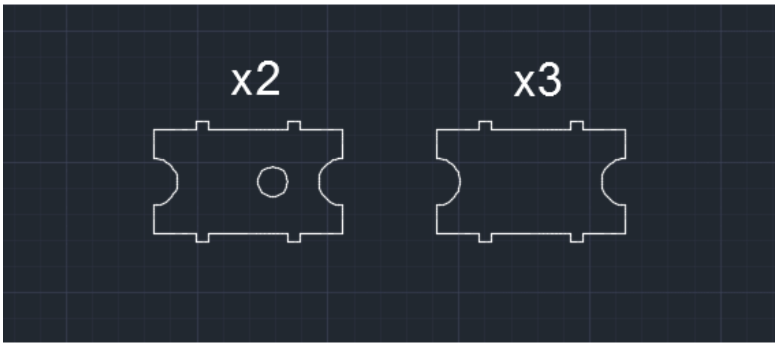


Figure 2.2: AutoCAD drawings of the main frame's side covers

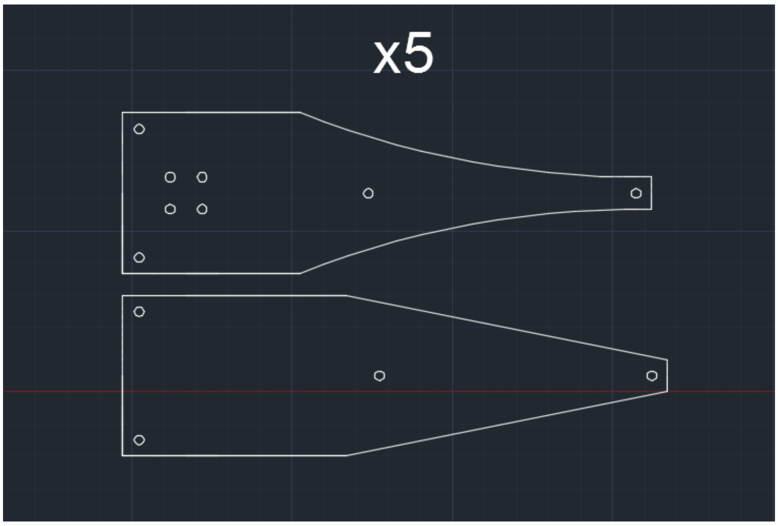


Figure 2.3: AutoCAD drawings of the foot end-effector

2.3 Electronic Circuit

To control various inputs and outputs (which we will discuss later) and make our robot move, we need to give it a brain. All robots have brains; even the smallest ground rovers do an exceptional job of adjusting rough terrain with a few switch sensors. Those tiny robots are fun to construct and demonstrate basics of robotics. However, more advanced robotic platforms have something called non-sensory actions which are programmed and performed in a given order. For accomplishing that, it is necessary to sequence the order of duties. However, the processing power that is demanded of such a sequencer is quite demanding. Apart from primary functions (Moving, avoiding collisions, negotiating terrain), there are some complex functionalities that the brain of the robot must accommodate such as real-time data filtering. For that reason, the brain of the robotic platform is using two microcontrollers ARM Cortex M4 running at 45MHz, each one for a specific workload.Except the processing units, the robotic platform is equipped with a sensory system to provide the necessary feedback from the environment as well as some input/output pins for external sensors/modules.

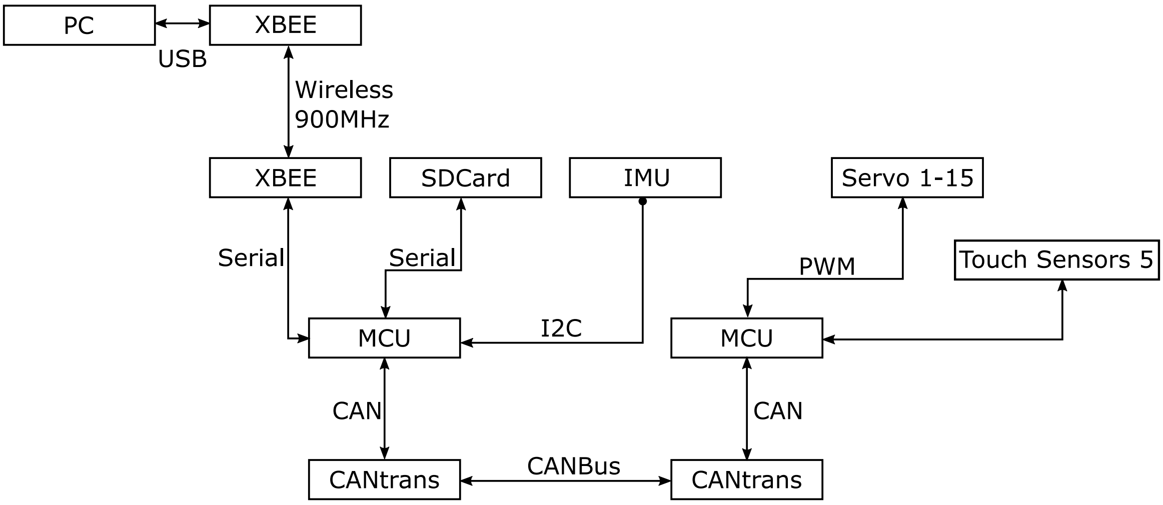


Figure 2.4: Devices connectivity map

For that reason we had to made a double sided custom pcb using a simple but effective CNC 3020. The following outline uses what we have concluded to be the most common tools. These tools are Eagle CAD, PCB-gcode, Mach 3 CNC, 0.1mm 30deg V-Shaped Engraving Bit, and of course random surplus carbide PCB drill bits. We tried to follow the safest way (ie large traces width). After designing the circuit using the EAGLE PCB software, we use the pcb-gcode plugin to produce the G-Code for the CNC afterwards using the Mach 3 software we started the milling process.

On a dual sided PCB there will often be a need for connecting a through-hole component pin to the top layer. Whenever it is reasonably easy to solder on the top of the PCB we solve it that way. But that is not always possible. Some components completely cover the top pad making soldering impossible. And then we need a plated through hole. To make those we use tiny hollow copper rivets made specially for creating PCB vias.

Below there are presented the EAGLE PCB Desing as well as the final result.

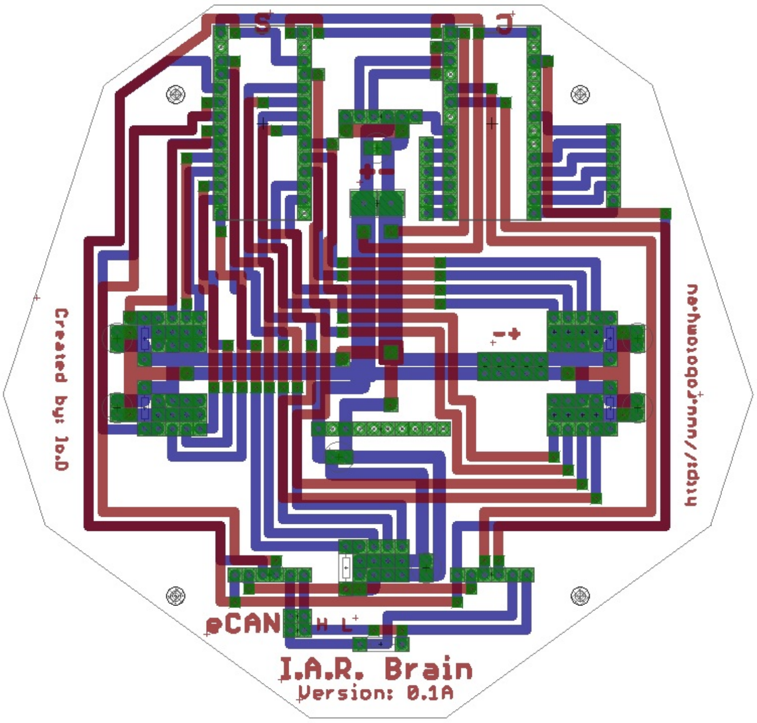


Figure 2.5: Eagle PCB board design

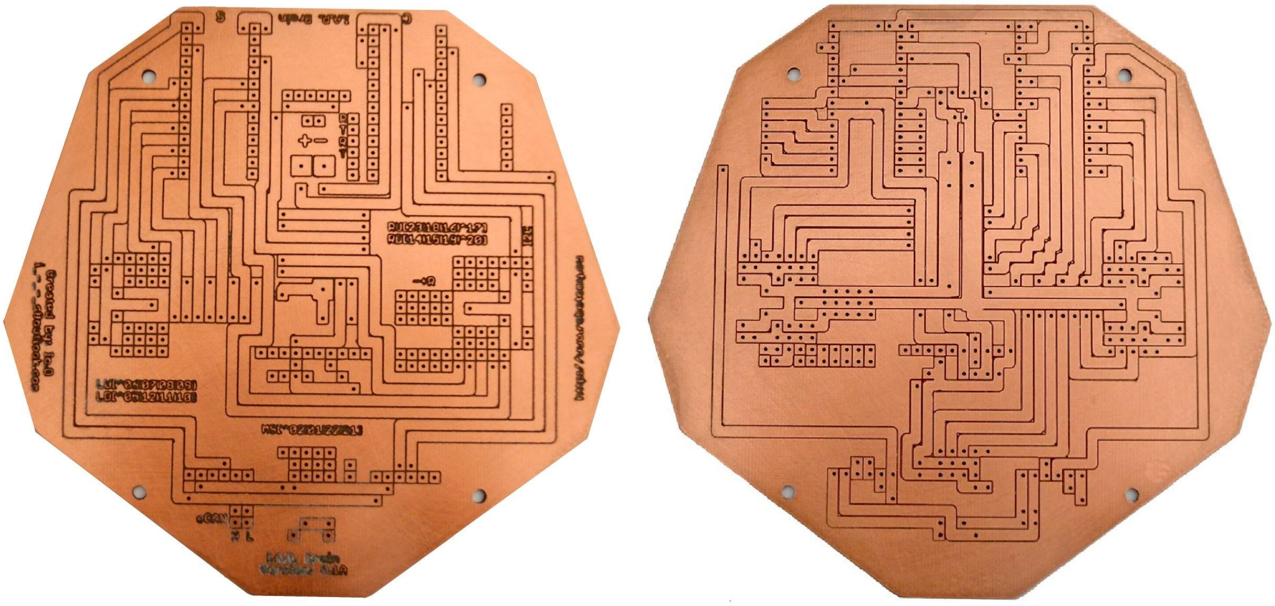


Figure 2.6: IARBrain PCB after the isolation process

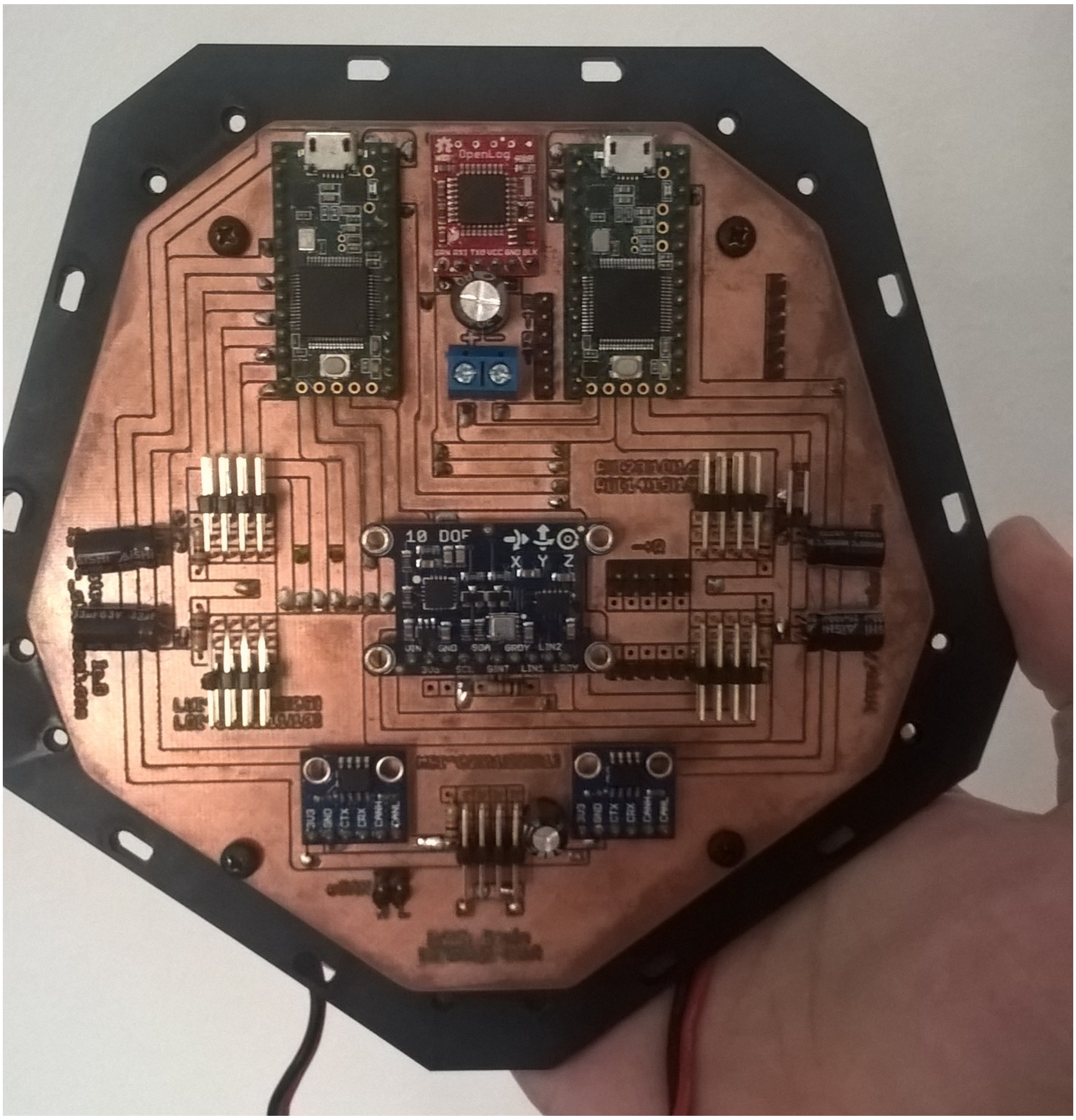


Figure 2.7: Assembled IARBrain v0.1 PCB

2.4 Construction Results

Based on the results mentioned above, an experimental robot was constructed. The assembly process of the individual parts was completed step by step following the original plans. Tightening and examination of the overall construction result was neccessary to avoid rupture forces and possible malfunctions. Finally , some Stress Test confirmed the original design idea and the functionality of the robot base. The constructed robotic platform which is almost 5kg can be stored in a 600x600x20 box. Below there are presented the assembled two sided PCB as well as the final result.

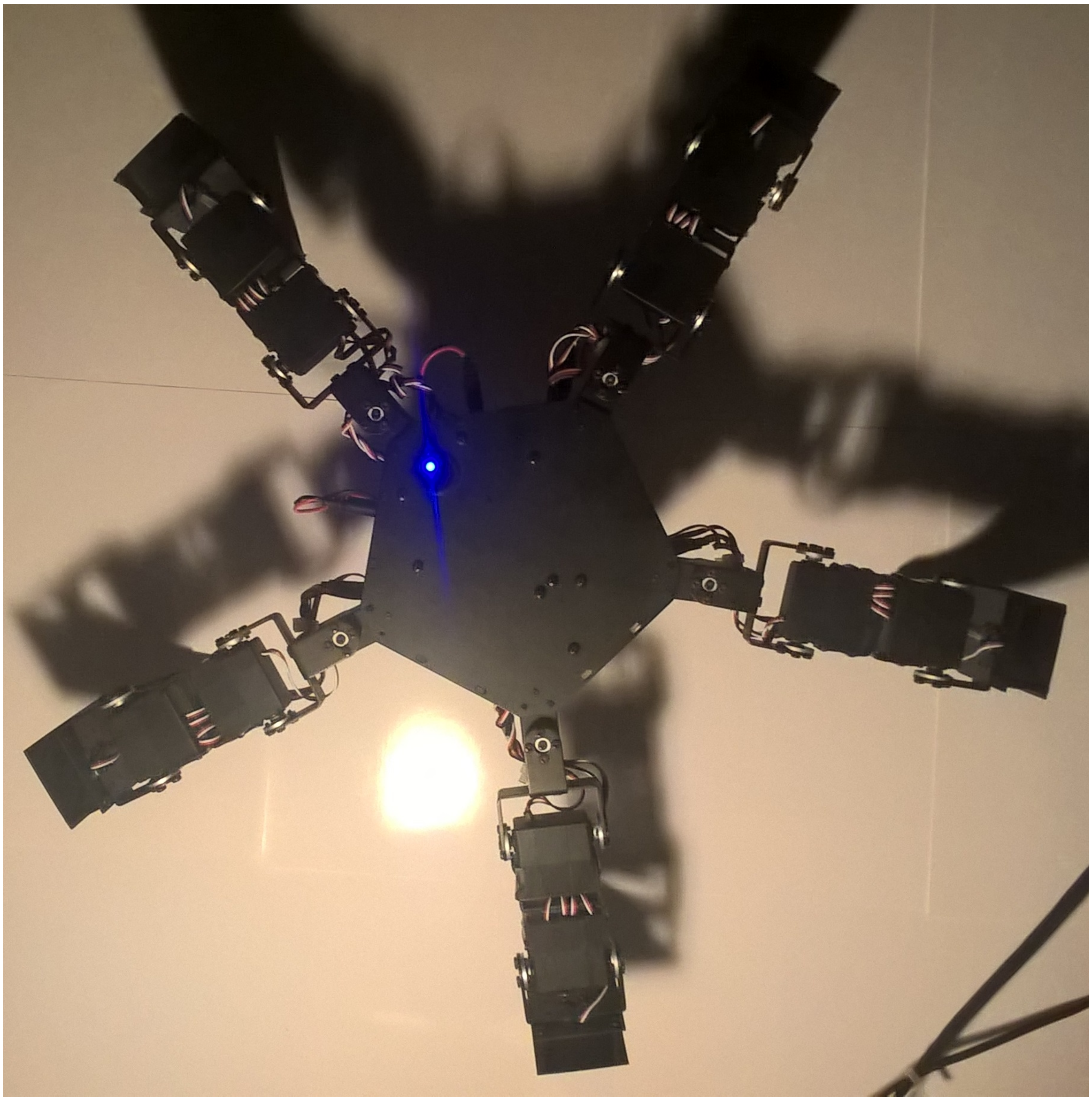


Figure 2.8: Assembled five-legged robotic platform

2.5 Analysis of Robot Kinematics

Kinematics studies the motion without consideration of any forces or moments that cause the motion. Robot kinematics refers the analytical study of the movement of a robot manipulator. Formulating the suitable kinematics models for a robot mechanism is very crucial for analyzing the behavior of industrial manipulators. There are two different spaces used in kinematics modeling of manipulators, Cartesian space, and Quaternion space. The transformation between them can be decomposed into a rotation and translation. There are many ways to represent rotation, including the following: Euler angles, Gibbs vector, Cayley-Klein parameters, Pauli spins matrices, axis and angle, orthonormal matrices, and Hamilton 's quaternions. Of these representations, homogenous transformations based on 4x4 real matrices (orthonormal matrices) have been used most often in robotics. Denavit & Hartenberg (1955) [14] showed that a transformation between two joints requires four parameters. These parameters are known as the Denavit-Hartenberg (DH) parameters which have become the standard for defining robot kinematics. Although quaternions compound an elegant representation for rotation, they have not been applied as much as homogeneous transformations. A dual quaternion can present rotation and translation in a compact form of transformation vector, concurrently. While the orientation of a body is described by nine elements in homogeneous transformations, the dual quaternions reduce the number of items to four. It offers a considerable advantage regarding computational robustness and storage efficiency for dealing with the kinematics of robot chains [15]. Robot kinematics can be divided into main classes, forward kinematics, and inverse kinematics.

Forward kinematics problem is almost straightforward, and there is not any complexity deriving the equations. Hence, there is always a forward kinematics solution of a manipulator. It is the method for determining the orientation and position of the end effector (x,y,z) coordinates about the center of mass, given the joint angles and link lengths of the robot arm (servomotor rotation). This equation is deterministic. We always know the position of the foot by the servo angle.

Inverse kinematics is a much more challenging problem than forward kinematics. It is the procedure when we have a desired end effector position but need to know the joint angles required to achieve it. The solution of the inverse kinematics is computationally expensive, and usually, it takes long time for the real time control of manipulators. Singularities and nonlinearities that make the problem harder to solve. Hence, only for a slight class of kinematically simple manipulators (manipulators with Euler's wrist) have complete analytical solutions.

The Forward Kinematics is not very helpful because having given a change of the servo's angle, only one effector moves in the chain. However, if we are given a change of the coordinates, the whole chain of effectors (servos) have to turn a certain angle to reach the desired position [16]. Moreover, also the movement tend to be more natural as well! Balance can be defined as the robots center of mass (affectionately referred to as its center of gravity) being its center of pivots (i.e. the edges of where its feet contact the ground). If the center of mass is above the center of pivots and between them, the robot will balance (almost an unstable equilibrium, if you are an applied mathematician. If the center of mass is above but outside the center of pivots, the robot will overbalance and fall. Two main solution procedures for the inverse kinematics problem are analytical and numerical methods. In the first solution, the joint variables are solved analytically according to given configuration. In the second solution, the joint variables are obtained based on the numerical techniques. In this section, we examinedthe analytical solution of the manipulators rather than the numerical solution. There are two approaches in the analytical method: geometric and algebraic solutions. The geometric approach is usually applied to simple robot structures, such as DOF manipulator with parallel joint axes or 2-DOF planar manipulator. For the manipulators with more links extend into three dimensions or more the geometry gets much more tedious. In this case, the algebraic approach is more beneficial for the inverse kinematics solution. There are some difficulties to solve the inverse kinematics problem when the kinematics equations are coupled, and multiple solutions and singularities exist. Mathematical solutions for inverse kinematics usually are not corresponding to the physical solutions and method of its solution depends on the robot structure.

In this pentapod robot, it is not advised control the legs directly, rather than controlling the position of the body. Any change of the body's position is translated into changes of each leg position. By using Inverse Kinematics, we can work out the angles of each servo, and the robot moves the way we wish. For that reason the implementation is separated into two main parts:

- Body Inverse Kinematic
- Leg Inverse Kinematic

The principle behind Body Inverse Kinematic is that if we move the centre of the body, it would change the coordinates of the feet, therefore it will change the servo angles. In Airplanes, generally there are 3 types of body movements:

- Roll
- Pitch
- Yaw

There is one more type of body movement for a legged robot, the translation (moving horizontally on the X, Y plane). For example, if we move the robot's body to the left, the end-effector of feet coordinates would shift to the left of the same distance from the center of the body.

The feet are stationary, which means that even if we move the body's position, the absolute positions of the feet will not change, but their relative positions to the center of the body have changed. Consequently, we need to take those changes into account and calculate the latest relative coordinates by using the body's I.K. procedure and then pass that results to leg's I.K. algorithm to calculate new servo angles.

Unfortunately, this Inverse Kinematics algorithm cannot support 3D coordinates (x,z,y). We can find the change in each dimension, and calculate the change of angles. Until now each dimension is calculated using very basic trigonometric equations and ideas, this mathematical model works, but it is not close enough to reality, producing a rough and unnatural movement. The reasons behind this results are described below:

- When calculating body rotation (roll and pitch), the usage of the trigonometric function tan to estimate change in y axis, but x and/or z will change as well. Even if this angle change is relative small we have to include some calculations to produce more accurate/natural movements by using a more advance and complex way to calculate change of coordinates from rotations Rotation Matrix

2.5.1 Geometric Solution Approach

Geometric solution approach is based on the decomposition of the spatial geometry of the manipulator into several other plane geometry problems. It is applied to the simple robot structures, such as, 2-DOF planer manipulator whose joints are both revolute and link lengths are l1 and l2. The components of the point P (px and py) are determined as follows.

$$p_x = l_1 c\vartheta_1 + l_2 c\vartheta_{12} \tag{2.1}$$

$$p_y = l_1 s\vartheta_1 + l_2 s\vartheta_{12} \tag{2.2}$$

where $c\vartheta_{12} = c\vartheta_1 c\vartheta_2 - s\vartheta_1 s\vartheta_2$ and $s\vartheta_{12} = s\vartheta_1 c\vartheta_2 + c\vartheta_1 s\vartheta_2$. The solution of ϑ_2 can me computed from summation of squaring both previous equations.

$$p_x^2 = l_1^2 c^2\vartheta_1 + l_2^2 c^2\vartheta 12 + 2l_1 l_2 c\vartheta_1 c\vartheta 12 \tag{2.3}$$

$$p_y^2 = l_1^2 s^2\vartheta_1 + l_2^2 s^2\vartheta 12 + 2l_1 l_2 s\vartheta_1 s\vartheta 12 \tag{2.4}$$

$$p_x^2 + p_y^2 = l_1^2(c^2\vartheta_1 + s^2\vartheta_1) + l_2^2(c^2\vartheta_{12} + s^2\vartheta_{12}) + 2l_1 l_2(c\vartheta_1 + c\vartheta_{12} + s\vartheta_1 + s\vartheta 12) \tag{2.5}$$

Since $c^2\vartheta_1 + s^2\vartheta_1 = 1$, the equation given above is simplified as follows.

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2(c\vartheta_1[c\vartheta_1 c\vartheta_2 - s\vartheta_1 s\vartheta_2] + s\vartheta_1[s\vartheta_1 c\vartheta_2 + c\vartheta_1 s\vartheta_2]) \tag{2.6}$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2(c^2\vartheta_1\vartheta_2 c\vartheta_2 - c\vartheta_1 s\vartheta_1 s\vartheta_2 + s^2\vartheta_1 c\vartheta_2 + c\vartheta_1 s\vartheta_1 s\vartheta_2) \tag{2.7}$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2(c\vartheta_2[c^2\vartheta_1 + s^2\vartheta_1]) \tag{2.8}$$

$$p_x^2 + p_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 c\vartheta_2 \tag{2.9}$$

$$c\vartheta_2 = \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2} \tag{2.10}$$

Since, $c^2\vartheta_1 + s^2\vartheta_1 = 1(i = 1, 2, 3......), s\vartheta_2$ is obtained as

$$s\vartheta_2 = \pm\sqrt{1 - (\frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2})^2} \tag{2.11}$$

Which lead us to the following two possible solutions for ϑ_2

$$\vartheta_2 = \text{atan2}\,(\pm\sqrt{1 - (\frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2})^2}, \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2}) \tag{2.12}$$

2.5.2 Algebraic Solution Approach

Unfortunately, for manipulators with more links and whose arm the extension into three dimensions creates a much more tedious geometry. Hence, the algebraic approach is chosen for the inverse kinematics solution. By recalling the previous equations we can find the inverse kinematics solution for a six-axis manipulator.

$$T_6^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_1^0(q_1)T_2^1(q_2)T_3^2(q_3)T_4^3(q_4)T_5^4(q_5)T_6^5(q_6) \tag{2.13}$$

To find the inverse kinematics solution for the first joint (q_1) as a function ofthe known elements of $T_{end-effector}^{base}$, the link transformation inverses are premultiplied as follows.

$$^{-1}T_6^0 = [T_1^0(q_1)]^{-1}T_1^0(q_1)T_2^1(q_2)T_3^2(q_3)T_4^3(q_4)T_5^4(q_5)T_6^5(q_6) \tag{2.14}$$

where $[T_1^0(q_1)]^{-1}T_6^0 = I$ (identity matrix). In that case the above equation in given by

$$^{-1}T_6^0 = T_2^1(q_2)T_3^2(q_3)T_4^3(q_4)T_5^4(q_5)T_6^5(q_6) \tag{2.15}$$

To find the other variables, the following equations are obtained as a similar manner.

$$^{-1}T_6^0 = T_3^2(q_3)T_4^3(q_4)T_5^4(q_5)T_6^5(q_6) \tag{2.16}$$

$$^{-1}T_6^0 = T_4^3(q_4)T_5^4(q_5)T_6^5(q_6) \tag{2.17}$$

$$^{-1}T_6^0 = T_5^4(q_5)T_6^5(q_6) \tag{2.18}$$

$$^{-1}T_6^0 = T_6^5(q_6) \tag{2.19}$$

There are 12 concurrent sets of nonlinear equations to be solved. Fortunately, the only unknown on the left-hand side of equation 18 is q_1. The 12 nonlinear matrix elements of right-hand side are either zero, constant or functions of q_2 through q_6. If the elements on the left hand side which are the function of q_1 are equated with the elements on the right hand side, then the joint variable q_1 can be solved as functions of $r_{11}, r_{12}, r_{33}, p_x, p_y, p_z$ and the fixed link parameters. Once q_1 is found, then the other joint variables are solved by the same way as before. There is no necessity that the first equation will produce q_1 and the second q_2 etc. To find a suitable equation for the solution of the inverse kinematics problem, any equation defined above can be used arbitrarily. Some trigonometric equations applied in the solution of inverse kinematics problem are presented in the following table.

	Equations	Solutions
1	$a\sin(\vartheta) + b\cos(\vartheta) = c$	$\vartheta = \text{atan2}(a, b) \pm \text{atan2}(\sqrt{a^2 + b^2 - c^2}, c)$
2	$a\sin(\vartheta) + b\cos(\vartheta) = 0$	$\vartheta = \text{atan2}(b, -a)$
3	$\cos(\vartheta) = a \sin(\vartheta) = b$	$\vartheta = \text{atan2}(b, a)$
4	$\cos(\vartheta) = a$	$\vartheta = \text{atan2}(\pm\sqrt{(1 - a^2)}, a)$
5	$\sin(\vartheta) = a$	$\vartheta = \text{atan2}(a, \pm\sqrt{(1 - a^2)})$

Table 2.1: IK trigonometric equations

2.5.3 Implementation of Inverse Kinematics

All the previously mentioned methodologies are applied to produce the correct angle of each servomotor of the robotic platform. The following equations present the implementation of inverse kinematics for a five-legged robot. For each leg the algorithm has to complete the following calculations.

First of all we have to calculate the updated total distance of the end-effector (2.20, 2.21, 2.22).

$$\begin{gathered} IKLeg_i.totalX = positionBodyOffset.X + coxaOffest_i.X \\ +positionInitLeg_i.X + positionOffsetLeg_i.X \end{gathered} \tag{2.20}$$

$$\begin{gathered} IKLeg_i.totalY = positionBodyOffset.Y + coxaOffest_i.Y \\ +positionInitLeg_i.Y + positionOffsetLeg_i.Y \end{gathered} \tag{2.21}$$

$$IKLeg_i.totalZ = positionInitLeg_i.Z + positionOffsetLeg_i.Z \tag{2.22}$$

Afterwards, we need to calculate the initial rotation of each leg R = Rx, Ry, Rz (2.23-2.28)

$$IKLeg_i.sinRotX = \sin(rotOffsetBody.X) \tag{2.23}$$

$$IKLeg_i.cosRotX = \cos(rotOffsetBody.X) \tag{2.24}$$

$$IKLeg_i.sinRotY = \sin(rotOffsetBody.Y) \tag{2.25}$$

$$IKLeg_i.cosRotY = \cos(rotOffsetBody.Y) \tag{2.26}$$

$$IKLeg_i.sinRotZRotZ = \sin(rotOffsetBody.Z) \tag{2.27}$$

$$IKLeg_i.cosRotZRotZ = \cos(rotOffsetBody.Z) \tag{2.28}$$

Now, we will calculate the body's position X,Y,Z axis using the tranformation matrix (2.29-2.31)

$$\begin{gathered}IKLeg_i.bodyIKX = IKBLeg_i.totalX * IKBLeg_i.cosRotY * IKBLeg_i.cosRotZRotZ \\ -IKBLeg_i.totalY * IKBLeg_i.cosRotY * IKBLeg_i.sinRotZRotZ \\ +IKBLeg_i.totalZ * IKBLeg_i.sinRotY - IKBLeg_i.totalX\end{gathered} \tag{2.29}$$

$$\begin{gathered}IKLeg_i.bodyIKY = IKBLeg_i.totalX * IKBLeg_i.cosRotY * IKBLeg_i.sinRotZRotZ \\ +IKBLeg_i.totalX * IKBLeg_i.sinRotY * IKBLeg_i.cosRotZRotZ * IKBLeg_i.sinRotX \\ +IKBLeg_i.totalY * IKBLeg_i.cosRotZRotZ * IKBLeg_i.cosRotX \\ -IKBLeg_i.totalY * IKBLeg_i.sinRotZRotZ * IKBLeg_i.sinRotY * IKBLeg_i.sinRotX \\ -IKBLeg_i.totalZ * IKBLeg_i.cosRotY * IKBLeg_i.sinRotX) - IKBLeg_i.totalY\end{gathered} \tag{2.30}$$

$$\begin{gathered}IKBLeg_i.bodyIKZ = (IKBLeg_i.totalX * IKBLeg_i.sinRotX * IKBLeg_i.sinRotZRotZ \\ -IKBLeg_i.totalX * IKBLeg_i.cosRotZRotZ * IKBLeg_i.cosRotX * IKBLeg_i.sinRotY \\ +IKBLeg_i.totalY * IKBLeg_i.cosRotZRotZ * IKBLeg_i.sinRotX \\ +IKBLeg_i.totalY * IKBLeg_i.sinRotZRotZ * IKBLeg_i.sinRotY * IKBLeg_i.cosRotX \\ +IKBLeg_i.totalZ * IKBLeg_i.cosRotY * IKBLeg_i.cosRotX) - IKBLeg_i.totalZ\end{gathered} \tag{2.31}$$

Here we will calculate the leg's position X,Y,Z axis using the tranformation matrix (2.32-2.34)

$$\begin{gathered}IKLLeg_i.transfX = (posCurrentLeg_i.X * \cos((i * 72))) \\ -(posCurrentLeg_i.Y * \sin((i * 72)))\end{gathered} \tag{2.32}$$

$$IKLLeg_i.transfY = posCurrentLeg_i.X * \sin((i * 72))$$
$$+posCurrentLeg_i.Y * \cos((i * 72)) \tag{2.33}$$

$$IKLLeg_i.transfZ = posCurrentLeg_i.Z \tag{2.34}$$

One step before calucating the angles of each link(servo) we have to find the Coxa Distance as well as apply the trogonometric equations provided at the Table 2.5.2 to produce the rotation of the end-effector (2.35-2.42)

$$IKLLeg_i.coxaFeetDist = \sqrt(IKLLeg_i.transfX * IKLLeg_i.transfX$$
$$+IKLLeg_i.transfY * IKLLeg_i.transfY) \tag{2.35}$$

$$IKLLeg_i.IKSW = \sqrt((IKLLeg_i.coxaFeetDist - dimensions.lengthCoxa)$$
$$(IKLLeg_i.coxaFeetDist - dimensions.lengthCoxa)$$
$$+IKLLeg_i.transfZ * IKLLeg_i.transfZ) \tag{2.36}$$

$$IKLLeg_i.IKA1 = atan((IKLLeg_i.coxaFeetDist - dimensions.lengthCoxa)$$
$$/IKLLeg_i.transfZ) \tag{2.37}$$

$$IKLLeg_i.IKA2 = acos((dimensions.lengthTibia * dimensions.lengthTibia$$
$$-dimensions.lengthFemur * dimensions.lengthFemur$$
$$-IKLLeg_i.IKSW * IKLLeg_i.IKSW)$$
$$/(-2 * IKLLeg_i.IKSW * dimensions.lengthFemur)) \tag{2.38}$$

$$IKLLeg_i.tAngle = acos((IKLLeg_i.IKSW * IKLLeg_i.IKSW$$
$$-dimensions.lengthTibia * dimensions.lengthTibia$$
$$-dimensions.lengthFemur * dimensions.lengthFemur)$$
$$/(-2 * dimensions.lengthTibia * dimensions.lengthFemur)) \tag{2.39}$$

Finally, we are able to calculate the angles of each servo.

$$angLeg_i.tibia = 90 - IKLLeg_i.tAngle * 180/PI \tag{2.40}$$

$$angLeg_i.femur = 90 - (IKLLeg_i.IKA1 + IKLLeg_i.IKA2) * 180/PI \tag{2.41}$$

$$angLeg_i.coxa = atan2(IKLLeg_i.transfX, IKLLeg_i.transfY) * 180/PI \tag{2.42}$$

2.5.4 GAIT cycle for five-legged robot

The gait cycle begins when a foot makes contact with the terrain and stops when that same foot contacts the ground again. Each cycle is broken down into several phases and periods to determine normative and abnormal gait. Legged robots usually utilize a collection of gait patterns to locomote over a variety of surfaces. Each feedforward gait is modified for a specific surface and set of operating conditions. To facilitate locomotion across a changing surface, a robot must be able to change stably between gaits while continuing to locomote [17].

There are at least three distinct approaches to the problem of motions generation for legged robots. The first one is to design a reactive system whose emergent behavior resembles the desired motion. While it is possible to encode the stability within a before-mentioned system, it is challenging to design the outgoing behavior, and these systems often lack common intuition. A second, opposite approach is to plan the individual motions for each leg and feet of a robot while ensuring the stability of the resulting overall motion. Legged locomotion, however, necessarily involves complex and often nonholonomic constraints relating to surface contact and conservation of momentum, making the planning approach very tough to implement.

As mentioned before, gait is a cyclic pattern that produces locomotion through a sequence of foot contacts with the terrain. The legs provide support for the body while the forces emerging from ground contact propel the robot. Gaits can be different in a variety of ways, and different gaits usually produce several styles of locomotion. The basic building block to define gaits is a motion pattern. It is a mapping from phase, a scaled version of time, to the desired robot configuration. For a gait, this function maps from phase space, P, to the configuration space of the whole robot, Q. Gaits are typically cyclic, so the domain, P, is topologically similar to the unit circle, S_1. If G is the space of all possible gaits, then a gait $g \subset G$ is a periodic, constant, and injective function from phase angle to desired robot configuration.[17]

On a five-legged robotic platform, the configuration space, Q, is naturally thought of as the Cartesian product of the individual configuration spaces for each leg, Q_i. Thus, it can be rewritten as a collection of functions, one for each leg.

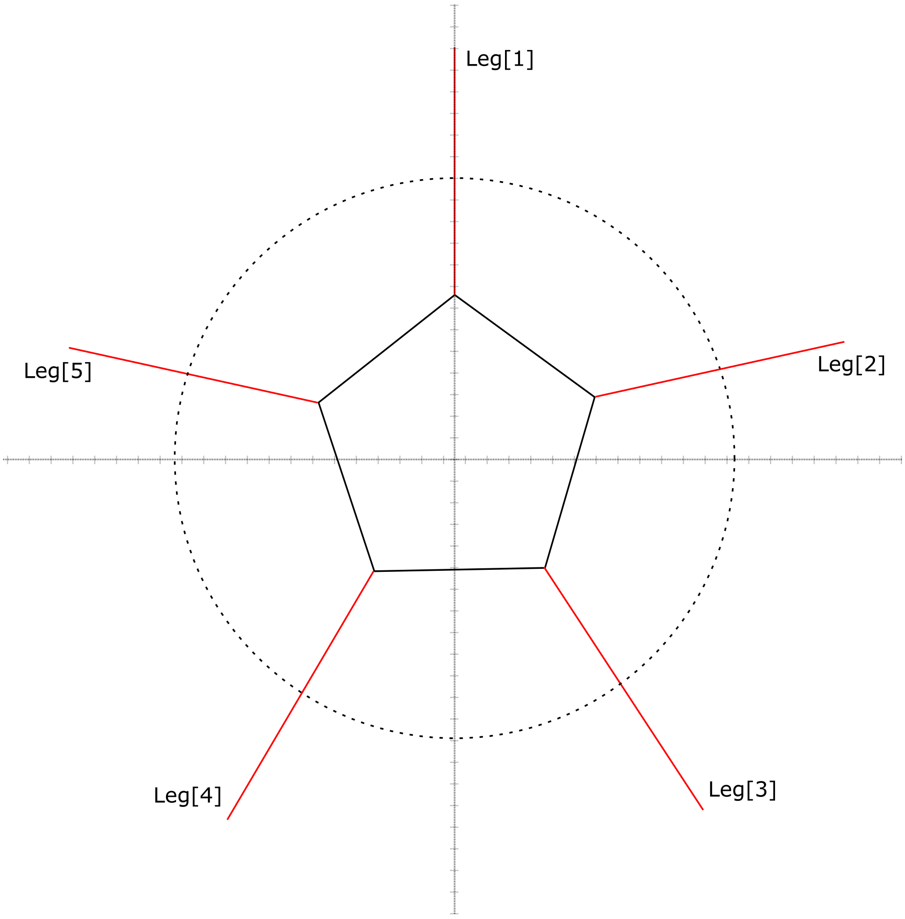


Figure 2.9: Illustration of the five-legged robotic platform used in GAIT sequencer

$$g : P \to Q_4 \times Q_2 \times Q_5 \times Q_3 \times Q_1 \tag{2.43}$$

Gait Leg Sequence				
4	2	5	3	1

Table 2.2: Gait Leg Sequence

While this description of a gait produces the desired motion patterns the legs, an important distinction must be made between the leg that is touching the ground, termed in stance, and the leg that is recirculating in the air, in flight. During stance, a leg pushes against the ground, generating forces that move the robot forward. A leg in flight returns to the configuration where stance starts again, completing a cycle of the gait.

The sequence in which legs start stance, as well as a count of the number of legs in stance, reveals much information about the structure of a gait.

Ignoring the exact spatial trajectory for each leg, there are useful parameters for describing gait exist in the timing of events in these trajectories. Those parameters provide a kind of semantic information about a gait, indicating the kind of gait, when certain legs undergo stance, as well as whether or not the gait will be accurate, providing proper support for the body of the robot as it locomotes. The following table provides the actual parameters used in the robotic platform.

Gait Setup	**Value(mm)**
LegLiftHeight	-25
TravelLengthX	35
TravelLengthZ	0
TravelRotationY	0

Table 2.3: Gait Setup

In brief, each leg is picked and moved forward during its own quarter-phase, and then moves backward during the other three quarter-phases. The overall operation results are very smooth, including even forward movement because all legs are in perpetual motion here. The robot's body remains nice and level. The stability of the body is related to the frequency of the legs being raised and placed, the faster, the less unstable will be. Of course, it is has something to do with the design of the feet as well, if the feet has a large contact with the ground it will stand much better.

2.5.5 Naturalizing the motion

One major improvement that we had to develop is the easeInOut transition. Objects in real life dont just start and stop instantly, and almost never move at a constant speed.

The simplest way of implementing this is to change the objects position at fixed intervals over time so that it appears to move from A to B as the clock ticks from 0 to T. This is known as linear motion and while it works it appears unnatural because in the natural world objects accelerate and decelerate as they move away from and towards their resting positions.

To simulate acceleration and deceleration on each movement of the platform we apply a transformation to each servo's position as it moves to ease it in and out of its resting positions. The result is a natural looking animation that is pleasing to watch and looks slick and professional. Even if there exist a variation of easeInOut algorithms, in our case the most suitable result is produced by the Quadratic easeInOut algorithm.

2.5.6 Demonstration of the IK Algorithm

In this subsection, we present all the above algorithms implemented in the five-legged robotic platform. Unfortunately due to the servo motors angle precision is limited to 1.8 degrees which is causing the results to be not perfect but acceptable.

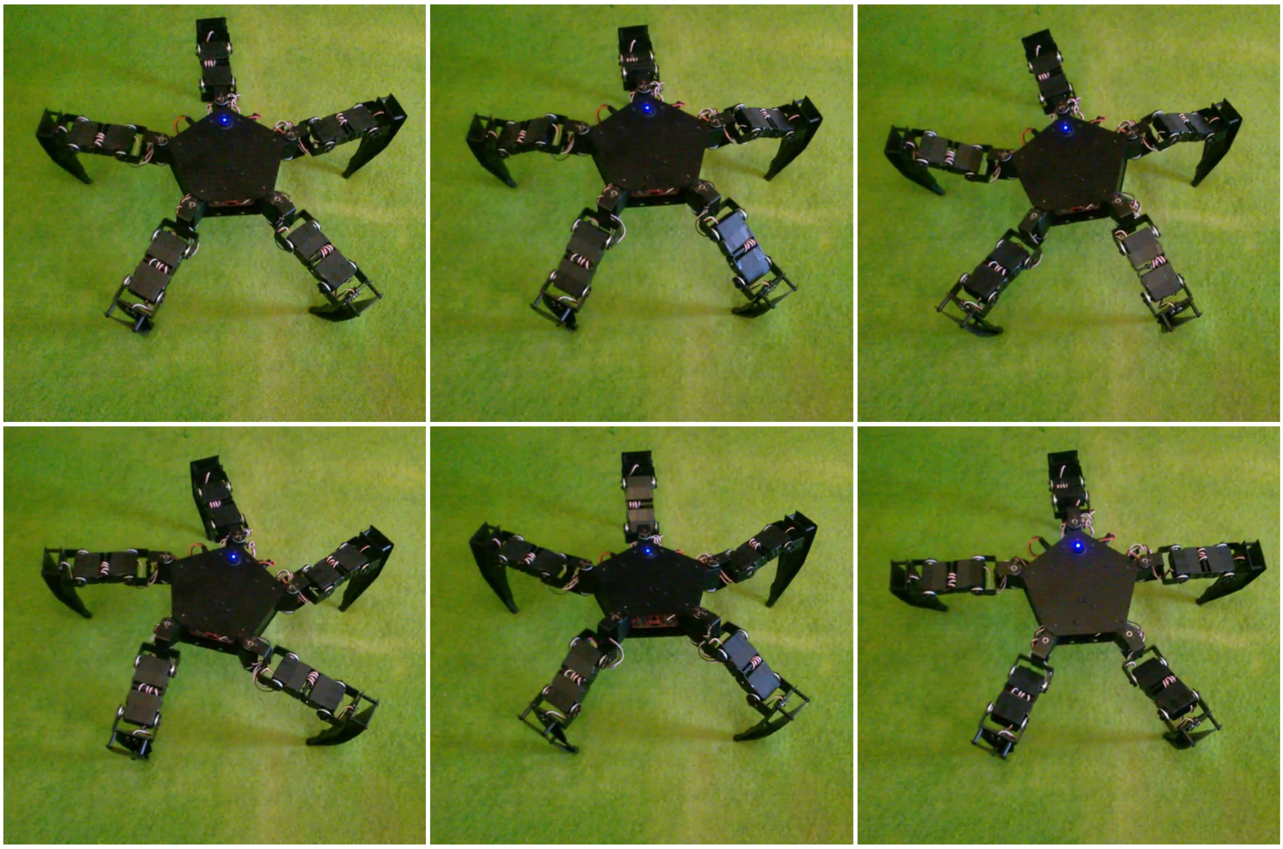


Figure 2.10: Demonstration of the IK Algorithm

Chapter 3

Software Design and Implementation

3.1 Introduction

In this chapter, we will discuss the overall software requirements and design. An analysis of concepts and techniques for design and construction of a reliable and maintainable software system will be presented as well as the program structure and design; program-correctness approaches, including testing; and event-driven programming. In details, we will discuss the real-time operating system, some necessary additions that were made and the underlying structure of the applications needed. Moreover, we will present an extended study of the memory management scheme that we are using to provide a flawless stable system operation as well as the communication protocol and the sensory system.

3.2 Real-Time Operating System

Most of the control systems, including sensor based control system used in robotics and industrial automation, have a realtime functionality and are based on real time operating systems (RTOSs). As it is well known, a real-time systems correct functionality is not given only by the correct outcome of the system but also by its correct timing functionality (i.e. all the time constraints must be met). We note the difference between hard real-time (HRT) constraints and soft real-time (SRT) constraints. While for the SRT applications a time constraint violation has an impact only in the degradation of the outcome quality (e.g. the loss of a number of frames in a video processing application leads to image quality degradation), for the HRT applications, the violation of a time constraint has catastrophic impacts (e.g. an overcome of the maximum allowed response time of an airbag system can cause loss of human lives)

In general, an operating system (OS) is responsible for managing the device resources of a computer and hosting tasks on the computer. An RTOS performs these tasks but is also specifically designed to run applications with very precise timing and a high degree of reliability. That can be especially important in analysis and automation systems where downtime is harmful, or a program delay could cause a safety risk. To be considered as ”real-time,” an operating system must have a known maximum delay time for each of the critical procedures that it performs (or at least be able to guarantee that maximum most of the time). Some of these procedures include OS calls and interrupt handling. Operating systems that can unconditionally guarantee a maximum time for these procedures are usually referred to as ”hard real-time,” while operating systems that only guarantee a maximum most of the time are indicated as ”soft real-time.” In practice, these strict categories have limited usefulness - each RTOS solution demonstrates unique performance attributes and the user should carefully investigate these features. To fully understand these concepts, it is helpful to consider an example. Imagine that you are designing an airbag system for a new model of car. In this case, a tiny error in timing could be catastrophic and cause injury. Therefore, a hard real-time system is needed; you need assurance as the system designer that no particular operation will exceed certain timing constraints. On the other hand, if you were to design a mobile phone that received streaming video, it may be acceptable to lose a small amount of data occasionally even though on average it is important to keep up with the video stream. For this application, a soft real-time operating system may suffice.

The main point is that, if programmed perfectly, an RTOS can guarantee that a program will be executed with very consistent timing. Real-time operating systems do this by giving programmers a high degree of control over the priority of tasks, and typically allow checking to make sure that critical deadlines are reached.

Nowadays, there exist several RTOSs used in different kind of applications. s-OS, eCos, Windows CE and RT-Linux are just a few RTOSs currently available. The selection of the RTOS to implement the proposed HST was based mainly on:

- Portability of the RTOS among different embedded systems,
- Availability of well documented source code and

- License to introduce modifications into the source code.

With these criteria in mind, we selected FreeRTOS which is a small footprint and widely used RTOS. FreeRTOS is an open source RTOS distributed under a modified GPL license. It is written in C, with small sections written in assembler. Its main features are its small footprint, its low memory and processing requirements and its availability in more than 33 processor architectures. FreeRTOS implements tasks as threads, each with its stack of configurable size, and supports processors with Memory Protection Unit (MPU). It implements multiple memory profiles, from static to dynamics ones with the support of malloc and free functions. FreeRTOS provides a preemptive scheduler, with a FIFO for each priority level. The scheduler guarantees that, for any given instant, it will execute the highest priority task in the ready list. If multiple tasks with the same priority are ready, a Round Robin policy may be enabled among them. The priority of a task is assigned by the designer when created, and can be modified at runtime.

3.2.1 Real-Time Applications as Tasks

A real-time application that uses an RTOS can be structured as a set of independent tasks. Each task executes within its context with no unplanned dependencies on other tasks within the system or the RTOS scheduler itself. Only one task within the application can be executing at any point in time, and the real-time RTOS scheduler is responsible for deciding which task this should be. The RTOS scheduler may therefore repeatedly start and stop each task as the application executes. Because of the task that has no knowledge of the RTOS scheduler activity, it is the responsibility of the real-time RTOS scheduler to ensure that the processor context (register values, stack contents, etc.), when a task is swapped in, is precisely that as the swapping out. To accomplish that, each task is provided with its stack. When the task is swapped out, the execution context is stored into the stack of the task so it can be exactly restored later on when the task is swapped back in. Each task can exist in one of the following states: Running, Ready, and Suspended.

3.2.2 Improved Task Control Block (TBC)

The Task Control Block (TCB) defined by FreeRTOS does not include attributes to hold particular parameters for generic task models as the proposed in this thesis. For this reason, it was defined an extended TCB, denoted improvedTCB, based on the attributes that a task in a Mixed Critical System may present. Moreover, each improvedTCB has a reference to one TCB to keep functional the FreeRTOS tasks control functions. The improvedTCB for each task holds the following parameters/States:

- A memory pointer to additional parameters
- An identifier for the task type (DRIVER, COREAPP, SMPLAPP)
- An identifier for the task state (INACTIVE, MEMREGISTER, INITIALIZE, READY, RUNNING, STOPPED)
- An identifier for the execution mode of each task (AUTO, MANUAL)

The purpose of this additional control block is the creation of a structured execution model as well as to provide a control system of the SIMPLE APPLICATIONs.

3.2.3 Inter-task Communication standards

Except the improvedTCB the overall software design includes some Inter-task Communication standards. Queues are the primary form of intertask communications. They can be used to send messages between tasks, and between interrupts and tasks. In most cases they are used as thread safe FIFO (First In First Out) buffers with new data being sent to the back of the queue, although data can also be sent to the front.

Binary semaphores are used for both mutual exclusion and synchronisation purposes. Binary semaphores and mutexes are very similar but have some subtle differences: Mutexes include a priority inheritance mechanism, binary semaphores do not. This makes binary semaphores the better choice for implementing synchronisation (between tasks or between tasks and an interrupt), and mutexes the better choice for implementing simple mutual exclusion.

Mutexes are binary semaphores that include a priority inheritance mechanism. Whereas binary semaphores are the better choice for implementing synchronisation (between tasks or between tasks and an interrupt), mutexes are the better choice for implementing simple mutual exclusion (hence 'MUT'ual 'EX'clusion). When used for mutual exclusion the mutex acts like a token that is used to guard a resource. When a task wishes to access the resource it must first obtain ('take') the token. When it has finished with the resource it must 'give' the token back - allowing other tasks the opportunity to access the same resource.

3.2.4 Implementation Details

Both microcontrollers are using the same implementation. However each one has it's own tasks and priorities. For example, the servocontroller is using 1 DRIVER application (CANController), 3 SIMPLE applications (Sequencer,Gait,Messenger) and one CORE application (Cortex). In contradiction the Governor is using 4 DRIVER applications (CANController, XBee, SDCard , IMU 9/10dof) 1 SIMPLE application (Messenger), and 1 CORE application (Governor).

3.3 Adaptive Memory Management

The RTOS kernel needs RAM each time a task, queue, mutex, software timer, semaphore or event group is created. The RAM can be automatically dynamically allocated from the RTOS heap within the RTOS API object creation functions. Dynamic memory allocation is a process that allows a program to distribute efficiently its memory space in situations of unpredictable events that need to be stored in memory due to unknown inputs.[18] Using dynamic memory allocation, while a program is running provides the ability to request more memory from the system. If there is enough memory available, the system will grant the program the right to use the amount it requests. However, as it previously mentioned, in some situations due to multiple allocation and deallocation actions with different sizes, dynamic memory allocation leads to some critical side effects such as internal and external fragmentation. Resulting in unexpected memory failures of the system even if the total available space is sufficiently large to fulfill the requirement . Both internal and external fragmentation forms are usually referenced as wasted memory.[19]

In this context, optimization techniques for efficient and adaptive memory management and data assignment have to be solved. In the literature, there exists an extensive number of references to this particular issue however this issue has not been solved for all application types, especially for the domain of real-time applications.

Usually, most developers are trying to avoid the use of dynamic memory at all for this reason, as it is mentioned in I. Puaut et al. [20] review, but it contradicts with the requirements that Autonomous Intelligent Devices have. Nonetheless, the use of dynamic memory in real-time embedded devices is important to be deterministic; the time was taken to allocate memory should be predictable, and the memory pool should not become fragmented. In this chapter, we introduce an another approach of Adaptive Memory Management, called hereby AMM, which mainly focuses on small embedded systems where memory is limited, and there is no MMU hardware support such as the ARM Cortex M4 microcontroller.

3.3.1 Forms of Memory Fragmentation

But what is exactly the Internal and External Memory Fragmentation? Using dynamic memory allocation while a program is running, provides the ability to request more memory from the system. If there is enough memory available, the system will grant the program the right to use the amount it requests. However, as it previously mentioned, in some situations due to multiple allocation and deallocation actions with different sizes, dynamic memory allocation leads to some critical side effects such as internal and external fragmentation, which can result in unexpected memory failures of the system even if the total available space is sufficiently large to fulfill the requirement. Both internal and external fragmentation forms are usually referenced as wasted memory. However, it is worthy to analyze the difference between those two. Internal fragmentation is a form of fragmentation that arises when allocation of memory is done only in multiples of a sub-unit. A request of arbitrary size is served by rounding up to the next highest multiple of the sub-unit, leaving a small amount of memory that is allocated but not in use. Internal fragmentation can be decreased by reducing the size of the sub-unit; such a reduction increases though external fragmentation. In contrast, external fragmentation is a form of fragmentation that arises when memory is allocated in units of arbitrary size. When a large amount of memory is released, part of it may be used to satisfy a subsequent request, leaving an unused part that is too small to serve any further requests.

Figure 3.1 illustrates three different fragmentation states that may occur in a system using dynamic allocations. In the first one (a), a best-case scenario is presented where there is no memory fragmentation, however in the second use case (b) it is easily observed that a sort of fragmentation occurs in the heap memory section. Finally, the last use case (c) presents a situation that leads to memory failure due to unavailable free memory space that may occur by fragmentation, which is considered as a potential issue in modern embedded systems development. However, as Johnstone et al. shows, well-designed allocators can efficiently manage fragmentation problems. In this direction since development kits that rely on a system heap can result in memory fragmentation, with a negative impact to system performance, industry promotes thread-free library with an integrated memory manager for a zero-heap solution for IoT devices.

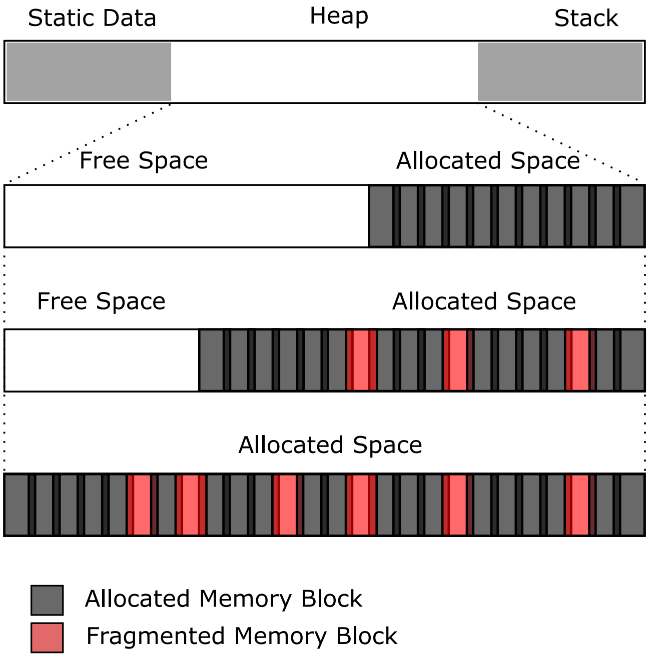


Figure 3.1: System memory time instances with diverse fragmentation states

Usually, an approach for avoiding memory fragmentations of the system with such a tendency is the use of system equipped with a memory management unit (MMU) alongside with a dynamic memory management algorithm. A memory management unit is a hardware component that handles all memory operations associated with the virtual memory handling. The foremost goal of a memory management unit is to provide a convenient abstraction using virtual addresses to ensure the availability of adequate memory resources. In other words, MMU is a hardware part that translates a virtual address to physical address. MMU-equipped embedded systems can remap the fragmented memory spaces into a whole large block or even perform defragment operations to merge those pieces into a continuous physical block. Therefore, fragmentation problems only cause performance issues. However, on MMU-less embedded systems this issue is complicated due to lack of support for virtual addresses, hence remapping.

3.3.2 Related Work

Researchers have anticipated several different approaches to solve the fragmentation problem in MMU-less embedded systems.Each algorithm is classified according to the way that it finds a free block of the most appropriate size. As analyzed in Masmano el al. [16], Sun et al.[21] and Wilson [22], these algorithms can be categorized in: Sequential Fit, Segregated Free Lists, Buddy Systems, Indexed Fit and Bitmap Fit.

Buddy systems approach attempts to split the available memory into two same-sized blocks respecting the efficiency issue and increasing the overall performance. This method usually creates fragmented memory parts after extended use of allocation and deallocation processes. In the buddy system, the memory is broken down into power-of-two sized naturally aligned blocks [23]. This approach greatly reduces external fragmentation of memory and helps in allocating bigger continuous blocks of memory aligned to their size. On the other hand, the buddy allocator suffers increased internal fragmentation of memory and is not suitable for general kernel allocations. This purpose is better addressed by the slab allocator.

Slab allocator, creates different sized blocks and matches the requested allocation to the closest one. The majority of memory allocation requests in the kernel is for small, frequently used data structures. The basic idea behind the slab allocator is that commonly used objects are pre-allocated in continuous areas of physical memory called slabs. Whenever an object is to be allocated, the slab allocator returns the first available item from a suitable slab corresponding to the object type. Because the sizes of the requested and allocated block match, the slab allocator significantly reduces internal fragmentation [24]. The advantage of this setup is that during most of the allocations, no global spinlock needs to be held. CAMA [24], which is a research follow-up of slab allocator, has a better performance by splitting and merging the block accordingly.

Two-Level Segregated Fit [16] (TLSF algorithm) seeks to implement a good-fit policy in order to fulfill the most important real-time requirements. The basic segregated fit mechanism uses an array of free lists, with each array holding free blocks within a size class. In order to speed-up the access to the free blocks and also to manage a large set of segregated lists, the array of lists is organized as a two-level array. The first-level array divides free blocks into classes that are a power of two apart (16, 32, 64, 128, etc.); and the second-level sub-divides each first-level class linearly, where the number of divisions is a user configurable parameter. Each array of lists has an associated bitmap used to mark which lists are empty and which ones contain free blocks.

MMU-Less Defragmentable Allocation Schema [25] has a significantly better performance in memory consistency by respecting the fact that each block should be moveable to avoid fragmentation. This approach attempts to mimic the hardware MMU operations by using a different allocation method. By using the address of the pointer as extra information in the descriptor of each block, MMU-Less Defragmentable Allocation Schema can move allocated blocks and update the referred address pointer to the new address space. However, this approach does not provide all the necessary characteristics to support more complex memory blocks such as linked lists, which makes it non-usable for many real-world applications. In addition to those approaches, there are many cases of application programs that include an extra memory management code called sub-allocator. A sub-allocator is an allocator functioning on top of another allocator. It usually obtains large blocks of memory from the system memory manager and allocates the memory to the application in smaller pieces. Sub-allocators are usually written to avoid the general inefficiency of the systems memory manager or to kate an advantage over the applications memory requirements that could not be expressed to the system memory manager. However, sub-allocators are less efficient than having a single memory manager that is well written and has an adjustable interface.

All the previously mentioned methodologies indeed decrease the memory fragmentation. However, they are bounded by the type of application and the system that will use them. In general, memory fragmentation without hardware MMU component is almost impossible to avoid, and usually a memory management scheme is developed according to the demands of the application that will use it. We should clarify that the proposed memory management scheme does not try to mimic a hardware MMU, nonetheless the goal is to provide fast and stable allocation algorithm enhanced with a defragmentation process for MMU-Less devices without the need of virtual addressing.

3.3.3 Memory Structure

As outlined before, there is not a single memory management scheme suitable for all application domains; algorithms for dynamic memory management can help but only for specific usage patterns. Real-time applications are quite different from conventional applications, and usually, each application requires a different memory management approach to achieve both optimal performance and stability. The objective of the proposed scheme is to minimize the memory fragmentation without sacrificing the overall speed of the system, while at the same time being able to support the requirements of applications from different domains.

In order to meet those requirements, the proposed memory management algorithm uses a combination of a modified TLSF methodology [16, 21] and the ability to move memory blocks[25]. AMM has been developed according to the following principles. First of all, the smallest memory block that the system can support including the blocks meta-data is 16 bytes, similar to previous research works [16, 21, 26, 27] Due to constraints of embedded systems due to limited available memory, it is not efficient to store extra information or to leave unused memory blocks for long periods. In the proposed memory management scheme, free fragmented memory blocks are removed immediately from the allocated space, by moving them to the border between allocated and free space.

This approach is only valid in embedded systems where the available memory is limited. The difference between the elapsed time which is required by the system to efficiently store a free block in a list and dispatch it, versus the elapsed time to move some occupied blocks, is almost insignificant. However, to obtain the maximum performance, the decision algorithm which is responsible for these actions has to provide an adaptive formation in order to eliminate un-necessary memory block movements. Usually, adaptation algorithms require a previously held calibration process for the needed parameters. For demonstration purpose, in this work the parameters are fixed values as defined in the algorithm.

In this scope, all new allocation requests are served as soon as possible. For each memory block request if there is available space, the request is served by inserting the allocated size of data and a pointer to its previous block (this information is called next as meta-data) and by returning a pointer to the corresponding memory address. The advantage of this process is to provide identical constant elapsed time; as shown in the next page, in Figure 3.2, it is constant and equals to three comparisons to successfully allocate a block. Essentially, this enables the use of dynamic allocation in time critical operations. At the same time, any method of the proposed memory management scheme should always check if defragmentation is needed in order to provide memory resiliency. In addition, the defragmentation process can be adaptive for keeping a balance between system resources and memory fragmentation level [21, 28, 22]. The algorithm can be configured and adapted in two ways. First, to control the margin between pointers and data in cases of high-frequency operations of dynamic allocation, since we reduce the need to do frequent re-organizations. Second, to control the percentage of the data section, either in the front or at the rear, where we perform the defragmentation of the blocks. For that reason, defragmentation is a series of small inexpensive steps in terms of time.

The memory data structure is a crucial part of the proposed scheme. In contrast to all other memory management approaches, our memory manager requires a different structure of how the meta-data and the blocks are stored in the memory. Figure 3.2 outlines the memory data structure organization. Pointers Section and Data Section are the main sections, which are separated by a small amount of free memory space, the Margin. The Pointers Section is responsible for storing the addresses of the pointers, which are associated with an allocated data block. In principle, an allocated data block can be referenced by many different variables. This pointed address of the pointers is updated when a block movement action is in progress. The system can actually save multiple pointers for each block in order to support more complex data structures in a user application than simple allocated blocks such as linked list, pointer of pointers etc.

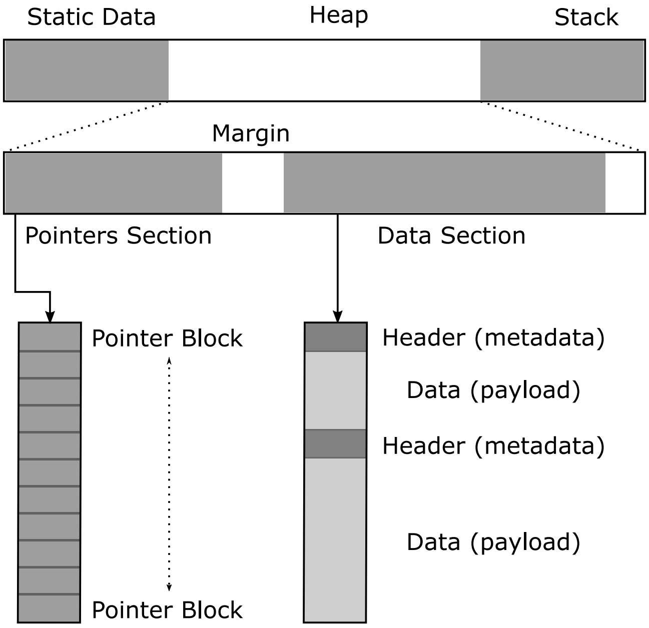


Figure 3.2: Organization of the memory structure

The Data Section is the memory space where the actual data blocks are stored. By providing almost the same API allocation / free methods as the default memory management scheme of LibC[29], developers are able to transit to the new memory management scheme easily. Moreover, a margin between those two sections is responsible to maintain an amount of free memory space for registering new pointer blocks. The optimal size of this margin can be configured for each application differently, since this affects the position of the Data Section and the number of operations for each allocation operation. If the configuration of the margin is not optimal then the system will try to adapt itself by moving some data-blocks from the beginning of the data section to the rear. Each allocated block has two main parts, the meta-data part, also known as block header and the data named hereby as payload. To be able to manage the memory blocks successfully, Adaptive Memory Management Scheme adds some useful information into the meta-data of each block. The header information contains a pointer to the previously physical occupied block and the size of the contained data. Each block size is always a multiple of four bytes to match word-length alignment and increase the system performance. Using the previous structure definitions, the proposed Adaptive Memory Management scheme provides the capability for supporting more forms of objects than simple allocations such as linked lists or even blocks of pointers. However, as it is clearly mentioned this allocation algorithm is optimized for embedded systems with limited memory space. To maintain the size of the margin to 32 bytes the memory management scheme can add a new block either in front or at the end of the Data Section. The Memory Management API provides one public function for allocation *memoryallocate* one public function for deallocation : *memoryfree* , two public functions for pointers : *memoryaddpointer*, *memoryremovepointer*, and three private functions for the management of the pointers section.

In order to create a request for a new allocation, the user has to call *memoryallocate*. This method takes as argument the size of the requested block and returns the address of the new allocated space. For the pointers address registration, developers have to call *memoryaddpointer* function passing as parameters the address of the pointer. This function does not allocate any data block but it is responsible only for the registration of the pointer in the Pointers Section. Finally, the provided methods *memoryfree* and *memoryremovepointer* are responsible for removing data blocks and pointer blocks. When a user calls the *memoryfree* method, the system always removes the associated pointers of the block. However, in some cases developers might want to assign or disassociate a pointer without clearing the allocated space.

3.3.4 Memory Management Algorithm

The algorithmic steps of the allocation process is trying to provide a minimal but efficient solution for memory allocation. The steps are presented in the Figure 3.3.

- Step 1: In the beginning the allocation process has to check if the defragmentation process is active.
- Step 2: If a fragmented block is found that has sufficient memory space to keep the requested block, then the allocation process deactivates the defragmentation process and returns the address of the newly created pointer. Otherwise, it continues to the next step.
- Step 3: If the margin between the Pointers Section and the Data Section is large enough to fit the requested block with a minimum of 32 bytes of free space left, then the allocator occupies the free space at the beginning of the Data block. Otherwise, it goes to the end of the Data Section and checks if there is sufficient space to allocate. The returned pointer is NULL in case of failure or the address of the new block in case of success.

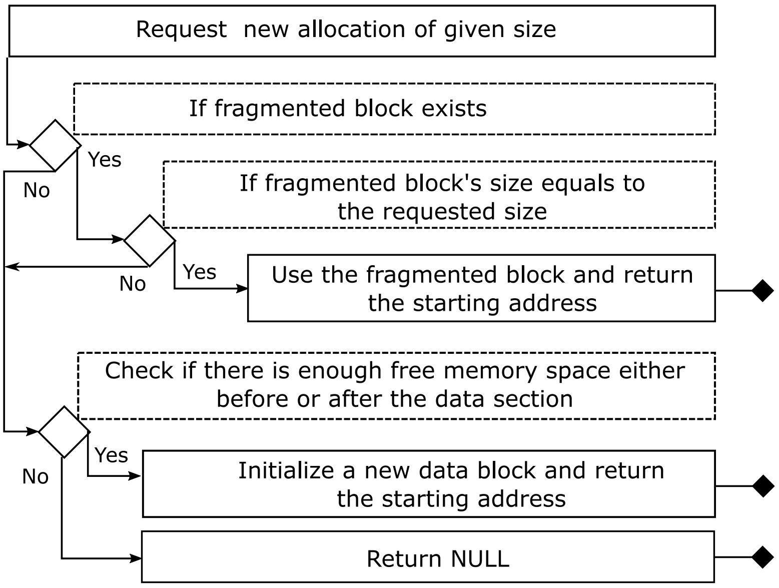

Request new allocation of given size
If fragmented block exists
Yes
No
If fragmented block's size equals to the requested size
Yes
No
Use the fragmented block and return the starting address
Check if there is enough free memory space either before or after the data section
Yes
No
Initialize a new data block and return the starting address
Return NULL

Figure 3.3: Memory allocation procedure

The deallocation process destroys an allocated memory block by using a referenced pointer to it and activates the defragmentation process, as explained in the following sections. Figure 3.4 depicts the algorithm. Separating the process of defragmentation from the process of deallocation gives us the ability to use this memory management approach in both bare metal and multitasking applications supported by real-time operating systems.

Unlike the previously mentioned allocation processes, the Adaptive Memory Management process seeks to keep the allocation algorithm as simple as possible to provide a balance between speed and efficiency.

The defragmentation process uses an optimal workflow for achieving the best utilization of the system. It is divided into small steps that can be carried out anytime during the systems uptime. However, the number of actions that will be executed depends on the fragmentation and the memory availability of the overall system. This approach complies with the requirements of soft real-time embedded systems. The algorithm used in defragmentation is described below:

- If the fragmented block lays in the first 10% of the data section, the remaining blocks between the fragmented one and the beginning of the data section should be immediately shifted (moved) left. The same principle applies if the block is in the last 10
- If none of the previous rules apply, the fragmented block should be swapped either with the last best-fit block found in the last 10% for the data section, or with the first best-fit block found in

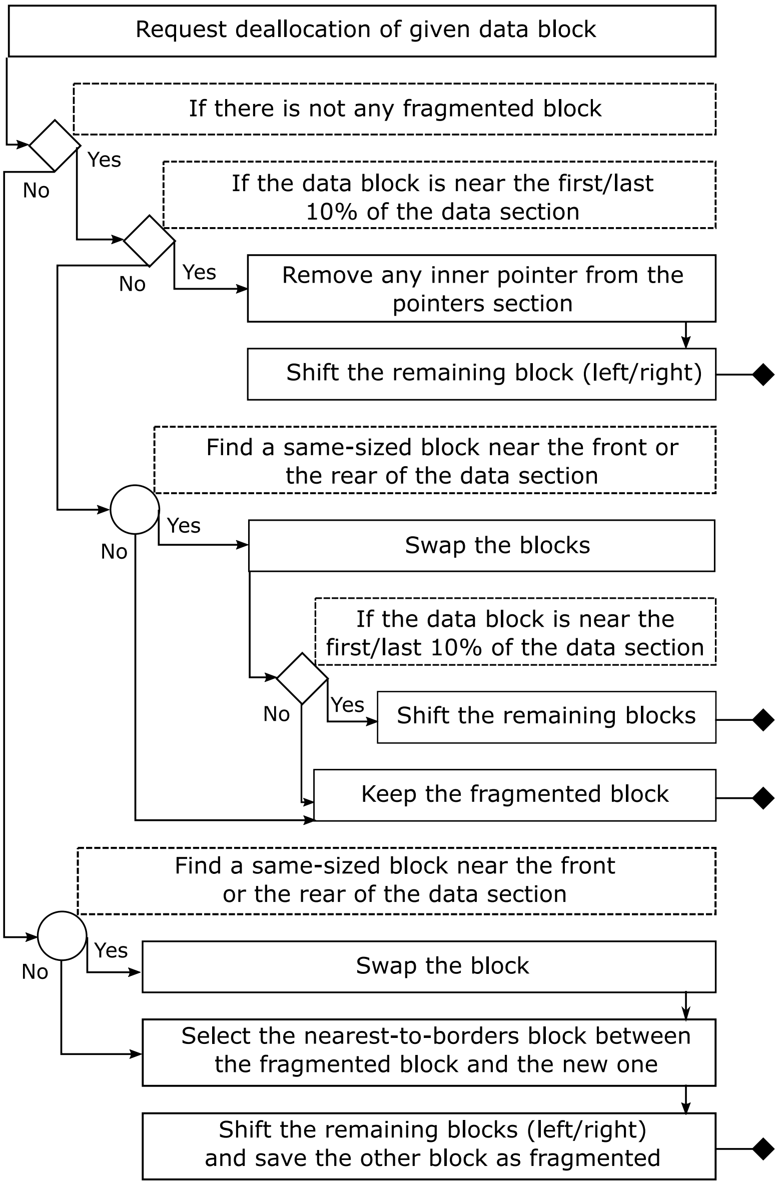

Request deallocation of given data block
If there is not any fragmented block
Yes
No
If the data block is near the first/last 10% of the data section
Yes
No
Remove any inner pointer from the pointers section
Shift the remaining block (left/right)
Find a same-sized block near the front or the rear of the data section
Yes
No
Swap the blocks
If the data block is near the first/last 10% of the data section
Yes
No
Shift the remaining blocks
Keep the fragmented block
Find a same-sized block near the front or the rear of the data section
Yes
No
Swap the block
Select the nearest-to-borders block between the fragmented block and the new one
Shift the remaining blocks (left/right) and save the other block as fragmented

Figure 3.4: Memory deallocation procedure

the front 10% for the data section and execute the first step of the algorithm again. As before, if there is no block to fit, the fragmented block is treated as if it already was in the first or last 10% of the data section.

Figure 3.5 presents an example of the defragmentation process in action. Each block represents a data block with an actual size (meta-data and payload). The defragmentation process begins by applying the rules mentioned above. As long as the block is in the first 40% of the data section the system swaps it with the first best-fit block at the beginning of the section. Afterwards, the system realigns (pushes) the remaining blocks to remove the fragmented one and updates the respective pointers in the pointers section.

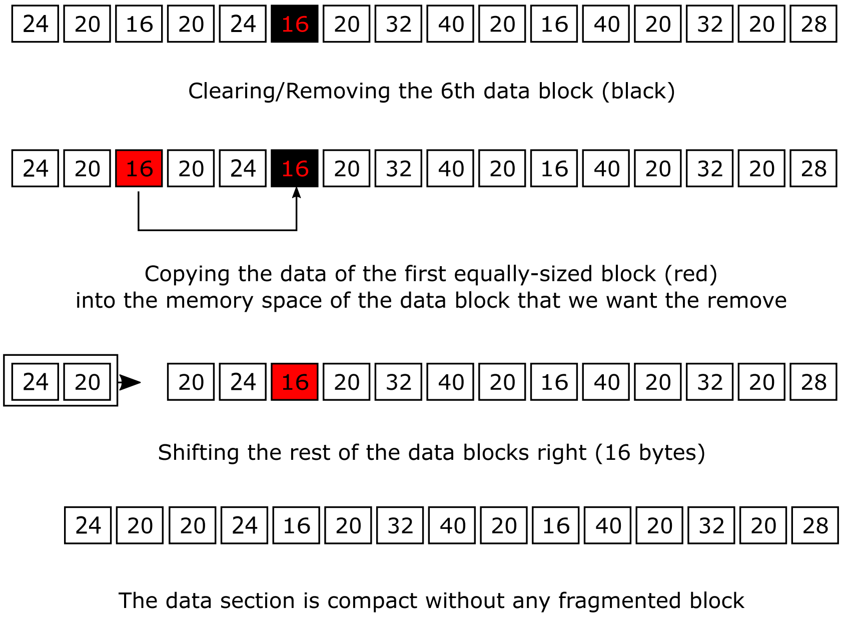

24 20 16 20 24 16 20 32 40 20 16 40 20 32 20 28
Clearing/Removing the 6th data block (black)
24 20 16 20 24 16 20 32 40 20 16 40 20 32 20 28
Copying the data of the first equally-sized block (red)
into the memory space of the data block that we want the remove
24 20 20 24 16 20 32 40 20 16 40 20 32 20 28
Shifting the rest of the data blocks right (16 bytes)
24 20 20 24 16 20 32 40 20 16 40 20 32 20 28
The data section is compact without any fragmented block

Figure 3.5: Example of the defragmentation procedure

In order to provide optimal decision rules for each system, a calibration process should be run when the system begins. This is a simple but effective process of defragmentation that assures there is no external fragmentation in the system.

3.3.5 Experiments and Limitations

In this subsection, we present experiments that showcase the efficiency and stability of the proposed memory management scheme. To this end, we compare the default memory allocation without defragmentation with the proposed adaptive memory management scheme. Since real-time application requirements are quite demanding, the results obtained have to comply with them. We implemented AMM on an ARM Cortex M4 (MK20DX256 Chip) microcontroller. Time measurements are expressed in CPU cycles using the provided functionality of the microcontroller for counting. The first experiment is using same-size blocks for allocation and free. In this test, we randomly allocate and deallocate memory space for one thousand blocks. The second experiment is using the same methodology but with random-size blocks ranging from four to twenty bytes. Both experiments are trying to mimic two different types of resource-hungry applications but mostly applications that require many dynamic allocations/free. The number of blocks and the overall size equals to the 65-70% of the systems memory. The following tables present the results of the allocation and free processes for each memory management algorithm, as well as the fragmentation level.

Memory Allocation	**Standard Allocation**			**AMM**		
	Best	***Worst***	***Mean***	***Best***	***Worst***	***Mean***
Experiment 1	327	475	401	175	207	196
Experiment 2	345	1258	876	182	215	192
Memory Deallocation	***Best***	***Worst***	***Mean***	***Best***	***Worst***	***Mean***
Experiment 1	273	348	301	127	456	367
Experiment 2	275	531	427	112	715	523

Table 3.1: Latency overhead in clock cycles

External Fragmentation	**Standard Allocation**	**AMM**
Experiment 1	12%	0%
Experiment 2	37%	0%
Internal Fragmentation	**Standard Allocation**	**AMM**
Experiment 1	0%	0%
Experiment 2	8%	0%

Table 3.2: Fragmentation Level

The main difference of those two allocation methodologies is the worst case of the allocation process. We could easily understand that if there exist fragmented blocks (Table III) in the memory space, the standard allocation methodology produces a highly degraded response time (Table II) in contrast to the adaptive memory management scheme which has a discrete stable elapsed. It is worth noting that the observed difference in CPU cycles in deallocation between the Standard allocator and the Adaptive memory management scheme is generated by the defragmentation process. The proposed memory management scheme is optimized for a wide range of use. However, to be utilized in a safe and optimal environment, some modifications are essential. This approach has downsides because of memory movement actions that need to be performed. It is clearly mentioned that the overall purpose is to support a safe dynamic allocation method in MMU-Less embedded systems with limited memory space without undergoing the performance of time critical operations. The demonstration does not comply with any security rule about memory isolation, which could lead to an undesired information exposure. The Adaptive Memory management scheme is optimized to be used in low-cost, power efficient devices with a small amount of available memory, however it could also be used as a sub-allocator for general purpose operating systems such as Unix/Linux. Finally, according to statistics regarding the average time latency for memory access, the proposed technique costs three to fourteen

CPU cycles and a memory copy using the default GLIBC memmove [29] method (Complexity O(n)) has around 500Mbps (megabits per second) transfer rate in an ARM Cortex M4 microprocessor. Of course, by using DMA data transfer method (800Mbps) the average time could be less, but there is a prerequisite that the hardware has to support that feature. A key issue in MMU-Less embedded systems is usually the limitation of the available memory to enable dynamic allocation and this can incur increased levels of fragmentation and as a consequence an unstable system. All the memory management schemes to our knowledge try to solve some aspects of these problems but, in extended use of dynamic allocations, usually fail to maintain a low fragmentation level. Using this memory management scheme the robotic platform will be able to operate stable, dynamic and fast.

3.4 Sensory System

A tele-supervised autonomous robot needs to have accurate information on its position, orientation, and velocity and the location of nearby obstacles. The robot needs multiple sensors to gather all the information required and a computer to read the sensor data and make decisions based on it. Information on the robot's environment can be acquired using laser distance sensors, ultrasonic distance sensors, infrared distance sensors or cameras utilizing machine vision. Other methods are also available, but they are not discussed here further. The most popular detection methods in robotic projects have been using a laser range finder and/or a computer vision system. Some studies have also used infrared or ultrasound sensors as alternative obstacle detection sensors. An intelligent robot has to have sufficient knowledge concerning the state of the outside world. For this it needs many sensors to gain information regarding its environment and status. To effectively use all of its sensors, a method is needed for integrating the information from different sources into a more usable form for the operating system.

In sensor fusion, the raw sensor data from the sensors is transformed into a more abstract numeric or symbolic representation. In robotic applications, it is important to detect and study the kinematic of objects. The movement or the displacement of an object is directly related to the position i.e., the object coordinates about a reference system. The displacement of an object in space from an initial position to a final position can be decomposed into linear displacement or angular displacement. The linear displacement refers to the distance between the initial and final position, and the direction of travel in the direction of travel, i.e., the straight line on which it is moving. The angular displacement refers to the angle between the initial and final position of the object and the direction of travel on the plane of movement. For that reason, the following sensors provide the minimum selection of sensors which can be used to provide the economic thus effective way of relative position detection system.

3.4.1 Proximity Sensors

Proximity Sensors are used to detect an object regardless of the distance from the sensor. These sensors are divided into two major categories depending on the mode of operation, detecting by contact or by measurement.

- The proximity sensors based on the contact of the sensor to the object, detecting whether an object is close enough to the robot then activates the appropriate robot circuit. The objects are not in contact with the sensor are ignored.
- The distance measuring sensors calculate the distance between the sensor and the object lying within the range.

The contact is the most common form of object detection. The contact sensors help a robotic system to perceive the contact or not with an object. Implemented using switches that change state when there is mechanical contact. The form is simple and economical, but with multiple uses. The switches are widely available in a wide variety, and easily connected to robotic systems control systems. This platform uses 5 contact sensors to control the connection between the floor and edge of each leg. So, it gives the opportunity to move on uneven level, reduce energy consumption and prevent a problem such as a multitude extensive pressure specifically feet from incorrect weight distribution. Below is a presentation of the legs of robotic platform with sensors, and some cases of motion.

3.4.2 Inertial Navigation Systems

The Inertial Navigation Systems (INS) are integrated sensors devices which record the acceleration and the rotation rate of a system of three axes, assisted of three accelerometers and three gyroscopes. Comparing GPS and INS notice the following. The INS do not require and do not depend on an external electromagnetic energy signal. So their use is feasible in areas outside of GPS range, eg inside buildings. However, the INS have problems due to the integration of the signals, a process that accumulates measurement errors. The amount of data generated by the sensors requires substantial computational resources to process them. The INS systems used in passenger aircraft, where the accumulation of error in measurements reaches 1800m per hour of operation. The maximum achievable accuracy approaching 0.1% of dianystheisas distance in terrestrial applications. The construction of a platform in which to maintain a constant orientation of the accelerometers system used gyro balancer device is very complex and very costly, which is prohibitive in most cases for applications in robotics. Nevertheless the progress of laser devices and optical fibers greatly reduced the cost and use of INS systems are now more affordable for applications in robotics.

The Intertial Measurement Unit (IMU) plays an important role of inertial navigation systems. The IMU inertial measurement unit is an electronic device that takes measurements of speed, the orientation of a vehicle and the gravitational forces, using a combination of accelerometers, gyroscopes and magnetometers sometimes.In that capacity, the data collected by the IMU sensors allow a computer to monitor the position of a vehicle, etc., using the calculation of their position.The IMU devices used in vehicles, boats, aircraft, unmanned aerial vehicles, spacecraft, satellites and equipment landing. An IMU device allows the GPS to operation when the signal is not available, such as when there is electrical interference. For example, supposingly an IMU device has been installed on a plane, when it finds that the vessel travels to the east for an hour with an average speed of 500 miles per hour, then the guidance computer will assume that the plane must be 500 miles east of the original place. In combination with an electronic map system, the steering system can be used to indicate to the pilot where is the plane. It is similar to a GPS system but without the need to communicate with external elements, such as satellites. For this reason, in the present robotic platform the minimum position reference means attributed to the use of adafruit 10DOF IMU which allows us to capture ten distinct types of motion or orientation related data and is consisted of the following sensors:

- LSM303DLHC - a 3-axis accelerometer (up to +/-16g) and a 3-axis magnetometer (up to +/-8.1 gauss) on a single die.
- L3GD20 - a 3-axis gyroscope (up to +/-2000 dps).
- BMP180 - A barometric pressure sensor (300..1100 hPa) that can be used to calculate altitude, with an additional on-board temperature sensor.

All of the sensors on the Adafruit 10DOF breakout communicate via a two-pin I2C bus. Unfortunately, a major problem that appears during the period of use of this system is the phenomenon of noise, which can cause considerably serious problems to the correct operation of the platform. Therefore all data need a filtration before use. An appropriate choice for a filter is the use of kalman algorithm.

The following section describes the kalman filter as well as the modified version which was used in the robotic platform in order to increase the overall performace.

3.4.3 Kalman Filter

Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. The filter is named after Rudolf E. Klmn, one of the primary developers of its theory.

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation, and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization. The Kalman filter has also found use in modeling the central nervous system's control of movement. The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty.

The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required. The Kalman filter does not require any assumption that the errors are Gaussian. However, the filter yields the exact conditional probability estimate in the special case that all errors are Gaussian-distributed.

Kalman Filter equations are composed of a lot of complicated matrix and vector math. However, after some research on the application domain for an accelerometer, a simple one-dimensional filter will do the job. The accelerometer puts out three-dimensional acceleration data. However, if we just want to get clean acceleration data from it all formulas, we simply deal with one dimension. So it was easier to use three different one-dimensional Kalman filters. It all ended in a small set of formulas:

$$p = p + q \tag{3.1}$$

$$k = \frac{p}{p + r} \tag{3.2}$$

$$x = x + k(m - x) \tag{3.3}$$

$$p = (1 - k)p \tag{3.4}$$

The first two formulas represent the prediction of the Kalman Filter. Moreover, since there is no information about the driving forces it is very simple. The second three formulas calculate the measurement update. The variables are x for the filtered value, q for the process noise, r for the sensor noise, p for the estimated error, k for the Kalman Gain and m for the measurement. The state of the filter is defined by the values of these variables. The filter is applied to each measurement and initialized with the process noise q, the sensor noise r, the initial estimated error p and the initial value x. The initial values for p are not very important since it is adjusted during the process. It must be just high enough to narrow down. The initial value for the readout is also not very important since it is updated during the process. However, tweaking the values for the process noise and sensor noise is essential to get clear readouts.

To validate the previously mentioned filter, the following figures present the raw data as well as the filtered data of roll of the robotic platform using an accelerometer combined with a gyroscope and a compass chip.

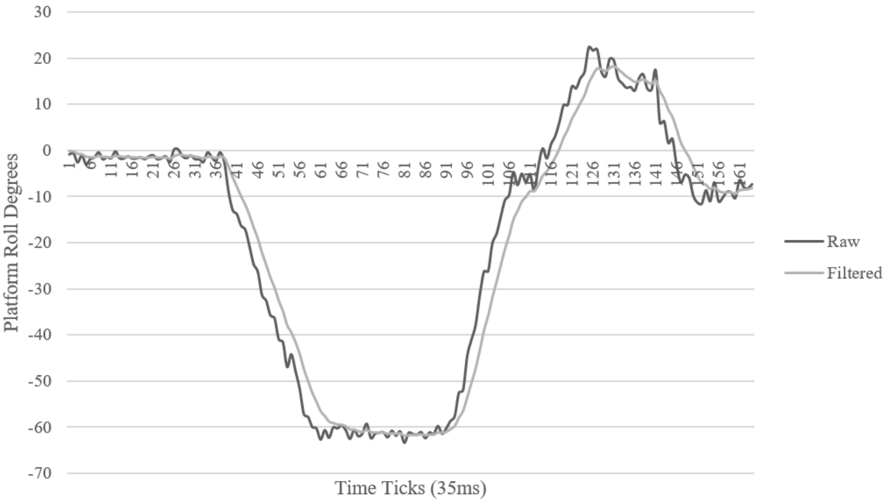


Figure 3.6: Kalman filter results

3.5 Communication Protocols

The past few years have seen the beginning of a trend to dramatically increase the embedded electronics content of automobiles, elevators, building climate control systems, jet aircraft engines, and other traditionally electro-mechanically controlled systems. In many large systems, this increasing electronics content is being accompanied by a proliferation of subsystems having separate CPUs.

The increase in the number of processors in a system is often driven by computation and I/O growth. In some development environments, the increase may also be driven by a need to ease system integration burdens among multiple design groups or to provide system flexibility through "smart sensors" and "smart actuators". But, whatever the reasons, once there is more than one CPU in a system there must be some means of communication to coordinate action.

The first protocol is UART (Universal Asynchronous Receiver Transmitter); it is a very common protocol used in embedded systems to connect the MCU to a computer or to connect the MCU to another MCU, and it uses 2 wires to transfer data, one for sending the data called (Tx) and the other one for receiving the data called (Rx). It used in making P2P networks (Point to Point).The second protocol is I2C (Inter-Integrated-circuit or I squared C) which is also referred to as TWI (Two Wire Interface); its a serial protocol that uses 2 wires (SCL and SDA) to transfer data between 2 or more MCUs. And the last one is CAN (Controller Area Network) which is a very common protocol in automotive industry as well as many industrial products.

3.5.1 I2C: Inter-Integrated Circuit

In the early 1980's, NXP Semiconductors developed a simple bi-directional 2-wire bus for efficient inter-IC control. This bus is called the Inter-IC or I2C-bus. At present, NXP's IC range includes more than 150 CMOS and bipolar I2C-bus compatible types for performing communication functions between intelligent control devices (e.g. microcontrollers), general-purpose circuits (e.g. LCD drivers, remote I/O ports, memories) and application-oriented circuits (e.g. digital tuning and signal processing circuits for radio and video systems). All I2C-bus compatible devices incorporate an on-chip interface which allows them to communicate directly with each other via the I2C-bus. This design concept solves the many interfacing problems encountered when designing digital control circuits. I2C has become a de facto world standard that is now implemented in over 1000 different ICs and is licensed to more than 50 companies. I2C bus is popular because it is simple to use, only upper bus speed is defined and only two wires with pull-up resistors are needed to connect almost unlimited number of I2C devices. I2C can use even slower microcontrollers with general-purpose I/O pins since they only need to generate correct Start and Stop conditions in addition to functions for reading and writing a byte.[30] The initial I2C specifications defined maximum clock frequency of 100 kHz. This was later increased to 400 kHz as Fast mode. There is also a High speed mode which can go up to 3.4 MHz and there is also a 5 MHz ultra-fast mode.

I2C uses only two wires: SCL (serial clock) and SDA (serial data). Both need to be pulled up with a resistor to +Vdd. There are also I2C level shifters which can be used to connect to two I2C buses with different voltages. Basic I2C communication is using transfers of 8 bits or bytes. Each I2C slave device has a 7-bit address that needs to be unique on the bus. Some devices have fixed I2C address while others have few address lines which determine lower bits of the I2C address. This makes it very easy to have all I2C devices on the bus with unique I2C address. There are also devices which have10-bit address as allowed by the specification. 7-bit address represents bits 7 to 1 while bit 0 is used to signal reading from or writing to the device. If bit 0 (in the address byte) is set to 1 then the master device will read from the slave I2C device. Master device needs no address since it generates the clock (via SCL) and addresses individual I2C slave devices.[30]

In normal state both lines (SCL and SDA) are high. The communication is initiated by the master device. It generates the Start condition (S) followed by the address of the slave device (B1). If the bit 0 of the address byte was set to 0, the master device would write to the slave device (B2). Otherwise, the next byte will be read from the slave device. Once all bytes are read or written (Bn) the master device generates Stop condition (P). This signals to other devices on the bus that the communication has ended, and another device may use the bus. Most I2C devices support repeated start condition. This means that before the communication ends with a stop condition, the master device can repeat start condition with address byte and change the mode from writing to reading.[30] I2C bus is used by many integrated circuits and is simple to implement. Any microcontroller can communicate with I2C devices even if it has no special I2C interface. I2C specifications are flexible - the I2C bus can communicate with slow devices and can also use high-speed modes to transfer large amounts of data. Because of many advantages, I2C bus will remain as one of the most popular serial interfaces to connect integrated circuits on the board. [30]

Using the I2C protocol, the robotic platform is able to communicate with some sensors.

3.5.2 RxTx: Serial Communication

Serial communication is a standard method for transmitting data between a computer and some peripheral devices such as a programmable instrument or another computer. Serial communication transmits data one bit at a time, sequentially, over a single communication line to a receiver. The serial protocol is the most popular communication protocol that is used by several devices for instrumentation; numerous GPIB-compatible devices also come with an RS-232 based port. This method is used when data transmission rates are very low, or the data must be transferred over great distances and also where the cost of cable and synchronization difficulties make parallel communication impractical. Serial communication is popular since most computers have one or more serial ports. Therefore no extra hardware is needed other than a cable to connect the instrument to the computer or two computers together.

This type of communication connects the robotic platform with the computer through an XBee module.

3.5.3 CAN: Controller Area Network

The all-round Controller Area Network, developed in the early 1980s, is an event-triggered controller network for serial communication with data rates up to 1Mbit/s. Its multi-master architecture allows redundant networks, which are able to operate even if some of their nodes are defective. CAN messages do not have a recipient address, but they are classified over their respective identifier. Therefore, CAN controllers broadcast their messages to all connected nodes and all receiving nodes decide independently if they are going to process the message. CAN uses the decentralized, reliable, priority-driven CSMA/CD (Carrier Sense Multiple Access/Collision Detection) access control method in order to guarantee the transmission of the top-priority message first.

In order to employ CAN in the environment of strong electromagnetic fields, CAN offers an error mechanism that detects transfer errors, interrupts and indicates the erroneous transmissions with an error flag so as to(and) initiate the retransmission of the affected message. Furthermore, it contains mechanisms for automatic fault localization including disconnection of the faulty controller.

In CAN there are two main Hardware Implementations, they are Basic CAN and Full CAN.

Basic CAN: Which has only one Message buffer for Receive and Transmit messages. The received message is accepted or ignored after acceptance filtering. The decision to process a message or to ignore it is also achieved by acceptance filtering. This acceptance filtering of the node is done by software in Basic CAN. To reduce the software load at the nodes, there is a possibility to ignore some messages by ignoring specific identifiers. This is realized by bit mask for the message identifiers.[31]

Full CAN: Which has 8 to 16 memory buffers for every transmitted or received message. Here the acceptance filtering is done by hardware and not by the software. Every buffer can be configured to accept messages with specific IDs. Since the acceptance filtering is done by hardware, the software load is greatly reduced. With different buffers for different messages ensures more time for the processing of the received messages and the transmitted message can be handled according to the priority levels. Configuring each buffer for every message ensures also the data consistency in Full CAN.[31]

Since any CAN node may begin to transmit when the bus is free, two or more nodes may begin to transmit simultaneously. Arbitration is the process by which these nodes battle for control of the bus. Proper arbitration is critical to CAN performance because this is the mechanism that guarantees that message collisions do not reduce bandwidth or cause messages to be lost. Each data or remote frame begins with an identifier, which assigns the priority and content of the message. As the identifier is broadcast, each transmitting node compares the value received on the bus to the value being broadcast. The higher priority message during a collision has a dominant bit earlier in the identifier. Therefore, if a transmitting node senses a dominant bit on the bus in place of the recessive bit it transmitted, it interprets this as another message with higher priority transmitting simultaneously. This node suspends transmission before the next bit and automatically retransmits when the bus is idle.

The result of proper arbitration is that a high-priority message transmitted without interruption is followed immediately by a low-priority message, unless of course, another high-priority message attempts to broadcast immediately following the same message. Since no messages are lost or corrupted in the collision, data and bandwidth are not compromised. CAN protocol specifies five different types of network errors.[31]

- Stuffing error - a transmitting node inserts a high after five consecutive low bits (and a low after five consecutive high). A receiving node that detects violation will flag a bit stuffing error.

- Bit error - A transmitting node always reads back the message as it is sending. If it detects a different bit value on the bus than the one it sent, and the bit is not part of the arbitration field or in the acknowledgement field, an error is detected.

- Checksum error - each receiving node checks the CAN messages for checksum errors (different rules apply for CAN 2.0 and CAN FD).

- Frame error - There are certain predefined bit values that must be transmitted at certain points within any CAN Message Frame. If a receiver detects an invalid bit in one of these positions a Form Error (sometimes also known as a Format Error) will be flagged.

- Acknowledgement Error - If a transmitter determines that a message has not been ACKnowledged then an ACK Error is flagged.

The CAN protocol supports two message frame formats , the only essential difference being in the length of the identifier (ID). In the standard format (version 2.0A) the length of the ID is 11 bits and in the extended format (version 2.0B) the length is 29 bits. Most version 2.0A controllers are tolerant of extended format messages, but essentially ignore them. Version 2.0B controllers can send and receive messages in both formats.

The functionality of hardware filters is very similar on many CAN devices. While receiving a CAN message, the identifier (and sometimes even the data) can be compared to a configured filter. Only if the incoming message matches the filter does the message get stored into a receive buffer. The major differences in filters are usually the width of the filter and if it is a match only filter or also allows a mask to be used. The filter width specifies how many bits of an incoming CAN message can be processed. For a standard CAN message identifier at least 11-bits are required. For an extended CAN message identifier its 29-bits. Where a match filter only allows you to do one exact match (for example exactly one identifier), a combination of match and mask allows for filtering on message groups. Usually a bit set in the mask register means that the corresponding bit in the CAN message is a "don't care" value for the acceptance filtering. If a bit is cleared, it MUST match the value in the match register. [32]

3.6 Secured Messaging System

The embedded or handheld devices are getting increasingly connected and are more and more involved in network communications. The users of these devices are now able to execute almost all the network/internet applications that run on a PC on these devices. These devices are also increasingly involved in transfer of secure data through public networks that needs protection from unauthorized access and thus the security requirements in embedded devices have become critical.[33, 34]

For that reason, we had to create a secured protocol that would be able to ensure the integrity of the transferred data. Because of the variety of communication protocols that the platform supports the main idea is to develop a communication system that could be easily adopted in Serial, I2C and CAN protocols by bypassing the particularities of each one. As described before Serial communication is a straight forward process (P2P streaming bytes) in contrasts to I2C and CANbus (data frames). This lead us to use a specified configuration for the CANbus described below.

Therefore the secure data must be scrambled in such a way that the data will be useless or unintelligible for anyone who is having unauthorized access to the secure data. This can be achieved with the help of cryptographic methods such as Encryption/Decryption, Key Agreement, Digital Signatures and Digital Certificates.

3.6.1 Description of the messaging system

First of all, we had to eliminate the filtering processes, for that reason, each node is using the Universal ID $0x00$. Lastly, all the data frames should have the same length (8 bytes) to avoid conflicts with the encryption process. Both Serial and CAN protocol are almost identical.

However, there are some other issues that should be solved. In case that the message has to be split into several frames we had to ensure that the reconstruction would not mess the frames. For that reason, we have to use one byte of the payload to determine the sequence number of each frame and one byte for the total number of frames. This also provides the ability to recognize delivery failures. Moreover, we have to ensure that each frame is listed to the appropriate list of frames of each message. By inserting a CRC8 as well as a random byte value in all the frames of each message, the system can determine which frame corresponds to which message. Even if this methodology occupies 4 bytes of the 8 available bytes of the payload, the overall system performance is fair enough. The following table provides the description of the frame's payload.

byte sequenceNo;
byte sequenceSize;
byte crc8;
byte rnd;
byte message [4];

Table 3.3: Data-frame payload description

3.6.2 The encryption mechanism

Cryptography systems can be broadly organized into two categories:

Symmetric encryption algorithms that use a single key that both the sender and recipient have. This key is kept secret among sender and receiver so that no intruder can steal the data to be transferred by encrypting it.

Asymmetric encryption algorithm or public-key systems that use two keys, a public key known to everyone and a private key that only the recipient of messages uses. Individuals who practice this field are known as cryptographers. Asymmetric encryption provides more security as compared to symmetric key encryption but in case of encryption speed, symmetric encryption is on lead.

Terminology	**Short Description**
Plain Text	The original text or message used in communication in called as Plain text.
Cipher Text	The plain text is encrypted in un-readable message.
Encryption	Encryption is a process of converting Plain text into Cipher text.
Decryption	Decryption process is the reverse of Encryption process,
Key	A key is a numeric or Alpha-numeric text (mathematical formula).
Key Size	Key size is the measure of length of key in bits, used in any algorithm.
Block Size	Key cipher works on fixed length string of bits.
Round	How much time encryption function is executed in complete encryption process.

Table 3.4: Cryptography terminology

The encryption we are using combines the benefits of both types of encryption algorithms; the general idea is to create a random symmetric key to encrypt the data, then encrypt that key asymmetrically. Once the key is asymmetrically encrypted, we add it to the encrypted message. The receiver gets the key, decrypts it with their private key, and uses it to decrypt the message. If all the devices are always using the same key, the attacker would be able to decrypt all messages encrypted with this key. Given enough computing resources, both symmetric and asymmetric encryption can be broken. The most basic way to attack a symmetric cryptosystem is brute-force attacks, where you mainly try every combination of a key. For a 128-bit key, there are 2$\hat{1}$28 combinations to attempt, which requires extensive computing resources. Other cryptanalysis attacks, including chosen-ciphertext and chosen-plaintext attacks, can be more efficient than brute-force, but they require a priori knowledge to work. For that reason, to protect the communication, the system randomly generates and exchanges symmetric keys during the run-time. This key is called session key (randomly generated and valid only for one session). If an attacker grabs the session key, he can decrypt only the messages from one session.

In this paragraph we will analyze this process step by step. The Governor (main microntroller) generates a session key (SESSION_KEY) and encrypts it with Servocontroller's public key (PUB_KEY_SC). The result is PUB_KEY_SC (SESSION_KEY), which is denoted by PART1. Then the message (MESSAGE) is encrypted with SESSION_KEY. The result is SESSION_KEY(MESSAGE), which is denoted by PART2. Finally PART1 and PART2 are sent to Servocontroller. Only the Servocontroller can decrypt PART1, because it is the only device knowing the corresponding private key (PRIV_KEY_SC). Servocontroller decrypts PART1 and gets the SESSION_KEY. Then it uses SESSION_KEY to decrypt PART2 and get the MESSAGE.

One main problem with the previously mentioned cryptographic mechanism is the randomness. The following subesction presents a way to provide real random values to the system.

3.6.3 Using Genetic Algorithms for Randomness

As mentioned before, randomness is one of the most crucial parts of the overall systems security. If random value is predictable, then any of the above-mentioned algorithms would fail. This could be solved if the system would be able to use Genetic Algorithms to produce random seeds.

A random number is generated such that it cannot be predicted, and which is difficult to reproduce sequentially and reliably. Pseudo-random number generators are used to generate a sequence of number that approximates the properties of random numbers. Pseudorandom numbers are practiced for their speed in number generation. Random numbers are numbers that occur in a sequence such that two conditions are met: The distribution of values are uniform across a defined set interval, and the prediction of future values by past or present ones is impossible.

A genetic algorithm is a randomized search and optimization technique guided by the principle of natural selection systems. Three basic operators used in Genetic algorithms contain selection, crossover, and mutation. The GA goes through the following cycle: Selection, crossover, and mutate until some stopping criteria are reached. Reproduction and crossover together give genetic algorithms most of their searching power.

Only a few genetic algorithms based cryptographic scheme have been proposed. The most notable one is an approach named ICIGA (Improved Cryptography Inspired by Genetic Algorithms) described by A. Tragha [35]. This new symmetrical block is ciphering where the session key is generated in a random process. This procedure is an enhancement of the system Genetic Algorithms Inspired Cryptography [36].

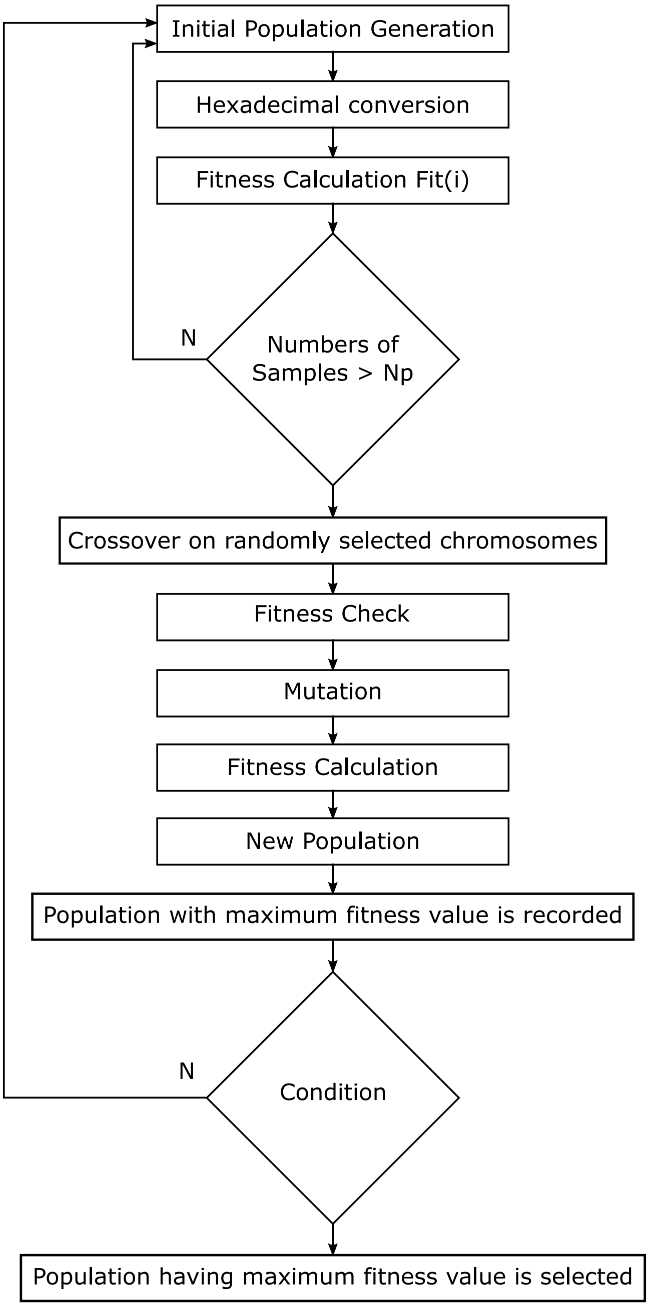
Using a genetic algorithm to produce random encryption keys have been studied a lot in the past. One of the best methodologies is presented by Aarti Soni and Suyash Agrawal [37]. They are using as fitness function the following equation:

$$F = n + (\epsilon/m) \tag{3.5}$$

The following process is used in key the generation process. The crossover and mutation operation is used along with Pseudo random number generators to make the key very complex. The process of generating the key from the Genetic Population has the following steps:

- STEP 1: A pseudo random binary sequence is generated on the basis of current date using millisecond function.
- STEP 2: The generated string or population is divided in to two halves.
- STEP 3: On the selected string crossover operation is performed to achieve good randomness among the key.
- STEP 4: After crossover operation the bits of the string are swapped again to permute the bit values.
- STEP 5: The same process is iterated two times.

The work has been implemented at the boot time of the robotic platform and as a secondary task to produce encryption keys when the system is idle. Following this fitness function and the following procedure, each microcontroller is able to create a series of random keys to protect the messages.



Initial Population Generation
Hexadecimal conversion
Fitness Calculation Fit(i)
N
Numbers of Samples > Np
Crossover on randomly selected chromosomes
Fitness Check
Mutation
Fitness Calculation
New Population
Population with maximum fitness value is recorded
N
Condition
Population having maximum fitness value is selected

Figure 3.7: Genetic Algorithm procedure for encryption keys generation

3.6.4 Conclusion and Limitations

The available security measures for secure transfer of data between two devices are mature enough to defeat any third party to decrypt and get access to the protected content. But the security measures available to protect the stored secure data, like secret keys, within the device are not yet foolproof. A tamper resistant protection mechanism in a device may require hardware circuit to zeroise the secret keys when a physical attack is made to the Secure SoC. The more the hardware security measures implemented in a device to protect its secret keys and other secure data, the more costly the device will be. Thus the hardware security measures implemented in the robotic platform are a trade-off between the cost of implementation and the cost of the data protected.

3.7 System's Sequence Diagrams

The following sequence diagrams give more details about the phases of system's booting process and the action taking process.

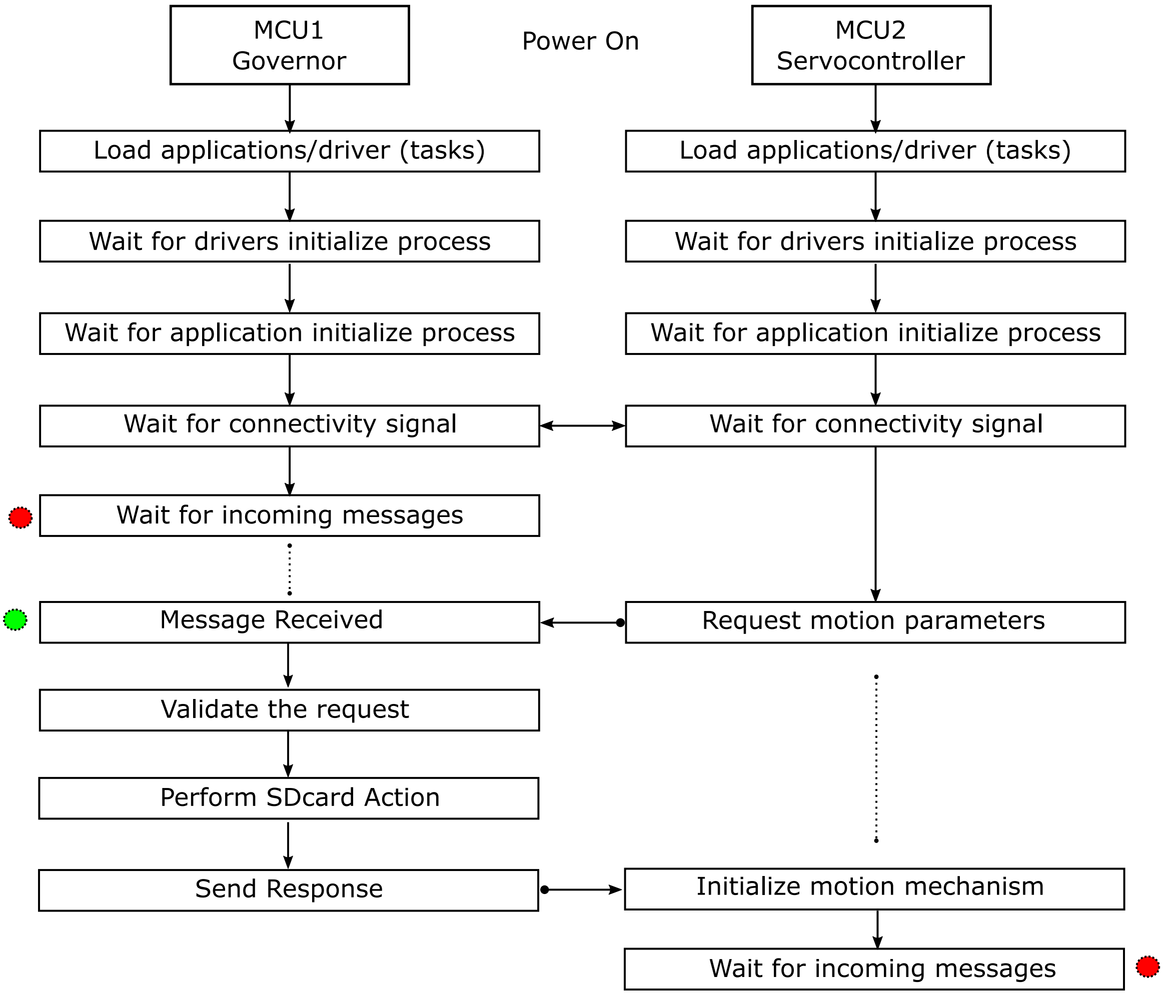

MCU1
Governor
Power On
MCU2
Servocontroller
Load applications/driver (tasks)
Load applications/driver (tasks)
Wait for drivers initialize process
Wait for drivers initialize process
Wait for application initialize process
Wait for application initialize process
Wait for connectivity signal
Wait for connectivity signal
Wait for incoming messages
Message Received
Request motion parameters
Validate the request
Perform SDcard Action
Send Response
Initialize motion mechanism
Wait for incoming messages

Figure 3.8: System's boot sequence diagram

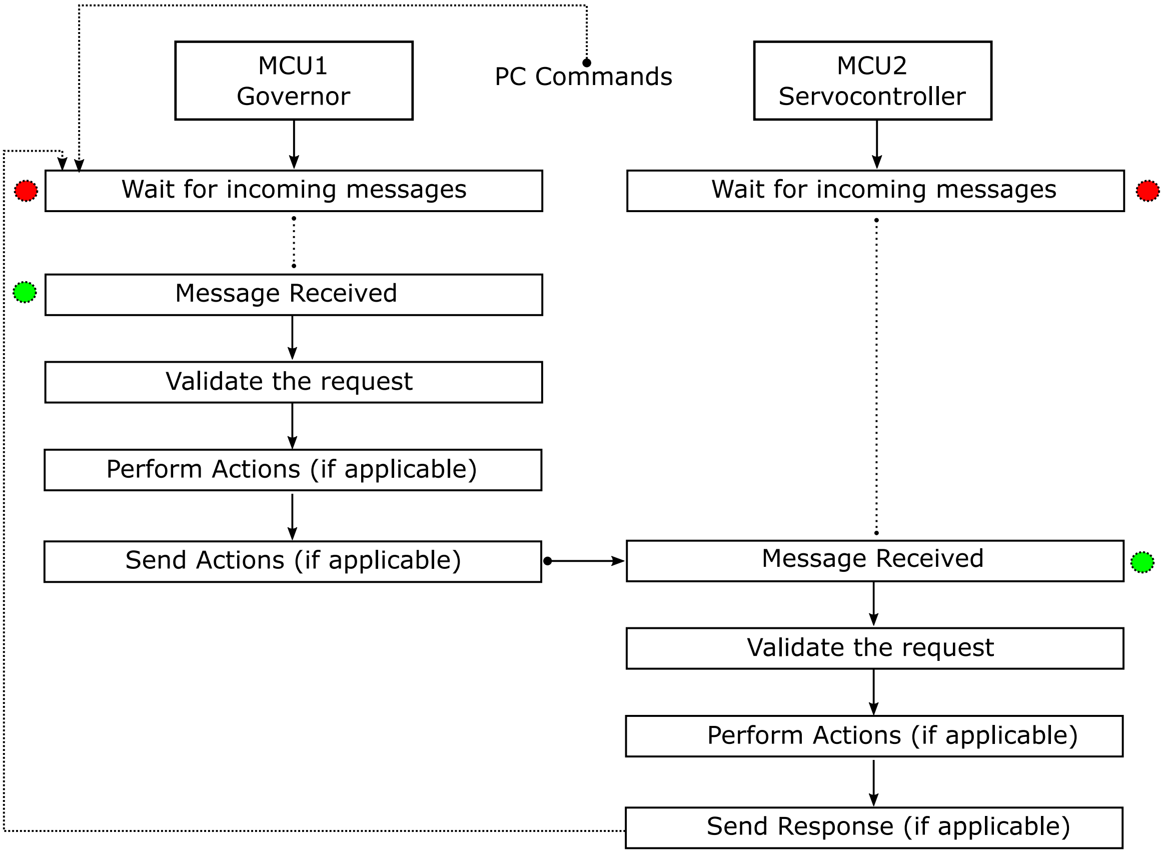

MCU1
Governor
PC Commands
MCU2
Servocontroller
Wait for incoming messages
Wait for incoming messages
Message Received
Validate the request
Perform Actions (if applicable)
Send Actions (if applicable)
Message Received
Validate the request
Perform Actions (if applicable)
Send Response (if applicable)

Figure 3.9: Actions taking sequence diagram

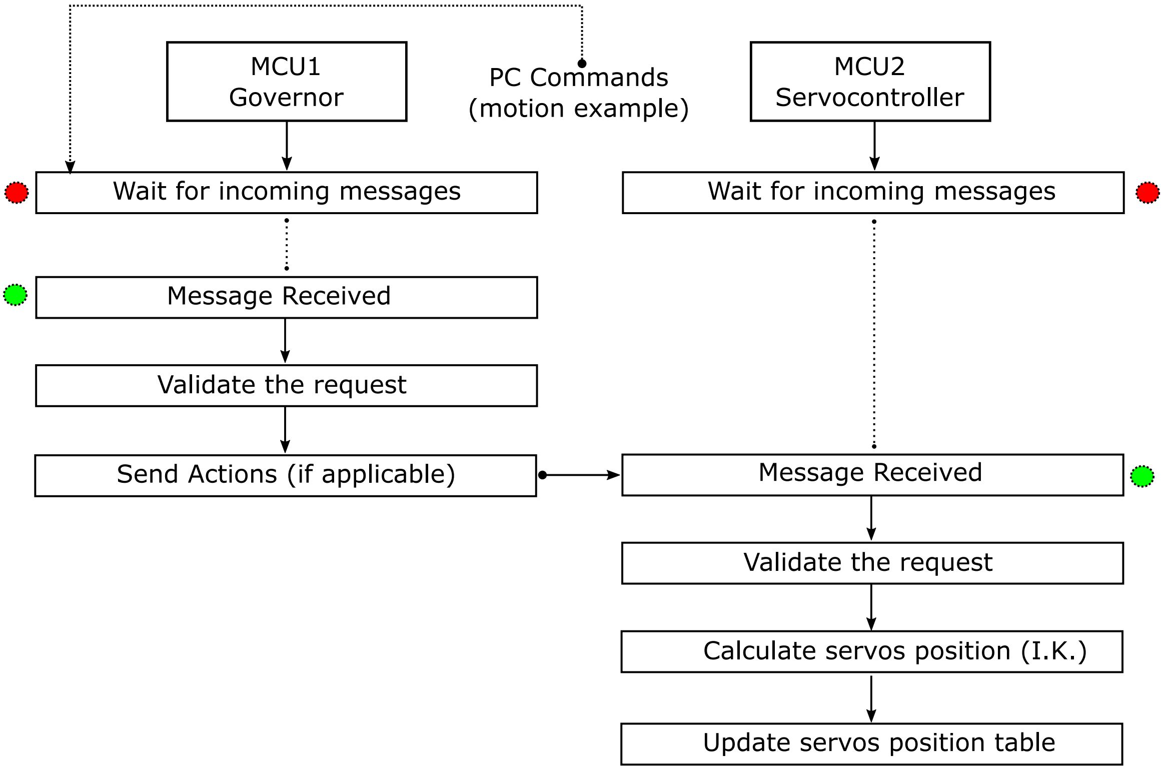

MCU1
Governor
PC Commands
(motion example)
MCU2
Servocontroller
Wait for incoming messages
Wait for incoming messages
Message Received
Validate the request
Send Actions (if applicable)
Message Received
Validate the request
Calculate servos position (I.K.)
Update servos position table

Figure 3.10: Example of Motion action sequence diagram

Chapter 4

Conclusion

This section presents the results obtained from the real experimental setup outlined above. Experiments are conceived to test compliance with time constraints at each level of the software architecture, from the high-level decision making to the low-level motion control algorithms.

4.1 Summary of Thesis Achievements

It is possible to construct an autonomous robot with the techniques available today. The main problem is that te equipment needed for a robust and accurate robot is too expensive for most commercial application of the robot. However, the prices are decreasing all the time. This master thesis presents an all-terrain autonomous legged robotic platform designed to provide a stable motion mechanism by interacting with the environment, in a given amount of time.

To construct an autonomous robot rather than just a remotely controlled vehicle, it was necessary to use the provided sensor data and transform them into motor control commands. Simple decision systems, as well as real-time methods and techniques, were combined to obtain a robot able to autonomously perform intelligent motion actions under real-time constraints. For that reason, a two-level software architecture was used, in which the high-level soft real-time agent-based software was in charge for decision-making and communication while a low-level hard real-time software was responsible for the motion control and the reactions of the direct environmental changes perceived as raw data from the sensory system.

The solution described in this master thesis highlights the strength of a modular approach solving complex and very different problems such as real-time inverse kinematics algorithm, messaging system, memory management as well as security over the CAN protocol. From a functional point of view, the system can achieve its goals on a regular basis, thus placing the real-time motion control system in better operational condition and prioritizing the messaging mechanism.

We are condent that a system with the features and advantages presented here can help to build cost efcient robots, which could be used by scientists and industry to further the progress of autonomous robots.

4.2 Future Work

Mobile robots are gradually leaving the laboratories to undertake service tasks ranging from providing surveillance of buildings and supervision of plants, to transporting patients and delivery items, cleaning houses and guiding people. Regardless of the assigned task, the basic capability of a mobile robot is to move efficiently (e.g., along short trajectories) and safely (i.e., without collisions) to its destination (or sequence of targets). While developing a generic computational/motion robotic platform for seeing remains the dream of many robotic application enthusiasts, significant progress can certainly be made by pursuing the fundamental problems of motion and interaction in a natural environment. Certainly, there are many plausible approaches for a legged robotic platform and the criterion to compare them is how efficient they could be in a variety of circumstances. Given the successful methodologies we have, next steps would be to find a way of modeling the interactions among different plugins(external componments) of the robot and integrate them effectively into a complete robotic system.

Believing that the robotics is still a growing field, and it holds huge potential. This field can be beneficial for the development of human society in the future. As the information age is taking many forms over the years, it has become apparent that the net productivity ratio needs to be increased. In the future, robots will be used to accomplish tasks of secretarial nature so the mankind can focus on more important tasks. It is my sincere hope that this work would provide some useful insights.

Bibliography

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics.* MIT press, 2005.

[2] N. Hockstein, C. Gourin, R. Faust, and D. J. Terris, “A history of robots: from science fiction to surgical robotics,” *Journal of robotic surgery*, vol. 1, no. 2, pp. 113–118, 2007.

[3] S. B. Niku, *Introduction to robotics: analysis, systems, applications*, vol. 7. Prentice Hall New Jersey, 2001.

[4] F. J. Martínez-López and J. Casillas, “Artificial intelligence-based systems applied in industrial marketing: An historical overview, current and future insights,” *Industrial Marketing Management*, vol. 42, no. 4, pp. 489–495, 2013.

[5] J. F. Engelberger, *Robotics in practice: management and applications of industrial robots.* Springer Science & Business Media, 2012.

[6] Y. U. Cao, A. S. Fukunaga, and A. Kahng, “Cooperative mobile robotics: Antecedents and directions,” *Auton. Robots*, vol. 4, pp. 7–27, Mar. 1997.

[7] J. Kramer and M. Scheutz, “Development environments for autonomous mobile robots: A survey,” *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[8] T. Lozano-Perez, I. J. Cox, and G. T. Wilfong, *Autonomous robot vehicles.* Springer Science & Business Media, 2012.

[9] M. Vukobratovic, *Introduction to robotics.* Springer Science & Business Media, 2012.

[10] E. R. Fuchs, “Global manufacturing and the future of technology,” *Science*, vol. 345, no. 6196, pp. 519–520, 2014.

[11] S. L. Moskowitz, *Advanced Materials Innovation: Managing Global Technology in the 21st Century.* John Wiley & Sons, 2016.

[12] C. Ranganathan, C. Ye, and S. Jha, "Market value impacts of information technology enabled supply chain management initiatives," *Information Resources Management Journal (IRMJ)*, vol. 26, no. 3, pp. 1–16, 2013.

[13] N. B. Ignell, N. Rasmusson, and J. Matsson, "An overview of legged and wheeled robotic locomotion," *Available from: Mälardalen University, Web site: http://www. idt. mdh. se/kurser/ct3340/ht12/MINICONFERENCE/FinalPapers/i rcse12_sub mission_21. pdf [Accessed: January 7, 2014]*, 2012.

[14] J. Denavit and R. Hartenberg, "Kinematic modelling for robot calibration," *Trans. ASME Journal of Applyed Mechanics*, vol. 22, pp. 215–221, 1955.

[15] J. Funda, R. H. Taylor, and R. P. Paul, "On homogeneous transforms, quaternions, and computational efficiency," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 3, pp. 382–388, 1990.

[16] M. Masmano, I. Ripoll, A. Crespo, and J. Real, "Tlsf: A new dynamic memory allocator for real-time systems," in *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pp. 79–88, IEEE, 2004.

[17] G. C. Haynes and A. A. Rizzi, "Gaits and gait transitions for legged robots," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1117–1122, IEEE, 2006.

[18] A. Crespo, I. Ripoll, and M. Masmano, "Dynamic memory management for embedded real-time systems," in *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, pp. 195–204, Springer, 2006.

[19] W. E. Croft and A. Henderson, "Eliminating memory fragmentation and garbage collection from the process of managing dynamically allocated memory," Oct. 22 2002. US Patent 6,470,436.

[20] I. Puaut, "Real-time performance of dynamic memory allocation algorithms," in *Real-Time Systems, 2002. Proceedings. 14th Euromicro Conference on*, pp. 41–49, IEEE, 2002.

[21] X. Sun, J. Wang, and X. Chen, "An improvement of tlsf algorithm," in *Real-Time Conference, 2007 15th IEEE-NPSS*, pp. 1–5, IEEE, 2007.

[22] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," in *Memory Management*, pp. 1–116, Springer, 1995.

[23] G. S. Brodal, E. D. Demaine, and J. I. Munro, "Fast allocation and deallocation with an improved buddy system," *Acta Informatica*, vol. 41, no. 4-5, pp. 273–291, 2005.

[24] J. Bonwick *et al.*, "The slab allocator: An object-caching kernel memory allocator.," in *USENIX summer*, vol. 16, Boston, MA, USA, 1994.

[25] K. Wang, *Design and Implementation of the MTX Operating System.* Springer, 2015.

[26] Y.-H. Yu, J.-Z. Wang, and T.-Y. Sun, "A novel defragmemtable memory allocating schema for mmu-less embedded system," in *Advances in Intelligent Systems and Applications-Volume 2*, pp. 751–757, Springer, 2013.

[27] G. Barootkoob, E. M. Khaneghah, M. Sharifi, and S. L. Mirtaheri, "Parameters affecting the functionality of memory allocators," in *Communication software and networks (iccsn), 2011 ieee 3rd international conference on*, pp. 499–503, IEEE, 2011.

[28] M. Masmano, I. Ripoll, and A. Crespo, "Dynamic storage allocation for real-time embedded systems," *Proc. of Real-Time System Simposium WIP*, 2003.

[29] S. Loosemore, R. Stallman, R. McGrath, A. Oram, and U. Drepper, "The gnu c library reference manual, free software foundation," *Inc, Boston, MA, USA*, 1999.

[30] P. Semiconductors, "The i2c-bus specification," *Philips Semiconductors*, vol. 9397, no. 750, p. 00954, 2000.

[31] S. Broyles and S. Corrigan, "Using can arbitration for electrical layer testing," in *international CAN Conference*, 2002.

[32] O. Pfeiffer, "Selecting a can controller," *Embedded Systems Academy. Disponível em:¡ http://www. esacademy. com/en/library/technical-articles-and-documents/can-andcanopen/selecting-a-can-controller. html¿. Acesso em*, vol. 6, 2010.

[33] M. Anoop, "Security needs in embedded systems.," *IACR Cryptology ePrint Archive*, vol. 2008, p. 198, 2008.

[34] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Moderator-Ravi, "Security as a new dimension in embedded system design," in *Proceedings of the 41st annual Design Automation Conference*, pp. 753–760, ACM, 2004.

[35] A. Tragha, F. Omary, and A. Mouloudi, “Improved cryptography inspired by genetic algorithms,” in *ICIGA, 2006 International Conference on Hybrid Information Technology (ICHIT’06).*

[36] A. Tragha, F. Omary, and A. Kriouile, “Genetic algorithms inspired cryptography,” *AMSE Association for the Advancement of Modeling & Simulation Techniques in Enterprises, Series D: Computer Science and Statistics*, 2005.

[37] A. Soni and S. Agrawal, “Using genetic algorithm for symmetric key generation in image encryption,” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 1, no. 10, pp. 137–140, 2012.