8-2006

Migration from Teleoperation to Autonomy via Modular Sensor and Mobility Bricks

Roselyne Dalanda Barreto
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Roselyne Dalanda Barreto entitled "Migration from Teleoperation to Autonomy via Modular Sensor and Mobility Bricks." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mongi Abidi, Major Professor

We have read this thesis and recommend its acceptance:

David Page, Seong Kong

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Roselyne Dalanda Barreto entitled "Migration from Teleoperation to Autonomy via Modular Sensor and Mobility Bricks." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mongi Abidi

Major Professor

We have read this thesis
and recommend its acceptance:

David Page

Seong Kong

Accepted for the Council:

Anne Mayhew

Vice Chancellor and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

Migration from Teleoperation to Autonomy via Modular Sensor and Mobility Bricks

A Thesis
Presented For The
Master of Science Degree

The University Of Tennessee, Knoxville

Roselyne Dalanda Barreto

August 2006

Acknowledgement

I would like to thank Dr. Mongi A. Abidi for giving me the chance to join the IRIS lab and for making the preparation and completion of this thesis possible. His encouragement and support throughout this experience have allowed me to have a great start in my professional career. I sincerely appreciate the opportunity you have given me and I am forever grateful.

I would like to acknowledge Dr. Andrei Gribok for advising me and guiding me through my first year at the IRIS laboratory. I would like to thank to Dr. David Page for helping me and keeping me focus towards the completion of this degree. I truthfully am thankful for the incredibly helpful and understanding advisor you have been for me. Your advice made me a better student and a better person.

I would like to mention my colleagues at IRIS and especially at IRIS West: Nikhil Naik, Tom Wilson, Kim Kate and Doug Warren for making my experience at the lab unforgettable. Special Thanks to Chang Cheng and Chung-Hao Chen for sharing their programming skills, their work experience and their personal experience. I am lucky to have work with all of you.

Last but not least I would like to thank my family for their constant emotional support. I am hereby thanking my father Philippe Barreto, my sister Vanessa Barreto and my brother Nilton Barreto. To my mother Marie Madeleine Spencer I say you have been the greatest influence in my life always helping place me where I need to be. There is no way to repay you but I would like you to know you are appreciated. Without all of you none of this would have been possible.

Abstract

In this thesis, the teleoperated communications of a Remotec ANDROS robot have been reverse engineered. This research has used the information acquired through the reverse engineering process to enhance the teleoperation and add intelligence to the initially automated robot. The main contribution of this thesis is the implementation of the mobility brick paradigm, which enables autonomous operations, using the commercial teleoperated ANDROS platform. The brick paradigm is a generalized architecture for a modular approach to robotics. This architecture and the contribution of this thesis are a paradigm shift from the proprietary commercial models that exist today. The modular system of sensor bricks integrates the transformed mobility platform and defines it as a mobility brick. In the wall following application implemented in this work, the mobile robotic system acquires intelligence using the range sensor brick. This application illustrates a way to alleviate the burden on the human operator and delegate certain tasks to the robot. Wall following is one among several examples of giving a degree of autonomy to an essentially teleoperated robot through the Sensor Brick System. Indeed once the proprietary robot has been altered into a mobility brick; the possibilities for autonomy are numerous and vary with different sensor bricks. The autonomous system implemented is not a fixed-application robot but rather a non-specific autonomy capable platform. Meanwhile the native controller and the computer-interfaced teleoperation are still available when necessary. Rather than trading off by switching from teleoperation to autonomy, this system provides the flexibility to switch between the two at the operator's command. The contributions of this thesis reside in the reverse engineering of the original robot, its upgrade to a computer-interfaced teleoperated system, the mobility brick paradigm and the addition of autonomy capabilities. The application of a robot autonomously following a wall is subsequently implemented, tested and analyzed in this work. The analysis provides the programmer with information on controlling the robot and launching the autonomous function. The results are conclusive and open up the possibilities for a variety of autonomous applications for mobility platforms using modular sensor bricks.

Contents

Tables

Figures

1 Introduction

1.1 Overview

Robotic Systems are a promising but challenging research area. While significant progress has been made in teleoperation, research on full autonomy is still limited. It usually focuses and performs better on autonomous navigation or simple assembly tasks. Even as the studies on autonomous systems advance, in certain cases, for safety and security reasons, removing the operator from the human-robot loop is not an option. Instead, assisting the human operator by adding autonomous functions to a teleoperated robot is the proposed goal of this work. There is usually a trade off when acquiring a degree of autonomy; studies have shown that the robot's effectiveness decreases as its autonomy increases [1], [2] and [3]. This project however proposes to keep both teleoperation and autonomy in the same system to minimize the trade offs and the restricted applications. In other words, the objective of this thesis is to integrate both teleoperation and autonomy into one system using mobility and sensor bricks.

Teleoperated robots are opposed to intelligent systems in that they absolutely require human guidance. Autonomous systems use sensor to independently perceive and act on their environment. This trend is referred to as active sensing as opposed to passive sensing when the robot is only collecting data. Hence going from one system to another involves adding sensors to the teleoperated robot. While the principle is simple, actually adding sensors to a robot is a complicated task. Teleoperated machines usually include a few simple sensors such as surveillance cameras. The visual output is typically sent to a control station from which the robot is being controlled. Adding sensors to the robot implies physically adding an on-board unit capable to access the sensors and control the robot. In other words it means a lot of hardware changes need to be done before autonomy takes place.

Autonomous robots also have built-in sensors such as range sensors and encoders that they use to localize and position themselves. The focus of most self-sufficient robots is to autonomously navigate through certain terrains or execute well-defined simple tasks. This means that even when certain robots are autonomous or semi-autonomous they are usually task specific. They have been built to meet a precise need. Therefore it seems like going from teleoperation to autonomy also requires a predefined goal. When the environment or circumstances change, the autonomous robot cannot adapt without adding different sensors and corresponding self-ruling functions.

The migration from teleoperation to modular autonomy emphasizes two main points. First this transition is simple and does not involve major hardware changes on the initial system. Second, the autonomous functions are not limited to a few basic sensors and a few specific applications. Overall the anticipated system combines teleoperation and autonomy in a very advantageous way as it does not choose from either technology but rather combines them into an easily convertible system. Before the theory and methodology, resides the motivation of this work. A few examples of applications will also help clarify the usefulness and need for this new hybrid robotic system.

1.2 Motivation

There are several motivation factors for creating a teleoperated robot with autonomous capabilities, the main factor being keeping the security and safety of the operator. Other issues are the modularity and flexibility of the system, which involves easily replacing, updating or combining several kinds of sensors for data collection and data fusion. Adding autonomy implies more efficiency and accuracy in a priori known environments. Indeed in well-defined circumstances robots are much more efficient and precise than human operator, especially untrained operators.

1.2.1 Safety and Efficiency

An appropriate example of robust teleoperated robots is the Remotec ANDROS robot series. The company specializes in reconnaissance and hazardous waste or bomb disposal robots. Remotec robots are designed to support the requirements and needs of federal, state and municipal law enforcement agency Explosive Ordnance Disposal (EOD) organizations [4]. ANDROS robots are designed to assist expert technicians when performing remote reconnaissance, access, render safe, “pick up and carry away” (PUCA) [4], and disposal during extremely hazardous explosive ordnance missions. Those robust machines are capable to navigate in rough terrains, climb up to 37 degrees stairs and withstand the force of minor explosions. Obviously in such delicate circumstances human operators must supervise the robots. Therefore the main reason to keep the teleoperation option in the new system is allowing the operator to take over at as soon as a situation becomes too dangerous to be handled autonomously by the robot. Beside the safety issue, certain tasks are just too complicated for the robot to undertake unassisted. As much as robotic autonomy is being developed there are still challenges that limit autonomous robots to simplistic tasks such as pulling, pushing and picking up objects. Even if complex engineering systems are implemented for a robot to identify, recognize and perform a task, an expert can be a lot more cost and time effective by guiding a less complex machine in performing the same exact task.

1.2.2 Modularity and Flexibility

Robotics is moving toward modularity, flexibility and portability. Modular robotics breaks unmanned systems up into several components. While components come together to create one system, they are each semi-independent unit that can be reused, replaced,

removed, added or separately updated. The first great advantage of such a system is its life-cycle cost. When one component breaks down the whole system does not go down as technicians can easily find a replacement for the defective part. Hence the entire system does not go to waste and there is no need to purchase a whole new one. Similarly when there is maintenance or new development on parts of the system, it does not completely become outdated and unusable since certain parts can be reused and reassembled with updated components. At a larger scale than just one machine, modular robotics allows parts to be interchangeable as needed between different robots. Another improvement in this technology is the flexibility that it offers. In the case of the migration from teleoperation to modular autonomy, the motivation is not to be so restricted in the kind of autonomous function to be added. Indeed in this case, modular sensor bricks can, not only be reused, replaced or independently be updated, but they can also be combined. The term brick refers to an independent sensor system. Physically attaching the sensors to the robot would limit the number and sorts of sensors to be linked to the system. Using sensor bricks allows using any desired number of sensors, which is important for several different kinds of autonomous functions, for data collection and data fusion. This way the autonomous robot is not so task specific and can be fit to various and diverse applications.

1.2.3 Autonomy and Capability

While part of the motivation behind this work is to keep the operator involved another element is to alleviate his or her work. Indeed while human guidance is very important and required in delicate bomb disposal operation, robots are generally a lot more efficient than human operator in simple tasks performance. An untrained person cannot operate most of the commercial robot. The operators must be trained and become expert technicians. With time and experience one can acquire the dexterity necessary to operate this heavy machinery. Adding semi-autonomous to autonomous functions to the control systems makes it easier for a new operator to execute certain tasks. Considering that in a priori known environment and well-defined circumstances a robot is more efficient than a human, giving the robot a certain degree of autonomy can also help even the best-trained operator. During operation, allowing the technician to focus on the most important aspects of a certain task and delegating the rest to the robot can be a useful autonomous feature. For example, while the operator is focusing on a manipulating task, the robot can autonomously keep its own survival by avoiding collisions. This is however just one illustration of autonomy. The greater motivation here is not to meet the need for a specific application. It is rather to allow communication between a robot and various sensors to execute diverse autonomous functions. The connection between a mobility platform and several different kinds of sensor bricks allows the implementation of as many autonomous functions from simple data collection to path planning and obstacle avoidance.

1.3 Applications

The main applications of the projected system are security and safety oriented. An obvious reason to operate or completely delegate a task to an unmanned system is to protect human beings from probable or anticipated danger. This type of application best fits organizations such as the Department Of Energy (DOE). Other applications are more general and could be applied to fields such as the Automotive Research Center (ARC) and others in which scanning processes are required.

Safety and security are at the core of DOE operations. For example DOE deals with chemical safety, nuclear safety and hazardous waste transport. Teleoperated or autonomous unmanned systems are desirable in such applications mainly to avoid contact between human beings and potential harmful contamination. Among several scenarios showing the usefulness of the anticipated system for the DOE, one major picture is the crucial surveillance of its facilities. A semi-autonomous agency with DOE, which could use surveillance robots, would be the National Nuclear Safety Agency (NNSA). NNSA enhances national security through the military use of nuclear energy. It not only improves nuclear weapons but also responds to nuclear and radiological emergencies in the US and abroad. To prevent terrorists from accessing dangerous material, a robot equipped with a surveillance camera and a range sensor can periodically go around the perimeter of a particular building and look for intruders or other anomalies. This way an operator does not have to constantly supervise the robot and will only be alerted in the case of an emergency. It is safe since a robot cannot get hurt; it relieves the operator who can focus on another important task. Moreover the robot behavior does not include loss of focus, fatigue and other human imperfections.

The ARC develops simulations and designs for mobile platforms to later be applicable in real environments. The work of ARC revolves around five thrusts areas. Throughout the different thrusts areas, the main emphasize remains the collection and fusion of data. Before modeling and simulation is possible data has to be scanned and analyzed. The teleoperated or automated scanning process is a very useful application for ARC. Having a robot communicating with the sensors involved is a practical and efficient way to collect data on a precise path and at a constant speed. Of course the scanning vehicle could be a car and not necessarily a robot. However as mentioned above driving a car can imply more errors in the process. Moreover for huge profiling mission sending "*a well-organized army*" of robots perceptibly has several advantages over sending drivers in automobiles. There is also a *reachability* factor in the scanning process. A robot can access remote areas that a human cannot or should not access. Aside from hazardous sites where it would be undesirable to send a human, robots can scan hard-to-reach areas such as the undercarriage of a car for example. The robot could also be a flying helicopter scanning over a certain field. Using mobile platform equipped with sensors can redefine and expand the meaning of scanning processes for ARC and other similar laboratories.

Under vehicle inspection is a scanning process of special interest at the Imaging, Robotics and Intelligent Systems (IRIS) laboratory. It combines the security aspect and the scanning process in an application for a teleoperated to autonomous system. The teleoperation removes the operator from the scene out of a safety concern in the case of hazardous material hidden in or under the car. Moreover scanning under a car with a mirror on a stick only covers thirty to forty percent or the under carriage of a car. Hence sending a robot with sensors under the vehicle is safer and more efficient than sending an operator to conduct the inspection. An inspector may drive the robot to the vehicle and then let the robot take over. Performing an autonomous scrutiny facilitates the task of the operator, who may at time loose sight of the robot. Having access to several sensors during the scanning process reveals more information than the eye can tell. For example visual data can be deceptive. A fake muffler placed under the car could actually hide a threat that a visual camera would not detect. Alternatively using an additional thermal camera automatically reveals important details about what part should or should not be hot while the engine is running. Similarly a range sensor can differentiate between the picture of a muffler or a real muffler by providing 3-D information. Such complete scanning can be extended to the inside of the car, or the inside of a suspicious room. Figure 1 illustrates the difference between a human and a robot inspector.

Scouting missions are another promising application for the robot. With a few images of a specific scene allowing the operator to simply point at objects of interests on a screen instead of driving the robot to different places would be a great autonomous function to a robust teleoperated robot. For example the operator would point to a suspicious object and request a close up picture of that object. The robot would then take over, drive itself to the target destination and come back with a picture. This mission implies that the robot is equipped with at least a visual and range sensor to be able to avoid obstacles, reach its target, collect data and *home* back to its original position. Such application is especially desirable for agency such as the Weapon of Mass Destruction Civil Support Team. Those military teams were established to protect U.S. citizen against the growing threat of chemical and biological terrorism. They are spread out in the different states to support state and local authorities in the event of incident involving weapons of mass destruction. They are equipped with personal protective suits and decontamination kits. They carry special equipment to detect the source of toxic agents, digital still and video camera and other sophisticated tools to help identify the nature of the threat. Then they return to their special communication van for modeling and simulation on the computer and for laboratory analysis of potential samples. Some limitations include the fact that a man should not stay in those suits for more than an hour, which implies several rotations between several agents before the scene is completely analyzed. The analysis only starts when the scouting process is over. Using an unmanned system equipped with several sensors including chemical and biological sensors in particular allows continuing the inspection as long as necessary. While the robot cannot get contaminated and can stay exposed longer than a human without perishing, the operators inside the van can start identifying the threat and react consequently. Guiding the robot when desired or pointing

Driver interrogation conducted by a human.

Driver interrogation conducted by a robot.

Trunk inspection conducted by a human.

Truck inspection by a robot.

Under vehicle inspection using a mirror.

Under vehicle inspection by a robot.

Figure 1: The images on the left describe a human operator conducting a vehicle inspection. The images on the right show a robot conducting the same inspection.

the robot to specific locations at time, they can thoroughly cover terrain through communication with the robot and its sensors.

1.4 Contributions

The concept of an adjustable autonomy has been examined before under different angles, especially the trade offs between an entirely autonomous system and a teleoperated robot. While this topic will be discussed later in the literature review, one can say that in general the work is usually done on a robot already built with the capability to support simple autonomous function. Hence the first contribution is to reverse engineer an industrial system and add a computer interface control to a commercial robot initially intended only for teleoperation. This first step is crucial in order to later add intelligence to that robot. It is an advantage because those robots are usually very robust and the only reason they do not include multiple sensors are practicality and cost efficiency. Again as extra sensors are added to a robot it becomes a little more task specific and therefore applications restricted. Those are not desirable features for a company building series of robots. A chemical sensor or a nuclear sensor is often meant for a particular application and would be not be useful for a basic surveillance robot for example. The capability to reverse engineer commercial robots and make them controllable through a computer interface is an inexpensive advantageous alternative to built different specialized robots. Such system can be converted first from the proprietary teleoperation to a complex teleoperation using several special sensors. Its functionality can later be expanded through relatively simple software changes versus complicated hardware changes.

The second contribution is an extended definition of a mobility platform as a mobility brick. Mobile platforms are not an innovation, however they are usually perceived as robots in themselves. Therefore they are either teleoperated or they integrate sensors with on-board intelligence to navigate autonomously. The contribution here is to blend in this definition with the whole brick concept. The notion of sensor brick has formerly been established [5]; the idea of a mobility brick in the larger scope of an interoperable Modular Robotic System is a major contribution of this thesis. Before any autonomy takes place, the mobility brick paradigm provides the information necessary for any operator to access the drive commands of the platform. In the greater picture of the Sensor Brick Concept, different robotic platforms are transformed in series of mobility bricks and are no longer controlled by their original proprietary controller. Instead, they become accessible by various control units located in a central unit or other bricks. Any computer equipped with the appropriate information can drive the mobility brick.

The last contribution is derived from this greater picture of the brick technology. Acquiring autonomy through mobility and sensor bricks widens the areas of applications especially when several sophisticated sensors are required. As different sensors can be rotated or combined together the robots become more versatile. This project does not propose to build a navigation or task specific autonomous robot but rather to allow

implementing several types or autonomy using the mobility and sensor bricks concept. Figure 2 shows the main objective of this work and highlights the claimed contributions.

1.5 Organization of this Thesis

This document will be organized as follow; Chapter 2 will be a literature review to acquire a general knowledge of the different kinds of teleoperated and autonomous robots. In other words this chapter will attempt to examine the state of the art in robot autonomy. Chapter 3 will discuss the core of the proposed concept. This chapter will describe the transition from teleoperation from the proprietary control box to a computer interface control box. After this reverse engineering section, Chapter 3 will explain the subsequent evolution towards an autonomous robot using the sensor bricks. Chapter 4 will look more closely into the previously defined sensor and mobility bricks concept. While Chapter 3 describes the theory of this research Chapter 4 will focus on the implementation of the proposed idea. Chapter 5 will then present how the resulting system has been tested and analyzed. Finally, Chapter 6 will conclude the thesis based on its results and suggest future work to be conducted on this system or other prototypes.

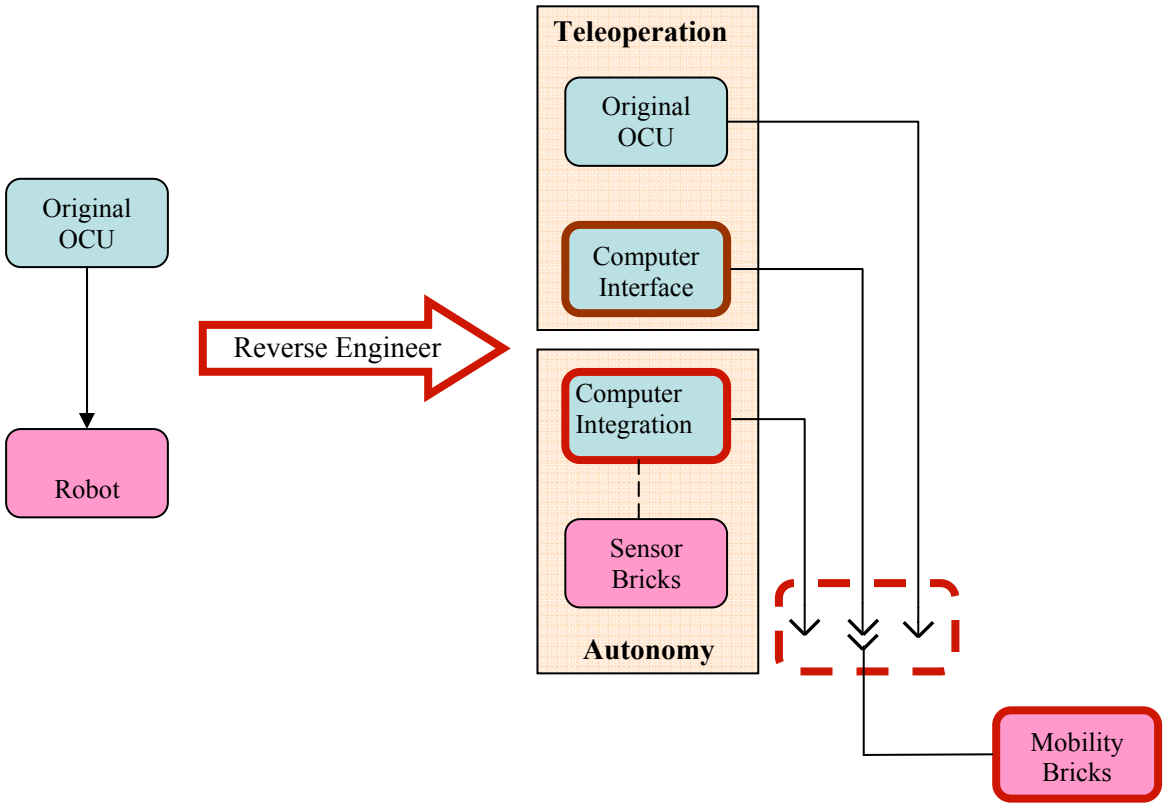


Figure 2: This diagram show the general objective of this thesis and underlines its contributions.

2 Related Works

Chapter 2 examines the state of the art of the research on autonomy. This chapter starts by giving an overview of the autonomy concept, its origin and meaning. It further discusses its principle and different levels. Finally the last section of this chapter summarizes and analyzes the different methods to implement autonomy.

2.1 Overview

The word “automation” comes from the Greek word “autonomos” In the etymology of the term autos means self and nomos means rule or law. That is to say that an autonomous individual makes its own rules as opposed to following those of an external governing power. Autonomous robots are aimed to be physical entities that can accomplish useful tasks without human intervention. They are supposed to operate in an unknown environment without receiving direct instructions from users. This is a very complex project; therefore it involves several objective difficulties. The first section of this chapter will help understand and explore the meaning of autonomy. This definition implies various real-time difficulties and possible solutions, which will also be discussed in the same section. The second section will look at the bigger picture about autonomous systems. Without going into detailed implementation and architecture, it will concentrate on the different levels of autonomy. As a complete evaluation of this broad subject is beyond the scope of this review, it will examine the work of a few researchers on robots with different degrees of autonomy. The last two sections will present important factors to consider in going from teleoperation to autonomy.

2.2 Principle of Autonomy

As mentioned before autonomous systems are physical systems capable of operating without direct human intervention. In other words it should perform its role while maintaining its own viability [6]. This definition can be compared to that of a living system, which also adapts to its environment for survival. Autonomous robots must adapt to their environment even if those change. To be autonomous first implies being automatic. In turn being automatic means sensing the environment and its impact on your existence. Autonomy goes beyond automaticity in the sense that an autonomous robot not only realizes the impact of the environment on its existence but further adapts to the environment changes to insure its survival. Moreover this adaptation process has to happen in real-time as opposed to studying an environment in advance before operating. For example it is less complex to program a robot to go around an empty room without human supervision or intervention than it is in a room filled with obstacles. It is also less difficult to accomplish the same task in a room with obstacles when the programmer has a priori knowledge on the room then it is when the environment is unknown. What will

the robot do if the dimensions are different, if it comes to a still obstacle, a moving obstacle? An automatic robot that ran in an unexpected wall will keep going against it until its motors burn, while an autonomous robot will know to go around the obstacle. This robot should not only know to adapt to the environment (avoid the wall) but also learn from it. In other words it should remember the characteristics of this obstacle in case it should take the same rout sometime later. In addition, according to Kasabov [7], if it does encounter the same obstacle a second time it should have learned enough about it to avoid it even more efficiently than the first time and this learning process should go on as the robot finds itself in the same or similar situations. An Intelligent Agent System (IAS) should then have parameters that represent short and long term memory, age, forgetting, etc [6]. Kasabov [7] states that an IAS should be able to analyze itself in terms of behavior, error and success.

2.2.1 Definition

Robotic systems are aimed to perform services. Autonomous robotic systems are aimed to perform the same services while maintaining themselves. Those systems have to self-govern themselves in order to insure their survival independently of external changes. They have to be adaptive because users and environment change frequently. They have to learn from the different circumstances they encounter and improve their existences.

2.2.2 Problems

From the definition above arise several problems, which reside in the process of acquiring data and learning from the environment. The fact that the autonomous robot is expected to adapt to its surroundings in real-time supposes that it is able to sense still and moving object with respect to itself. Moreover *autonomaticity* supposes that the robot already has a general knowledge about how to interpret and handle the sensors' information.

2.2.3 Sensing

There are several issues in the sensing part of robot automation. The quality of sensor information is influenced by sensor noise, the limited field of view, the condition of observation, and the inherent difficulty of the perceptual interpretation process [6]. Thus those issues have to do with how the robot receives the information from sensors. Assuming the robot knows how to handle a specific situation, noise-corrupted information may lead to a malfunction in the robot operation. Similarly limited field of view may engender problems in the robot navigation and operation. The condition of observation such as illumination, angle of view, motion and others all have to be taken into consideration by the robot when sensing information to avoid misinterpretation. It is easy to see how ambitious and complicated it is to want to attribute the human notion of sensing to a robot.

2.2.4 Control

Making sense of sensor information leads to the problem of controls in autonomous robot. Once again for the robot to correctly interpret sensor information it needs to have prior general knowledge about the environments in which it operates. An obvious difficulty is that it is impossible to have complete and exact prior knowledge about these environments. Real-world environments are characterized by a large amount of uncertainty that even human beings cannot predict. Therefore it is impossible to prepare the robot to handle all the situations that it will encounter. Problems can range from the effects of varying environmental conditions on the robot sensors and traction performance through to the need to deal with the presence of unexpected situations [6].

The goal for an autonomous robot control then would be to start with a basic database about possible situations and later learn and improve its database with experience. This simple, logic idea brings up another important issue in control systems called epistemic actions. Epistemic actions require that the programmer build the "basic" database mentioned earlier with additional rules so that the right actions are fired at the right time. Epistemic actions not only need to be fired when information in the database is missing but also when it is out of date [8]. Indeed the information in the database has to be kept coherent with the sensory systems and the external world for appropriate reaction from the robot. This cache system problem, called model coherence problem is an instance of a more general problem in computer and robot design. Real robotic systems consist of several sub-systems, clusters of sensors, motors and databases operating in parallel, on separate processors. Often these simple processors don't have their own operating systems. Therefore all their information must be consistent with one another and the external world as they are gathered in one central operating system.

In summary automated reasoning systems are typically built on a task-oriented model of programming. Some basic prior knowledge is stored in a database of assertions in a number of logical languages, indexed perhaps by predicate name [8]. The robot receives information from sensors and functions by query. It asks the system about the findings of the sensors, about how to handle them, finds the answer and reacts in the appropriate manner. Ideally if the result of the query is missing or out of date it adds or modifies the database. How this is done, or even how the database is filled in the first place is the essence of the problems in control architectures of autonomous robots.

2.2.5 Control Architecture

Control architectures for autonomous robots should be able acquire sensory information, adapt to unstructured changing environments and fire the desired reactive behavior. This behavior can be to move in a certain direction or it can be task oriented. Autonomous robots should be modular i.e. subdivided into smaller modules that can each easily be replaced or updated without major software modifications to the main system. All those facts need to be taken into account when building those robots. Several architectures have been proposed including four main ones [9]:

- In the NASREM architecture, developed by Albus, [10, 11], the information passes through several processing stages until the system understands the current situation. Then the commands are sent again to several modules until the desired action is taken at the lowest levels.
- The subsumption architecture proposed by Brooks [12] is layered in several communicating levels of competence. This architecture brings the robot to a higher level of competence using higher layers with access to the lower layers of operation. Lower layers operate at simpler and more basic levels.
- The Task Control Architecture (TCA) architecture, developed by Simmons [13, 14] includes a general purpose central unit linked to several task-specific modules. This system is interconnected; there is no higher layer but rather a common central control responsible for getting the right sensing information to the right reactive module.
- The Local Area Augmentation System (LAAS) architecture, proposed by Alami et al. [15], is composed of three layers. The highest level does the global planning; the middle layers receives tasks from the higher level, supervise their execution while being reactive to unexpected events. At the lowest level, the third layer does elementary robot tasks and functions such as perception or motion.

Those four architectures are used separately or together to implement four main paradigms for autonomous robots control systems.

- Sense-model-plan-act (SMPA). This basic idea is used in most autonomous system. However the single route between sensing and acting causes undesirable delays between the two. Hence this primary paradigm has been improved by horizontal and vertical decompositions.
- Vertical decomposition, as its name indicates, vertically splits into hierarchical levels. Sensing information and commands all flow up and down between the different layers. Reasoning at higher levels implies more planning and less interaction with the environment, hence less delays in reactions. This paradigm however requires protocols between layers which reduces modularity and expandability.
- Horizontal decomposition, as suggested by its name, this paradigm remains at a simple low level of operation. It is very reactive because it uses the idea of a central unit communicating with several modules without a hierarchical structure.
- Reactive systems are based on the fact that behaviors tend to be little or not goal directed [9]. These systems neglect the model-plan part of the SMPA paradigm and follow more of a sense act behavior.

Different architectures can be used to perform the same tasks. Some paradigms perform better in different circumstances. While various scientists argue in favor or against particular one, numerous researchers are supporters of hybrid architectures. In other

words, they avoid the disadvantages of specific architectures and make the best of advantages of others by combining them in the same systems.

2.2.6 Online Learning

An important general concept for autonomous robot is the concept of o*nline learning* versus simulation. According to the definition of autonomy a fundamental requirement for automated systems is that it be able to carry out tasks in the real world and in real time. Moreover it must be able to adapt to its environment as it performs those tasks. Still several scientists test their robots by using simulation. Results of simulation can easily be criticized. Indeed numerous successful simulations will fail on real robots because of the following reasons [6, 16].

- Numerical simulations do not usually consider all the physical laws of the interaction of a real agent with its own environment, such as mass, weight, friction, inertia, etc.
- Physical sensors deliver uncertain values, and commands to actuators have uncertain effects, whereas simulative models often use grid-worlds and sensors that return perfect information.
- Physical sensors and actuators, even if apparently identical, may perform differently because of slight variations in the electronics and mechanics or because of their different positions on the robot or because of changing weather or environmental conditions.

Learning online enables the robot to adapt to real conditions and circumstances as adaptive behaviors can only emerge from coupling the agent with its environment and not from simulation.

Having underlined the advantages of online learning versus simulation in automation and before looking into advanced research topics in this areas here are the most popular methods to develop robotic agents.

2.2.7 Methods to Implement Autonomy

2.2.7.1 Path Planning

Once a robot is given a goal position; it determines a collision-free path to navigate through from its initial to its final position. This process is called path planning and can be categorized into global and local path planning. Global path planning is usually used when the environment is known a priori. It is used with simple exact model of a real environment. Again as discussed above, real-world environment are never simple enough. Therefore for autonomous system local path planning is preferred. It is more practical and is based on obstacle avoidance. The simplicity of this idea is attractive but it has its problem also. Basing the motion only on avoiding obstacle could cause the vehicle to get stuck in corners. For example if two different sensors are near obstacles and trying

to move the robot in different directions it would get stuck. These situations would translate in local minimum in the local path planning algorithms. Second, this method causes unstable motion near obstacles. Typically there is no speed and direction control integrated in those avoidance algorithms, which leads to sudden and unstable motion when deviating the vehicles. The key idea here is adding mapping to this mechanism so that the vehicle knows more about its surrounding when reaching an obstacles. As this method does not however provide the robot with situation-action rules other methods propose an alternative.

2.2.7.2 Neural Networks

Neural networks have the advantage to learn by example, generalize from those examples and apply their resulting algorithms to specific situations. However these neural networks require substantial data sets and representative patterns to characterize their environment during training [3]. Moreover the number of hidden layers and nodes are parameters that are fixed by the programmer. On the one hand, too many hidden units overfit the training data and fail for testing data. On the other hand too few hidden layers produce a behavior that is too generalized. Hence the programmer would have to have a lot of experience with real environment and circumstances to implement the right number of layers. Again even if it were possible to store enough data to navigate in unstructured changing environment it would require a substantial amount of data.

2.2.7.3 Fuzzy Logic

This method seems to better solve the problem of unstructured changing environment. Indeed Fuzzy Logic method provides means for mapping sensor information in real-time. Again this is a major requirement for autonomous robots which must have a degree of self-government in unknown environment without human intervention. The success of Fuzzy Logic Control (FLC) is owed in a large part to the technology's ability to convert qualitative linguistic descriptions into complex mathematical functions and the ability to deal with various situations without analytical model of the environment [8]. In other words this methodology takes away the problem of having to mathematically describe an environment that is unstructured, changing and described only qualitatively, inexactly and uncertainly by sensors. This approach translates particular situation into functions that the autonomous agent can understand. It is then particularly useful in the case of obstacle avoidance. When compared with path planning this method provides actual mapping of the surroundings and provides the vehicle with more information about what to do after avoiding obstacles. When compared with the neural network approach, the algorithm is not based on experience but on expert knowledge. This is because the rules are based on physical meaning rather than on training and experimenting. Hence this approach seems the best fit for autonomous robot navigation. However physically describing very complex environments remains a challenge even for human experts. Instead of a large amount of data (neural networks approach), a large amount of rules have to be constructed to begin with. It is a tedious, time consuming process and leaves a lot of room for research and improvement.

2.2.7.4 Evolutionary Algorithms

This method is inspired by the principles of natural evolution and genetics [8]. Genetic Algorithms (GA) uses processes such as selection, recombination and mutation to solve robotic problems. Unlike fuzzy methods, evolutionary algorithms are based on global and not iterative searches in the solution space. They are more efficient and faster than iterative searches. The optimization process is simple and does not care about the problem itself just about what solution best solves it.

The GA approach enriches the optimization environment for fuzzy systems [8]. As mentioned before fuzzy rules can be very complex. However the GA approach assumes that the solution space is complete and fixed and therefore cannot learn Online. An attempt to solve this problem has been to develop the Experience Bank. As the robot encounters problems it searches through this Experience Bank, if one of these experiences solves the problem the search ends, otherwise each experiences are "graded" on how well they fit this particular situation. The highest fitness experience is used as a starting position to the lower level GA that is used to generate new solutions to the current situation [8]. This approach preserve the advantages of easy searches and speeds up the process of creating new solution space sections by starting at the best possible place in this space. Scientists were inspired by the mechanism of biological organism which functions according to two important processes, evolution and life long learning. Evolution takes place naturally and affects the nature of the organism while life long learning takes place at an individual level. Similarly the Fuzzy-Genetic system (the Associative Experience Engine) [8] develops the evolution process; an online technique is then added to implement the life long learning process.

2.2.7.5 Reinforcement Learning

This method is based on direct trial and error interactions with a dynamic environment. Reinforcement Learning (RL) is a learning strategy that does not characterize a learning problem. RL just considers possible behaviors, finds one reaction that performs well for one particular problem and applies it. It does not try and optimize solution; it does not care to know if another solution would have been better from a long term output point of view. It is just important that it has a lot of action-reaction contact with its environment. In other words the RL method looks at different situations as utility problems rather than optimization problems. After interacting with its environment the robot must gather and store lessons from its experiences. This approach contrasts with the traditional supervised learning algorithms. Indeed RL works towards immediate rewards while supervised learning works toward the best output from a given input. Another difference from supervised learning is that on-line performance is important: the evaluation of the system is often concurrent with learning (which seems attractive in online learning in unstructured environments) [8].

2.3 Levels of Autonomy

Robotics systems range from teleoperated to fully autonomous. In teleoperated systems the human operator has full control over the robot's behavior while there is no human intervention in fully autonomous systems. Telerobotics describe robotic systems, which despite human guidance have a degree of autonomy.

2.3.1 Telerobotics

Teleoperated robots require continual operator intervention to successfully perform a task [17]. This type of robotic systems is used in delicate applications such as clearing hazardous waste. In those dangerous cases, human operators are needed to maneuver the robots to prevent accidents. Often operators only have a virtual contact with the robots and need to be experienced and extremely precise in guiding the robots. Such operation is recognized to be difficult on the users. In an attempt to solve this problem, Conway [18] introduces the concept of "teleautonomy". He discusses methods to generate intelligent actions at distance. This method blends autonomy with human intervention. The degree of autonomy that the robot should have is the subject of discussions especially in hazardous tasks. Telerobotics blends human supervision with robot local intelligence. While researchers debate on how much autonomy the robot should have in performing a task, one safe focus in this matter is the robot equilibrium and survival. Sian, Yokoi, Kajita and Tanie illustrate this point in several papers [19, 20] on which this section will focus. Apart from walking pattern generation, the work previous to theirs has generally focused on arm or head manipulation of a static body. The few work on whole body motion has attempted to convert the body motion of an operator into command for the robot. This work however is very involved, requires complex interface and still causes problems such as difficulty in generating stable motions in real-time, due to geometrical and dynamical differences between human operators and humanoid robots. These authors propose a switching command based whole body teleoperation based on simple joystick interface. The concept of integrating operator's intention and robot autonomy is derived from human motions, which are a mixture of conscious motions and unconscious motions. When accomplishing a specific task, we consciously take actions while unconsciously taking others to accommodate us in our task. For example while running one focuses on his or her leg movements and unconsciously moves ones arm to stay balanced. Based on this idea Sian, Yokoi, Kajita and Tanie [19] have developed a system where depending on the objective of each task, the operator selects the specific part of the robot he wishes to operate with. The operator does so with simple joystick control without worrying about the robot balance. This autonomous motion of the robot is insured using a trajectory of the target manipulation point and the robot balance as the criteria for whole body motion generation. These scientists propose a method that divides the body structure in several mechanisms corresponding to the main joints of the robot. The operator controls the motion and velocity of each joint he wishes to operate. The robot maintains itself using two principals.

- *Balance Autonomy* Based on the measurement of the Zero Moment Point (ZMP) this mechanism allows the robot to remain within the support polygon for dynamically stable motion.

- *CoM Position and Torso Orientation Modification Autonomy* Based on the measurement of the center of mass (CoM) and the orientation of the torso, the stable reachable area of a humanoid robot can be extended [19].

The same authors added to their work by proposing an intuitive foot operation [20]. As the previous one, this approach is based on the fact that with current recognition and decision-making technology, human supervision is still necessary for humanoid robots. Hence this method proposes to incorporate the operator's foot command with the robot's autonomy in maintaining balance. Most of the work on humanoid robots' autonomy concentrates on generating walking pattern. The issue of real-time foot operation is rarely discussed [20]. This paper emphasizes that one great advantage about humanoid robots is that they have feet that can be used for more than walking. For example they can be used to step on specific areas or to push an object around. In those cases it is important to facilitate the operator's work by allowing him to only transmit intuitive commands to the robot and let the robot maintain its own balance. The contribution here is the Foot Operation Autonomy.

2.3.2 Semi-Autonomy

Semi-autonomy also referred as scripted autonomy describes the systems in which the user triggers behaviors in the robots through voice, touch or other mechanism. These automated reasoning systems are typically built on a transaction-oriented model of computation [8]. Knowledge is acquired from the environment and stored in the robot database. Then the system translates queries sent by an operator into logical assertions. Even assuming it is possible to clearly describe several queries so that the robot can answer them, the problem occurs about how to fill the database. In the case of greatest interest, the robot does not know its surroundings and its environment changes frequently. However automated systems are not prepared for such cases, they assume all the information they need is already stored in the database before operation. Semi-autonomous behaviors can be triggered by different mechanisms. Language is a favorite for researchers. This section will focus on such systems. They are called tagged behavior-based systems.

2.3.2.1 The Bertrand System

The Bertrand system [21] is a database-free logic programming system that answers block-world queries using real blocks and a real-time visual system [8]. This system answers questions such as "Is there a blue object above a red object?" using visual routine processor. The robot searches for blue block, and then looks right below. It finds a red object and drives to the chosen block. If it had chosen the wrong blue object it would have backtracked and searched for another blue object.

2.3.2.2 The Ludwig system

Ludwig [22] is another simple natural-language question answering system based on colors and special relations. It is grounded in real-time vision. It is different than the Bertrand system in that it consists of parallel network communicating finite state machines. Semantic analysis, synthetic analysis and visual processing occur in the pipeline and the system keeps track of the relationship between those programs.

2.3.2.3 Improvement of the Bertrand-Ludwig Architecture

Hence Tagging provides an alternative mechanism for coordinating the different representation of an object [8]. This approach does not require a complete database of objects nor does it require passing complicated symbolic expressions between components of a system. Moreover tagged behavior-based systems add to the traditional qualities (simplicity, parallelism and efficiency) of behavior-based system by providing additional flexibility and programmability. While the Bertrand-Ludwig architecture works well for queries it does not work so well for controlling actions. It is also dependant on the scene as the robot is seeing it a precise moment. If the scene changes, the robot does not notice it. *A forward-chaining inference* system is needed to continually re-compute computed inference. Tagging is only a partial solution to the problem of representing predicate-argument structure. However not only does it apply to the mechanism of the reasoning of several robots used nowadays but also improves it. The robot Kludge [8] was programmed using the feed-forward tagging scheme to follow simple natural language, such as "get the green ball". Kludge is equipped with a 25-MIP DSP board with an attached frame grabber and video camera. In addition Kludge uses an odometry system which tracks the location of the object it identified using its visual system in real-time. Kludge then uses this combination of mechanism to fire different behaviors, like following a blue ball for example.

The unifying theme of these systems is to import useful features of traditional symbolic AI systems into behavior-based systems without also importing the model-tracking and model-coherence problems [8]. The goal for Cerebus [8] is to integrate the advantages about the previous automated systems but also to be capable of limited reasoning. This semiautonomous state gives the robot the ability to do the tasks of the Kludge project and also to access its own internal state.

2.3.3 Full Autonomy

Fully autonomous systems describe systems operating without human intervention. Such robots can only be implemented with the use of sophisticated sensors. They are divided in two main categories: systems that have knowledge about their environment and systems that operate in completely unknown environments. The robots with a priori knowledge about their surroundings are still autonomous because they operated without input from the user. They have learned various behaviors in a virtual environment before autonomously repeating the same behaviors in real time.

2.3.3.1 Learning by Demonstration

Yeasin and Chaudhuri [23] propose an approach to program a robot by demonstrating a task several times in front of a binocular vision system. The motivation behind programming a robot by demonstration is simple and compelling: a user knowing how to perform a task should be sufficient to create a program to replicate the task [23]. This idea of integrating perceptual information with human skill can help in developing a flexible autonomous robot. Previous work, similar to works by Ude [24], has concentrated on tracking the object the human was working on instead of tracking the human hand. The key new idea is to help a robot observe a human performing a task, understand it and generate the corresponding program to perform the same task. A similar work has been done by Kuniyoshi [25] *et al* to track the motion of a human hand for program generation. The endeavor of this paper is to develop a fast, efficient and robust vision system, which is capable of extracting the sufficient statistics from the visual data, captured during the demonstration of the task [23]. The proposed system is composed of five major blocks:

- Data acquisition,
- Vision,
- Trajectory reconstruction,
- Task description, and
- Command generation module.

The data acquisition module captures the training data and the vision module extracts the sufficient statistics from it to generate automatic commands for the robot controller [23]. The next module reconstructs an optimal path from the vision information. The task description module subdivides the tasks in smaller task to facilitate the work of command generation module. Billard and Matari [26] evaluate a model of human imitation of abstract, two-arm movement. Input to this system are data from human arm movement recorded using a video and marker based tracking systems. The goal here is to provide the robot with *on- and/or off-line learning* or *adaptive* capabilities. Instead of adapting through reprogramming the robot would adapt through demonstration. This method is appealing, once again because demonstration is a natural and simple way for human-robot interaction. Instead of trying to guide the robot through a task, the operator who might not be familiar with the robot can just demonstrate how she/he would perform the task. This approach makes the robot's motion much smoother and flexible because it directly follows human instruction. Several works on the same topic have been very task specific whereas recently the focus is more on mechanisms of imitation in natural systems. In other words this work ultimately aims for the implementation of a humanoid robot. The endeavor is to, on the one hand, build biologically plausible models of animal imitative abilities, and, on the other hand, develop architectures for visuo-motor control and learning in robots which would show some of the flexibility of natural systems [26]. Billard and Matari [26] evaluate the model's performance at reproducing human arm movements. A simplified biologically inspired model of primate imitative ability is

developed. It has different levels of attention for repeating known movements or learning new movements. The recognition of the direction and orientation and the motion tracking mechanism are based on visual information. The recognition of the direction and orientation and the motion tracking mechanism are based on visual information. The motor control, which activates the muscles, is hierarchical and composed of artificial neural networks.

In a similar work, Aleotti, Caselli and Reggiani [27] argue that for tasks whose essential features are known a priori, demonstrating in a virtual environment may improve efficiency and reduce the trainer's fatigue. It presents experiments in simple virtual tactile fixtures in pick-and-place tasks. This approach assumes that trajectory will eventually be computed by path planning based on actual location of objects and status of the working environment. In the proposed robot teaching method, an operator, wearing a dataglove with a 3D tracker, demonstrates the tasks in a virtual environment [27]. The system translates actions into commands for the robot manipulator. Then the recognized task is performed in a simulated environment for validation before being executed in the real environment. This teaching by showing method includes three main phases:

- Task presentation - The user wearing the dataglove executes the intended task in a virtual environment.
- Task analysis - The system analyzes the task and extracts a sequence of high-level operations, taken from a set of rules defined in advance.
- Mapping - The synthesized task is mapped into basic operations and executed, first in a 3D simulated environment and then by the robotic platform.

The robot is controlled by the programming by demonstration (PbD) application in a six Degree Of Freedom (DOF) Puma 560 manipulator. A 2D vision system recognizes objects in the real workspace and detects their initial configuration. The whole application is built on top of a Common Object Request Broker Architecture (CORBA)-based framework, which interconnects clients and servers while providing transparent access to the various heterogeneous subsystems [28].

2.3.3.2 Autonomous Self Reconfiguring Robots

A parallel approach to robot autonomy it that of self-reconfiguring robots. Those robots do not necessarily move but rather change their shape autonomously. An example on this kind of robots would be the work of Rus and Vona on crystalline robots [29]. Crystalline robots consist of modules that can aggregate together to form distributed robot systems [29]. They are equipped with an actuation mechanism, which permits automated shape metamorphosis. Self-reconfiguring robots consist of a set of identical robotic modules that can autonomously and dynamically change their aggregate geometric structure to suit different locomotion, manipulation, and sensing tasks [29]. The goal for assuming different geometric shapes is to adapt to different environments. Self-reconfiguring robots are subdivided into two groups: heterogeneous and homogeneous. Heterogeneous

systems consist of different modules and homogeneous systems consist of identical modules. In this case there is no path planning instead researchers implement local and global self-reconfiguring planning. Rus and Vona introduce a new approach to homogeneous self-reconfiguring robots based on a module called *Crystalline Atom*. Inspired by muscles and amoebas, this module is actuated by expansion and contraction. By expanding and contracting the neighbors in a connected structure, an individual module can be moved in general ways relative to the entire structure [29]. In previous papers the same authors presented this crystalline module [30] and a robot system composed of 10 Crystalline Atoms [31]. Their latest work shows that Crystalline Atoms satisfy sufficient conditions for a self-reconfiguring robot system. Traditional work in this area involves cellular robotics like Fuduka's method to coordinate a set of specialized modules [32]. More work in this area includes a type of locomotion added to the metamorphosis. Murata et al proposed a system of modules that can achieve planar motion by walking over one another [33, 34] and moved to 3D motion [35]. The self-reconfiguring planning in most cases is somewhat similar or inspired from one another except in the case of Rus and Vona [29] who propose a new algorithm suitable for their new actuation capabilities.

2.3.3.3 Homing

Homing is a term borrowed from biology, where it is usually used to describe the ability of various living organisms such as insects, to return to their nest after having traveled a long distance along a certain path [36]. In robotics, this term is used for the ability of the robot to its initial (*home*) position after performing a certain task. The tendency in recent researches is to move to *visual or visual guided* homing instead of *sensory* homing for reasons that will be discussed in the second part of this section.

2.3.3.3.1 Non-Visual Homing

There have been various efforts of solving the problem of autonomous navigation in robotics. Robots are usually equipped with non-visual sensors, such as range sensors. The position of the robot is constantly recomputed with respect to an arbitrary absolute coordinate system.

A good example of distance sensing is the work of Bizzantino *et al.* They describe a work on a large laboratory testbed for space robotics, able to execute hierarchically organized complex activities, to increase the degree of autonomy of the system [37]. This approach uses a set of laser distance sensors to localize the true grasping positions. The resulting system is able to perform several tasks such as closing a drawer without human intervention and even when the robot is off the expected position by a few centimeters. Space robots are probably one of the few applications where a high degree of autonomy is not just desirable but rather mandatory. Indeed even if men wish to do the robot's work, space is not a natural environment for them and makes their performance of any task very difficult. Previous similar attempt (CAT) [38] to solve this problem, especially in grasping tasks, has been very dependant on the exact position of the object to be

grasped. Errors of just a few millimeters caused failure in grasping and collisions. The contribution of Bizzantino and al. has been to exploit the information of a set of laser distance sensors to allow autonomous motion of the arm even if for unexpected reasons the object to be grasped has been moved of several centimeters. Indeed an important part of space robotics is SPARCO (SPAce Robot Controller), which provides powerful motion control instructions. However, as mentioned before, this system does not handle anomalies (non nominal configurations). If the object to grasp accidentally moves a few millimeters from the expected position, collision occurs or an operator is needed to readjust the settings. The new idea is to add a new action called CHECK. The CHECK action completes all the initial actions in SPARCO by automatically and in real time learning the actual robot position. This is done through a special configuration and processing of distance sensors provided in this paper. In this new approach the manipulated subjects are out of the nominal place of several centimeters, against the few millimeters of the basic SPARCO release [37].

The previous example focused on grasping motion. Another area of interest in research is just the mobility platform concept, which can be used for other purposes such as carrying or dragging objects. In modular robotics that is a very important concept more specifically in the Imaging Robotics and Intelligent Systems Lab in the Electrical and computer Engineering Department at the University of Tennessee. Its sensory system is based on modular brick sensors that have their own sensor, processing, power and communication units. Hence the mobility platform becomes a mobility brick capable of standing on its own. The general idea is that the mobility brick should be able to pick up and carry a of the sensor brick and bring back scans to a passive operator. There are two components of intelligent mobility: mobility capabilities and mobility control [39]. Those two components of a mobile platform are self-descriptive; the first refers to its characteristics and the second to its intelligence. Moore and Flann add the mobility control is also subdivided in two components:

- First managing the “local dynamic” interactions between the vehicle and the forces it encounters, intelligent *vehicle-level control* algorithms must be developed and implemented to optimally maneuver the vehicle.
- Second, *mission mobility and path planning* is concerned with “global” motion planning and navigation and is used to determine the part a vehicle should take to pass through a given rejoin to achieve its objective.

For easy motion mobility platform usually incorporate omnidirectional wheel, hence the name Omnidirectional Vehicle (ODV). The Utah State University (USU) has developed a series of T ODV including the ARC III, a 45-lb small-scale robot [40] and the T2 [41], a 1480-lb robot. ODV are an innovation in mobile robotics as traditional concept use vehicle steering mechanisms [42, 43]. This kind of vehicle uses skid steering and is constrained by the direction of travel. The T series are not really omnidirectional as it takes a finite time to turn a wheel to a new angle. However this type of vehicle is

equipped with smart wheel with three degrees of freedom: height, steering and drive. This combined with very fast turn rate motor result in a motion that is effectively omnidirectional. Other researchers have developed ODV concepts that include fewer (3-4) wheels [40, 44, 45, and 46]. The T series uses six wheels to get more power and tired surface on the ground. Some researchers have also carefully built and studied the particular behavior of different robots [47, 48 and 49]. The T series differs from these previous works in the design of the smart wheel, with the slip ring that allows infinite rotation of each wheel in either direction [39]. Hence Moore and Flann contribution is the development of a task-based trajectory planning strategy combined with a first-principles-derived, model-based controller developed specifically to exploit the mobility capabilities of their specific robots. This strategy is characterized by a hierarchical task decomposition approach. At a high level, a knowledge-based planner and a special algorithm generate the vehicle path as a sequence of basic maneuvers. At the vehicle level those maneuvers are converted in time-domain trajectories. At the lowest level linear controllers drive the wheel's low-level drive motor and steer the angle motor controllers.

Sun and al. present a modeling and analysis method for the motion planning and control of mobile robot systems in a hybrid fashion. Robotic systems obtain environmental information from perceptive sensors and respond to the perceptions to execute the task through decision and control process [50]. The *Hybrid* Automata perceptive model and reference have both discrete and continuous components. The discrete layers enable the robot system to plan and modify original path through switching. However in the case of an obstacles for example, the reference is blocked and the robot only resumes its operation after the obstacles are moved. Previous works on hybrid systems and hybrid automata [51, 52, and 53] have lead to Brocket's proposal of a motion description language for kinetic state machines [54]. In this model machines are the continuous analog of finite automata. However all the previous approaches use time as a reference for both the discrete and continuous parts of the system. Sun proposes a new general approach to sensory homing: instead of time, the control input is the new reference which is a function of real-time sensory measurements as it is crucial for the reference to keep evolving and not being blocked by unexpected obstacles. The hybrid perceptive references enable both the continuous part and the discrete part of the system to deal with unexpected events [50]. The discrete part keeps the perceptive reference evolving through modifications of the original information. Both, the discrete and the continuous part are integrated by the hybrid perceptive motion reference. This system has been implemented using a Mobile Manipulator-Phantom Joystick teleoperation system consisting of a Nomadic XR4000 mobile robot, a puma560 robot arm mounted on a mobile robot and a phantom joystick controller. There are two PCs on board of the mobile robot (platform and arm) and one for the joystick controller. The robot and the controller communicate through the Internet.

2.3.3.3.2 Visual Homing

Several approaches to robots autonomy have been inspired by insect physiology. Most of these works were inspired by the pioneer work of Cartwight and Collet [55] on insect behavior. According to them insects base their navigation on their memory of certain landmarks locations. More work (Srinivasan et al. and Lambrinos et al.) [56, 57] later reinforces the idea that insect navigation is based on specific landmarks and uses it for robots navigation. These works use the fact that some insects would actually make detours in their way to a target position just to pass by familiar locations. Works by Lambrinos, Franz, Moller and Cassinis et al. [58, 59, 60, and 61] on autonomous robot navigation exploits interesting findings about insects' behavior. First insects use vision to travel and they use very large fields of view. Second, they use certain parts of this stored visual data to localize themselves. Finally and very interestingly 3D structure is not a prerequisite for their navigation. Most of these works especially exploit the benefits of the wide field of view. Argyros, Bekris and Orphanoudakis [36] take those facts into consideration and propose a new approach to homing that exploits omni-directional vision. Their method is based on a panoramic camera. It uses the fact that there are at least three correspondence features between two panoramic views taken from two different positions. Then, a control mechanism helps the robot move from one frame to another. More explicitly when homing is activated the robot starts tracking specific features of its environment (corners). The robot selects intermediate target positions (IPs) on its original path. These IPs are visited sequentially as the robot travels toward its final destination. A tracking mechanism allows the robot to compare images along its path. It only needs to match 3 corners in order to move from one position to the next. This homing scheme is based on the extraction of low-level sensory information, namely the bearing angles of corners. Argyros et al. [36] implemented this method on a robotic platform and the *home* position is reached with an accuracy of a few millimeters after a journey of several meters. This approach could be qualified as homing as a solution for global navigation. Homing has also been proposed as a solution for local navigation problems [62]. Koku et al. [62] point out that several homing approaches methods usually use egocentric navigation as a part of their homing algorithms [63] and propose a homing method based solely on visual data. The proposed method is called Egocentric Navigation or ENav. Other methods used egocentric navigation in addition to some absolute information including the work of Gaussier, Pinette, Moller and Dai [64, 65, 66, and 67]. ENav is based only on egocentric navigation. The robot has a perception of a target position and a similar perception of the position where it currently is. It then scans the view for object it recognizes and creates egocentric representation for them. The comparison between the two helps direct it towards the right destination. This information is referred to as the Sensory Egosphere (SES). A similar representation called the Landmark Egosphere (LES) describes the target position. The robot is given the LES and frequently creates the SES. This mechanism is usually done through egocentric representation where landmarks are indexed based on their angular separation with each other. A heading vector **h** computed by this method points the robot toward the target point. The robot generates and updates the **h** vector and starts moving towards the

target point until the SES and the LES are close enough to each other. The results obtained from this method are beyond satisfactory as the error in reaching the destination does not exceed the size of the robot. The particularity of this method is that is eliminates the need for absolute landmarks and is based only on perception. Since it does not involve mapping, the heading vector **h** does not have to point toward the final direction, it just has to get a robot a little closer to the target point.

2.4 Autonomous Methods Evaluation

The research on autonomous robots is broad. As those research areas apply to various different topics, they do not always compare to each other. Nor do they always complete each other but rather they apply to different circumstances. This section will try to categorize the methods to implement autonomy presented in section 3. It will point out the ways and the circumstances where a certain method is preferable over another. In other words it will present the applications and the limitations of those methods. Then it will attempt to explain the trade off between teleoperation and autonomy. Finally this analysis will propose an alternative to choosing, which is the purpose of this work.

2.4.1 Task Oriented Autonomy

Methods like Tagging and Programming by demonstration are mostly task oriented. Some behavior-based methods may have commands such as "Go to the blue block", however the focus resides in recognizing and reaching the block rather than path planning and obstacle avoidance. It is common to see behavior-based systems as incompatible with autonomous operation. Hence research in this area is quite limited. Yet Horswill [8] points out that it is not only compatible but also *implementable*. His project shows that it is possible to program a robot to respond to users' simple voice commands and perform different tasks. The main reason why this method is neglected is the difficulty in programming behavior-based systems. While reasoning systems use very advanced languages, compilers and development environments, behavior-based system are still written in simple languages such as C++. Programmers have to write codes in both languages and then couple them, which is painful and error-prone, hence not practical. Some equivalence between the two languages is needed to make the programming and debugging easier and more efficient. Assuming such protocol is created, behavior-based systems would still need to be couple with online-learning methods to be very useful. Indeed they are task-oriented and so far they mostly assume a database of behaviors already stored in the robot's memory. This is when PbD (Programming by Demonstration) becomes handy. This method could be used to fill up the database in the tagging systems. Programming by demonstration goes beyond filling databases in robotic systems. It does so simply by demonstrating a task to the robot. Any user can later demonstrate rather complicated task for the robot to reproduce. Of course PbD is still being studied and still needs improvements. The initial work on PbD often required special hardware - not available for everyone - such as the *dataglove* with 3D tracker used in the University of Parma, Italy [12]. While researchers are going towards visual

tracking, other problems with this method involve singularity of trajectory and the complexity of the underlying recognition and interpretation of movements. Another problem with this method is that often the task has to be demonstrated several times before the robot can execute it, which is not always possible in real time.

Both Behavior-based and PbD robots aim at task-oriented applications. Both methods want to alleviate the responsibility of the user. In real-time end users with little or no specific expertise might be required to program robot tasks. In fact most of the research in humanoid robot aims at making robots a part of everyday human lives. Whether it is to simply "fire" a behavior by voice commands for example or program a robot by simply demonstrating a task, the goal is to produce a robot that is easy to use. The problem with behavior-based methods as mentioned before is that the programming is very tedious and does not yet incorporate online learning. So the user only has fixed preprogrammed and specific tasks to execute. On the other hand, programming by demonstration allows more flexible tasks as the operator simply demonstrates the chores to the robot. This method does not only simplify the operator's job but also the programmer's. However in this case, sophisticated hardware is generally required and as this method is still improving, it generally takes several demonstrations to execute a simple task.

2.4.2 Mobile Autonomous Robots

The quest for a fully autonomous system has been one of the ultimate scientific goal in the robotics community. However, most of the time, due to the limitation of the current recognition and decision making technology the need for human intervention or supervision is still essential in the unstructured real world. Teleautonomy and homing are both going around this problem in their own distinct ways.

Teleautonomy attributes the robot with partial autonomy to ease the burden on the operator. Although the operator may need expertise to manipulate the robot, the goal of teleoperation is to make the operation as intuitive as possible. This method aims at allowing the operator to concentrate on the core of the task being performed while allowing the robot to autonomously maintain its balance and existence. By integrating the operator's intuition and the robot's autonomy, telerobotics covers a very large range or application in real-time and in changing unstructured environments. Homing also avoids the task-oriented aspect of a number of autonomous systems by focusing on navigation problems. In this case autonomous systems process perceptions (the environmental information obtained from onboard sensors) and respond to the perceptions by changing the original path planning and control schemes. Homing is subdivided in two categories, sensory and qualitative homing. Sensory homing is based on the use of lasers or range sensors to acquire 3D information and encoding information necessary to reconstruct or map the environment. This method is then very concerned with sensor accuracy and structure of the environment to ensure a collision free path for the robot. Errors and most importantly cumulative errors in the sensing process can cause damages to the robot and make this method quite unreliable over time. Indeed sensors can give different readings

depending on location for example and the robot can easily get confused. Moreover in this case, localization is a prerequisite of navigation since the use of a map suggests that the robot should know where objects are with respect to itself. Certain robots do not cover distances long enough to require maps. All those issues lead to more research as well as to another homing method often referred to as visual homing. Indeed visual homing eliminates the need for reliable sensors and uses vision which is much more informative. This method provides the information "where is what" [13] which can directly be used by the robot. This approach is then range-free. It is also map-free and does not perform localization algorithms. The comparison of adjacent frames just provides the robot with a vector to follow towards a target position.

Even though telerobotics and homing do not directly compare, they approach the autonomous systems in similar ways. Neither of those two methods is focusing on fully autonomous tasks performance. Telerobotics focuses on providing the robot with a degree of autonomy while being operated. Homing focuses on providing the robot with autonomy before or after performing a task, so that it can safely return or reach a home position. It is easy to imagine how both methods could actually complete each other. Assuming the robot autonomously knows where to go before and after a specific task, it could safely reach those locations and wait for instructions. Then the operator could perform that task without worrying about balancing or stabilizing the robot no matter how complex the task is. Both methods have their advantages and drawbacks. Both methods have a common advantage. They perform very well in real-time. On the one hand, while telerobotics eases the burden on the operator but still requires his intervention and supervision, homing does not. On the other hand while teleoperated systems can perform tedious, complicated and sometimes dangerous tasks, homing focuses solely on navigation.

2.4.3 Tradeoff between Teleoperation and Autonomy

Robotic systems range from fully teleoperated to fully autonomous. While a number of scientists try to solve specific problems along this wide spectrum of systems, others analyze the trade off in going from one to another. No system is better than another one in terms of human dependency; they simply apply to different circumstances and various degrees of complexity. While research is going forward in autonomous systems, studies on teleoperated robots are more mature. The intuitive method to deal with avoiding obstacles, dealing with communications delays, reacting to unexpected event and other issues is to have human supervision. Teleautonomy is the next step in supervisory control. While the operator only focuses on a specific task, the robot has a few basic self-maintaining functions. One step further, fully autonomous systems do not require human input. However along this progress the operational level of complexity has decreased. Jacob W. Crandall and Michael A. Goodrich continue working on adjustable autonomy [1] and argue [2] that in certain circumstances such as search and rescue or hazardous waste removal, human operators are desirable. Unfortunately there is a limit to the number of robot one man can efficiently handle. To analyze this issue, they present a

theoretical framework for understanding how the expected performance of a particular interaction scheme changes as the robots are neglected and as world complexity increases [2]. Their way of doing do is to create a framework for characterizing the efficiency of human robot interaction. In other word, they prove that the human robot performance degrades as the operator neglects the robots, as the information and control scheme deteriorate and as the complexity of the world increases. Their ultimate goal is to use this study to better design control schemes to deal with those issues. Myra Wilson and Mark Neal also examine the tradeoff between the increasing design and implementation effort necessary as the system moves through the continuum from teleoperated to autonomous and the amount of human intervention required [3]. Telerobotics describes system where despite remote human supervision the robot has a degree of local autonomy. This case of study uses a human shepherd herding a robotic sheep using robotic dogs. The behavior of the sheepdogs varies from teleoperated to highly autonomous. This study reinforces a few ideas. It is difficult for the operator to predict at distance the best behavior to initiate locally (at the robot end) until the system is well understood. The appropriate behavior can drastically ease the burden on the operator. However, engineering effort and complexity in the system can easily have a negative impact on the systems requiring more effort for implementation and more effort from the operator. This study confirms the intuition that, complete automation is a time consuming and complex engineering exercise [3]. Scientist generally agree that the reduction in operator intervention will only decrease at the cost of an increasing engineering effort to fully understand the systems itself and the real environment in which the robot interacts.

2.4.4 New Architecture for Teleoperated and Autonomous Unmanned Systems: JAUS

In recent attempts to improve the research and applications of autonomous robots, the Office of the Secretary of Defense Joint Robotic Program has developed a common, domain level architecture into consumer, military and industrial unmanned systems. More and more academic, military and commercial systems are adopting this Joint Architecture for Unmanned Systems (JAUS). The Spartan Advanced Technology Concept Demonstration SPAWAR Systems Center, San Diego developed a series of new JAUS messages, including radar data transport and dynamic (on-route) re-configuration of the waypoint route [68]. The Department of Mechanical and Aerospace Engineering of the University of Florida at Gainesville participated in the DARPA Grand Challenge that was held in March 2004. The system architecture of their system is also based on JAUS [69]. As robotics is heading towards modularity, more and more products are aiming to be JAUS compliant. Indeed JAUS promotes technology reuse and insertion. It defines a set of reusable “components” and their interfaces [70]. Modularity is desirable for two main reasons. First, if components are easily portable and attachable to a certain system, whenever a component breaks down it is easily replaceable. It does not impact the whole system and when it is fixed it is easily reusable by the same or another system. Second, as technology advances, new parts can easily replace old ones or be inserted in addition to previous components, as the architecture in place already supports more advanced

capabilities [70]. Hence an implied advantage of this new architecture is simple interoperability between different systems provided they all are JAUS compliant. In other words provided they exchange the corresponding messages. To ensure that a system can become JAUS compliant, the architecture cannot be dependant on a particular software or hardware technologies. It is purely and simply based on messages exchanges between components. As research on teleoperated and especially autonomous systems presents numerous challenges, JAUS is a practical way to try and eliminate problems, such as system dependencies.

3 Migration to Autonomy

The broad spectrum of robotic system expands from teleoperation to fully autonomous robots. The ultimate goal of this work is to avoid parts of the restriction involved with belonging to either end of this range by producing a three state machine able to be either fully teleoperated or fully autonomous. The intermediate state being a computer interfaced teleoperation. This state is slightly more flexible than the proprietary teleoperation in the sense that it can be combined with modular sensor brick interfaces. The self-ruling state goes one step further by using those sensor bricks to acquire a degree of autonomy.

3.1 Original System

The transition starts with the original robust commercial system. The Original systems targeted by this work are robust systems used mainly during delicate, potentially perilous circumstances such as explosive or hazardous waste disposal. Commercial and military teleoperated robots are being produced to keep men out of harm's way during such dangerous missions. Teleoperation implies that the system absolutely require human input to function. A technician remotely maneuvers the robot using an Operator Control Unit (OCU). This unit is usually easily portable or wearable. It can communicate via cable or wirelessly. Figure 3 shows the wearable control panel for the ANDROS Mark VA. In this case the control panel consists mainly of mechanical toggle switches and potentiometers. The OCU takes the user input and transmits the data to the robot over a serial RS-232 line. The switches allow for simultaneous control of multiple functions – The toggle switch signals are independent until they are multiplexed and sent to the robot in a serial data stream [71]. The software is simple, robust and processes data quickly. This control system is very efficient. The emphasis is not on being fancy and sophisticated but rather on building a reliable elementary system that works. This is the general motivation for this type of robots control units. The mobile platform is similarly very robustly built. Figure 4 shows the Remotec ANDROS Mark VA. Hence on the one hand, the system can be trusted to survive difficult situation. However on the other hand the operation of this robot should not be taken lightly either. Before driving the robot, one usually has to go through a special training. An untrained operator cannot operate this heavy piece of machinery. For example when the communication is not wireless, the operator has to constantly avoid running into the fiber optic cable. The cable is very resistant yet getting caught into the robot's track can definitely damage it.

Figure 3: This figure show the original mechanical control panel for Remotec ANDROS Mark VA.

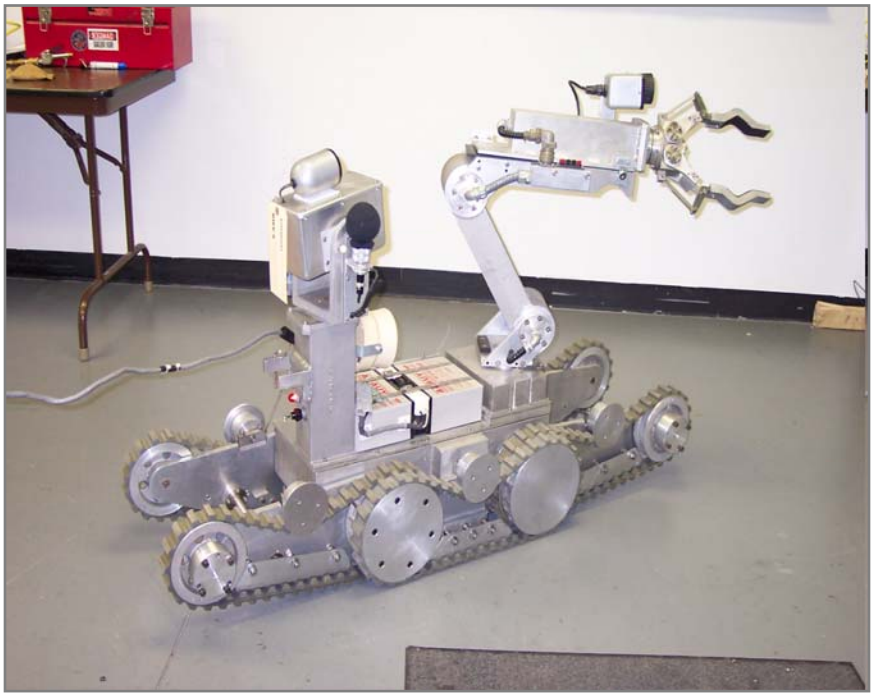


Figure 4: This figure is a photograph of the Remotec ANDROS Mark VA.

Also while driving the robot while it is in line of sight is quite possible for a first time user, this is not the case when the robot is in another room or simply when the operator's view of the robot is obstructed. In general an experienced operator remains at a safe standoff distance either because the scene is hazardous or because it is simply inaccessible (caves, tunnels and collapse structures). Meanwhile, the robot relays real-time video, audio and occasionally other sensors' readings. The advantages of such systems are clearly apparent as they allow the operator to identify threats (mines, bombs, enemy…) and to prepare before being exposed to it. Examples of such robots are the ANDROS from Remotec, TALON produced by Foster Miller and the PackBot series manufactured by iRobot. Naturally those robots are particularly well-built systems. They can be operated in rough terrains and withstand minor explosions (See Figure 5). The ANDROS can climb up to 37 degree-stairs while the PackBot scout is designed to survive a 2 meter drop onto concrete [72]. Such features only begin to illustrate the survivability of typical reconnaissance, surveillance and Explosive Ordnance Disposal (EOD) robots. Standard accessories include driving, manipulating and surveillance cameras. Global Position System (GPS) is also occasionally available. Those types of sensors are basic for these robots and do not restrict their domain of application. That is why, to avoid narrowing their clientele, manufacturers usually do not attach other more specific operational requirements. Instead, they provide the user with additional power outlet and Accessory Interface Mount (one or two). The interface for those extra sensors is not embedded in the original system. The functionality of supplementary accessory has to be simple. The user could add weapons as an accessory to the ANDROS for example; the corresponding switch is simply "Fire Weapon". More complicated accessory and interfaces have to be specially ordered and integrated into the basic design. Teleoperated systems rarely include degrees of autonomy. If they incorporate autonomous function at all, it is usually a self-status checking, such as checking the charge of the battery or the status of the wireless communication or other similar self-maintenance functions.

To summarize the characteristics of typical inspection teleoperated systems, their control unit and the actual robots are very robust. They absolutely require the supervision and intervention of an operator to function. They do not normally include various sophisticated sensors neither do they usually provide involved autonomous functions. They are mainly designed for real time hazardous missions providing the operator with accurate immediate information about suspicious scenes and objects. Since the robot has limited or no local intelligence a clumsy operator can lead to disasters. In addition, even though there are different types of such robots being produced in the industry, there is no interoperability between systems manufactured in different companies. Each system is constructed using its own proprietary control system that is not directly translatable to other system; payloads and manipulation appendages are generally not interchangeable [5]. These hardware-based systems are also each difficult to upgrade. When there is a situation, only technicians with complete knowledge of the robots' engineering can debug the problem. This process is time and labor consuming. Moreover as new models are created previous versions become obsolete.

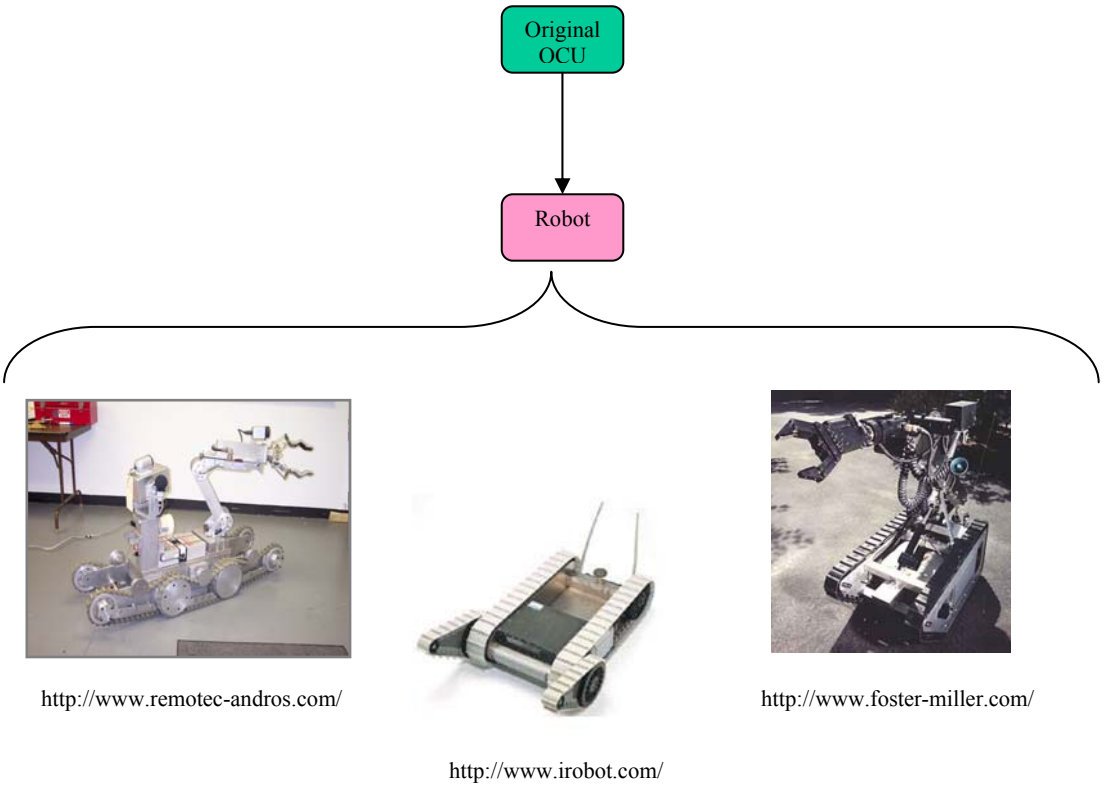


Figure 5: This diagram shows examples of teleoperated robots and their basic structure.

3.2 Reverse Engineering

One of the goals of this project is to bring a commercial robust system from being hardware-based to software-based without major changes to the original robot. The OCU has to be easily reconnected to the robot when desirable. In this specific example, the ANDROS Mark VA control unit is still entirely functional. No alterations were made to the robot or to its OCU. Instead the original control panel was analyzed and decoded in details in order to later mimic the control system from a regular PC.

The original OCU for a commercial robot is like a black box for people who are unfamiliar to the manufacturers' design. Moreover assuming an engineer has special knowledge about a specific design (schematic, circuit design, source code…), this knowledge is not directly applicable to another robot. Therefore the best way to analyze the native control panel is to determine the output signals by observation. The work described in the Implementation section primarily applies to the Remotec ANDROS Mark VA. However similar process should apply to a comparable robot. Depending on the particular system used the original OCU can more or less easy to be analyzed, mapped and decrypted.

The process of capturing and retransmitting analog signals to a robot can be relatively involved and may vary with different systems. There may be impedance matching effects between OCU and robot. Impedance matching is the special connection between an additional impedance to an existing one in order to accomplish a specific effect, usually maximize a performance. It is recognized that to maximize the output power from a source to a load the impedance of the load has to be the conjugate of that of the source. If there is a similar specific association in the original system, replacing the OCU by a computer may not be as trivial. Different devices are not always compatible with each other in terms of voltages, frequency, noise tolerance etc. Even details such as the length of a cable can make a difference in the accuracy of signals transmission. However there are ways to solve this problem. There are standard electronic protocols that enable different systems to "talk" to each other. The first important step in identifying the right transition is to methodically and systematically characterize the native controller. Visualizing and measuring the voltage and current levels helps settings targets for what the new system needs to reproduce or in other words what the robot is expecting from the new control box. Once the conversion is known extra hardware or software interfaces can help build the gap between the proprietary OCU and the new control computer. Those changes may happen at the peripheries of the original system or in the programming they are not considered to be major changes in the original system. The engineering inside the control panel and the robot are to remain unchanged and the robot must remain operable by its original OCU.

Certain teleoperated systems are already using regular processors for controls, in which case sending commands from a computer interface is clearly achievable. It is still

desirable and necessary to reverse engineer the control panel because even in those cases, the control units and the mobile platforms are not interchangeable between different systems. This step is en essential step in order to create an independent unit capable of connecting with different additional sensor and mobility bricks.

3.3 Computer Interface Teleoperation

Reverse engineering the commercial system is important in order implement a computer-interfaced teleoperation. This step allows establishing a connection between the original system and sensors that were not initially built in without major hardware changes. Through certain of those modular sensors, the robot can acquire autonomy. Although that is the main goal of this work, the computer interface along with the sensor and mobility bricks significantly expands the functionality of the original system. Indeed while keeping the robustness of the industrial robot it provides the operator with additional information about the scene to be inspected. The user drives the robot into desired places, gets information from the sensors and reacts consequently. Each system is quasi-independent, meaning each brick can be operated on its own or join efforts with other systems to complete a particular mission or inspection. The Imaging Robotics and Intelligent Systems laboratory has formulated the Brick Concept. Hence this section will begin by presenting the sensor brick notion followed by the mobility brick idea and finally the main OCU.

3.3.1 Sensor Brick

Modularity is an important piece of modern robotics. The brick concept follows this new trend in its definition and its composition. It is an independent sensor unit. Considering an under-vehicle robot for example, different bricks can easily be placed on, removed from or combined together on the mobile platform at the operator's will. If one brick fails, the others are not affected. On the one hand the performance of one brick does not depend on that of another one; on the other hand components of different bricks are easily replaceable and interchangeable in the case of failure of a component. Indeed the brick design includes four elementary blocks:

- Power,
- Acquisition (Sensor),
- Intelligent Systems and Computing (Pre-Processing), and
- Communication.

The sensing process starts with the Sensor block, which acquires the data. The pre-processing block then processes this information. This step may involve low-level processing of the acquired raw data such as noise removal or image sharpening for visual data for example. The Communication block then relays the information to the operator and the Power block fuels the whole system. Figure 6 shows an iconic representation of the brick. Three bricks have been in their entirety in the IRIS lab, a thermal brick, a visual brick and a range (laser) brick. (See Figure 7)

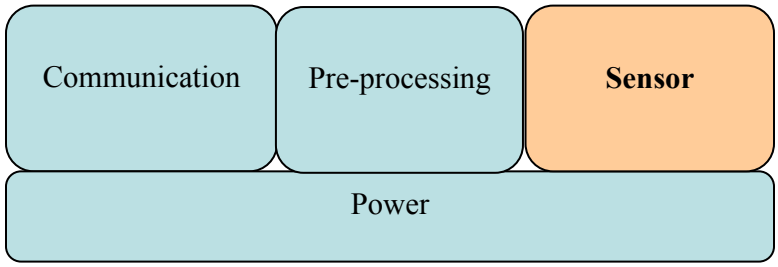


Figure 6: This iconic representation of the sensor brick design shows the different blocks that compose it.

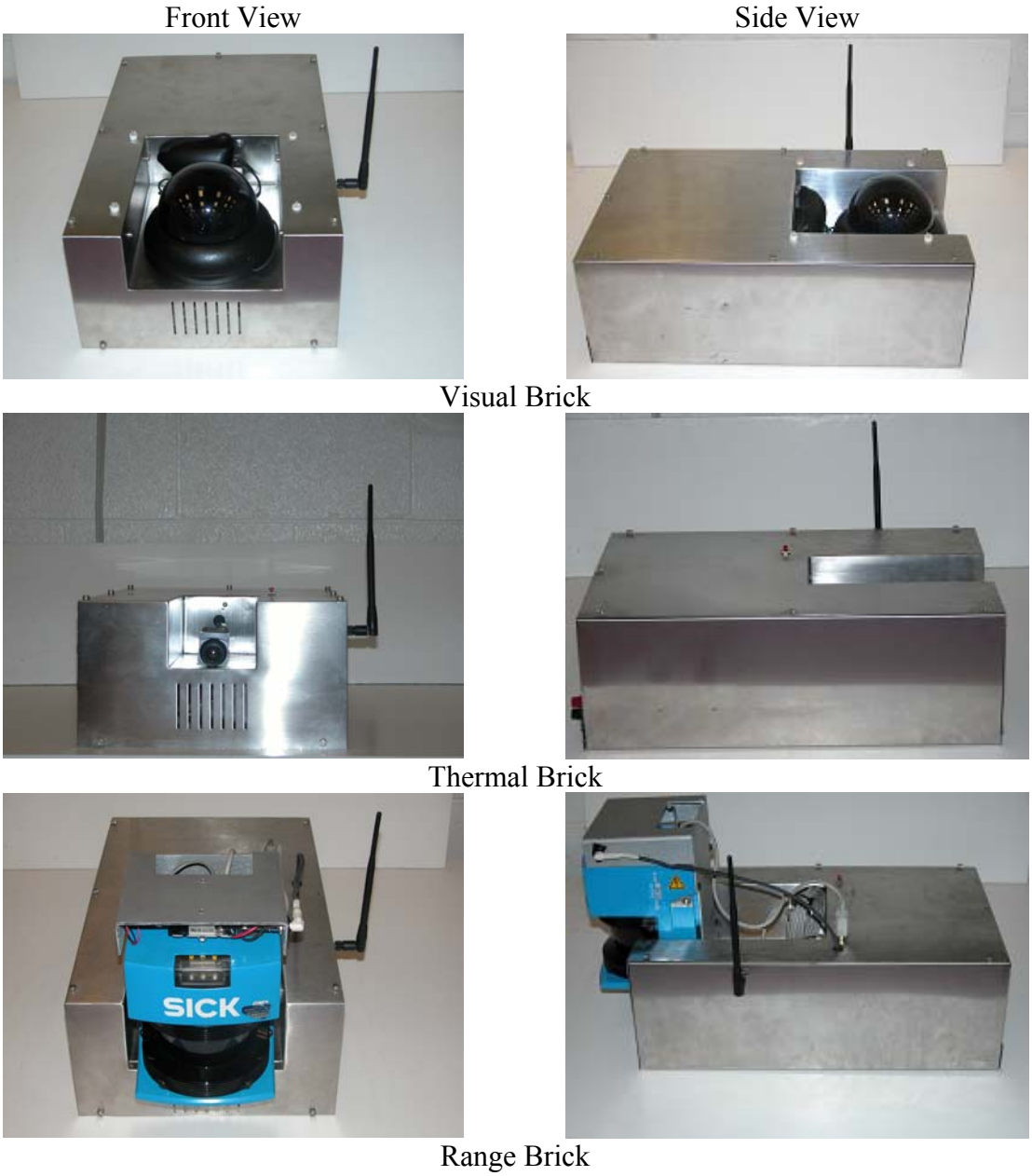


Figure 7: These pictures show sensor bricks implemented at the IRIS laboratory [73].

3.3.2 Mobility Brick

The mobility brick is defined similarly to the sensor brick. The acquisition block is simply replaced by a mobility block. The brick can be seen as a mobile platform. It is a robot which primary function is teleoperated or autonomous navigation. Several of those robots already have on-board sensors and encoders. The emphasis in the mobility brick concept is that the robot's main purpose is to drive. Other subsystems such as sensor and manipulator may later be added to it to increase its capabilities. It can be compared to the JAUS primitive driver. Its functionality only includes basic driving and related mobility functions. An example for the Remotec ANDROS would be lifting and lowering the tracks on the robot. The Mobility brick can be independently operated using the communication and pre-processing blocks to access it. The user has two options. He or she can either directly send commands to the robot from a computer; or simply remote login to the brick and drive it from the on-board computer. Figure 8 shows the structure of the mobility brick based on the sensor brick design.

3.3.3 Main Control Unit

The operators control the sensor and mobility bricks from a central computer. The sensor bricks transmit pre-processed information to the remote located central control computer. The human operators monitor the inspection process, compare the images or data from the various sensor modalities, and gather information to make future decisions with respect to the system's state (what sensors to use next, what should be the next sensor, the order of use of the sensors, when to stop the process, etc) [5]. This is done either by logging in remotely to the local computer on the brick or by communicating with the brick using the appropriate Graphical User Interface (GUI). At the moment, each mobility brick and sensor brick provide the user with a different GUI. As mobility is being integrated in the brick concept, the system is moving towards incorporating several sensor and robot information into the same interface. Having a unique GUI is a convenience not a requirement. Indeed the fact that each brick provides the user with a control system allows it to be used by different platforms or vice versa. There can be as many monitoring windows as there are bricks used during on inspection including the mobility brick.

The control unit can be compared to the native control panel of the commercial robot as it relays real-time information to the operators. The technicians then analyze the data and act subsequently. The main advantage here is the easy access to more sensors than in the original system. Moreover in the case of a hazardous scene inspection for example, instead of sending a robot and waiting for its return, the operators can directly determine the nature of the threat and react consequently.

The ideal situation for the control unit would be to broadcast messages to all bricks using a special message header. Each brick should have the ability to decode those messages

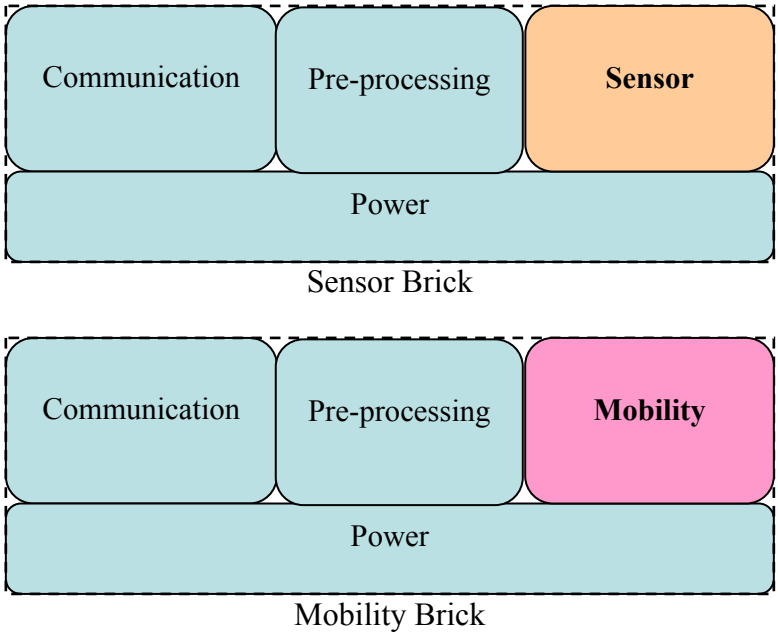


Figure 8: This image describes the mobility brick following the sensor brick design.

and decides its source and destination as well as its desired effect when applicable. The sensor brick are passive sensors. Their sole purpose is to acquire data and make it available for the user. The robot may carry the sensor during an entire mission, or place them at specific places and leave them to be picked up later on. The sensors may already have been placed overhead or in the corner of the room. Again, the modularity and the wireless connection between subsystems allow very flexible applications and scenarios. Meanwhile the central computer is at the highest level of the control structure, overseeing and organizing the overall operation. As the number of bricks increases and the level of autonomy increases, the control scheme gets more involved. In this initial state however, the user logs into the passive bricks for display of the sensor data. Similarly the operator can log into the mobility brick or control it from the main control unit. The greater picture is to have an *army* or sensor and mobility bricks reporting to a central OCU. Figure 9 illustrates this idea.

3.4 Computer Integration

Industries focusing on safety and hazardous tasks do not produce robots with advanced autonomous functions and sophisticated built-in sensors. As previously discussed, there are still serious challenges in building a fully autonomous, efficient and practical robot. Moreover a majority of tasks in this particular domain are potentially harmful for human being. Hence, it would be too risky to entirely give the decision making to the unmanned system. The operator is still required at least to supervise the robot's actions. Those are two main reasons why commercial safety robots are generally not autonomous. Another reason why they do not incorporate more sensors is that it narrows down their clientele. All the clients can use a teleoperated robot and manipulate it to meet certain requirements and needs. Not all users however need a chemical sensor on an autonomous robot or a nuclear sensor for example. Since it is not nor cost-efficient neither time efficient to produce enough robots for different areas of applications, industries produce basic and robust teleoperated robot. In acquiring autonomy, the goal is to preserve the robustness and durability of those systems by transforming them into mobility platforms. In other words they become mobility bricks that can easily be combined with sensor bricks. The wireless communication between bricks allows the transition from teleoperation to autonomy to be simple and without major hardware changes to the original systems. Indeed the brick can be stacked up on or carried by the robot or not even on the robot itself. While the Ethernet connection allows the data to flow between the different subsystems, the computer integration insures that each brick is able to analyze and interpret the data and react accordingly.

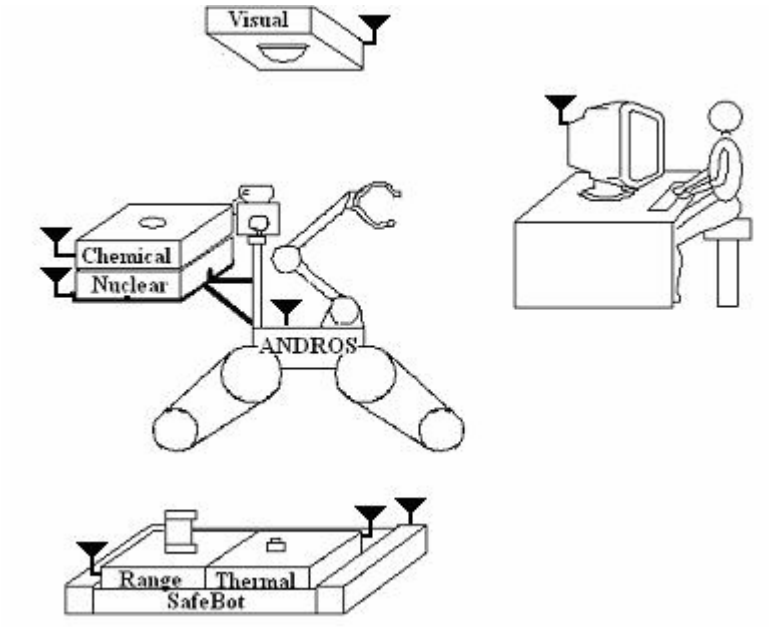


Figure 9: This figure shows a central computer, robots and bricks communicating wirelessly.

The technician can either use the teleoperation or when the complexity of the task decreases he or she can delegate it to the robot. The mobility brick then takes the input from the operator, decides which brick to use and what to do next. There are several ways to achieve autonomy, unmanned system typically use encoders, lasers or visual sensors to "feel" their environment. By using sensor bricks, the options are not limited, different algorithms can be used on the same platform to produce different autonomous functions. Visual homing can be performed using the visual brick; distance-based autonomy can be achieved using the range sensor. A scouting mission requiring that the robot reaches a target takes a visual scan and returns to its original position while avoiding obstacles along its path would require at least the range and the visual bricks. This scenario can be seen on Figure 10. While several methods to achieve autonomy can be implemented on the robot using the appropriate sensor bricks, the transformed robot can be use for autonomous data collection. For example the task may be as simple as going around a room recording data and transmitting it back to the operating station in real time. The unmanned system simply follows its path carrying a nuclear, chemical or biological sensor. During autonomous navigation or data collection, the operator has the authority to stop the process at any time. This is a safety measure; the operator has to be able to interrupt the robot in the case that an anomaly occurs.

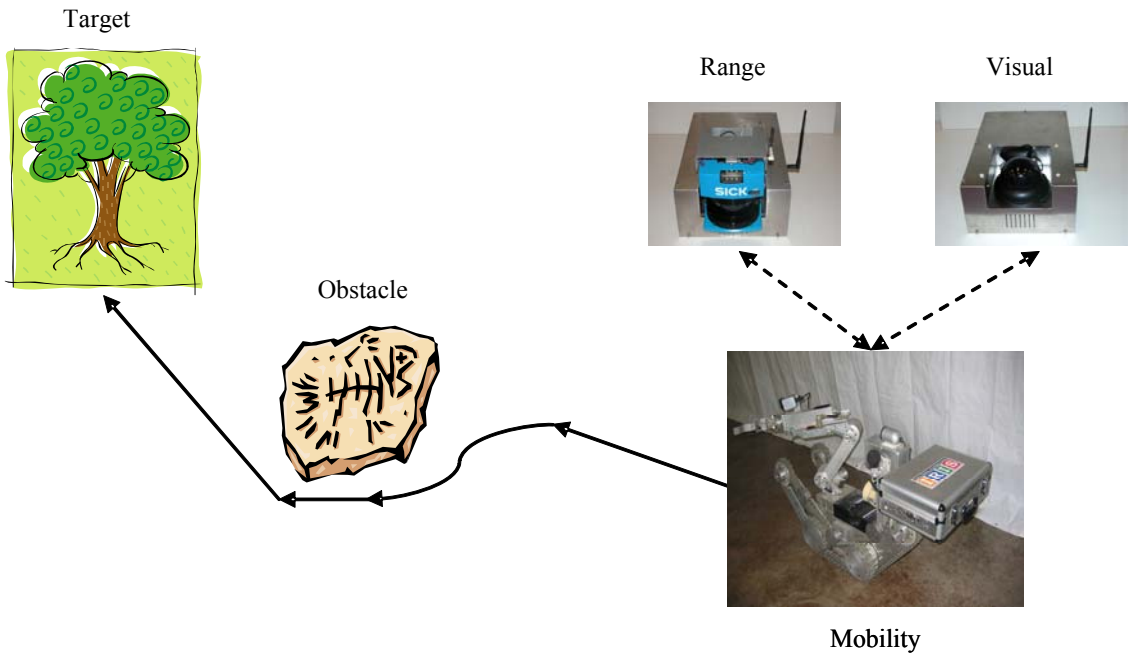


Figure 10: A robot can be sent on a scouting mission equipped with one or more sensor bricks.

4 Implementation

While Chapter 3 paints a general picture of this work, Chapter 4 will go into details in the implementation of the proposed system. It will start by preliminary work on the ANDROS Mark VA and describe the reverse engineering process as well as the migration toward mobility brick and autonomous system.

4.1 Preliminary Work on the ANDROS Mark VA

In the fall of 2003 a senior design group in Electrical and Computer Engineering at the University of Tennessee started reverse engineering this particular system. This Basic process includes two main steps: Capturing the signals from the Native controller and storing those strings inside a C++ program.

4.1.1 Previous Work on the ANDROS

The fall 2003 team aimed at replacing the original OCU of the ANDROS by a software-based control system. Without specific knowledge about the engineering of the robot, the group had to mimic the control signals from a touch-screen. The first step was to intercept the signals coming out of the control box. The OCU transmits RS-232 signals to the robot. To capture those signals the team opted for a program named KERMIT 95. Kermit 95 (K95) is an extensible file transfer protocol first developed at Columbia University in New York City in 1981 for transferring text and binary files without errors between diverse types of computers [71]. Among the different types of connections that this program can establish one of particular interest for this project were the serial port connections. This feature allows establishing an RS-232 line between the PC and the OCU. The control panel is powered separately from the robot, and connected to a PC where a Kermit terminal is running. The user indicates the COM port and the baud rate and the program automatically reads the serial data. The KERMIT software allows the user to save the output in a text file. Hence the user can record and later analyze different signals corresponding to as many toggle switches. Different control character strings represented the different commands. The programmer then stores the strings into a Paradigm C++ Lite program, which the TERN LCD touch-screen provides as a software package. This software is simply a C++ package with various additional predefined functions to facilitate the use of the touch-screen. The data was transmitted from the touch screen to the ANDROS using MAXSTREAM wireless Stand Alone Radio Modems. The units were formatted to transmit RS-232 data at a 1200-baud rate.

Though parts of the logic behind the strings were understood, at the time the program could not generate the strings. Instead the commands were stored unchanged from the OCU. This was a considerable limitation since they were recorded under a specific set of settings: Light OFF, Arm Speed HIGH and Vehicle Speed HIGH. While the characters

changing for different functions and different settings were quickly and easily identifiable, without the corresponding modified check sum character, the string was incorrect. Therefore the touch-screen only sent fixed strings to the robot. Each user press of a button sent one string. This method turned out to be slightly unpractical since to generate a smooth motion the user had to keep clicking on the button at a fast pace. Nevertheless this project established the first and crucial step towards a more sophisticated software-based control system.

4.1.2 Moving towards a PC-Based OCU

The next step was to enhance the Senior Design project. Indeed in order to say that the original control box has really successfully been replaced, the new system would have to be closer to a "clone" of the analog box, which had not yet been done. The touch-screen button did not reproduce the toggle switch effects of the native controller. The code did not generate the string commands. Instead it copied and pasted a limited set of commands. Before rectifying those two main dissimilarities, a new controller was identified as a better fit for a software-based controller. Indeed the software used in the touch screen was only applicable when using that specific touch-screen. It was not portable to another touch screen or a computer. Moving toward a PC would keep the advantages of the touch-screen and increase the portability of the program while establishing the ground for later acquiring a degree of autonomy. Those advantages are underlined below.

- In moving to PC the user does not loose the descriptive aspect of having a screen with images and help menus. Instead this aspect is emphasized, as more and more people are familiar with computer.

- A C++ program provides more flexibility in programming and allows feasible transition between RS-232 to 802.11 g wireless system for example. This transition allows the incorporation of this system in existing networks. (Modular bricks in the IRIS lab)

- Ultimately the goal is to give ANDROS a degree of autonomy. This would later require having a certain control unit on board. Hence moving to a computer-based control system allows installing a local intelligent system while controlling the robot from a remotely located second computer.

The change in devices starts with a stand-alone software such as C++. Microsoft Foundation Classes, or MFC, is a Microsoft library that wraps portions of the Windows API in C++ classes, forming an application framework. Classes are defined for several of the handle-managed Windows objects and also for predefined windows and common controls. This software eases the burden on the programmer by providing great features such as very easy ways to create dialog boxes. MFC sorts the classes and variables, creates header and source files etc. It helps programmer focus on the functions they are

trying to implement rather than on organizing the environment and properly linking their files and libraries. The code from the touch screen was not directly transferable to the MFC project, however adding classes that allow the computer to open, and write to the serial port was fairly simple in this new programming environment. The process of capturing the strings and storing them into a new Graphical User Interface is illustrated on Figure 11.

4.1.3 Enhanced Reverse Engineering

As shown in Figure 11, the reverse engineering process begins by breaking the original connection from the robot to its controller and connecting the controller to a regular PC instead. To improve the former work on the ANDROS, this work starts by reproducing the Kermit experiment and capturing new strings.

4.1.3.1 The Kermit 95 Experiment

This experiment starts by establishing a communication between a regular PC and the control box using Kermit. Once the send/receive cables were identified on the control box, the PC and the box were connected using an RS232 cable. Figure 12 illustrates this setup. The signals to be sent to the robot can be captured and read on the screen of the computer through a Kermit terminal. The K95 software came with instructions on how to read and capture the screen of signals from different sources such as selecting the serial port and setting the speed of the exchanges. On the "session" has started, it is possible to capture the Kermit Terminal content in a text file as illustrated in Figure 13. As shown in this figure, the control box loops through as 21-character string constantly. Different characters change depending on the selections made by the user. For example the highlighted string in Figure 2-2 corresponds to the control command *Torso Left*. As the user presses *Torso Right*, the 10th character changes from a 4 to an 8.

Example:

0A000C2004908C82C0Ëññ -- *Torso Left*
0A000C2008908C82C0Ïññ-- *Torso Right*

Using those exact strings and saving them into a C++ program causes the robot to react. The first attempts to control the robot used a copy and past method and used unchanged captured strings. While the robot was controllable from a regular computer equipped with Visual Studio .NET application, the functionality in the first version of the GUI was still the same as in the touch screen. There were two limitations in the previous versions of this GUI. The first one was due to the special ASCII character at the end of each strings, the second was in the functionality of the program itself.

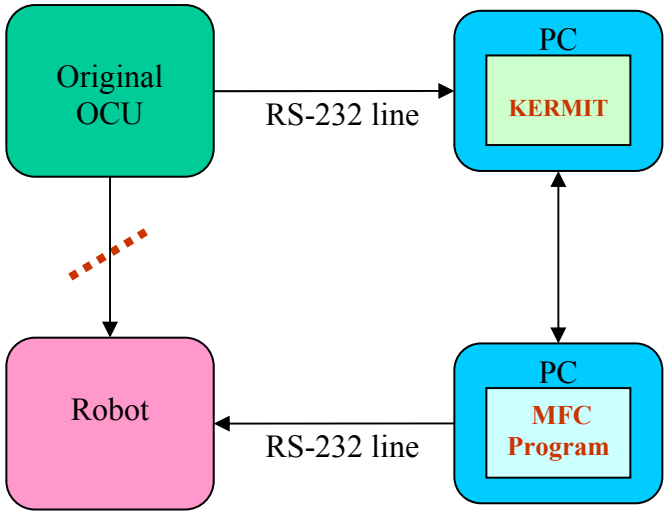


Figure 11: This diagram illustrates the process of extracting data from the native controller and storing it into a C++ program.

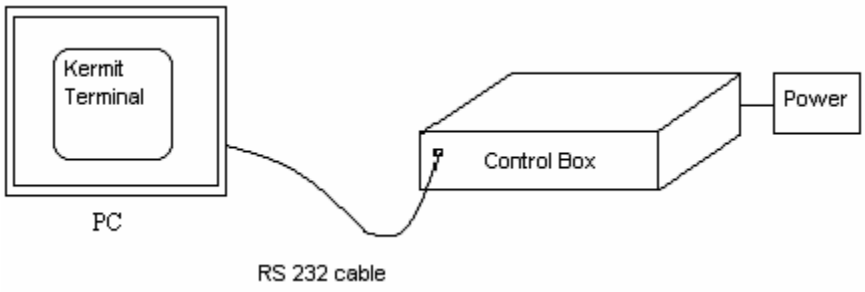


Figure 12: This picture shows the set up for capturing the strings.

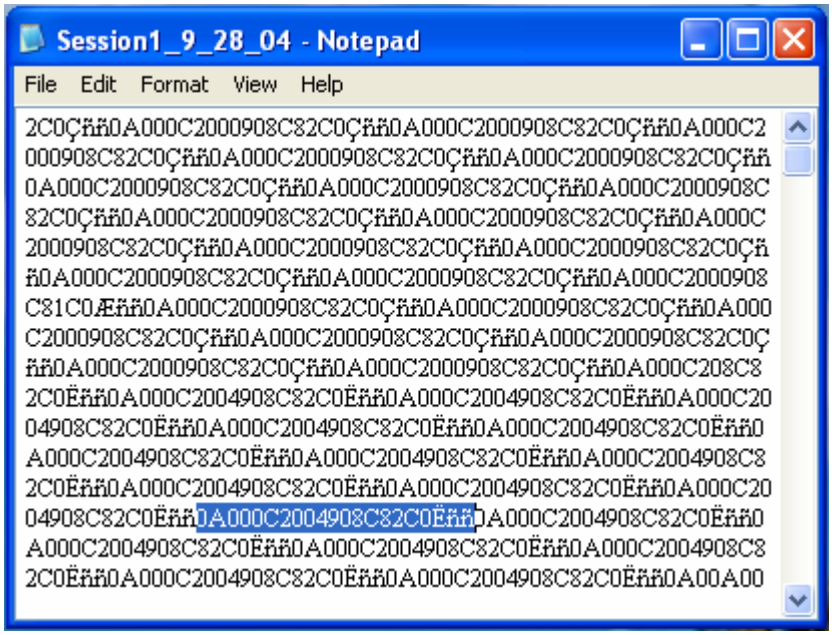


Figure 13: This window is an example of a saved Kermit session.

4.1.3.2 The character sum

Even though only specific characters changed in the string for different body motions, driving motions or settings options, whole new strings had to be sent for different commands. Each character could not be independently changed since the original program was unable to predict the appropriate corresponding character sum at the end of the string.

The first attempt to solve this problem was to have a database of all the basic commands under different set of settings. Even though memory is not an issue with the storage capacity of computers nowadays, this method affects the flow of the program and decreases its efficiency. Moreover it increases the time that one has to spend acquiring data from the original system and the number of strings they are able to send back to the robot. To be more specific, assuming a total of 22 basic body and driving motion commands and assuming only 3 settings variables; if we reduce those variable to take just 2 values (HIGH and LOW or ON and OFF) this process already implies capturing ($22 * 2^3$) = 176 strings. Allowing those settings variables to take more values or increasing their number makes this number go up exponentially. Again with the space and processing speed available in computers now, that may not be a major problem however in terms of data acquisition and programming having a database can easily become tedious, time consuming and error prone.

A closer look to the ASCII character revealed that it was indeed a check sum. Without the appropriate check sum, the robot does not respond to the command. This mechanism constantly checks the validity of the commands before sending them to the robot. The sum of the 18 previous characters after the separators (ññ) is compared to this last character before sending the packet to the robot. The check sum is not a hexadecimal accumulation of the other characters but the sum of their ASCII codes. Table 1 show the ASCII codes for the letters and numbers that compose the first 18 characters of a string.

Table 1: The first 18 characters of a string each correspond to an ASCCI code.

Character	ASCII Value	Character	ASCII Value
0	48	8	56
1	49	9	57
2	50	A	65
3	51	B	66
4	52	C	67
5	53	D	68
6	54	E	69
7	55	F	70

Example using Table 1:

0A000C2008908C82C0Ïññ-- *Torso Right*

To generate the command Torso right from a constant signal, the programmer only needs to change the 10^{th} character of the string and then compute the sum:

S = 48 + 66 + 3*48 + 67 + 2*48 + 56 + 57 + 48 + 56 + 67 + 56 + 50 + 67 + 48

Once the sum S is divided by 256, the ASCII character corresponding to the residue is the correct check sum to send to the robot.

With this simple change the last character is always correctly predictable. This important modification allows an easy expansion of the project. Characters can be changed individually for different functions or settings provided the correct sum is generated at the end of the string. In other words there are no longer restrictions in number of functions or setting to duplicate as the strings are not stored in the program but generated at will.

4.1.3.3 The functionality

The first version sent one string with each click of the mouse just as the touch-screen did. The problem with this method is that it is really difficult to accomplish continuous motion. It is rather jerky and to approach a smooth motion the operator has to keep clicking fast without interruptions. Needless to say it is unpractical and not intuitive.

The second version achieves smooth motion by starting a timer with just one click. When the timer is started, it continuously sends a string at a fixed rate until the user stops the timer. Even though this method seems a lot better especially for driving motion for example it is still not practical. The Remotec ANDROS robots do not have limit switches. The motor stalls for a period of time but the operator has to then stop the motion by releasing the corresponding switch. The problem with the timer is that the operator has to actually stop the timer by clicking on another button. Otherwise, the robot can run into obstacles, and/or simply try a motion beyond its physical capabilities. Therefore not only is this method not intuitive but it is dangerous for the environment and for the robot.

The latest version of the GUI fixes those problems by using bitmap images instead of the buttons provided by MFC. This program captures the signals from the mouse and when the mouse is pressed down on a specific image it sends the corresponding strings. It does so continuously until the mouse is released. This is the best analogy to the mechanical switches on the original control box. This version of the GUI is shown on Figure 14.

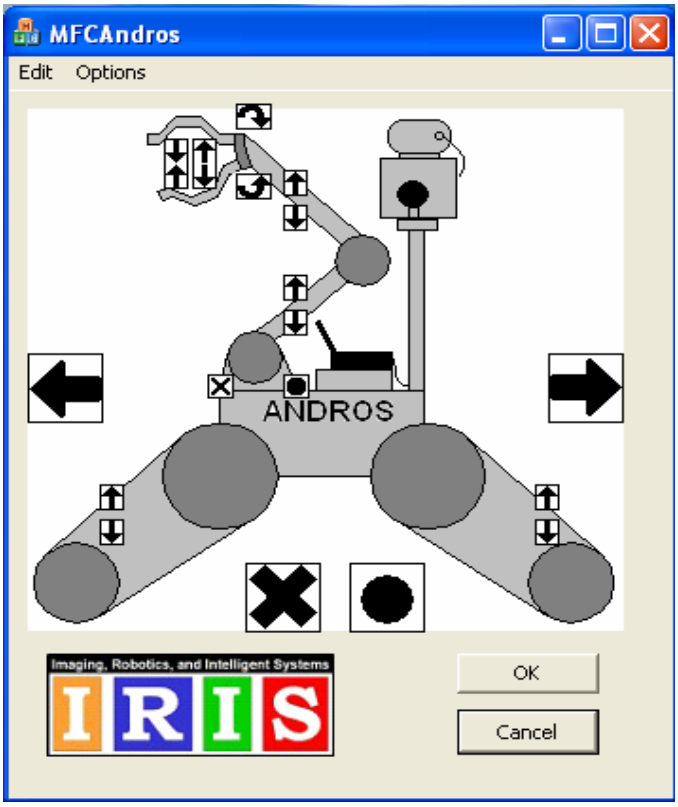


Figure 14: The main window of the MFC Graphical User Interface for the ANDROS Mark VA includes body functions in addition to vehicle drive motions.

4.2 Making the ANDROS a Mobility Brick

4.2.1 Wireless Connection

Once the logic of the strings was completely understood and duplicated the data was still sent to the robot using the radio modems. The idea is to move to a mode of communication that is not specific to the ANDROS but really easily integrated into the Sensor Brick Concept. Also there was not yet an on-board intelligence. The data was sent from the COM port of the computer to the radio modems and then finally to the robot, creating a wireless RS-232 line between the PC and the ANDROS.

The mobility brick is defined similarly to the sensor brick. It is a mobile platform with its pre-processing unit, communication unit and power. It is an independent unit that can be operated on its own (teleoperated) or part of a more sophisticated autonomous system. The communication block in the sensor brick typically remotely transmits data using standard IEEE 802.11g. The wireless communication card (wireless Ethernet) is attached to the computer motherboard. Any number of networks can be utilized with wireless Ethernet devices [5]. Therefore the transition from the teleoperated robot to the mobility brick implies the elimination and replacement of the RS232 wireless line by an Ethernet connection (see Figure 15**)**.

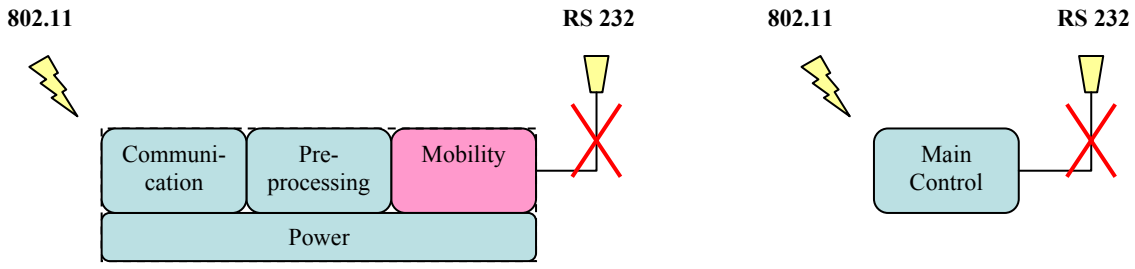


Figure 15: The transition to the mobility brick implies the replacement of the RS-232 radio modems by an Ethernet connection.

4.2.2 JAUS Primitive Driver

4.2.2.1 JAUS Overview

The Joint Architecture for Unmanned Systems (JAUS) is the architecture defined for use in the research, development and acquisition of Unmanned Systems [70]. This new architecture is introduced in part I of the Reference Architecture (RA) Specifications. Part II is the Message Definition and Part III is the Message Set. Both Parts explain in more details how to go about creating a JAUS compliant system as JAUS is in a few words a component based, *message passing* architecture [70].
To implement a JAUS compliant system one must understand JAUS language and appropriately attribute the title of systems, sub-systems, node, component, instances and messages.

- A System is a sub-grouping of sub-systems.
- A Sub-system independently performs one or more functions within a system.
- A Node defines a processing capability within a sub-system.
- A Component provides a unique functionality for the unmanned system and within a node.
- An Instance allows redundancy and duplication of components.
- A Message is composed of a header and data transmitted between components.

The system, group of cooperating sub-systems is the highest level of the JAUS hierarchy. At the second level, each independent and distinct sub-system performs one or more functions within the system. The node is an assembly of hardware and software parts that support a particular function in a sub-system. In other words, a node connects to a device using the appropriate hardware and software required to operate that particular device. The component/instance level is the lowest level. Generally speaking the component is an executable task [70]. Instances allow component redundancy. JAUS indicates the acceptable names and ID numbers for all systems, sub-systems and components. While the engineers have a lot freedom to implement and name their systems, sub-systems and nodes, all the components currently supported by JAUS are already specified. Hence if the Developer chooses to add any of those components there already is a specific ID number attributed to each one of them. There also is a degree of freedom of adding new components and ID numbers. The JAUS components are grouped as follows [68]:

- Command and control components,
- Communication components,
- Platform components,
- Manipulator components, and
- Environmental components.

4.2.2.2 Mobility Brick/Primitive Driver

A platform component of particular interest for this work is the Primitive Driver. The definition of the mobility brick resembles that of the JAUS Primitive Driver. This component performs basic driving and platform related mobility functions including operation of common platform devices such as the engine and lights [70]. Hence it appears that our mobility brick concept is that of the Primitive Driver. To be more specific, the mobility brick can be identified to the Primitive Driver once it is JAUS compliant. Since messaging is the base of this architecture a JAUS compliant system has to be able to read and interpret JAUS messages as well as sending JAUS messages. In this case, the Primitive Driver is the bottom of the typical Unmanned System Diagram. Therefore the mobility brick mainly has to be able to receive and decode JAUS messages. The pertinent information in the message header allows the pre-processing block to decide whether or not it is the desired destination of a message and react consequently when applicable.

JAUS is a software-architecture; components are not devices. This feature allows the mobility brick to an independent subsystem. It can be controlled independently of other subsystems or it can receive commands from an external device. We can identify two components in the teleoperated system for the ANDROS: a *System Commander* (ID 40) and a *Primitive Driver* (ID 33). While those ID numbers are predefined in the RA, there is more freedom in numbering systems, subsystems and nodes. The control program for the mobility brick includes two main functions, the commander (sender) and the driver (receiver) components. The *System Commander* generates the JAUS messages and the *Primitive Driver* is the receiver, which decodes those commands and performs the corresponding driving commands. In other words in the main program when the user presses a button, JAUS strings are created and sent to a *Receiver* function which distinguishes between the header and the actual data to be transmitted to the mobility platform. Again at this point, because of the simplicity of the system and the lack of feedback from the robot, much of the header information is not yet pertinent to the current system. For example given the small size of the data, there is no need for data sequence at this point. The code only checks for the destination of the message and sends the string to the robot only and only if it is the intended target. This makes a simple compact system, which can later be expanded as more components, and nodes are added.

4.2.2.3 First Result

The first integration of the new software implemented and the mobility platform is shown on Figure 16. The program is stored on a laptop; other computers may communicate with the local computer using 802.11g while the connection between that computer and the robot is established through an RS-232 link. The laptop is secured in a foam-filled casing and the casing is fixed on a plate solidly attached to the robot. This system is simply a mobility platform it does not include sensors, it does not perceive its environment and does not take decisions. At this point it is an enhanced teleoperated system.

Figure 16: This picture shows the version of the mobility brick without sensor brick.

4.3 Autonomous Navigation and Directed Imaging Robot (ANDIbot)

Once the original system has been reverse engineered, and once it has adopted a new wireless communication it is ready to acquire autonomy. The user may teleoperate the ANDIbot directly from a computer by running the controller program; or he/she can log into the brick from a remote computer and control the robot from using a local program. He or She may also use the *System Commander* program from the Main Unit with the *Primitive Driver* code located on the robot. The control program in the Main Unit includes an option for autonomous mode. Once this option is selected, the main computer takes over the control of the robot until the operator intervenes or until it encounters an anomaly. This computer uses the sensor brick to guide the mobility brick. The sensor gets the information from the environment; the information is processed and exchanged between the pre-processing blocks of both bricks. This information is used to determine the robot's next move. For practicality reasons, in the implementation both bricks will share the preprocessing block as shown in Figure 17. In other words instead of installing two on-board computers, the same processors gets the information from the sensor and drive the mobility brick.

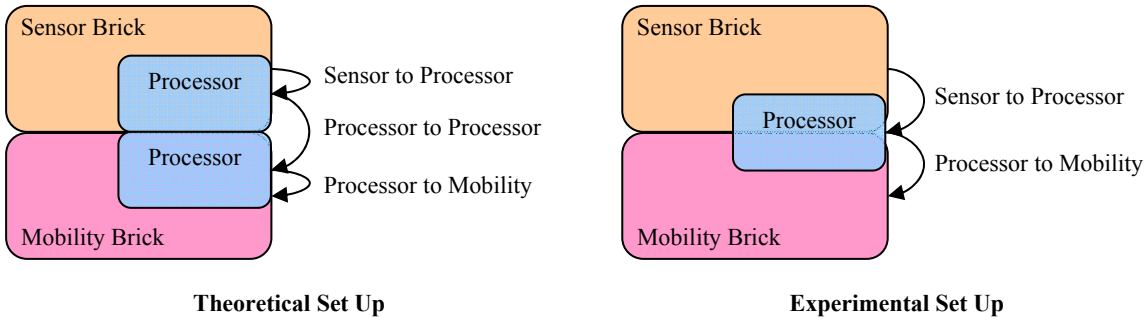


Figure 17: The Experimental set up for the ANDIBot uses a shared processor between the sensor and mobility brick.

4.3.1 Wall Following Algorithm

4.3.1.1 Configuration of the scanner on the robot

The implementation of this function starts by the positioning of the scanner on the robot. The goal is for the robot to autonomously follow a wall. In order to do so, the robot has to constantly locate the wall but also stay clear from obstacles in his direct path. The first assumption is that the wall should always be to the left of the robot. This decision was made arbitrarily since to cover a perimeter a person could as easily walk along the wall clockwise as counter clockwise. Hence the view to the left of the robot is preset as the wall. The front view is used to avoid collisions with objects on the robot's path or to detect a possible upcoming corner. The scanning angle that maximizes both the front (d_{front}) and side view (d_{wall}) of the robot is to be use for this application. Let D_o be the radial distance to detect obstacles. This distance represents the desired distance between the robot and the wall. This distance is flexible and will be discussed subsequently. Let θ be the difference in orientation between the range (R) scanner and the robot as shown in the diagram in Figure 18 then:

$$\cos(\theta) = \frac{d_{front}}{2*D_o} \qquad \text{and} \qquad \sin(\theta) = \frac{d_{wall}}{2*D_o}$$

Therefore to have d_{front} equal to $d_{wall,}$ the sine and the cosine of θ have to be the same. In other words the angle that maximizes both views is a $\theta = 45°$.

Once the orientation of the scanner has been determine, the exact position of the scanner on the robot has to be set. The second assumption resides in the height of the scanner relative to the ground. Any height on the 50 cm tall robot is a reasonable height to detect the wall. However in terms of obstacle avoidance, an argument can be made that obstacles will not necessarily be at the scanner's height. For the time being the assumption will be that obstacle will be tall enough (approximately 60 cm) to be detected by the scanner placed on top of the robot. This is a practical assumption, since the chosen height is not significantly above than ground level. Moreover the ANDROS is an all terrain robot capable of climbing over small obstacle without loosing its balance. Therefore the scanning will be one horizontally and at the fixed height of 60 cm.

The last adjustment made in the scanner-robot configuration is the switch between its front and its back. Indeed the placement for robot accessories is in the back of the robot. When the scanner is placed in this position, the robot's arm has to be "parked" or completely lowered so it is not in the way of the laser. To avoid confusion, the scanner will be facing away from the arm and the reverse motion will become the forward motion in autonomous mode. This feature will be emphasized in the hardware implementation section.

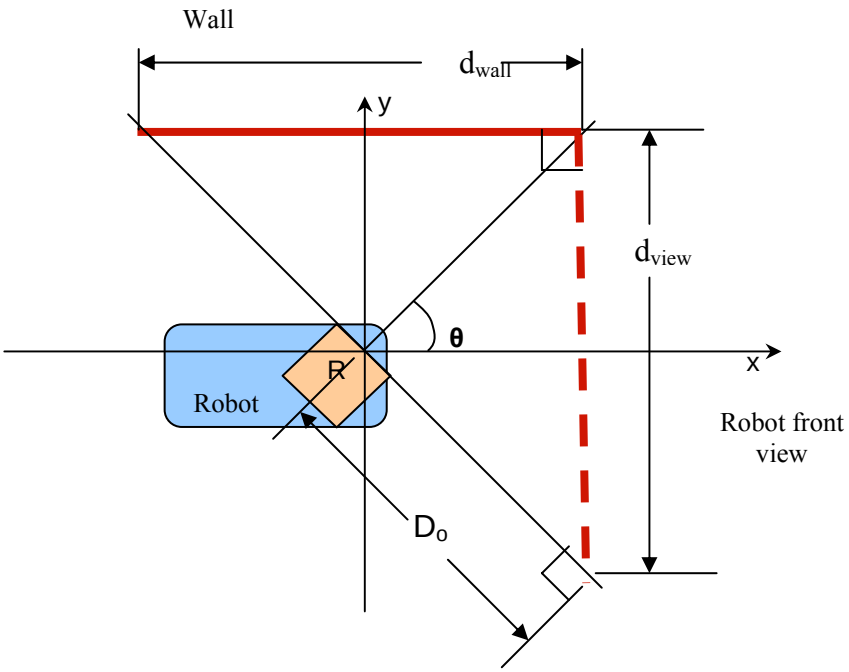


Figure 18: This diagram represents the top view of the system. The robot is equipped with the range scanner (R).

4.3.1.2 Pseudo-code and diagram

The wall-following algorithm is simple. On the left side of the robot, the ANDIBot is looking for the wall; it tracks down the beginning and the end of the wall within the chosen scanning angle and computes their difference. More specifically it computes the difference between their projections onto the y-axis. Within a certain threshold this difference indicates that the robot and wall are aligned with one another. Beyond this adjustable threshold, the *YY* difference indicates whether the robot is excessively going towards the wall or away from the wall. If this is the case, the robot takes the corresponding step to correct its position relative to the wall. Meanwhile, the ANDIBot is constantly checking for obstacle in its front view as well. Whether it is a corner of the wall or another obstacle it is ready to turn right to avoid collision. If at a certain point of time an obstacle suddenly comes too close to the robot – adjustable by the programmer – the robot will come to a complete stop. The program runs in an infinite loop until such obstacle stops it or the operator does. Figure 19 is a diagram displaying the different variables used in the program and Figure 20 shows how they are used in the system flowchart. They are:

- **d**: This distance is the difference in the projections of the right and left ends of the wall. The wall is only detected within a certain angle to the left of the robot. The d or *yydifference* determines how aligned the robot is with the wall. A zero value would be the ultimate case where the robot is perfectly parallel to the wall which is not the most practical case. While an excessively greater value (beyond 1m) will cause the robot to drive in zigzag.

- **D**: This distance is the distance at which the robot starts detecting a corner or an upcoming obstacle. The maximum reliable obstacle detection for the sensor brick is 4m. The ANDIBot is 1m long and that much distance has to be available during a turning operation. Hence the bounds for this variable are 1-4m.

- $\boldsymbol{\theta_w}$ and $\boldsymbol{\theta_f}$ are the angles within which the robot is searching for the wall and oncoming obstacle. While the maximum determined are 90 degrees to the left and 90degrees to the front, practical values may be adjusted depending on the desired performance.

- Other controllable variable not shown on the diagram include $\boldsymbol{\Delta\theta}$. This variable represents how much the robot corrects its trajectory when it is not parallel to the wall or when it encounters a corner. This value is a number of strings that correspond to a small angle. Also once it has established that it can go forward, the robot has to decide how far to travel before checking on new sensor information. This $\boldsymbol{\Delta l}$ is also a number of strings that corresponds to a distance.

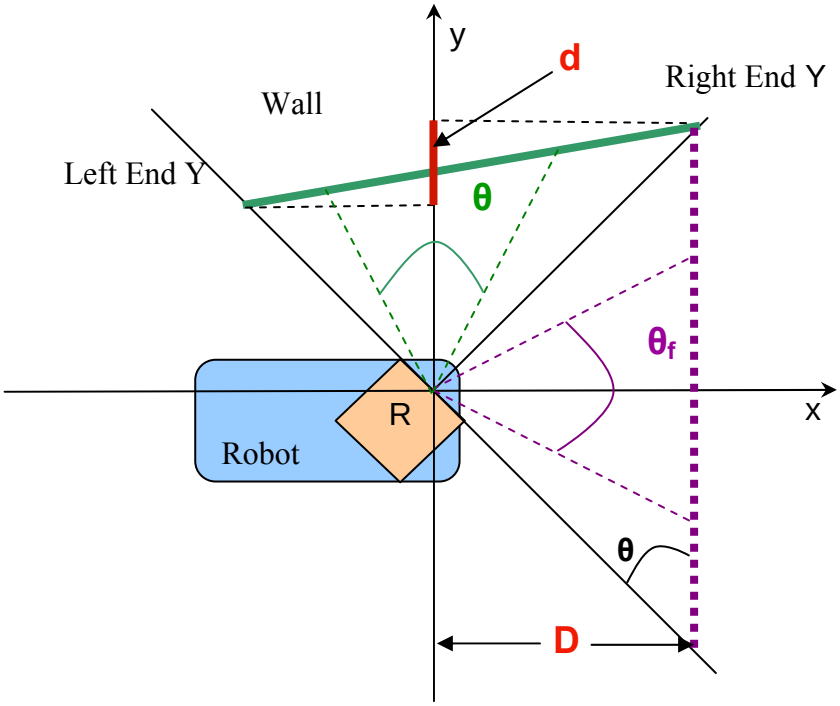


Figure 19: This diagram shows the top view of the system with the variables used in the wall following algorithm.

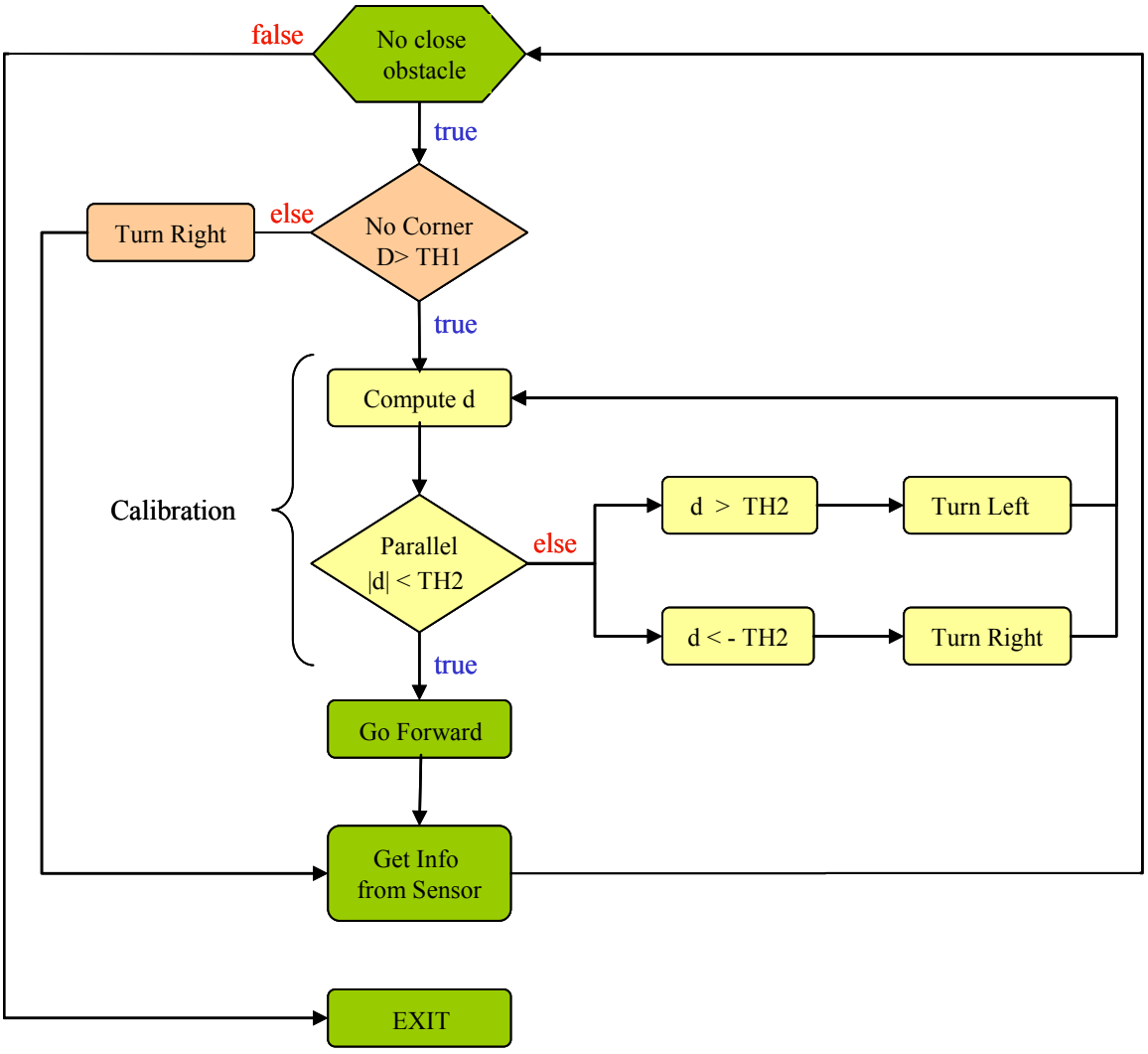


Figure 20: This flowchart shows the logic used in the wall following algorithm.

4.3.2 Hardware Implementation

The hardware implementation for the ANDIBot consists mainly of mounting the ANDROS robot and the range sensor together. The brick sits on an aluminum base with the same length and width. This aluminum plate is fixed on a stronger piece of aluminum for support, which is in turn is attached to the robot through a solid bar. The way that this bar is bolted onto the plate creates the 45 degree difference in orientation needed between the robot and the scanner. To prevent the brick from sliding off the base, four pieces of aluminum constrain its motion relative to the robot. In addition a rubber film placed onto the plate increases the friction between the sensor brick and the plate. Finally a strap solidly ties the sensor down its support. Figure 21 shows the aluminum base for the brick before being mounted on the robot. Figure 22 shows the mobility platform with and without the brick.

As previously mentioned the facade of the robot is inverted to avoid obstruction from the arm. In other words the back of the robot becomes the front. The scanner faces away from the robot arm as shown in Figure 23. The backward and forward directions are switched and the autonomous navigation is possible independently of the position of the manipulator.

Figure 21: The brick carriage simply consists in an aluminum plate with constraints and an aluminum bar that links it to the robot. The red strap and the rubber film under the brick increase its stability.

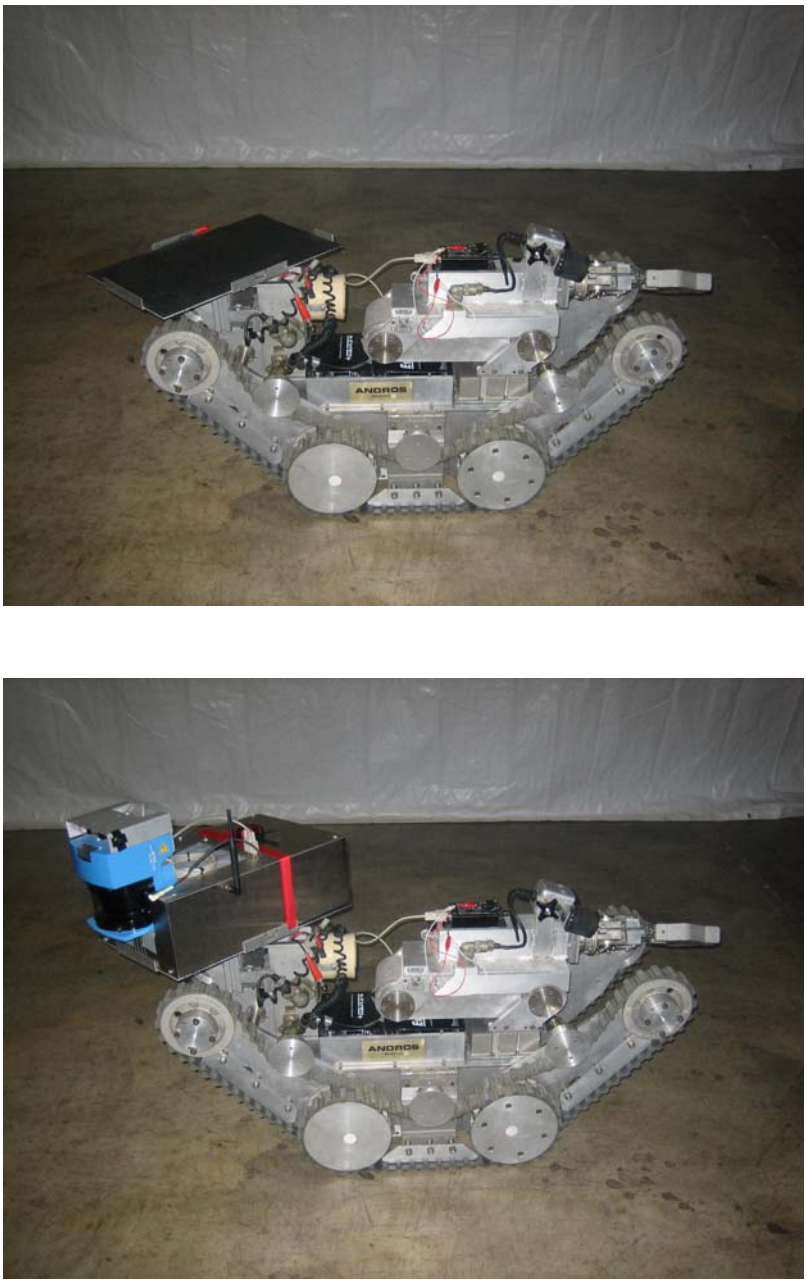


Figure 22: The top picture show the ANDROS without the brick and the bottom picture includes the range brick strapped to the robot.

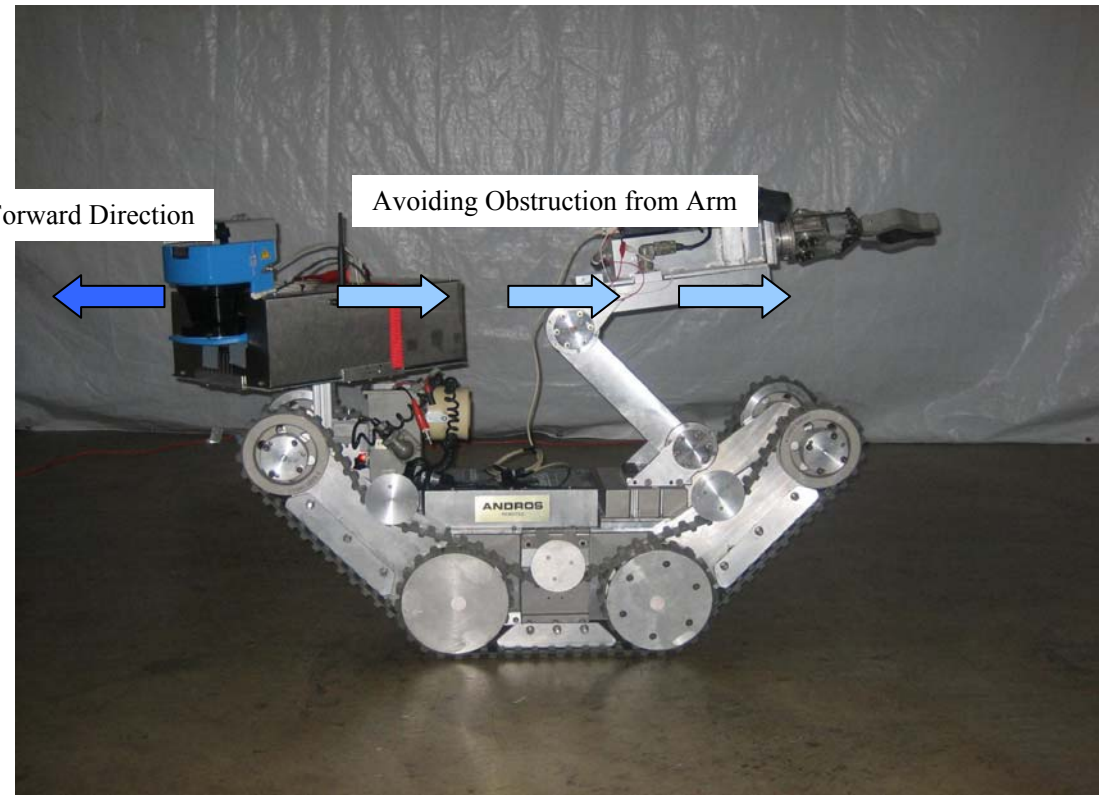


Figure 23: The back of the robot becomes the front in autonomous mode to avoid obstruction of the laser from the arm.

5 Experimentation

This chapter describes the analysis on the ANDIBot. The reverse engineering process implies that the programmer does not have direct access to the motors, the controllers or a of the robot's electronics and mechanics. The only accessible control commands are character strings. While those strings do provide information in terms of speed and direction, they do not give distance data. This is understandable since the robot was originally teleoperated. However, the ANDIBot needs units to decide how far to move in distances and angles. Traditionally autonomous systems use encoders. The first objective of this analysis is to provide the programmer with information that can be used similar to an encoding system. The second part of Chapter 5 will analyze the performance of the ANDIBot. Finally the last section of this chapter will summarize the observations made throughout the analysis.

5.1 Characterization of the Strings

While the characterization of the strings is not an absolute measure of the robot's motion, neither is an encoder. This is where the range sensor brick comes into the picture, which constantly re-evaluates the robot's location. The battery level of the platform and the type of floors it is traveling on are two important factors that can affect the accuracy of the platform's motion.

5.1.1 Straight Motion

For this motion, the number of strings ranges from 5 to 30 in 5-string increments. For each set of commands (5, 10, 15, 20, 25, and 30 strings) there are ten trials, or a total of 60 data points in each direction.

For safety reasons, the upper limit of 30 strings corresponds to approximately 1m, which is twice the obstacle threshold set in the algorithm. Moreover, once the robot receives a set of strings, the only way to stop it from moving is to shut the system down. Considering a dynamic environment it is dangerous to send an amount of strings corresponding to a long distance.

5.1.1.1 Description of the experiment

The final version of the ANDIBot GUI sends command to the robot as long as the user is pressing down on a bitmap image. This method does not allow counting the number of strings actually sent to the robot, at least not easily. Hence for this experiment an older version of the GUI that uses buttons provides the programmer to insert a *for* loop in the

button code and send a specific amount of strings in a single click. A mark on the floor indicates the starting point of the robot and distance traveled after each of the ten subsequent moves for the same command. Figure 24 shows the methodology for measuring the distances on a flat concrete floor (low friction) and the higher friction floor, a carpet. The instrument used in this case is a tape measurer marked to the tenth of an inch.

5.1.1.2 General observations

The first two graphs shown in Figure 25 represent the forward and backward motion on a full battery and are representative of the data sets at 90% of the full battery level. This rough data shows a certain consistency of the measurements before averaging. With the exception of the first data point in each set, which is taken, as the robot first starts moving, the data points seem to closely follow a constant distance. This observation is further confirmed in Figures 26 and 27, which show how the same data points vary around the average in inches and in percentage of the average distance. The fluctuation in inches reaches a maximum of 4.5 for the forward motion and 2.6 for the backward motion. In terms of percentage those values correspond to a fluctuation of less than 10% in most cases. This difference is obviously more visible within in the short distances, notably in the case of a 5-string command. Figures 26 and 27 also confirm a higher discrepancy between the starting point and the average. In the case of short distances this phenomenon is more obvious. The difference comes from the fact that with the short distances the 10 experiments did not require going back to a starting point. The robot seemed to cover the smallest distance after it was turned on, and then stabilized around a higher almost constant value. Therefore two important trends complement each other. The ANDROS responds differently at start up then once it has received a couple of commands, which also explains why it is the most inconsistent with short commands. By the time the robot accelerates the command is over. Sending short packages of strings is equivalent to sending short impulses and causes a rather jerky motion of the robot.

Figure 28 shows an almost linear relationship between the strings and the average traveled distances for both directions. It also uncovers a small difference between the forward and backward motions. The computed average difference is 3.91%, which means the system is not exactly symmetric in straight motions. Measurements errors are not the only cause of this variation; experiments also show that the robot does not move perfectly in a straight line. This is more visible as the robot travels long distances. It shows that despite the apparent straight motion caused by the forward and backward strings they have been captured at a small angle off the right directions. The original joystick of the ANDROS is a sensitive analog joystick and trying to capture 4 discrete directions (Forward, Backward, Right and Left) with exactitude is not a simple task. In this case, the error shows in the nearly 4% average difference between the two directions.

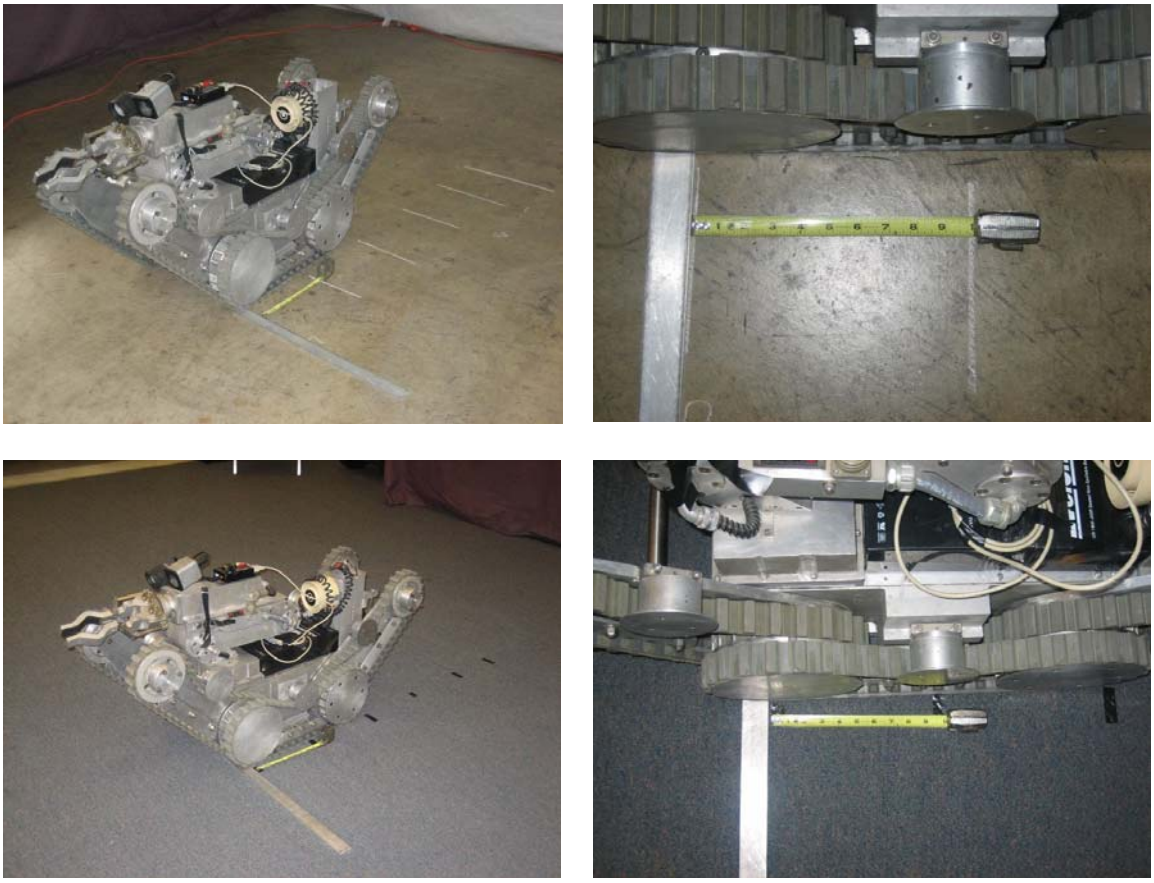
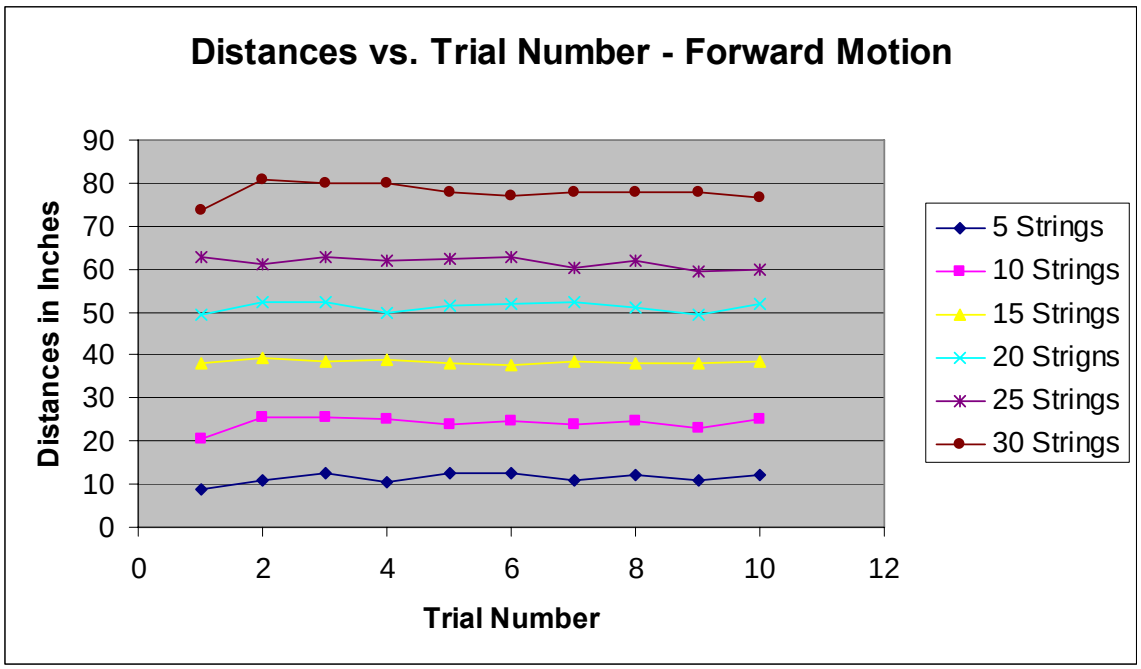


Figure 24: The pictures on the left show the methodology in measuring the covered distance for the same number of command strings. The pictures on the right are close up images of the same process on both types of floors.



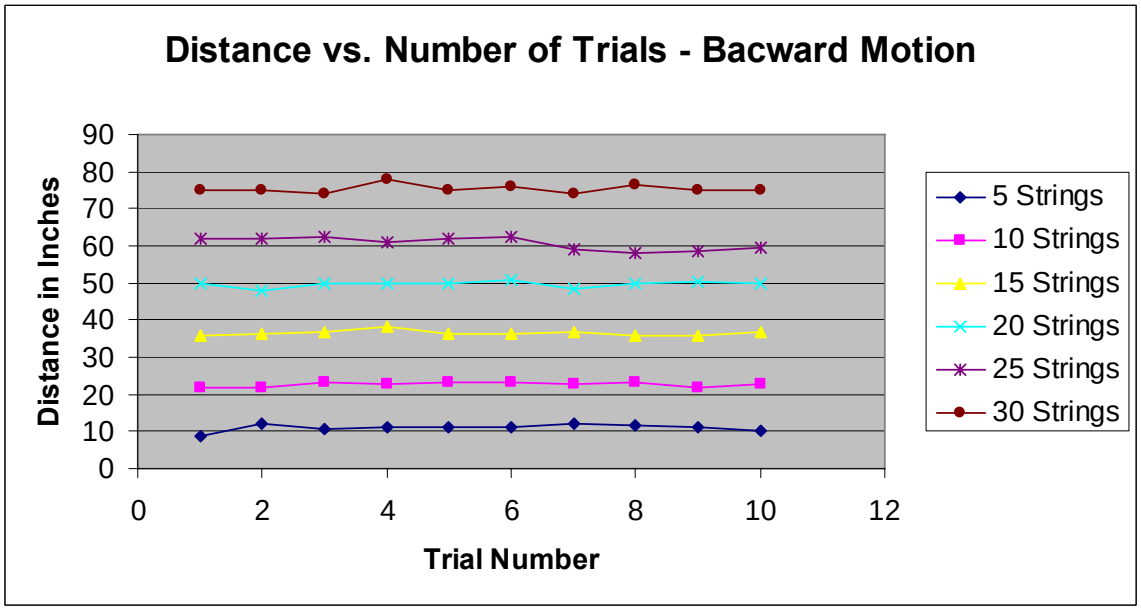
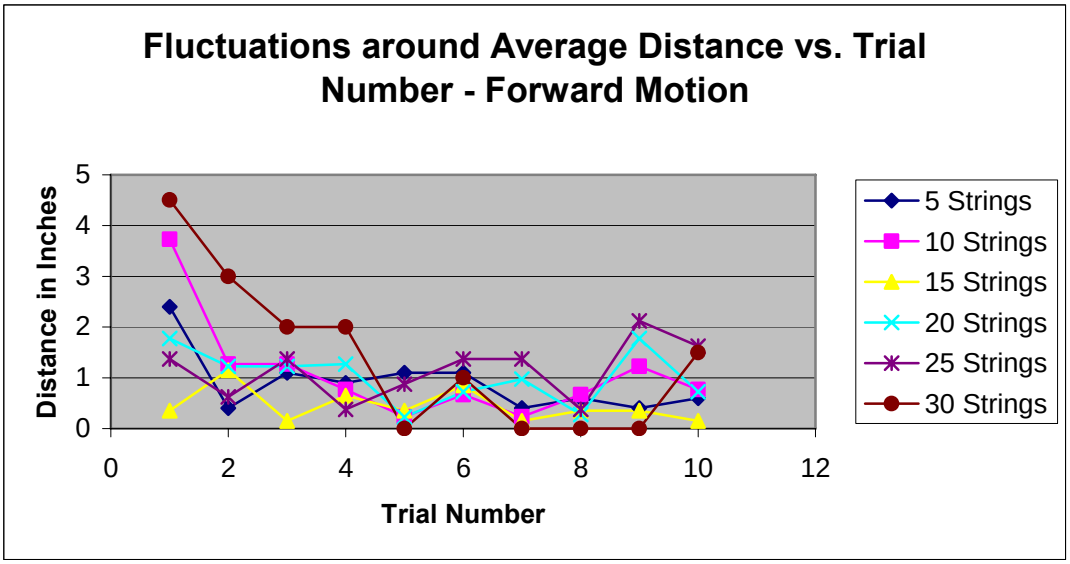


Figure 25: The rough data for backward and forward motion on a full battery appears fairly consistent.



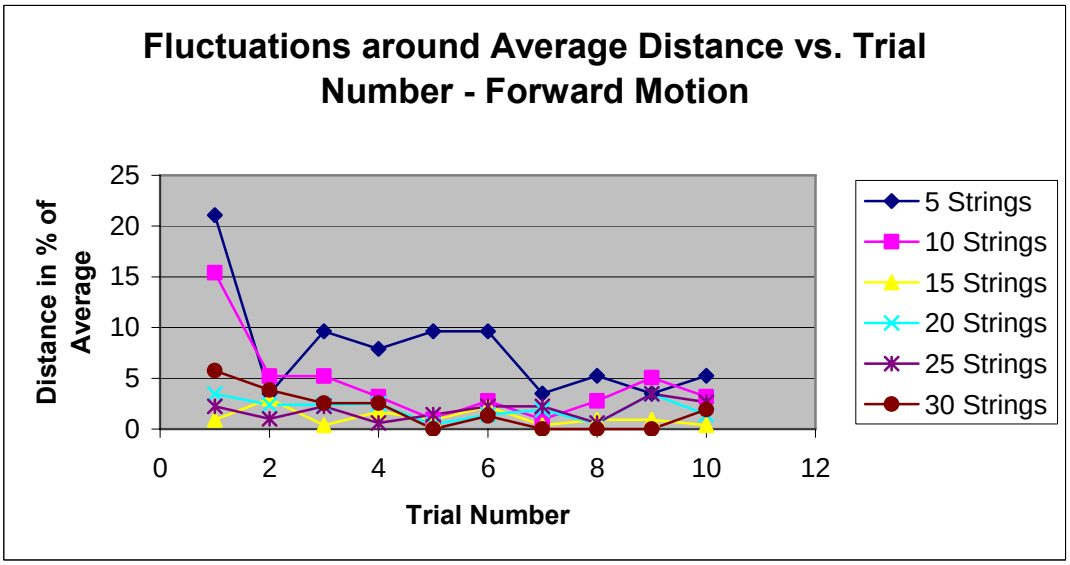
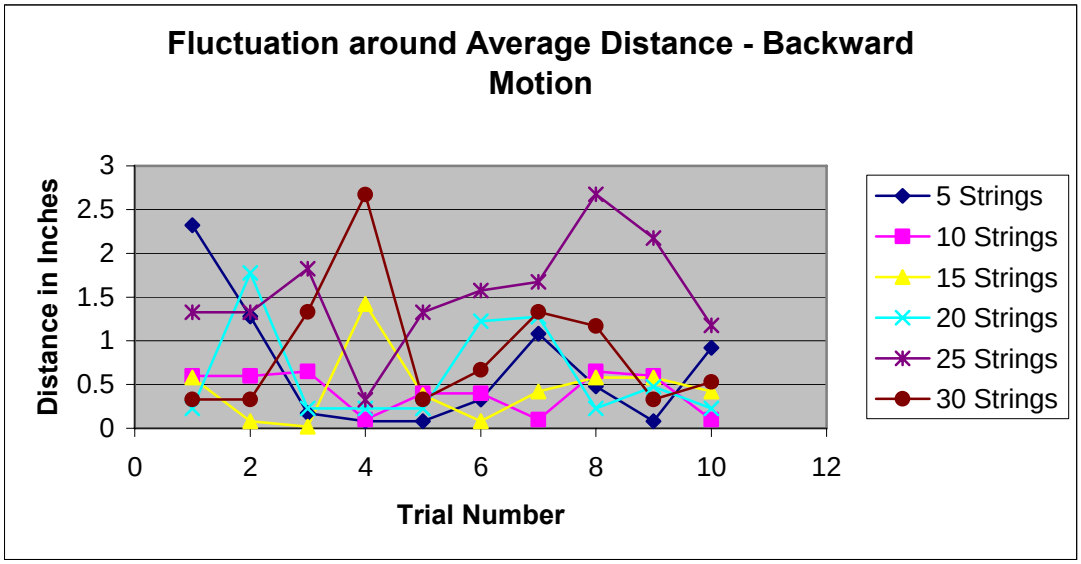


Figure 26: The fluctuation of the rough data around the average covered distance for each different amount of command strings confirms the apparent data consistency except for the shortest distance traveled.



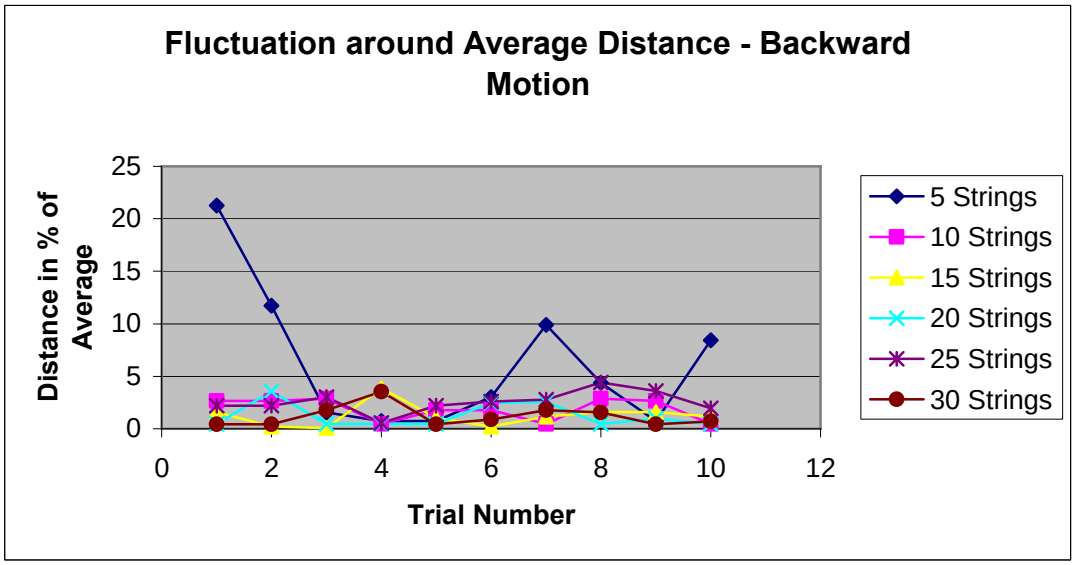


Figure 27: Approximately the same results occur for the reverse motion as for the forward motion

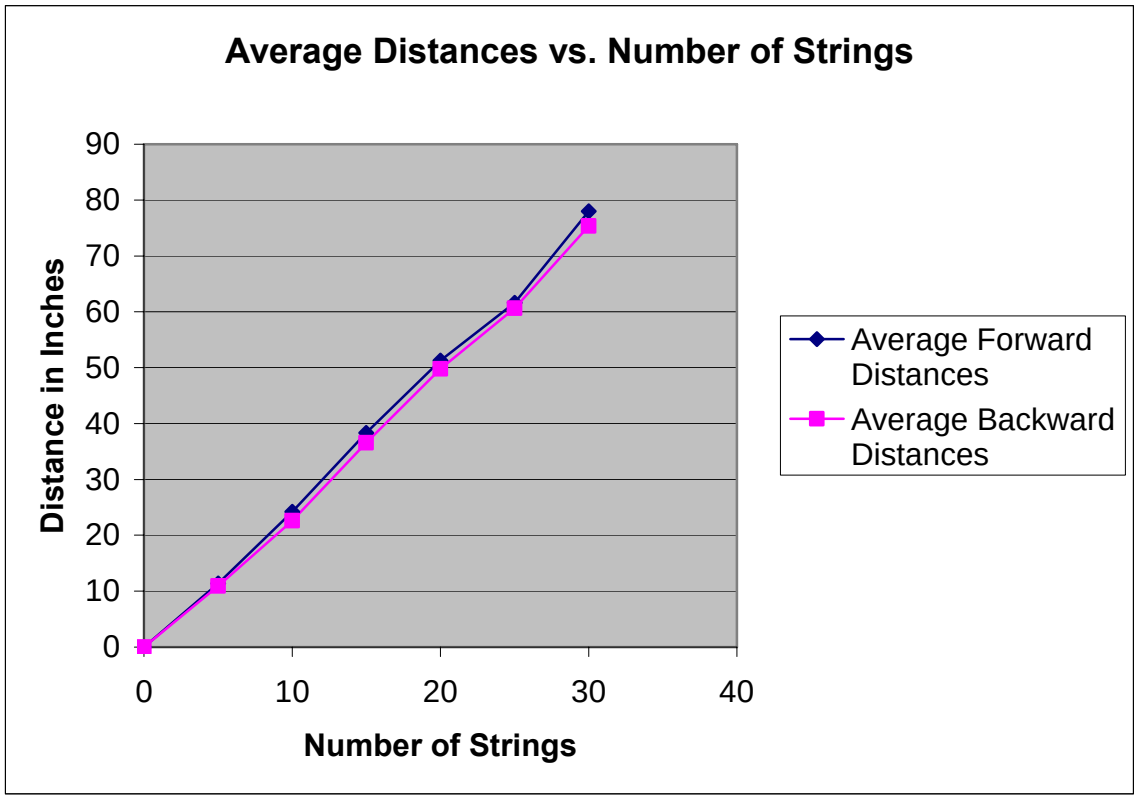


Figure 28: The relationship between the distance and the strings is almost linear.

5.1.1.3 Differences of performance across the floors

Figure 29 shows the average forward and backward motions per number of strings on the flat concrete and on the carpet. The effect on both directions is not exactly the same but it is minimal as shown twice in Figure 30. This experiment verifies the decrease in performance on a higher friction floor. The difference is a significant roughly 13% and becomes more and more visible as the traveled distance increases. The difference reaches its maximum for the 5-string commands as in previous experiments.

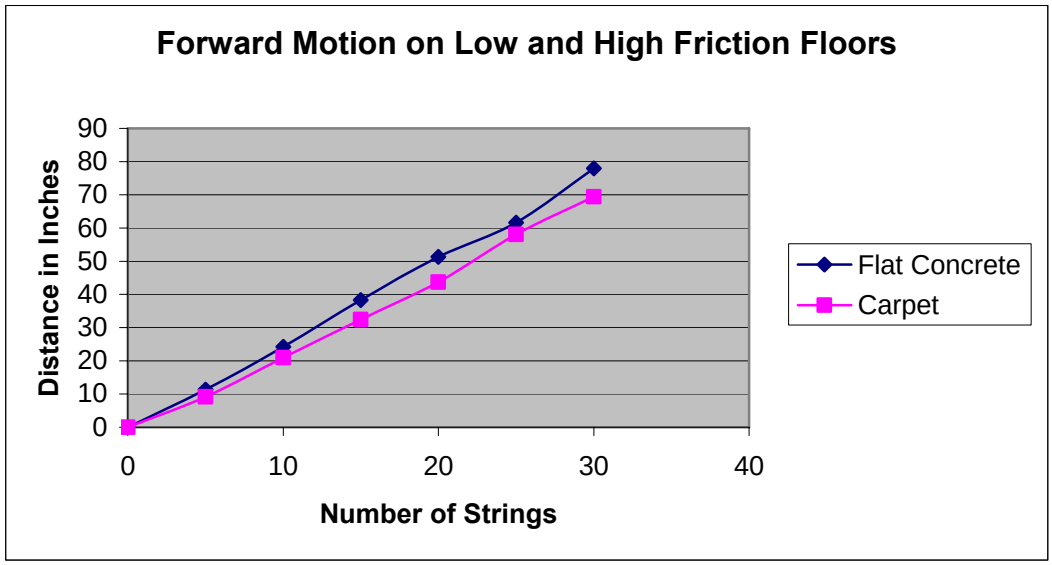
5.1.1.4 Variation with battery level

The ANDROS requires a minimum of 24V to guaranty normal operation. As the voltage drops under 24V, its efficiency quickly decreases. In this case a full battery will be approximately 25V. A low battery level can go down to 15V however the robot starts giving signs of malfunction before then and it is not recommended to operate the ANDROS under 24V. Repeating the forward and backward experiments on concrete at a battery level of 23V will give an idea of the effect of a decrease in supply voltage. Figure 31 shows the difference in performance when going from a full battery to a voltage of 23V. The decline in distance covered per number of strings is greater than in the case of different types of floor. The discrepancy is a considerable 17% as shown in Figure 32.

The variation due to battery levels combined with that caused by the increased friction in the previous experiment is equivalent to a decrease in covered distance of nearly 27.79%. This significant decline explains why the same experiment on carpet required charging the battery. At that point the robot is simply barely functional. A battery of 23V did not allow properly driving the ANDROS on carpet. The battery level variation confirmed that in order to be efficient, the robot has to be at least 24V. A smaller voltage may cause awkward behavior. This effect is worst on rougher terrains. As the friction increases, the robot needs more power to move and needs to be at least the nominal voltage level. Another pertinent confirmation on Figure 31 is that the difference in performance due to battery level clearly becomes more obvious as the distance increases. The operator may not realize there is a problem for short commands, however as the number of strings decreases the lag quickly becomes greater.

5.1.2 Turning Motion

For this motion the upper limit of strings reaches 180 degrees, which is twice as much as the maximum turning angle implemented in the wall following algorithm. For each set of commands (5, 10, 15, 20 and 25 strings) the experiment is repeated 10 times for a total of 50 data points.



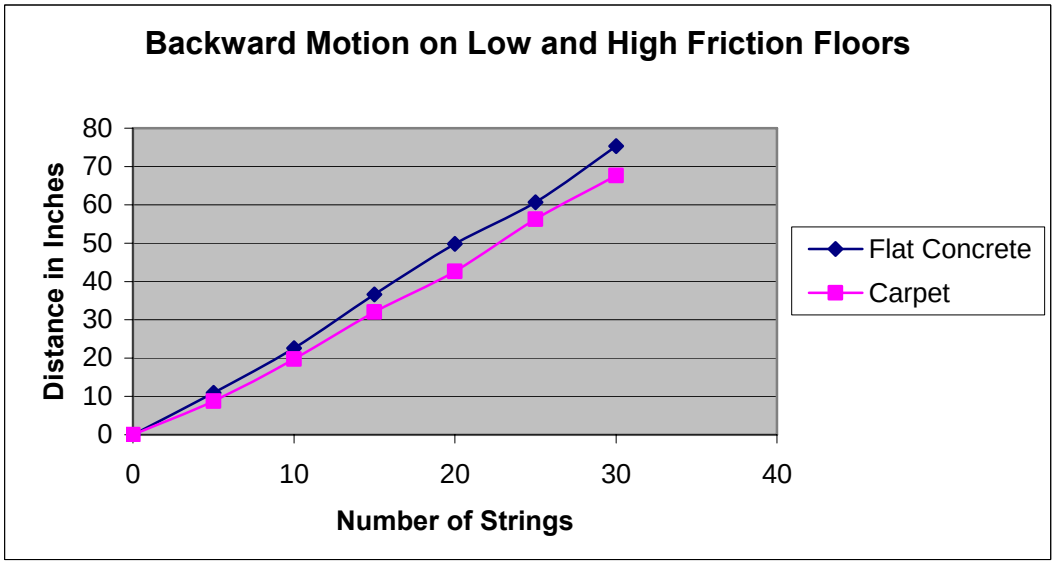


Figure 29: The effect of the floor on the robot's performance is clearly visible in both directions.

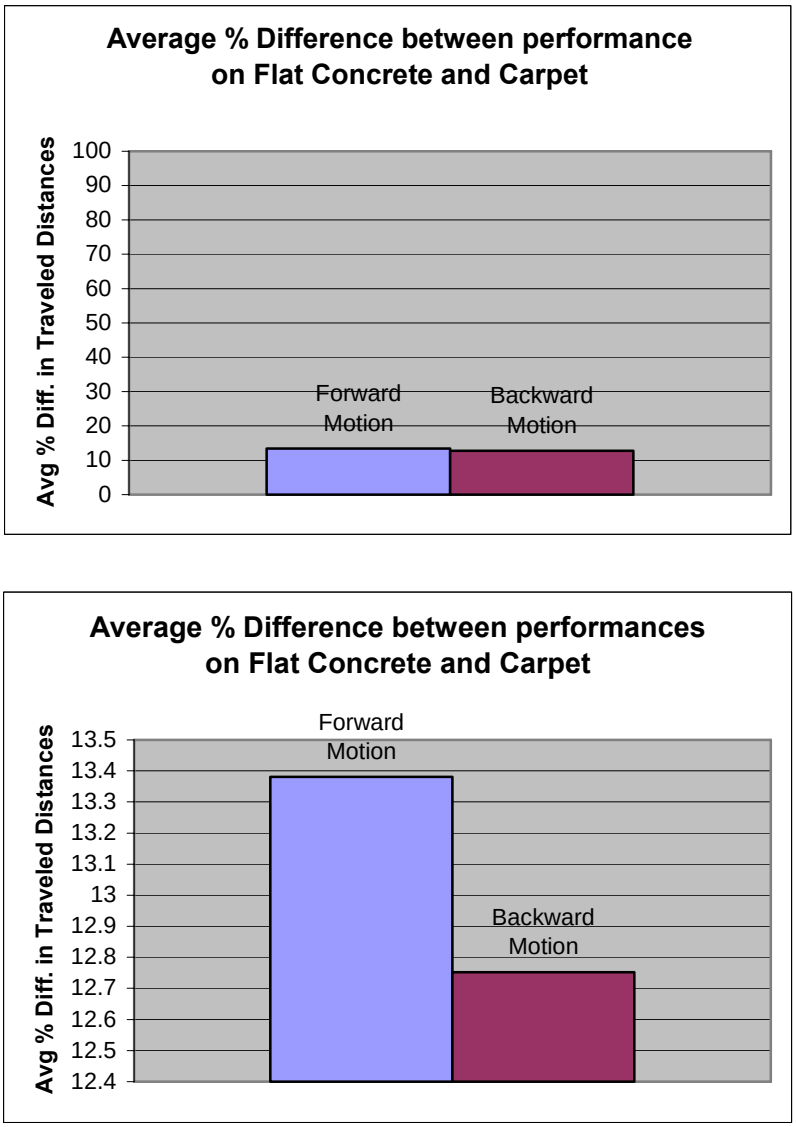
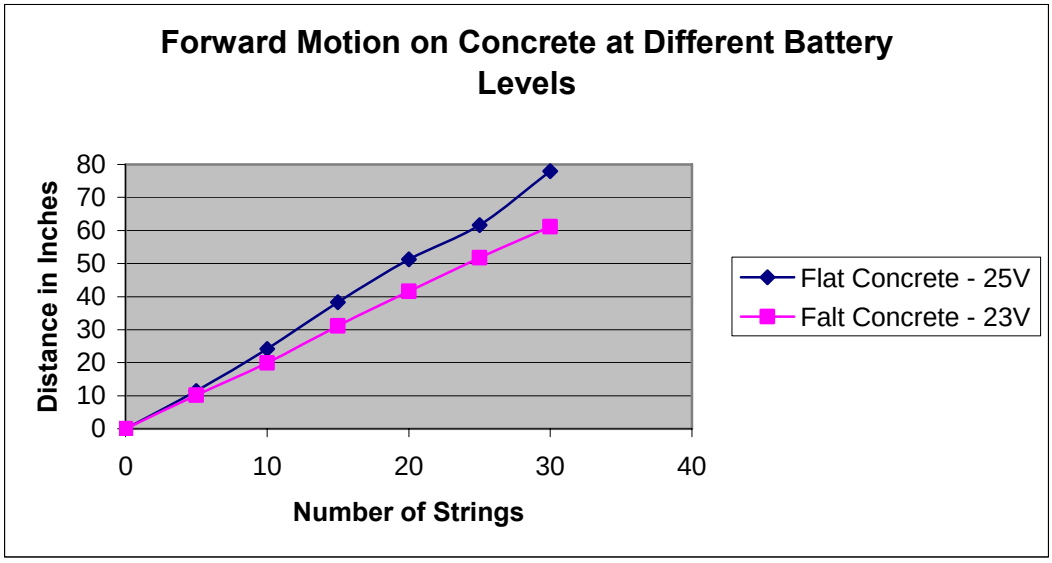


Figure 30: The difference in performance on concrete and on carpet is a considerable 13%. The bottom picture shows a slight difference between the forward and backward motion, not noticeable in the top picture.



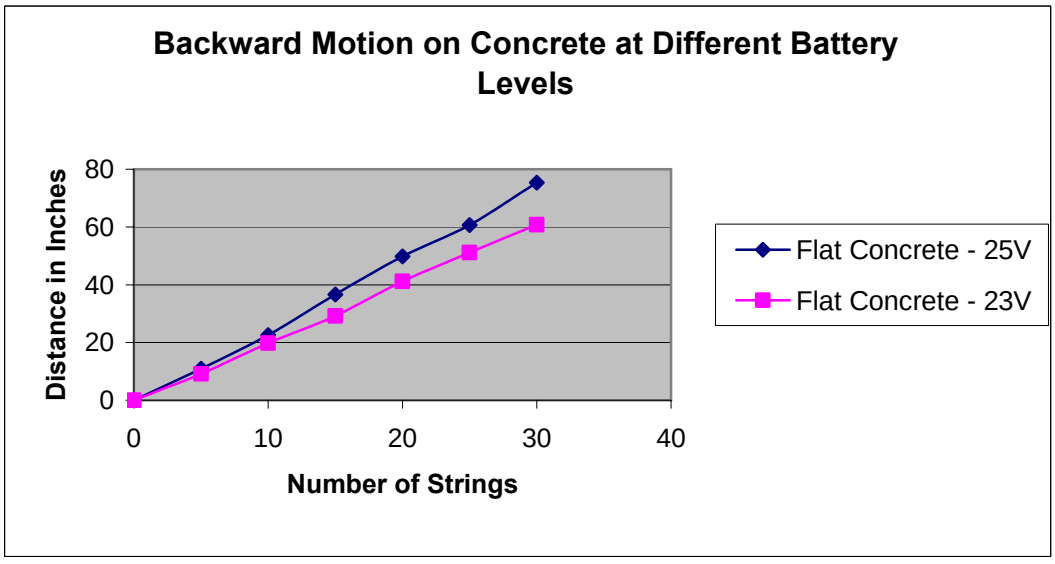
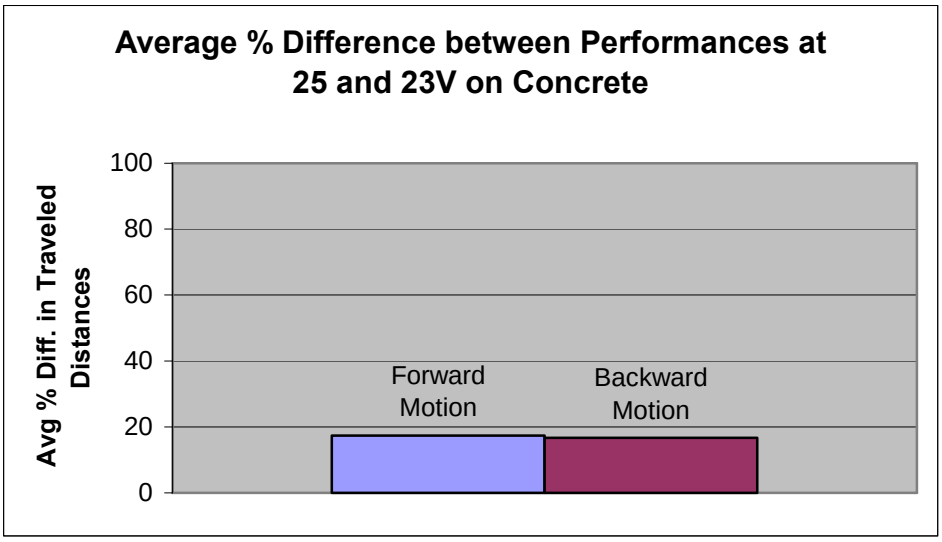


Figure 31: The effect of a low supply voltage is even more visible than that of the type of floor.



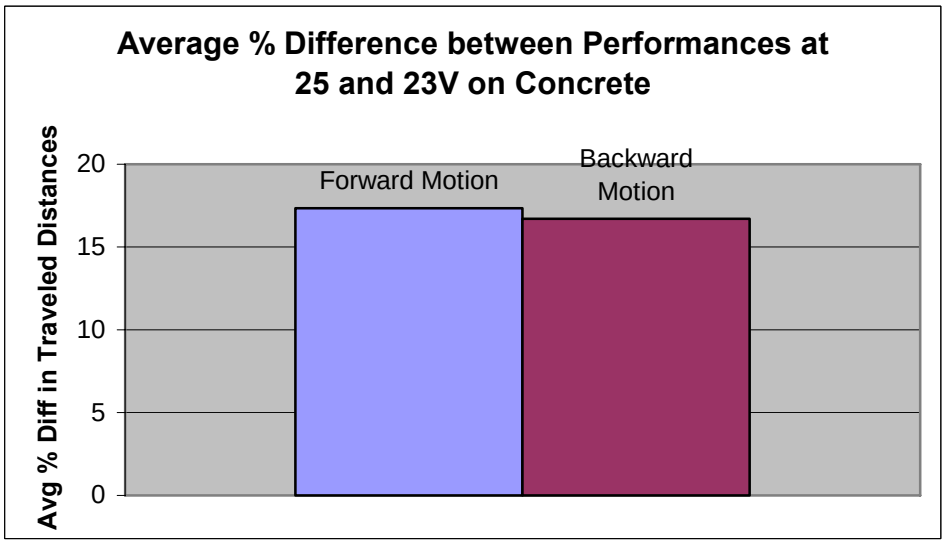


Figure 32: The effect of a low battery is roughly 17% in both directions.

5.1.2.1 Description of the experiment

The principle for collecting data with the turning motion is the same as for the forward and backward motion except that set-squared protractor replaces the tape measurer. Figure 33 shows pictures of the process on concrete and on carpet.

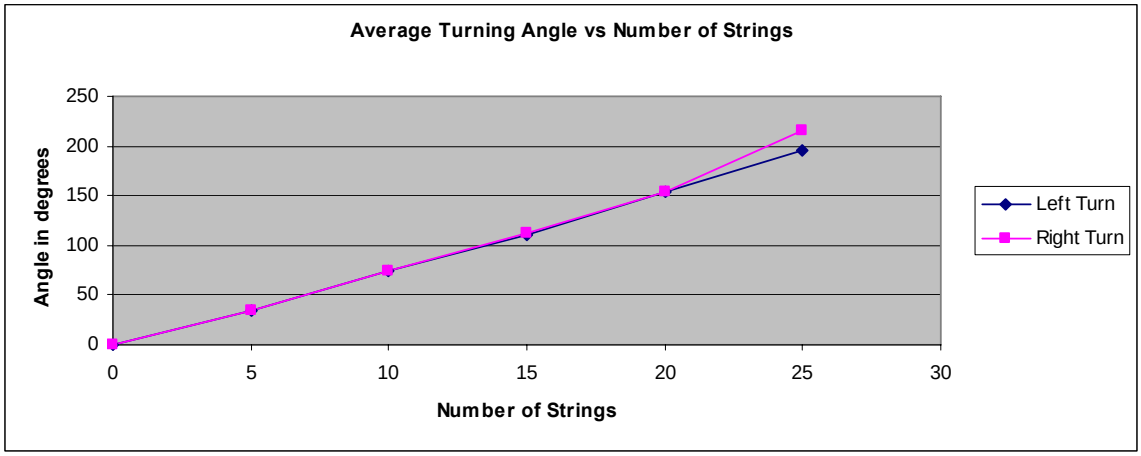
5.1.2.2 General Observations

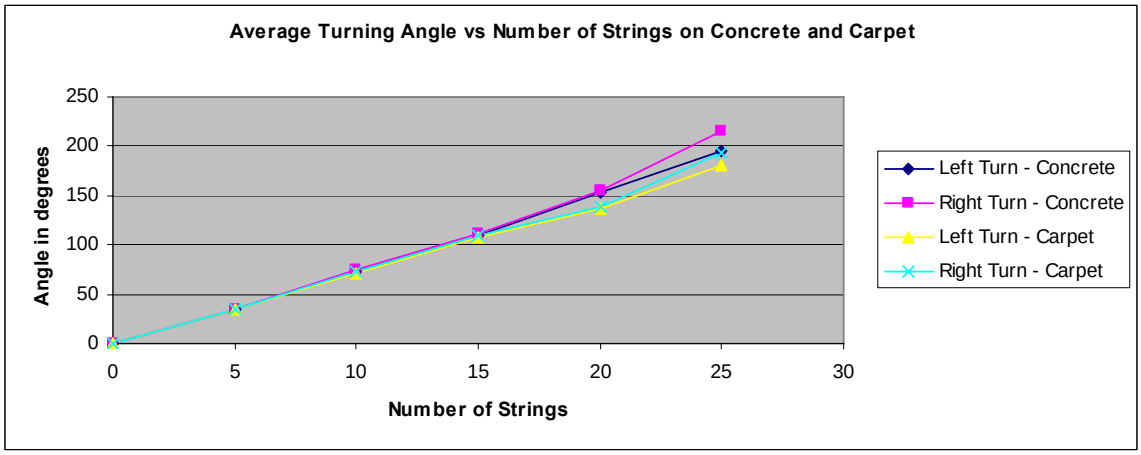
The analysis was the same as for straight motions. The protractor measures angles with a precision of half a degree. Measurements of motion for each amount of strings yielded the same angle except for the first data point in each set. Therefore there was little or no variation between the ten trials' angles. Figure 34 shows a more symmetric system in terms or turning than in straight motion. The 4% distance difference in forward and backward motion translates into a difference in left and right turning angles of less than 2% except for the 25-string commands. Figure 34 shows the results on concrete and carpet as well as the same measurements at 25 and 23V. These graphs reinforce the conclusions made in the forward and backward motions. While the effect of a high and friction floor is noticeable, the impact of the battery level is more significant and reaches peaks of 30% in turning motion. The turning motion appears more linear and predicable below 15-string commands. Below that value the difference in performance on different floors is also barely noticeable (less than 5%).

5.1.3 Summary

This evaluation shows that the string commands do not allow accurately predicting the robot's motion. The amount of friction decreases the forward and traveled distance by approximately 10%. This percentage translates in a lower value in terms of turning motions. The previous observation is valid on a full battery. A decrease in power supply voltage considerably slows the robot down – an average of 17%. The combined effect of a low battery and a high friction floor causes the robot not to be functional. This is a corroboration of an expected phenomenon. Robots' tracks provide a very good grip on the floor. This is a desired feature especially in robot such as the ANDROS, which is capable of climbing stairs and navigating through rough terrains. However it takes more power to drive the robot on tracks. On a low battery the robot did not drive properly on carpet. Hence the improvement in newer version of ANDROS robot: removable tires. The study still provides the programmer with useful data when choosing the motion steps for autonomous navigation particularly on lower friction grounds. The robot is more stable and predictable when it receives more than 5-string commands. However, as the number of strings increases the asymmetry of the motion, the effect of the floors' friction and the battery level become more important.

Figure 33: The pictures on the left show the methodology in measuring the covered angle for the same number of strings. The pictures on the right are close up images of the same process on both types of floors.





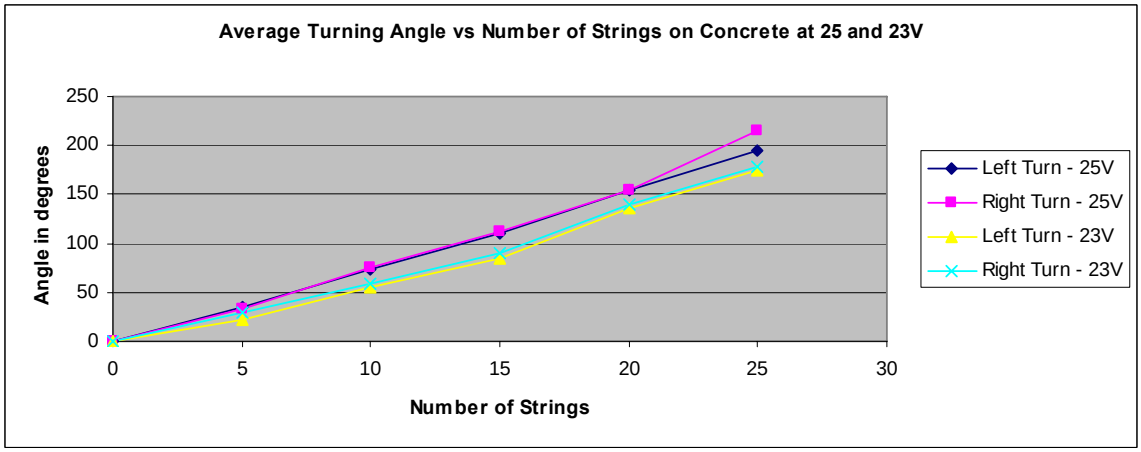


Figure 34: The turning motion reinforces the conclusion made in the forward and backward motions.

5.2 Characterization of the Algorithm

This section provides information about the wall following algorithm. Once launched, the robot autonomously follows the closed loop described in Figure 35 until stopped by the operator or a sudden close obstacle.

There were two main development from the algorithm described in Chapter 4. The first attempt to enhance the algorithm through the calibration process was not conclusive. The second attempt to do so though the turning process returned better results.

5.2.1 Calibration Process

This process describes the robot's alignment to the wall. The variable d determines how parallel the robot is to the wall. Ideally the left and right projections of the wall onto the y-axis would yield a d equal to 0. The wall following algorithm does not require such accuracy. To have an idea of how the number of moves needed to be parallel to the wall varies with d, Figure 36 displays the result of a calibration experiment. Since there is a correlation between the number of moves (1 string for small stepping angles) and the angle, measurements will be done using the same angle of 45 degrees. The trend shown in Figure 36 represents the average of 5 data points for each d. A computer program consistently and accurately displays the number of moves to reach d and the only error involved in this process is the operator's accuracy in placing the robot at the correct angle. The variation in moves is never more than 2. During each experiment a clock also chronometers the calibration time. The average time per move is 2 seconds. The variable d varies from 10 to 50 cm, which is 50% of the minimal distance between the wall and the robot. In other words if the robot is one meter from the wall, a half meter error can still be tolerated before reaching the obstacle threshold. Just as expected Figure 36 shows that as d increases the number of moves decreases with a 40% reduction as d changes from 10 to 50 cm. This different is roughly equivalent to a 0 to 20 degree-angle difference in directions between the robot and the wall.

The basic version of the wall following function takes accurate data from the sensor. In other words, the robot waits on exact position feedback from the range sensor before aligning itself to the wall or taking a turn. Several experiments showed that the average time the robot took to do one single loop was nearly 4 minutes with the lowest number of moves. In a first attempt to speed this process up, the robot stopped checking on accurate sensor data during the calibration. Table 2 shows the resulting time (in min) for the original program and the fast calibration code necessary to go around the testing scene once. This experiment was repeated 20 times; 10 measurements are shown in Table 2.

Figure 35: The three pictures show different shots of the testing set up.

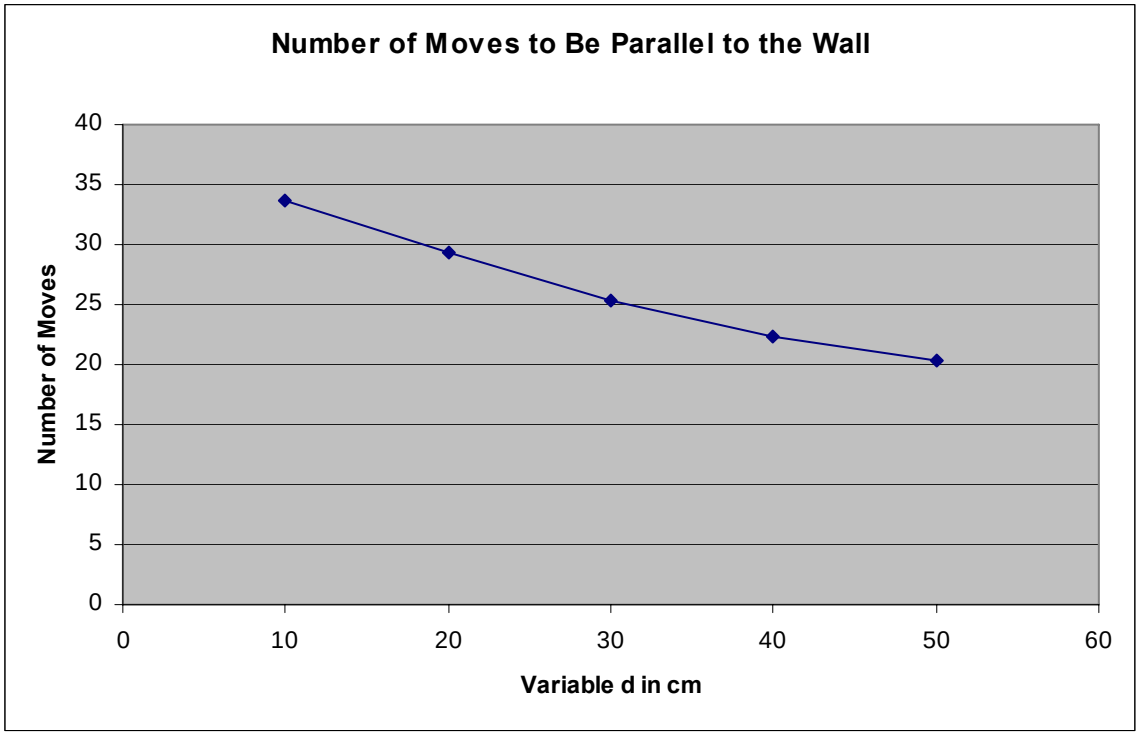


Figure 36: The number of moves to be parallel to the wall decreases as d reaches 50 cm.

Table 2: The time comparison between the original algorithm and an accelerated calibration program shows a problem with the new version of the code.

Trial Number	1	2	3	4	5	6	7	8	9	10
Basic Algorithm	4	3.6	4.33	4.5	3.8	4	3.75	4.125	4	3.9
Fast Alignment	3	2.5	6	∞	2.5	∞	4	∞	3	∞

Table 3: The time comparison between the original algorithm and an accelerated turn program shows a nearly 50% reduction in time per loop.

Trial Number	1	2	3	4	5	6	7	8	9	10
Basic Algorithm	4	3.6	4.33	4.5	3.8	4	3.75	4.125	4	3.9
Fast Turn	2	2.5	2.5	2	2.5	2.5	3	2.5	2.5	2

The accelerated calibration first seems to show promising results as the robot took less time to complete a loop with this algorithm. However sometimes it took more time, as it remained stuck in the calibration process. In other words it would continuously turn left and right without going forward. In certain cases it would eventually get out of the situation. An infinite time indicates that the robot did not get out of the calibration process unless moved by the operator.

This experiment emphasized the importance of acquiring correct data during the calibration process even at the cost of time.

5.2.2 Turning Process

Since accelerating the calibration process did not successfully produce a better algorithm, the second development in the wall following function is an attempt to accelerate the turning process. The results of this second attempt to improve the ANDIBot's performance are recorded in Table 3. Not having the correct information during calibration causes the algorithm to be unreliable and unpredictable. However not knowing the accurate information during the turning process only causes the robot to over turn; the robot receives several command strings before checking back on the sensor information. This solution turned out to be a better idea and consistently improved the robot's performance by nearly 50% of the earlier time per loop.

After each over correction, the ANDIBot has to come back towards the wall. Even so, the time saved by the fast turning considerably outweighs the time spent readjusting to the correct path.

While the first section of the analysis helps characterize the string commands, this second section describes the two key processes involved in the wall following algorithm. The study of those mechanisms has help improve the algorithm, specifically the amount of time necessary to go around one loop. The next chapter will conclude this work and propose further possible enhancements.

6 Conclusion

6.1 Summary

This research has successfully reverse engineered the telecommunications of the ANDROS Mark VA robot. This knowledge has been used to create a computer-interfaced Control Unit. Controlling the robot using a computer has allowed the integration of the new mobile platform in the modular Sensor Brick Concept as a mobility brick. The integration of sensor bricks has upgraded the original teleoperation to a more flexible system. The ability to wirelessly be connected to different modular sensors makes the robot more adaptable to different scenarios of operation. Furthermore it allows acquiring autonomous capability. This autonomous behavior acquired in such a way is not restricted to one application; instead it varies with the type and number of sensor bricks used.

A particular application of a wall following algorithm has been developed to illustrate that idea. The mobility brick uses a range sensor brick to continuously follow a wall. The resulting system has been tested and analyzed. The string commands sent to the robot can be used as an encoding system to determine each one of the robot's move. Since those commands are not absolute, the range sensor constantly checks on the robot's position. The main processes involved in the algorithm are the calibration process used to align the robot to the wall and the turning process. Both mechanisms have been analyzed to maximize the performance of the algorithm. Overall, this example accomplishes its goal of transforming a teleoperated robot into an autonomous robot while keeping its original features.

6.2 Future Work

The current system is able to continuously follow a wall. In this experiment the wall is continuous. Improving this algorithm would allow the robot to detect and enter doors. One major challenge remains in following a fence instead of a wall. Another enhancement could be that the robot finds a path between the wall and obstacle that might be on its right. In other words enhancing this algorithm would imply covering all the possible situations the robot might encounters while covering a perimeter. Furthermore the ground in this experiment is a flat concrete ground; a slight progress could be of taking the robot outdoors in a real environment and testing its performance. This future work generally consists in increasing the complexity of the robot's environment.

So far the algorithm is very much local and the robot constantly faces a new situation and takes the corresponding decision. There is no mapping involved. The robot strings evaluation allows making longer and more precise moves and could be used as an encoder. Evidently this method would be less reliable as it depends on battery level. However there are several applications such as the one implemented in this work that do not require as much accuracy as explosive disposal operation for example. This would allow programmer to consider implementing path-planning algorithms on this system.

At the moment the system can receive messages from one computer at the time. The wireless connection is manually established between the server and the client. While the client does not consider where the commands are coming from, the controller only sends information to one single client whose IP address is coded in the program. This method can be extended to broadcasting to several mobility bricks from one single control unit.

ANDIBot is an autonomy capable robot. One autonomous function has been implemented in the wall following algorithm. The key in implementing this algorithm has been to be able to use the captured and analyzed information from the ANDROS in order to control the robot. The mobility class in the program provides access the drive commands of the robot. Therefore several more applications can be implemented and added onto the ANDIBot. For example adding the visual, thermal or nuclear bricks to the range scanner would allow various autonomous data collection missions. The range sensor does not have to be used; visual homing could be implemented using the visual sensor brick and appropriately reusing the mobility brick.

References

[1] J. W. Crandall and M. A. Goodrich, "Experiments in Adjustable Autonomy," in *Proceedings of the Intl. Conference on Systems, Man, and Cybernetics*, 1624-1629, 2001.

[2] J. W. Crandall and M. A. Goodrich, "Characterizing Efficiency of Human Robot Interaction: A Case Study of Shared-Control Teleoperation," in *Procedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Systems* EPFL, Lausanne Switzerland. October 2002.

[3] M. Wilson and M. Neal, "Diminishing Returns of Engineering Effort in Telerobotic Systems." *IEEE Transactions on Systems, Man, and Cybernetics* – Part A: Systems and Humans, 31(5): 459-465, September 2001.

[4] Operation and Maintenance Manual for ANDROS Mark VA Robot System, Remotec Inc., Clinton, TN, 2001.

[5] Tom Wilson, "A comparison or the Sensor Brick Concept as a Modular System Architecture to the Realtime Control System as the Operational Architecture," M.S. thesis, University of Tennessee, Knoxville, TN, USA, 2005.

[6] H. Hagras and T. Sobh, "Intelligent Learning and Control of Autonomous Robotic Agents Operating in Unstructured Environments," University of Bridgeport, CT, USA, Tech. Rep. 2002.

[7] N. Kasabov and M. Watts, "Neuro-Genetic Information Processing for Optimisation and Adaptation in Intelligent Systems," in *Neuro-Fuzzy Techniques for Intelligent Information Processing*, N. Kasabov and R. Kozma, Heidelberg Physica Verlag 1999, pp. 97-110.

[8] I. Horswill, "Tagged Behavior-Based Systems: Integrating Cognition with Embodied Activity," *IEEE Intelligent Systems*, pp 30-37. September/October 2001.

[9] A. A. D. Medeiros, "A survey of control architectures for autonomous mobile robots," *Journal of the Brazilian Computer Society*, vol. 4, No. 3, Campinas Abr. 1998.

[10] J. S. Albus, H. G. McCain and R. Lumia, "NASA/NBS standard reference model for tele-robot control system architecture (NASREM)," Tech. Rep. 1235, National Institue of Standards and Technology, 1989.

[11] J. S. Albus, "Outline of a Theory of Intelligence," *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):473-509, May 1991.

[12] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, RA-2(1):14-23, March 1986.

[13] R.G. Simmons, "Concurrent Planning and Execution for Autonomous Robots," *IEEE Control Systems Magazine*, 12(1): 46-50, 1992.

[14] R.G. Simmons, "Monitoring and Error Recovery for Autonomous Walking," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, Raleigh, USA, July 1992.

[15] R. Alami, R. Chatila, S. Fleury, M. Ghallab and F.F. Ingrand, "An Architecture for Autonomy," in IJRR, Special Issue on "Integrated Architectures for Robot Control and Programming," Spring 1998.

[16] L. Steels, "When are Robots Intelligent Autonomous Agents?" *Journal of Robotics and Autonomous Systems*, vol. 15, 1995, pp. 3-9.

[17] M. Wilson and M. Neal, "Diminishing Returns of Engineering Effort in Telerobotics," *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol.31, No. 5, September 2001.

[18] L. Conway, R. A. Volz, and M. W. Walker, "Teleautonomous systems: *Protecting* and coordinating intelligent action at a distance," *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 146–158, Apr. 1990.

[19] N. E. Sian, K. Yokoi, S. Kajita and K. Tanie, "Whole Body Teleopeation of a Humanoid Robot Integrating Operator's Intention and Robot's Autonomy," in *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada. October 2003.

[20] E. Sian, K. Yokoi, S. Kajita, H. Saito and K. Tanie, "A Stable Foot Teleoperation Method for Humanoid Robots," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*. New Orleans, LA. April 2004.

[21] I. Horswill, "Visual Routines and Visual Search: A Real-Time Implementation and an Automata-Theoretic Analysis," in *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Francisco, 1995.

[22] R. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, 1986, pp. 14–23.

[23] M. Yeasin and S. Chaudhuri, "Toward Automatic Robot Programming: Learning Human Skill from Visual Data," *IEEE Transactions on Systems, Man, and*

Cybernetics – Part B: Cybernetics, vol. 30, No. 1, February 2000.

[24] A. Ude, "Trajectory generation from noisy positions of object features for teaching robots robot paths," in *Robotics and Autonomous Systems*, vol. 11, No. 2, pp. 113–127, 1993.

[25] Y. Kuniyoshi, H. Inoue, and M. Ibana, "Learning by watching: Extracting reusable task knowledge from visual observation from human performance," *IEEE Transaction on Robotics and Automation, v*ol. 10, pp. 799–822, June 1994.

[26] A. Billard and M. J. Matari, "Learning human arm movements by imitation: evaluation of a biologically inspired connectionist architecture," in *Robotics and Autonomous Systems* 941, 2001, pp 1-16.

[27] J. Aleotti, S. Caselli and M. Reggiani, "Leveraging on a Virtual Environment for Robot Programming by Demonstration," appears in: IROS 2003 Workshop on Robot Programming by Demonstration. Las Vegas, Nevada. October 2003.

[28] S. Bottazzi, S. Caselli, M. Reggiani, and M. Amoretti, "A Software Framework based on Real-Time CORBA for Telerobotic Systems," in *IEEE International Conference on Intelligent Robots and Systems*, 2002.

[29] D. Rus and M. Vona, "A Basis for Self-reconfiguring Robots using Crystal Modules," in *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2000.

[30] D. Rus and M. Vona, "Self-Reconfiguration Planning with Unit Compressible Modules," in *Proceedings of the1999 IEEE International Conference on Robotics and Automation,* pp 2513-2520, Detroit, MI, 1999.

[31] D.Rus and M. Vona, "A Physical Implementation of the Crystalline Robot," submitted to the 2000 *IEEE International Conference on Robotics and Automation,* San Francisco, CA, 2000.

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

[33] K. Tomita, S. Murata, E. Yoshida, H. Kurokawa, and S. Kokaji, "Reconfiguration method for a distributed mechanical system," in *Distributed Autonomous Robotic Systems 2,* pp 17-25, Springer Verlag 1996.

[34] E. Yoshida, S. Murata, K. Tomita, H. Kurokawa, and S. Kokaji. Distributed

Formation Control of a Modular Mechanical System," in *Proceedings of the 1997 International Conference on Intelligent Robots and* Sy*stems,* 1997.

[35] S. Murata, H**.** Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. "A 3-D Self-Reconfigurable Structure," in *Proceedings of the 1998 IEEE International Conference on Robotics and Automation,* Leuven, 1998.

[36] A. A. Argyros, K. E. Bekris and S. C. Orphanoudakis, "Robot Homing based on Corner Tracking in a Sequence of Panoramic Images," Computer Vision and Robotics Laboratory, Institute of Computer Science Foundation for Research And Technology – Hellas (FORTH), Heraklion, Crete, Greece, Tech. Rep. 2001.

[37] P. Bizzantino, M. De Bartolomei, G. Magnani and G. Visentin, "Space Robot Autonomy Based on Distance Sensors," in *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, Leuven, Belgium. May 1998.

[38] CAT (Columbus Automation and Robotics Laboratory Testbed) developed by ESA (European Space Agency) currently installed at the Robotics and Teleoperation Laboratory of ESTEC (European Space research and Techology Centre) in Noordwijk, The Netherlands.

[39] K. L. Moore and N. Flann "A Six-Wheeled Omnidirectional Autonomous Mobile Robot", expanded version of "Hierarchical task decomposition approach to part Planning and control for an omnidirectional autonomous mobile robot", in *Proceedings of 1999 IEEE International Symposium on Intelligent Control/Intelligent Systems and Semiotics*, Cambridge, MA, September 1999.

[40] E. Poulson, J. Jacob, B. Gunderson, and B. Abbot, "Design of a robotic vehicle with self-contained intelligent wheels," in *Proceedings of SPIE Conference in Robotic and Semi-Robotic Ground Vehicle Technology*, vol. 3366, Orlando, FL, pp. 68-73, April 15-16, 1998.

[41] C. Wood, M. Daidson, S. Rich, J. Keller, and R. Maxfield, "T2 omnidirectional vehicle mechanism design," in *Proceedings of the SPIE Conference in Mobile Robots XIV*, Boston, MA, pp. 69-76, September 1999.

[42] M. Davidson and C. Wood, "Utah State University's T2 ODV Mobility Analysis," in *Proceedings of the SPIE Conference on Unmanned Ground Vehicle Technology*, vol. 4024-12, pp. 96-105, Orlando, FL, April, 2000.

[43] S. Rich, J. Keller, and C. Wood, "ODV mobility enhancement using active height control," in *Proceedings of the SPIE Conference on Unmanned Ground Vehicle*

Technology, vol. 4024-16, pp. 137-145, Orlando, FL, April, 2000.

[44] H. McGowen, "Navy omnidirectional vehicle (ODV) development and technology transfer opportunities," Coastal System Station, Dahlgren Division, Naval Surface Warfare Division, Tech. Rep.

[45] M. Asama, M. Sato, H. Kaetsu, K. Osaki, A. Matsumoto, and I. Endo, "Development of an omnidirectional mobile robot with 3 DOF decoupling drive mechanism," *Journal of the Robotic Society of Japan*, vol. 14 no.2, pp.95-100, 1997 (in Japanese).

[46] A. Mutambara and H. Durrant-Whyte, "Estimation and control for a modular wheeled mobile robot," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 1, pp. 35-46, January 2000.

[47] A. Rodic and M. Vukobravotic, "Contribution to integrated control synthesis of road vehicles," *IEEE Transactions on Control System Technology*, vol. 7no. 1, pp. 64-78, January 1999.

[48] R. Colyer and J. Economou, "Comparison of steering geometries for multi-wheeled vehicles by modeling and simulation," in *Proceeding of the 37th IEEE Conference on Decision and Control*, pp. 3131-3133, Tampa, FL, December 1998.

[49] J. Economou and R. Colyer, "Modeling of skid steering and fuzzy logic vehicle ground interaction," in *Proceedings of 2000 American Control Conference*, pp. 100-104, Chicago, IL, June 2000.

[50] Y. Sun, N. Xi and Y. Wang, "Modeling and Analysis of Perceptive Robot Controller Based on Hybrid Automata," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA. April 2004.

[51] J. Lygeros, K. H. Johansson,S. N. Simic, J. Zhang, and S. S. Sastry, "Dynamical Properties of Hybrid Automata," *IEEE Transactions on Automatic Control* vol.48, No.1, January 2003.

[52] H.Ye,A.N.Michel and L.Hou, "Stability Theory for Hybrid Dynamical System," *IEEE Transactions On Automatic Control,* vol.43,No.43, April, 1998.

[53] S. Pettersson and B. Lennartson, Controller Design of Hybrid Systems, "Hybrid Systems," vol. 1201 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1993.

[54] R. W. Brockett, "Hybrid Model for Motion Control Systems," in *Perspectives in*

Control, Eds. H. Trantelman and J.C.Willems, pp.29-54, Birkhauser, Boston, 1993.

[55] B.A. Cartwright and T.S. Collett, "Landmark Maps for Honeybees," *Biology Cybernetics*, 57, 85-93, 1987.

[56] T.S. Collett, "Insect Navigation *en Route* to the Goal: Multiple Strategies for the Use of Landmarks," *The Journal of Experimental Biology*, 199, 227-235, 1996.

[57] M.V. Srinivasan, S.W. Zhang, M. Lehrer and T. S. Collett, "Honeybees Navigation *en Route* to the Goal: Visual Flight Control and Odometry," *The Journal of Experimental Biology*, 199, 273-244, 1996.

[58] D. Lambrinos, R. Moller, T. Labhart, R. Pfeifer and R. Whener, "A Mobile Robot Employing Insect Strategies for Navigation," *Robotics and Autonomous systems*, 30, 39-64, 2000.

[59] M. Franz, B. Schilkopf, H. Mallot and H. Bulthoff, "Where Did I Take That Snapshot? Scene-based Homing by Image Matching," *Biological Cybernetics*, 79, 191-202, 1998.

[60] R. Moller, "Insect Visual Homing Strategies in a Robot with Analog Processing," *Biological Cybernetics*, special issue: Navigation in Biological and Artificial Systems, Vol. 83, No. 3, 231-243, 2000.

[61] R. Cassinis, A. Rizzi, G. Bianco, N. Adami and P. Mosna, " A Biologically-Inspired Visual Homing Method for Robots," *Workshop AIIA-IAPR-IC*, Ferrara, 1998.

[62] A. B. Koku, A. Sekmen and D. M. Wilkes, "A Novel Approach to Robot Homing," in *Proceedings of 2003 IEEE Conference on Control Applications*, vol. 2, pp. 1477 -1482.

[63] T. S. Levitt and D. T. Lawton, "Qualitative navigation for mobile robots," Al, 44: 305 -306, 1990.

[64] P. Gaussier, C. Joulian, S. Zrehen, J.P. Banquet and A. Revel. "Visual navigation in an open environment without a map," in *IEEE International Conference on Intelligent Robots and Systems*, 545-550, 1997.

[65] B. Pinette, "Qualitative homing," *in Proceedings of IEEE International Symposium on Intelligent Control*, 318-323, 1991.

[66] R. Moller, D. Lambrinos, R. Pfeifer, T. Labhart and R. Whener, "Modeling and navigation with autonomous agents," in *5th International Conference on Simulation*

of Adaptive Behavior, 185-195, 1998.

[67] D. Dai and D. T. Lawton, "Range-free qualitative navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 783-790, 1993.

[68] J. Ebken, M. Bruch and J. Lum. "Applying Unmanned Ground Vehicle Technologies to Unmanned Surface Vehicles," SPAWAR Systems Center, San Diego, CA. Tech. Rep. 2005.

[69] C. D. Crane III, D. G. Armstrong Jr., M. W. Torrie and S. A. Gray, "Autonomous Ground Vehicle Technology Applied to the DARPA Grand Challenge," *ICCAS* 2004.

[70] JAUS Working Group, 2002, "Joint Architecture for Unmanned Systems (JAUS): Reference Architecture Specification", Version 3.0, Volume 2, The Joint Architecture for Unmanned Systems, http://www.jaugs.org, Feb 2003.

[71] M. Huff, R. Barreto, M. Dhikiri, J. Miltenberger, Q. Obitayo, B. Dhillon, J. Taylor and T. Wilson, "Remotec Control Panel Upgrade," Tech. Rep. 2003.

[72] iRobot Incorporated, (2005). http://www.irobot.com/governmentindustrial.

[73] N. Naik, "Infrared Imaging Sensor Brick for Modular Robotics," M.S. thesis, University of Tennessee, Knoxville, TN, 2004.

Appendices

Appendice A: Mark VA Repair

During the testing of the ANDIBot system the ANDROS robot broke down. The problem was first noticed when the robot would not make a turn in either direction. The tracks appeared to be turning but the robot would only go forward and backward. To start the debugging process the robot was lifted up onto a table. The ANDROS was then positioned on two pieces of wood placed between the tracks so that they could spin freely. This set up allowed noticing that only one of the tracks was actually rotating, giving a significant clue as to why the robot would only go two directions. With the robot turned off, the broken track – right side – allowed a person to easily rotate it manually, which explained why it appeared to be working on the ground. It was simply being dragged by the left drive motor.

Those two simple steps seemed to indicate a problem with the driving DC motor for that side. Figure 37 shows the wiring diagram of the DC Shunt Motor. It differs from the series motor in that the field winding is connected in parallel with the armature instead of in series. When the power is turned on, the high resistance of the shunt coil keeps the current flowing in the main outer loop. The armature draws current to produce a magnetic field strong enough to cause the armature shaft to start turning. Once the armature begins to turn, it produces back EMF which in turn causes the current in the armature to start decreasing. The amount of current the armature will draw is directly proportional to the size of the load when the motor reaches constant full speed. Without the stat current however the magnetic field is non existent and the shaft does not turn. Therefore one should not be able to freely rotate the tracks when the power is off.

The first suspicion was then that the right drive motor was broken and needed to be replaced. However once the robot was opened for further debugging as sown in Figure 38 that same motor was visibly rotating. The problem had to be the mechanical connection between the motor and the right track. Indeed there is a metal shaft connecting the two which teeth had been stripped over time. To confirm that the problem had been identified the *hub* (see Figure 39) from the left side was placed on the right side and the track was properly driven by the motor when the robot was turned on. Finally a new metal piece was ordered from Remotec and machined by Doug Warren at the IRIS lab.

Once a piece has failed, the repairing task does not stop at isolating the defective part but goes in determining what happened and how to prevent the same problem from occurring again. There are several reasons that the erosion happened on the right side and not the left track. The metal on that side could have been defective; the right side motor being further away from the center of gravity of the robot could have experienced more vibrations than the other side. Those reasons combined with time could have caused the stripping of the teeth. However in this particular case, there was one clear problem with

the part, only the top half of the splines was flattened. It seemed as the contact between the shaft and the drive collar was not maximized. Some careful measurements determined that the original part was incorrectly sized – cut too short – eventually causing failure. Both parts are shown in Figure 39 below the two parts they are supposed to connect.

Since the place has been successfully isolated, debugged and corrected, the ANDROS is fully functional.

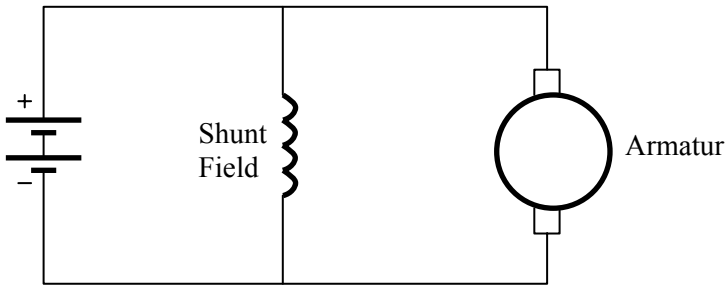


Figure 37: This image shows the wiring diagram for a DC shunt motor.

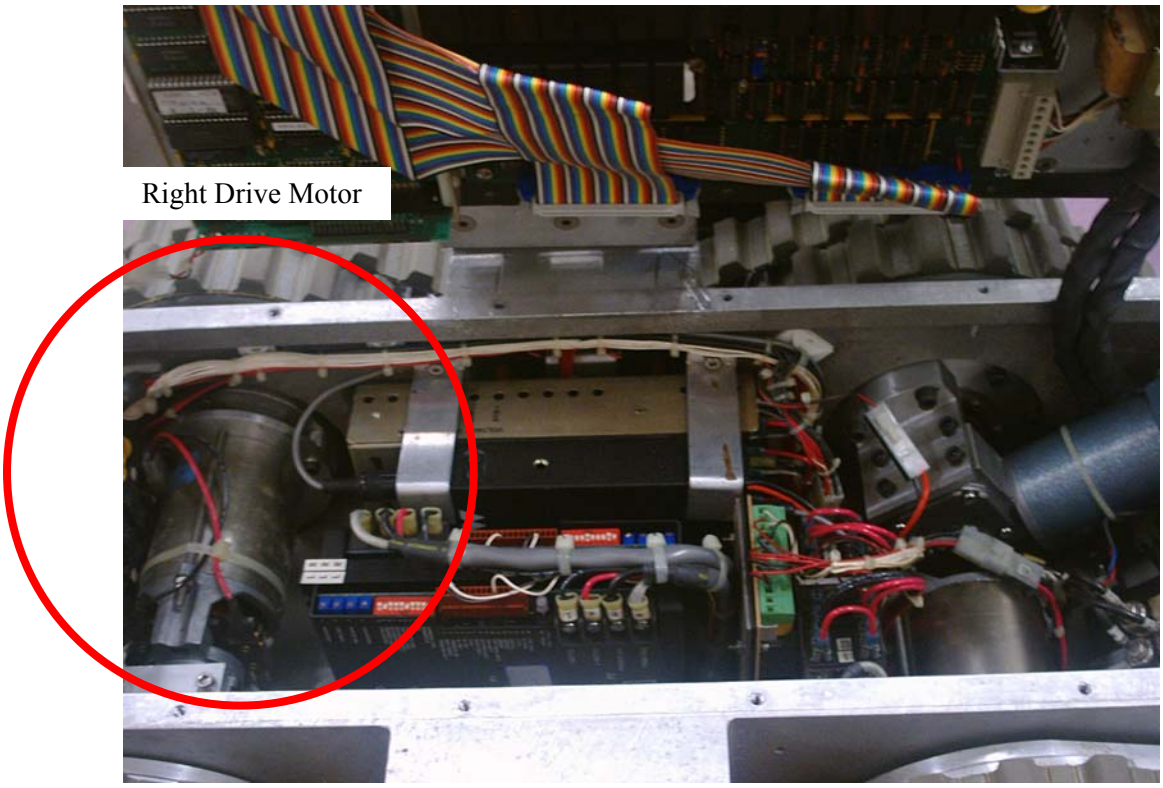


Figure 38: The debugging process showed that the motor was rotating and suggested that the problem was mechanical not electrical.

Figure 39: A closer look to the mechanical shaft (to the right) connecting the drive motor to the hub showed that the original part has been cut too short.

Appendix B: ANDROS F6A

This appendix describes the work done on another Remotec ANDROS robot. The same work was done as in the case of the Mark VA with exceptions that are covered in this section. The goal of this research is to help reproduce the accomplishments on the Mark VA on a newer version of the ANDROS.

1 Overview

In the summer of 2004, the IRIS lab acquired a newer version of the ANDROS, the F6A shown in Figure 40. The improvements include:

- A much faster baud rate (9600 vs. 1200)
- A 6 inches arm extension capability. This feature allows more delicate operation of the arm
- 4 removable wheels. This feature allows the robot to adapt easier to different terrains. While the tracks provide a strong grip to the floor for operation such as climbing stairs, they can make turning operation difficult on rough terrains. They also require more power than the tires
- An additional camera between the robot's tracks allows the operator to avoid collision with obstacle that might not be visible on the surveillance camera
- A more intuitive control system with a drawing of the robot
- Finally the new robot incorporates a few more sophisticated functions such as night vision and better graphics

This robot initially came with a fiber optic cable and was later replaced by a wireless system. This is a great improvement since the operator had to constantly keep track of the cable so that it does not get caught in the robot's tracks. However this wireless system only works with the native controller. Therefore to make a second mobility brick out of this robot would still require reverse engineering and transition to another wireless system such as 802.11g to be able to communicate with a computer.

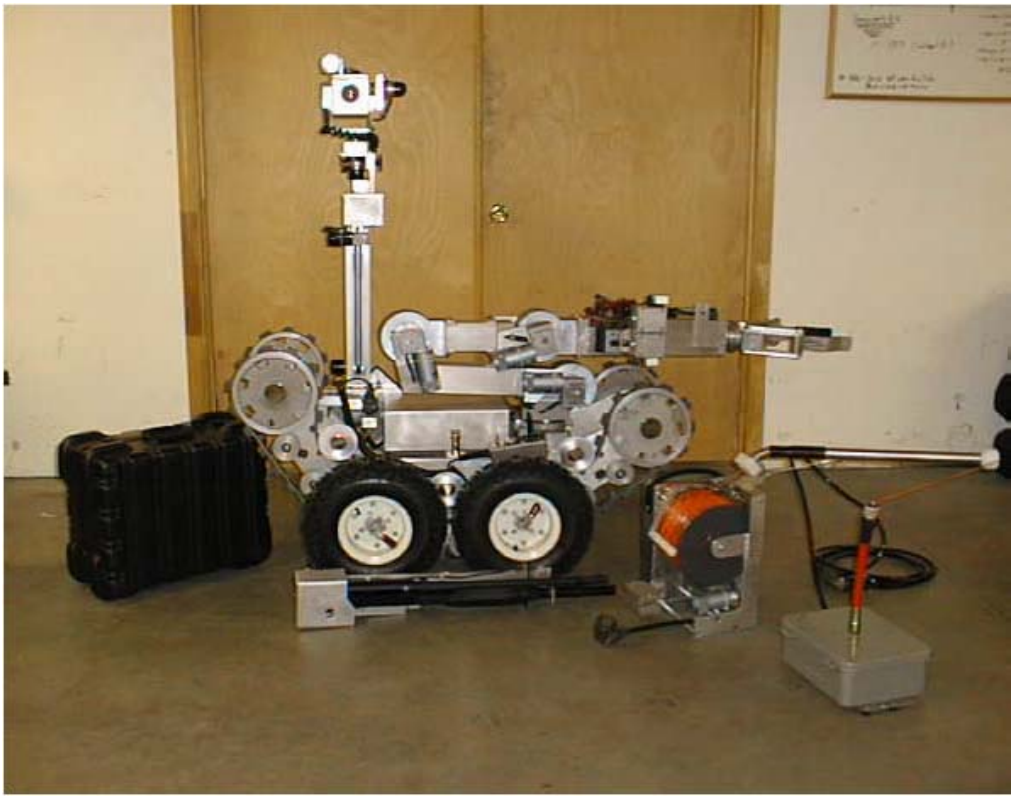


Figure 40: The ANDROS F6A includes new features and accessories.

2 Mapping

The mapping for the F6A starts as the one for the Mark VA which was decoded in the fall of 2003. Its objective is to capture the RS 232 signals emitted from the initial control box using Kermit 95.

The new OCU comes with a Wire Diagram. In an attempt to access the body motion and drive motions certain connections are not relevant. In other words there is not yet an interest for the Radio Power, the Audio Controller, the Video out etc. There are only two wire connections that could potentially be important: the "receive" and "weapon enable". In the initial mapping however only the "transmit" and "ground" lines are connected to the "receive" and "ground" lines from an RS 232 cable and plugged into the COM port of the computer.

This first KERMIT session on the new robot confirmed two improvements from the previous system. First, the session, captured at 9600 bauds, confirmed a faster baud rate. Second, the data strings were longer as expected since the F6A is a newer version of the ANDROS robot and includes more functions (and therefore more characters in the command strings).

3 Results

The mapping for the F6A was easier then the original mapping of the Mark VA. Indeed knowing how the check sum works allows us to study the character that change for each body function under a single set of settings versus repeating the mapping for all the desired set of settings. The settings could then be revised separately. Later, when the same body motion characters are changed under different settings the programmer can just change the corresponding settings characters and accurately predict the ASCII character.

Hence, the body functions and vehicle drive functions were captured under the following settings: Light OFF, Vehicle Speed HIGH and Arm Speed HIGH.

Table 4: This table contains the character changes for body motions.

Constant Signal	ññ0A00182000907F7F3F0000FFFF02CF02ý
Front Track Up	ññ**1**A00182000907F7F3F0000FFFF02CF02þ
Front Track Down	ññ**2**A00182000907F7F3F0000FFFF02D002ê
Rear Track Up	ññ**4**A00182000907F7F3F0000FFFF02D002ì
Rear Track Down	ññ**8**A00182000907F7F3F0000FFFF02D002ð
Torso Left	ññ0A0018200**4**907F7F3F0000FFFF02D002ì
Torso Right	ññ0A0018200**8**907F7F3F0000FFFF02D002ð
Shoulder Up	ññ0A0018200**1**907F7F3F0000FFFF02D002é
Shoulder Down	ññ0A0018200**2**907F7F3F0000FFFF02CF02ÿ
Elbow Up	ññ0A001820009**4**7F7F3F0000FFFF02D002ì
Elbow Down	ññ0A001820009**8**7F7F3F0000FFFF02CF02□
Wrist Up	ññ0A001820**1**0907F7F3F0000FFFF02CF02þ
Wrist Down	ññ0A001820**2**0907F7F3F0000FFFF02CF02ÿ
Wrist Extend	ññ0A00182000907F7F3F0004FFFF02D102í
Wrist Retract	ññ0A00182000907F7F3F0008FFFF02D102ñ
Wrist Roll CW	ññ0A001820**8**0907F7F3F0000FFFF02D002ð
Wrist Roll CCW	ññ0A001820**4**0907F7F3F0000FFFF02D102í
Grip Open	ññ0A001820009**2**7F7F3F0000FFFF02D102ë
Grip Close	ññ0A001820009**1**7F7F3F0000FFFF02D102ê

Table 5: This table contains the changes for vehicle drive motions.

Left	ññ0E0018200090**FF803**F0000FFFF02D002æ
Forward and Left	ññ0E0018200090**FDF1**3F0000FFFF02D002ó
Backward and Left	ññ0A0018200090**8080**3F0000FFFF02CF02Ó
Backward	ññ0A0018200090**807B**3F0000FFFF02CF02ä
Backward and Right	ññ0A0018200090**807A**3F0000FFFF02CF02ã
Right	ññ0A00182000907**E7A**3F0000FFFF02CF02÷
Forward and Right	ññ0A00182000907**E7E**3F0000FFFF02CF02û
Forward	ññ0E001820009077**FF**3F0000FFFF02D102í

Once the body function and vehicle drive characters were mapped we identified the characters changing with 3 main settings, the same we previously used for the Mark VA GUI: the Vehicle Speed, the Arm Speed and the light.

Table 6: **This table contains the changes corresponding to the speed settings.**

Vehicle Speed	
HIGH	ññ0A00182000907F7B3F0000FFFF02D202æ
MEDIUM	ññ0200182000907F7B3F0000FFFF02D202×
LOW	ññ0200182000907F7A3F0000FFFF22D202Ø
Arm Speed	
HIGH	ññ0A00182000907F7A3F0000FFFF02CF02ø
MEDIUM	ññ0A00182000907F7A1B0000FFFF02CF02ò
LOW	ññ0A00182000907F7A000000FFFF02CF02ß

4 Programming changes

So far then the main changes between the two Remotec ANDROS robot systems seem to be the speed and length of the strings. Hence the modifications in the program are first the baud rate after opening the port and then the size of the buffer. What used to be a 21-string character (buffer size = 22) became a 35-string data. Thus the changes in the main file are fairly simple. The source file that needs the most adjustments is the program that generates the special ASCII character. This check sum obviously is calculated for more characters even though the logic is confirmed to be the same. This portion of the code did not change as much as anticipated. Indeed the new features in the F6A have been implemented to the right of the string, leaving the previous functions unchanged. For example the first character controls the front and rear tracks the same way in both robots. This element emphasizes the fact that the main code did not have to be adjusted beyond the size and speed of the data.

5 Debugging

Despite apparent similarities between the two systems, the program for the F6A did not cause a reaction from the robot during testing.

5.1 Software testing

To narrow down the possibilities of problems the debugging process starts by eliminating possible software issues. A simple and crucial test consists in making sure the PC does send out the correct data at the correct baud rate. Connecting both COM ports of the PC to each other using a null modem and K95, the programmer can confirm that the new code is sending strings at 9600 baud. A simple test further eliminates possibilities that the code might not generate the accurate check sum. Instead of generating the strings in the code, the programmer can directly copy and past a K95 captured string. This way, the data going to the robot is exactly the data coming out of the control box. A last software suspicion was that the robot might expect a constant signal in between string commands.

However adding a constant signal to the code still did not create a response from the F6A. After those simple check points, it appears to be that the problem might be a hardware problem.

5.2 Hardware testing

Possible hardware problem include wire connections, hand-shaking between the OCU and the robot and impedance matching.

Wire connection – A major hardware issue to solve is to make sure data is getting to the robot. A previous test has shown that the data is in fact being sent through the serial cable, the only question remaining is whether or not it is getting to the right inputs at the robot. Once again a wiring diagram from Remotec indicates the Receive and Ground lines on the robot. To double check the connection is a simple process: connecting only the Transmit and Ground lines from the OCU to the corresponding Receive and Ground from the robot. If the robot moves, it establishes that it is indeed receiving the data from the OCU through the anticipated inputs. This test confirmed that the right connection is established between the computer and the robot. The set up for this experiment is shown in Figure 41.

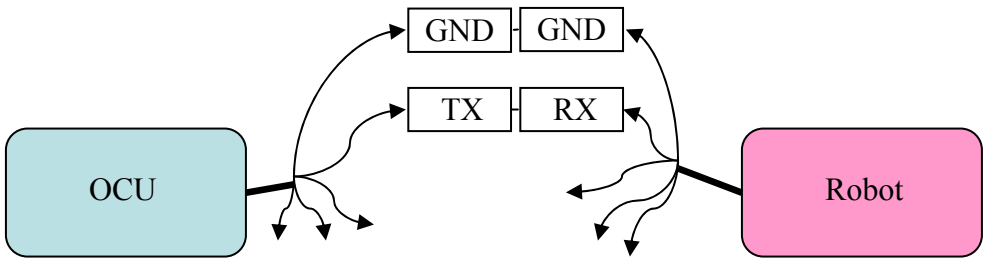


Figure 41: This testing set up is verifying the wire diagram.

Hand-shake/ Feedback – Since the original mapping data is in fact directly being sent to and received by the robot, the reason why it is still not responding is puzzling. One

thought is that the robot is able to identify the source of the strings by a hand-shake mechanism through the primary connection or through other wires. The previous experiment however has shown that only two wires are involved in the transmission of the strings. There is no feedback necessary from the robot, before it moves. The Receive line from the OCU and the Transmit lines from the robot are not connected. Neither are the other wire connections. However one needs to mention here that certain times, after performing this experiment, the F6A would be temporarily unresponsive to its own OCU. The following error would appear on the display: "*Looking for Robot communicator*". At this point the fact that there is no feedback is established; however the possibility that the robot recognizes its OCU is not excluded.

Voltage or Impedance – The previous experiments indicate that the strings are being transmitted from the computer to the robot. They are reaching the right inputs at the robot end and yet not generating motion. Therefore we can conclude that the robot does indeed notice a difference between the PC and its OCU. It seems to be a voltage, power or impedance problem. The voltage output of the new OCU is shown in Figure 42. These voltages levels are higher than a regular PC can output. This was not the case for the Mark VA and is definitely part of the problem in this case.

6 Conclusion

The F6A is a newer more sophisticated version of ANDROS Robot than the Mark VA. It offers various new features, despite keeping the same basic engineering. The only two noticeable differences from a programming point of view are the length of the strings and the baud rate. From a hardware stand point however there is a discrepancy between the computer and the new robot's OCU. This works suggest that solving this situation implies successfully matching the PC and the OCU impedances.

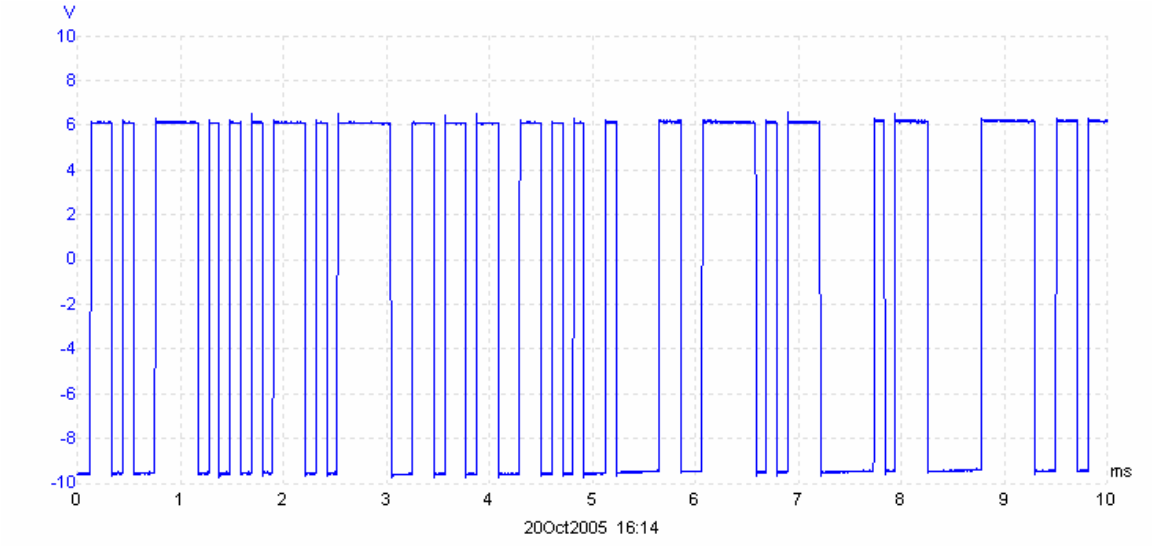


Figure 42: This signal represents the output from the F6A OCU.

Vita

Roselyne Dalanda Barreto was born on May 14, 1980, in Dakar, Senegal to Marie Madeleine Spenser and Philippe Barreto. She received the Bachelor of Science in Electrical Engineering in May 2004, from The University of Tennessee, Knoxville. She accepted a graduate research assistantship in the Imaging, Robotics and Intelligence (IRIS) Laboratory in August 2004. The completion of this Master's degree is scheduled for August 2006.