Robot Self-Configuration Using a Physical Test Harness

by

Javier Alejandro Castro

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2008

Author .....
.....
Department of Electrical Engineering and Computer Science
January 18, 2008

Certified by.....
.....
Seth Teller
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by.....
.....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Robot Self-Configuration Using a Physical Test Harness

by

Javier Alejandro Castro

Submitted to the
Department of Electrical Engineering and Computer Science

January 18, 2008

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Robot control software packages require a configuration step prior to use. The configuration requires that robot parameters such as the dimensions of the robot, the radius of its wheels, and the location of sensors in body coordinates be provided to the system. Typically, this is accomplished through manual measurement. This thesis describes a method for automating the configuration of essential robot parameters – specifically the size of the wheel radii, the dimensions of the chassis, and the location of the wheelbase with respect to the body frame – and compares the results of a preliminary configuration system for the CARMEN robot navigation toolkit to the parameters determined via user measurement. The method is able to estimate the parameters of morphologically analogous robots, for which the shape and sensor types are given, through the use of a physical test harness. The targeted family of robots consists of rectangular, two-wheeled, differential drive robots that are equipped with quadrature phase encoders and current-sensing capabilities.

Parameters are discovered by placing the robot in a known physical environment and moving it throughout the enclosed area, performing experiments from which each of the parameters can be calculated. The resulting self-configured parameters are then compared quantitatively to user-measured parameters through several methods including a complete system comparison using the University of Michigan Benchmark (UMBmark) as the standard for comparison. The results demonstrate that while the self-configured parameters do not match user-measured values perfectly, the proposed method remains an adequate technique for automating the configuration of microbot-class robots for use with robotics toolkits.

Thesis Supervisor: Seth Teller
Title: Professor of Computer Science and Engineering

Acknowledgments

Many thanks to my family, friends, advisor and peers who have all supported me through this entire process, never let me give up, and never gave up on me.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

-

Chapter 1

Introduction

Providing accurate measurements of robot parameters is typically the first task required in the use of a robot control package. Whether the purpose of the toolkit is localization and navigation or the actuation of a manipulator, the configuration of the target robot must be provided in order to provide high-level control over the underlying low-level aspects.

In general, the information that is required must be provided by the robot's user agent. This poses a problem because humans are error prone, and the configuration step is tedious. If the system is dependent on human input, the user will eventually make a mistake. Also, because the task is manual, it can degrade the user experience.

This thesis presents a method for implementing a self-configuration system for morphologically analogous robots. An example from this class of robots is shown in Figure 1-1. Configuration is achieved by performing experiments in a simple controlled environment called a physical test harness.

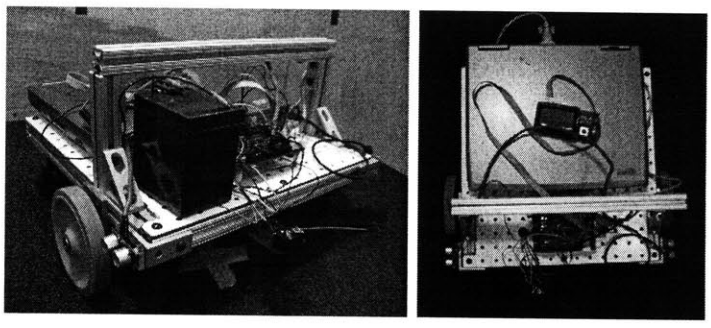


Figure 1-1: The type of robot that is used in both *Robotics: Science and Systems* and this thesis is the *microbot*. These photographs show an example microbot equipped with sonar sensors, a bump sensor, a laptop, an ORCBoard, a crossbar, wheels, and a battery.

1.1 Motivation for Self-Configuration

One of the primary reasons for developing a self-configuration system is so that it can be used with a multitude of robots with similar morphology. This scenario can be found whether experimenting with a variety of approaches for solving a particular problem, or simply over the course of a class in robotics.

Robotics: Science and Systems I (RSS), is the first of a pair of laboratory courses at the Massachusetts Institute of Technology (MIT) which teach the basics of autonomous mobile robotics. The course is targeted towards undergraduate students in their third and fourth years of study. RSS begins with the construction of a *microbot*, such as the one depicted in Figure 1-1, and covers robotics, ranging from basic dead reckoning to more advanced topics, including visually guided movement using a camera and grasping of objects using a manipulator. The course culminates in an event called the "RSS Grand Challenge", where the students are charged with a difficult robotics task, e.g., building a shelter from items that are located throughout an unknown region. At this point in the course, the students are allowed to modify the design of the original microbot so that it can perform the task better. This scenario, provided by RSS, is a prime example of a practical application for an automated robot configuration system.

More importantly, each step taken towards solving the self-configuration problem leads to incrementally better layers of abstraction. Better abstraction results in software that is portable across different robot platforms. This portability results in an increase in the amount of code that can be reused and the rate at which research can proceed.

1.2 A Method for Self-Configuration

The method presented in this thesis provides a mechanism for automating the configuration step that is required by robot control packages. The parameters which this method focuses upon are the size of the wheel radii, the dimensions of the chassis, and the location of the wheelbase with respect to the body frame.

Discovery of these four parameters provides basic information which can be utilized in the discovery of other robot parameters. Once the size of the wheels is known, the robot is able to translate at commanded velocities, allowing the robot to have some control over its location and movement. Determining the chassis dimensions is the first step towards

defining a coordinate system within which sensor locations can be mapped. The second step, determining the origin, is accomplished by finding the wheelbase location in body coordinates. In addition to defining the coordinate system, knowing the chassis dimensions allows motion planning algorithms to choose collision-free paths.

The test harness method for self-configuration works under a set of assumptions which pertain to the test harness, the toolkit being configured, and the robot itself. These requirements are described in detail in Chapter 4. To summarize these requirements, the shape and dimensions of the test harness are assumed to be known ahead of time; the robot control kit is assumed to possess basic navigation subroutines; and the morphology and available sensors of the robot are known.

The targeted robot morphology is derived from the microbot used in RSS, consisting of rectangular robots with two-wheeled, coaxial, differential drive and the ability to provide odometry information through quadrature phase encoders and current draw.

The test harness should have a length that differs from its width, and it should be sturdy enough to sustain collisions from the robot being configured.

Self-configuration begins by placing the robot within the confines of the test harness. The front of the robot must be placed as close to the center of the test harness area as possible. It should be oriented so that the direction of travel is along the length of the test harness. Once the robot has been placed in a suitable pose and the configuration process has been started, the robot moves within the test area in an attempt to experimentally determine the value of each parameter that is being configured. This is accomplished by executing predefined behaviors and calculating values from the observed events.

Acquiring the dimensions of the test harness still requires user-measured parameters, but there is an added benefit of using the configuration system rather than performing measurements by hand. Multiple robots are able to configure their own parameters at the cost of a small, one-time measurement. This is advantageous if many robots are being configured, or if one robot will be modified frequently, with each change leading to a similarly structured robot with differing configuration parameters.

1.3 Conventions and Coordinate Frames

Many different conventions are used throughout the field of robotics.This section describes them.

The family of robots targeted by this particular configuration method have a rectangular frame, and have a differential drive which consists of two coaxial wheels. References to "body coordinates" and "robot-centric coordinates" correspond to a right-handed coordinate frame, as depicted in Figure 1-2. The positive x-axis points to the right of the robot, the y-axis points to the front, the z-axis points upward and the coordinate origin is the midpoint of the rear wheelbase.

While the morphology of robots is that depicted in Figure 1-3a, the rendering in Figure 1-3b will be used in sketches in order to clearly denote which end is the 'front' of the robot.

In this thesis, all distances are measured in meters, time in seconds, and angles in radians.

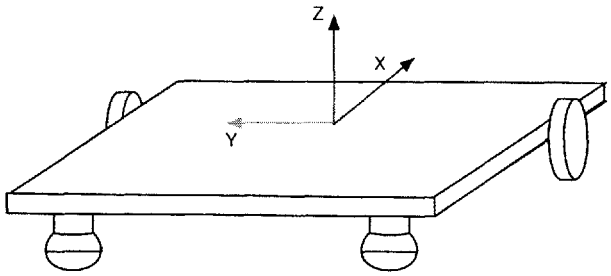
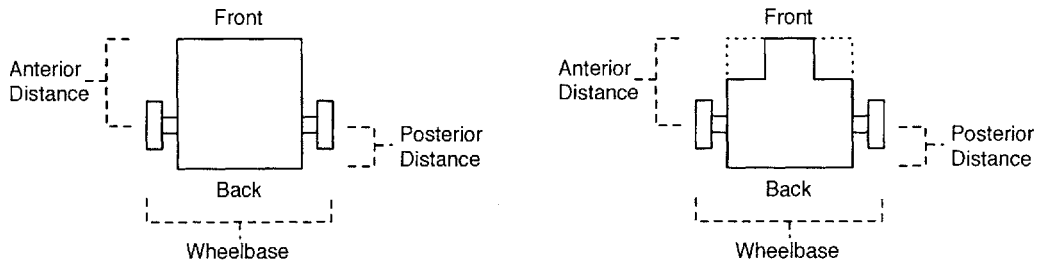


Figure 1-2: The origin of the robot-centric coordinate system is located at the center of the wheelbase. The right-handed coordinate system has the x-axis pointing forward, the y-axis pointing left, and the z-axis pointing upward.

1.4 Document Overview

Chapter 2 includes an overview of some popular robotics toolkits, focusing on the abstraction that they provide and the parameters they require. The chapter also provides an introduction to developmental robotics which serves to show the place where the proposed self-configuration method fits in the scope of the field.

Chapter 3 describes the self-configuration method in detail. It describes the assumptions we make, the specifications of the test harness, and a description of the parameter discovery process for wheel size, chassis dimensions, and wheelbase location.



(a) Exemplar Morphology

(b) Clarified Morphology

Figure 1-3: The sketch in Figure 1-3a is an outline of an example member from the target family of robots. The image in Figure 1-3b is not a member of the supported set of robots, however it will be used in figures throughout in order to facilitate the differentiation between the front and back of the robot.

Chapter 4 provides a description of the procedure for implementing a preliminary system which uses the self-configuration method. The toolkit implements the method described in Chapter 3 for CARMEN, using an RSS II microbot as the test subject. The chapter also includes details necessary for modifying CARMEN for use with the configuration procedures.

The results from the attempted self-configuration of the microbot are reported and analyzed in Chapter 5. The chapter includes an outline of the difficulties encountered during the implementation of the self-configuration system. It also shows that the method for discovering the basic parameters of a microbot-class robot is successful by comparing the results of the configuration routine to the real world, user-measured values.

Chapter 6 reflects on the results of the system implementation, focusing on the strengths and weaknesses of the method. This includes suggestions for future work as well as closing remarks.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Background Information

The design of a self-configuration system is strongly motivated by the current state of robotics control packages. The popular toolkits represent the state of the art with respect to code reuse for robots. An automated configuration system for robots would facilitate the use of toolkits and support the sharing of prepackaged robot tasks. Section 2.1 provides a brief look at CARMEN, Orocos, and PADI, three robot toolkits that are in widespread use. The section focuses on their intent, level of abstraction, and required configuration parameters.

The self-configuration method falls under a category of robotics called *developmental robotics*. An introduction to developmental robotics is given in Section 2.2. State of the art techniques in self-discovery are described in order to show the potential for extension of the method presented in Chapter 3. Finally, this chapter's discussion concludes with a description of a widely used method for comparing odometric performance across different techniques and systems.

2.1 Robotics Toolkits

Robotics toolkits are one of the major motivators for the design of this self-configuration method. Each toolkit, regardless of its primary purpose, requires that a description of the underlying robot be provided through a set of configurable parameters. The intent is that the robots can then be controlled at a more abstract level, which results in software that can be ported across different robot platforms.

The study of different levels of abstraction for robot control architectures is not new.

Rodney Brooks of MIT's Artificial Intelligence laboratory addressed the issues of designing software control packages for mobile robots as early as 1986 [5]. Essentially, the ideas are the same as the standard paradigms used throughout traditional software design: try to make the interface to the lowest levels uniform across all instances so that each low-level unit can be seamlessly exchanged while not modifying the high-level entity.

Using a high level of abstraction results in the ability to use the same software across different robots. CARMEN, Orocos, and PADI are examples of toolkits which are in widespread use and employ abstraction to promote code sharing and reuse among robotics researchers and hobbyists alike.

2.1.1 Carnegie Mellon Robot Navigation Toolkit (CARMEN)

CARMEN was created in 2002 for the purpose of facilitating the sharing of implemented navigation algorithms between research groups [11]. Most importantly, the aim was to have this shared code run on multiple platforms; *platforms* includes robot morphologies as well as the operating systems and hardware architectures that CARMEN executes within. CARMEN is widely used today and is the library that interfaces with the low level operations that are provided by the ORCBoard in the self configuration method presented in this paper.

CARMEN is divided into modules that reside in three different levels: base level, navigation level, and the user level. The different applications, which comprise CARMEN, communicate via the inter-process communication (IPC) system [14]. Since the upper reaches of the toolkit involve configuration of the robot, only the base level is used for this configuration system.

Each CARMEN module can have its own set of parameters. Some of the parameters listed on the CARMEN website[1] include the diameter of the wheels, the location and orientation of sonar sensors, the length and width of the robot's bounding box, odometry errors, and laser offsets. There are many other parameters, but these are the most pertinent to the goals of the self-configuration method being discussed.

[1] A partial list of parameters can be found `http://carmen.sourceforge.net/config_param.html`.

2.1.2 Open Robot Control Software (Orocos)

Orocos is an open-source robot control system intended to be reusable and available across both robot and computer platforms, and to define a public and widespread application programming interface (API), with abstract programming capabilities [6]. The software system is modular in design so that code reuse is strongly encouraged. The developers claim that software reuse among roboticists is not common primarily because it is not easy to do [6]. Each robotics group uses its own API and data representations. As a result, porting software across robot systems remains a difficult task.

The *middleware components* that Orocos provides are a step towards being able to easily share commonly used algorithms for kinematics, motion planning, and more, in a manner very similar to the application modules that CARMEN provides. These components of Orocos are designed to be programming language oblivious by following a specification that is closely related to CORBA[2], a solution that uses mediators to interface a variety of applications across different networking medium [17]. In other words, the components can be written in any language as long as the CORBA interface specification is followed.

Orocos has five primitives which are used to create different *contexts*. These primitives are Event, Property, Command, Method, and Data Port [15]. The different *tasks* of Orocos run in separate threads which are called periodically. This same structural behavior is implemented by CARMEN's message handlers, the difference is that each module in CARMEN runs in its own process space.

This robot control package is fully customizable and is divided into a number of useful primitives. The device interface of Orocos is divided into two parts: logical and physical interfaces. It proposes that the physical interface is device dependent and hence, not portable, whereas the logical interface is portable. As a result of being so customizable, the device interfaces can include any information desired. In effect, Orocos can provide at least as much information as CARMEN.

2.1.3 Player Abstract Device Interface (PADI)

PADI is a successful abstraction of algorithm from the physical aspects of the underlying robot [16]. PADI is another open-source project which strives to define a set of interfaces

[2]CORBA stands for Common Object Request Broker Architecture.

which can be used to write portable and reusable robot code. The system was originally implemented in two different parts. They consist of the `Player`, which acts as the interface to all of the low-level hardware of the robot, and the `Stage`, the simulation environment. The `Player` provides the abstract interface to the robot. The robot can be either simulated or real. The robot can then be controlled either in reality, or in the `Stage`. There is now a third component of the project, a three dimensional simulation environment called `Gazebo`.

`Player` is prepackaged with generic drivers that include support for a number of sensors as well as robot platforms. The robot platforms include the iRobot Roomba and the Nomad robot. Among the sensor interfaces are support for sonar sensors, bump sensors, and laser range finders. The robot platforms have configurable parameters that include the length and width of the robot's bounding box and the location of the robot's center of rotation. Odometry information is available for robot devices that support it.

2.2 Developmental Robotics

Developmental robotics is an umbrella term that encompasses the on-line development of a robot control system. Robots which exhibit this post-startup learning phase are classified as developmental robots [18]. Research in developmental robotics has yielded results which range from robots which learn to walk with their legs, to systems which can sustain damage and then gracefully recover and continue to function [19] [2].

2.2.1 Basics of Developmental Robotics

Two programming paradigms are prevalent in developmental robotics. They are *autonomous development* and *manual development* [18]. The differences between the two are defined by the interactions between the *user agent* and the *robot agent*. The user agent is the end user of the robot system. This can be either the robot programmer or a person interacting with the robot. The robot agent is the program which controls a robot. It is essentially the brain of the robot.

Autonomous development is a mode of learning which consists of two phases. The first phase consists of the programmer creating a developmental program; a program that is task-nonspecific. This program is necessary because the system developer does not know the types of tasks that the robot agent will need to learn while on its journey to self-discovery.

The second phase occurs when the program is started. At this point, the programmer no longer modifies the developmental program that was implemented in the first phase. All that can be done is allow the robot to discover its sensors and actuators much like a human baby would do just after birth. The robot agent can either be taught (or told) by humans or other robots, or learn on its own.

Manual development also consists of two phases, but differs from autonomous development in that the programmer knows, during the design phase, the nature of the tasks that the robot will encounter. He can then implement the robot agent, as necessary, for the specific task.

As a result of the tasks being known during the programming phase, it is typical for robot agents created in this mode to be entirely autonomous after the program has been started. The agent may still employ machine learning techniques once the program has started, the difference is that the architecture of the robot agent is task-specific.

The self-configuration method, which is presented in this thesis, is of this second genre of development. The tasks are known ahead of time, as they are specified by the parameters described within the CARMEN initialization file, and the robot carries out the tasks autonomously, learning its configuration parameters as the process is executed.

2.2.2 Examples

Self-configuration is a subset of the general problem of self-discovery. The self-discovery problem involves having a robot discover itself and the world around it much in the way that a newborn baby must do after birth. More explicitly, a robot agent is initiated without knowledge of its morphology, which sensors it has, where it is, or any knowledge of how to perform even the most basic tasks of locomotion or collision detection. Drivers are given, but the robot must discover which control vectors allow the actuator to achieve a particular goal.

One way for a robot to discover its sensors and actuators is to apply random control vectors to its actuators, and then observe the results with the sensors [12]. The underlying assumption is that if an actuator can affect the environment, then the sensors will be able to detect this change. From this information, it is possible to build *sensoritopic maps* which show the "informational relationship" [12] between sensors. This information can help in the bootstrapping process of a robot, where it can learn about how its sensors and motors

relate, and can eventually learn to follow paths and localize itself on a map.

A framework for self-discovery is proposed in [8]. Through the use of mutual information, a robot learns how to control its hand after 5 minutes of human interaction. The framework is an iterative process which consists of three steps. The first is exploring the space of motor control vectors. This step is similar to the method described in [12]. Once the space has been observed, the robot agent creates categories in which the results of the first phase can be placed; this is the discovery phase. The discovery which occurs in the second step can then be used in the third step to adapt its sensors to the learned events. The framework is then demonstrated by applying it to the discovery of a robot's hand and fingers.

2.3 The University of Michigan Benchmark: UMBmark

The goal of the self-configuration method is to determine the values of a target robot-system's configuration parameters. In order to gauge the level to which this has been achieved, it is necessary to have a systematic method for comparing the performance of the self-configured values to those of a manually configured system.

In the harness method presented in this paper, the most important information is odometry. Since only quadrature phase encoders and motor current-sensors are available, it makes sense that the primary comparison be dead-reckoning.

One method for comparing odometric performance between systems is the University of Michigan Benchmark (*UMBmark*). It has been used in numerous papers as a metric used to compare different methods for performing the same task [7], [9], [1], [4].

The UMBmark is a test which attempts to measure the amount of *systematic error* that is inherent in a robot's odometric system. These are errors which are predictable and can be modeled [3]. Examples of this type of error include the unequal acceleration from the left and right motors, having wheels of different sizes, the resolution of encoders, etc.

The benchmark is performed by driving the robot in a 4 meter square. This task is performed in both the clockwise and counterclockwise directions to eliminate any asymmetric properties of the robot system. Once a lap has been completed, the final internal position is then compared to the final real-world (absolute) position, giving a resulting value which is referred to as the error on return.

The trials are repeated a number of times in order to create two groupings of the

error on return values. The groupings represent the results of performing the UMBmark in the clockwise and counterclockwise directions. The average value of each grouping is then calculated and used to represent the maximum systematic error of the robot when traveling in the given direction. The maximum of the two error values is then taken to be the maximum systematic error for the robot. The more closely clustered that the elements of the respective groupings are, the easier it is to model and compensate for the systematic error inherent in the system.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

A Method for Self-Configuration

In order to design a solution for the self-configuration problem, one must first decide which parameters can be solved for, which parameters any given parameter is dependent upon, and which assumptions can and cannot be made. Once these preliminary issues have been decided, the proposed solution for self-configuration must be demonstrated to be valid.

The method works as follows. The user agent tells the configuration system which sensors are available to the robot. The choice of sensor is limited only by the set of discovery procedures that have been implemented for the particular robot control package. Next, the front of the robot is placed at a predetermined location, in such a way that the translational motion is parallel to the long sides of the physical test harness. Figure 3-1 illustrates the presumed starting position. Finally, the system is started and the specified parameters are discovered.

The remainder of this chapter begins by discussing both the assumptions that were made and avoided in the development of the self-configuration method. Then, the nature of the requirements necessitated by the physical test harness are itemized. The chapter concludes with a discussion of test harness properties and a description of the configuration routine.

3.1 Assumptions

Automated configuration of a robot's parameters is a subset of the general self-discovery problem. Assumptions are made in order to have a starting point for configuration. Once a basic implementation has been tested and proven correct, the assumptions can be relaxed in order to create a more general configuration technique. By designing the initial

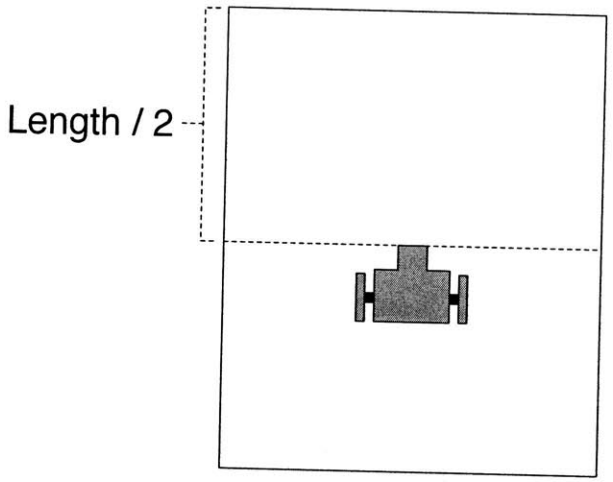


Figure 3-1: The self-configuration routine begins by placing the front of the robot as close to the center of the test harness as possible with its wheelbase perpendicular to the long axis of the harness. The test harness is represented by the solid rectangle surrounding the microbot. In this figure, the front of the robot is placed exactly over the optimal starting position.

configuration method with appropriate assumptions, a generalized process can be achieved. The configuration requirements are presented here so that it can be demonstrated that the assumptions made about the robot, the control package being configured, and test harness are reasonable.

The morphology of the targeted robot family is derived from the robot used in RSS: the microbot. These robots are rectangular in shape and have two, coaxial motors that are arranged in a differential drive configuration. The two, motored wheels of the robot must have the same radius. If the wheels are of two different sizes, then the control toolkit would not be able to drive in a straight line without knowing the sizes of the wheels, or at least the ratio between them, before the configuration sequence. Even so, the presented configuration method is not designed to solve for this case and would likely give a value which works but does not physically represent either wheel. Rather than deal with this complication, the assumption that the two wheels are approximately the same size is made.

It is necessary to make assumptions about the robot control toolkit that is being configured. The most basic assumption is that the toolkit can execute specified translational and rotational velocities. Additionally, the ability to drive in a straight line is also taken as a given capability of the robot application programmer's interface (API). Even if these properties are not provided by the default toolkit, these types of commands are simple

enough to implement given some form of feedback from the motors. In the case of the microbot, feedback is provided to CARMEN in the form of both the amount of current that the motors are drawing, and the number of ticks that have been observed by the quadrature phase encoders.

The dimensions of the test harness must be known. In the system implementation presented in Chapter 4, a wooden test harness, which is 1 m wide, 1.22 m long, and 0.39 m tall, is used by the robot to configure its parameters. The test harness ground surface should have enough friction so that a robot can detect a collision through odometric information. The test space is assumed to be clear of obstacles.

A number of robot agents have the ability to discover their sensor array without prior knowledge [10] [12] [13]. Instead, these systems discover the types of available sensors by applying different vectors to their actuator ports and then creating sensorimotor maps based on the resulting sensor feedback. A sensorimotor map is a way of visualizing the informational relationships between sensors and actuators. Based on this work, it is reasonable for this self-configuration method to take the available sensor types as given. The types can either be provided by the user agent, or through one of these discovery mechanisms.

3.2 Physical Test Harness

The physical test harness is necessary because it provides a reference point which serves to ground the estimated values in relation to something. The method was designed for RSS students to use on their morphologically analogous robots. Successful configuration by means of the presented routine is dependent on the fact that the parameters of the operating environment are known ahead of time.

3.2.1 Description of Physical Test Harness

The test harness is simple to construct. It is necessary only to have barriers which can stop robot movement and can be detected by the available sensors. There should be enough room for the robot to execute its configuration routine. Motions which the method depends on are harness traversal and in-place rotation. Materials for the implemented system's test harness consist of existing wooden boards as shown in Figure 3-2. Each of the four pieces is 1.22 m long; they are overlapped in order to achieve a rectangular region of the following

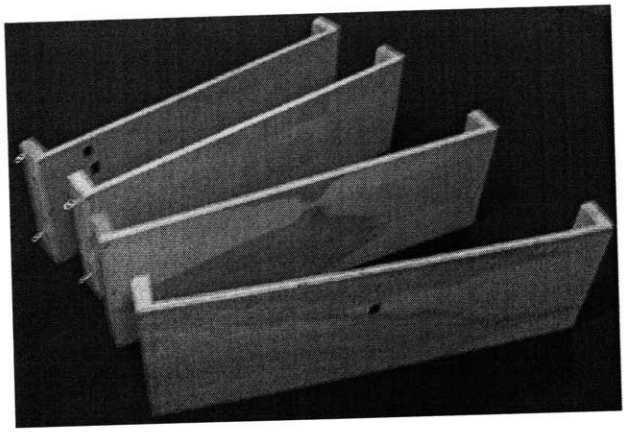


Figure 3-2: The pieces used in the assembly of the physical test harness are taken directly from the RSS class. Originally, they served as maze segments in the RSS Grand Challenge.

interior dimensions: 1.000 m in width, 1.220 m in length, and 0.386 m in height. Depicted in Figure 3-3 is the assembled test harness used in this study.

A rectilinear test harness was chosen because construction is simple, the length and width are two different known values, and corners are a natural feature with a distinct signature. While the width and corners are not utilized in this particular method, these features can be utilized in future extensions of the system which attempt to localize sensors on the robot frame.

Using a rectangular, but not square, bounding box also has the advantage that iterations of the configuration sequence can be repeated using two different, known values. Additionally, width is a local minima when measuring distance from the robot to a given side of the test harness. This information is useful for determining the location of sensors such as sonar sensors and laser range-finders.

3.3 Attaining Motion Control and Estimating Coordinate Space

We chose to focus on one feature at a time, then incrementally add other unknowns, in discovering the configuration parameters of the robot. Properties of the robot's sensors would be difficult to discover if the robot could not move in a controlled manner to perform configuration experiments. It is by knowing the radius of its wheels and the width of its wheelbase that a coaxial differential drive robot can accurately execute translational and

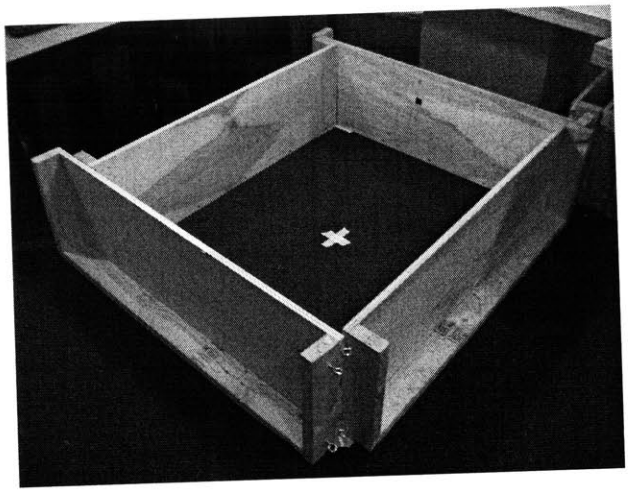


Figure 3-3: Photograph which illustrates the physical test harness that is used

rotational velocities. The remainder of this section presents a method which can determine these two parameters and will additionally provide the location of the wheelbase in body coordinates.

3.3.1 Wheel Radius Discovery Routine

Knowing the length of the robot's wheel radius is essential for controlling translational velocity when movement is based solely on odometry data. It is possible to experimentally determine the velocity of the robot by using a suitable robot control package and the known parameters of the test harness. Experimentally determining the velocity makes it possible to compute the wheel radius because these two values are directly proportional to one another through a fixed constant. Once the wheel radius is known, it becomes possible to learn the values of other configuration parameters.

Understanding the Relation Between Wheel Radius and Velocity

A motor controller can calculate the amount of distance traveled by counting the number of odometry *clicks* that have been reported by quadrature phase encoders. A given motor equipped with quadrature phase encoders has a resolution that depends on the number of clicks per revolution of the motor axle. This resolution is the smallest number of radians that the encoder can measure and is given by the equation

$$k = \frac{2\pi}{C}, \tag{3.1}$$

where C is the number of clicks per revolution of the motor axle, and k is the encoder resolution.

For small angles, such as the ones spanned by odometry clicks, the distance along the arc is approximately

$$d_{arc} \approx r\theta, \tag{3.2}$$

where d_{arc} is the distance along the arc, r is the radius of the circle, and θ is the size of the angle which determines the arc. In the case of motors equipped with quadrature phase encoders, θ is given by

$$\theta = k\{\#\ of\ clicks\}. \tag{3.3}$$

The relationship between the wheel radius and the distance traveled per click can be seen through equations 3.2 and 3.3.

Robot toolkits implement commanded velocities by ensuring that a specified number of odometry clicks occur from sample to sample. This means that velocity, as implemented by robot toolkits, can be written as

$$v = \frac{d_{arc}\{\#\ of\ clicks\}}{\Delta t}, \tag{3.4}$$

where d_{arc} is the distance traveled by each encoder tick. By plugging Equation 3.2 into Equation 3.4, it becomes apparent that r is directly proportional to v through known constants.

A click-dependent constant, α, can be named such that

$$\alpha = \frac{\{\#\ of\ clicks\}}{\Delta t}\frac{2\pi}{C} \tag{3.5}$$

and accordingly,

$$\frac{v}{r} = \alpha. \tag{3.6}$$

Equation 3.6 describes the relationship, for a given number of odometry clicks, between a velocity, v, and its associated radius, r. Knowing this, the robot toolkit can be utilized to determine the actual radius. This can be accomplished by guessing a radius, r', and attempting to traverse a known distance at a commanded velocity, v'. Measuring the

resulting actual velocity allows the true radius, r, to be determined through the relation

$$\frac{v}{r} = \frac{v'}{r'}. \tag{3.7}$$

The relationship in Equation 3.7 is possible, because α is the same for both radius-velocity pairs. The wheel radius discovery utilizes this information to determine the robot's wheel radius.

Method for Wheel Radius Configuration

The routine begins by placing the robot at a known distance from a chosen harness wall such that there is enough room on either side of the robot for a complete rotation. The distance from the chosen wall is referred to as d. Next, a guess of the actual wheel radius is made. The guessed wheel radius will be referred to as r'.

Next, a starting time, t_{start}, is recorded and the robot is commanded to translate at a velocity of v'. The actual velocity at which the robot translates is dependent on the length of the actual radius, r, and can be expressed by manipulating Equation 3.7 to give

$$v = r\frac{v'}{r'}. \tag{3.8}$$

The robot should continue to move at v, until a collision occurs. The time of the collision is recorded as time t_{coll}. Now, all of the information required in calculating v is known, and it can be calculated by

$$v = \frac{d}{t_{coll} - t_{start}}. \tag{3.9}$$

This actual velocity can now be used to calculate the actual radius size, r, by

$$r = r'\frac{v}{v'}. \tag{3.10}$$

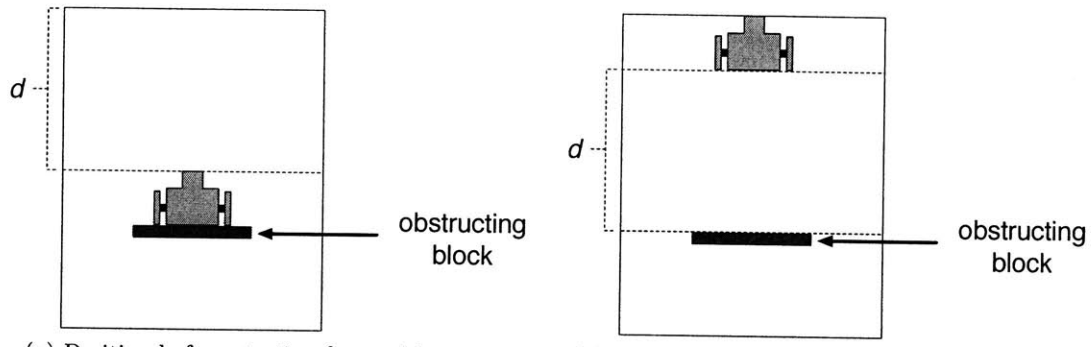
The success of this configuration routine is dependent on several variables. First, since the user agent is responsible for placing the robot on the starting position, it is possible that the distance the robot actually travels will not match the assumed distance d. Second, the source for the start time and the collision time is important. There cannot be a large delay between the recorded start time and the actual time where the robot begins to move. The

same applies to the time of collision. It is important that the time recorded when a collision occurs, is close to the actual time of collision. In order to gather a robust measurement, it is possible to repeat the experiment, and gather more data, by manually resetting the robot to the starting position.

The primary weakness of the wheel radius configuration routine is that it is not autonomously repeatable without modification of the test harness. The reason for not being autonomously repeatable is that, initially, the distance from the front of the robot to the wall of the test harness is known – the user agent puts the robot down to the best of his ability. However, once the routine has completed one iteration, the robot is at the boundary of the test harness. It is now not possible to return to the starting position without incurring some amount of error. If the original guess (r'_w) is used and the robot travels in the reverse direction for a duration of $t_{start} - t_{coll}$, then error results from inaccurate measurement of time, and perhaps an asymmetry in the acceleration of the forward direction versus that of the backward direction. Likewise, if the newly calculated r'_w is used and the amount of time traveled at a commanded velocity is used to approximate position, error is incurred due to the fact that the guess of wheel size will be reinforced by the result of the second trial reaffirming the means used for its setup.

One solution to this weakness is to start the robot at the edge of the test harness rather than the center. While this ensures that the wheel radius configuration routine is repeatable, it comes at the cost of needing to measure the distance, d. This cost is both the same as measuring the chassis length of the robot manually, and must be repeated whenever a robot of a different chassis length is used. It was decided to start the robot in the center of the test harness so that the cost of measuring d will occur only once, given that the chassis length of the robot used with the test harness is less than the test harness length minus d.

A second solution is as follows: once the robot has been placed at the specified starting position, place a sturdy structure behind the robot. This idea eliminates the need for knowing the length of the robot's chassis by limiting the travel distance to d in both the forward and backward direction. Figure 3-4 contains sketches which illustrate this second idea. This choice was not used in this thesis for the sake of favoring a completely autonomous solution requiring no human interaction beyond the initial setup and starting the configuration sequence.



(a) Position before starting forward iteration (b) Position before starting backward iteration

Figure 3-4: The wheel radius configuration routine, as it is implemented in this thesis is not autonomously repeatable by the robot. A proposed solution to this issue is to place an obstructing block behind the robot after the robot has been placed at the starting location. Implementing this solution would allow the robot to repeat this configuration experiment until a satisfying value is achieved. Figures 3-4a and 3-4b show the initial position for the forward and backward direction, respectively. In the solution's simplest incarnation, the human operator can place (and remove) the block. Alternatively, upgrades to the test harness could potentially provide an automated method for placing the block.

3.3.2 Chassis Length Discovery

The chassis length discovery routine is a simple, repeatable experiment which determines the value of l. This value is determined by measuring the amount of distance that the robot is able to travel across the test harness without observing a collision. The ability to translate at a specified velocity is important in determining the amount of distance that has been traveled.

Determining l begins with the robot starting flush with a wall of the test harness. Next, it is commanded to move towards the opposite test harness wall, at a velocity, v, until a collision is observed. The robot should measure the amount of time taken from starting the traverse, t_{start}, to the point in time where a collision is observed, t_{coll}. Given t_{start}, t_{coll}, and the span of the test harness that is being traversed, call it d, the length of the chassis can be calculated as

$$l = d - (v * (t_{coll} - t_{start}) \tag{3.11}$$

This method for determining the chassis length depends on being able to translate at a commanded velocity v, and that the length of d is known. These properties make this procedure ideal for following the completion of the wheel radius configuration routine that is described in Section 3.3.1. After the wheel radius has been configured, the robot is left in a suitable position for this chassis length configuration procedure.

The chassis configuration method is repeatable. This allows for a mean and variance to be established over the calculated chassis length values. These two values can be used to determine when the chassis configuration step is complete, for example, by repeating until the reaches a specified threshold.

Theoretically, the method will yield an accurate measurement of the robot chassis. In practice, sources of error for this configuration routine can start as early as during the configuration of the wheel radius. If the wheel radius is incorrect, then the commanded velocity will not be accurate and the resulting distance traveled will also be inaccurate. Also, as mentioned previously in the wheel configuration routine, the accuracy of the recorded t_{start} and t_{coll} values can affect the resulting calculation of l. As will be demonstrated in the results chapter, these issues do not have an large impact on the resulting chassis length.

3.3.3 Wheelbase Discovery

The goal of this method is to configure the width of the robot's wheelbase. The way in which this is accomplished simultaneously approximates the position of the wheelbase in body coordinates, i.e., the distances from the wheelbase to the front and back of the robot along the y-axis of the robot-centric coordinate system. This section describes how the robot can experimentally measure d_f and d_r, as defined in Figure 3-5, and how these values can be used to determine the wheelbase, b, and its location in body coordinates, given by l_f and l_r.

Discovering the width of the robot's wheelbase requires that the wheel radius and chassis length have already been discovered. The wheel radius must be known so that the robot can move a specified distance. The chassis length is needed to calculate the unknown values of l_f, l_r, and b.

Solving for the Wheelbase Width and Position

The wheelbase discovery routine depends on being able to find two values, d_f and d_r. These two values are the amount by which the radii, r_f and r_r, exceed the body frame. Figure 3-5 shows the triangle of interest.

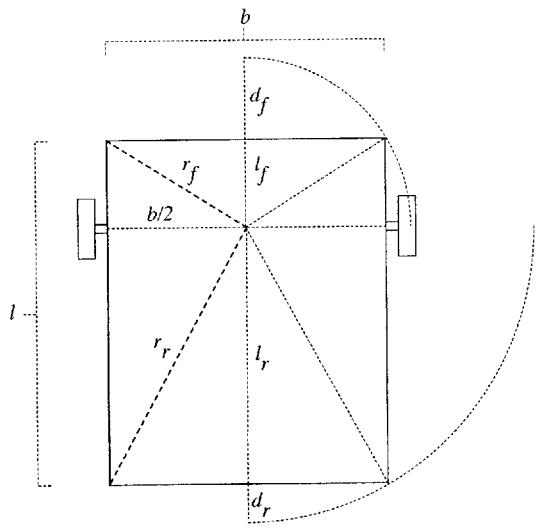


Figure 3-5: A number of distances on the robot chassis are referenced throughout this paper. The length and width of the robot chassis are denoted by the values l and b, respectively. The distances from the wheelbase to the front and rear of the robot frame are given by l_f and l_r. The distance from the origin of the robot coordinate system to the front and rear corners of the robot are radii denoted by r_f and r_r. The difference between r_f and l_f gives the value d_f, where d_f is the distance that is measured by the wheelbase discovery routine. The same is true for d_r, only with respect to the rear of the robot chassis.

Pythagoras' Theorem states that the values of r_f and r_r can be expressed as

$$r_f^2 = l_f^2 + (\frac{b}{2})^2 \tag{3.12}$$

$$r_r^2 = l_r^2 + (\frac{b}{2})^2 \tag{3.13}$$

The radius r_f exceeds the chassis length, l_f, by a distance d_f. An analogous situation is true for r_r, with its corresponding values. The distances d_f and d_r are experimentally determined by the wheelbase discovery method described in Section 3.3.3. The relationship between these values can be expressed as

$$r_f = l_f + d_f \tag{3.14}$$

$$r_r = l_r + d_r. \tag{3.15}$$

The locations of the variables in equations 3.14 and 3.15 are shown in Figure 3-5.

The fifth equation of the system is given by the fact that the sum of ℓ_f and ℓ_r is

constrained to the total length of the chassis. This relationship is expressed by

$$l = l_f + l_r. \tag{3.16}$$

The unknown variables l_f, l_r, r_f, r_r, and b, can be expressed in terms of one another through manipulation of the above equations. The following is a derivation focused on finding an expression for l_f in terms of the known values: l, d_f, and d_r.

Substituting the expression for r_f, given by Equation 3.14, into 3.12 gives

$$(l_f + d_f)^2 = l_f^2 + (\frac{b}{2})^2. \tag{3.17}$$

The quantity $(\frac{b}{2})^2$ can be expressed in terms of r_r and l_r by using Equation 3.12. This allows Equation 3.17 to be rewritten as

$$(l_f + d_f)^2 = l_f^2 + r_r^2 - l_r^2. \tag{3.18}$$

Applying Equation 3.16 to Equation 3.18 allows the expression to become

$$(l_f + d_f)^2 = l_f^2 + r_r^2 - (l - l_f)^2. \tag{3.19}$$

Next, r_r^2 can be expressed in terms of l_r by manipulating Equation 3.13. Substituting into Equation 3.19 results in

$$(l_f + d_f)^2 = l_f^2 + (l_r + d_r)^2 - (l - l_f)^2. \tag{3.20}$$

The expression is reduced to one unknown by using Equation 3.16 to substitute for l_r, resulting in

$$(l_f + d_f)^2 = l_f^2 + (l - l_f + d_r)^2 - (l - l_f)^2. \tag{3.21}$$

Now, it is possible to solve for l_f. First, expand the equation by distributing the terms

$$\begin{aligned} l_f^2 + 2l_f d_f + d_f^2 &= l_f^2 + (l^2 - ll_f + ld_r) + (-ll_f + l_f^2 - l_f d_r) + \\ &\quad (ld_r - l_f d_r + d_r^2) - (l^2 - 2ll_f + l_f^2). \end{aligned} \tag{3.22}$$

Next, remove all terms which sum to zero, giving

$$2l_f d_f + d_f^2 = -2l_f d_r + 2ld_r + d_r^2. \tag{3.23}$$

Finally, move all terms containing l_f to the same side, and divide to solve the system with

$$l_f = \frac{2ld_r + d_r^2 - d_f^2}{2(d_f + d_r)}. \tag{3.24}$$

Or, equivalently, solving for l_r instead of l_f gives

$$l_r = \frac{2ld_f + d_f^2 - d_r^2}{2(d_f + d_r)}. \tag{3.25}$$

These two values, l_f and l_r, give the location of the wheelbase with respect to the robot frame. The width of the wheelbase is then given by Equation 3.12 or 3.13.

Method for Determining the Wheelbase

The previous section describes how to derive b, l_f, and l_r when the values of l, d_f, and d_r are known. It has also been previously stated that l and r_w are assumed known in order to carry out this configuration routine. It is reasonable to assume that these values are known because if they have not been configured, then the corresponding configuration process can be carried out.

It remains to be shown how d_f and d_r can be determined. For the sake of clarity, only the measurement of d_f will be described in detail because the procedure for d_r is analogous.

The configuration routine begins by assuming that the front of the robot is flush with one of the harness walls. Next, the robot control package is configured with a guess of the wheelbase width, b'. This allows the robot to rotate, regardless of the fact that the rotational velocity commanded will not necessarily be equivalent to the observed rotational velocity.

Starting from this setup, make a guess for d_f and call it d'_f. Next, move the robot away from the harness wall a distance of d'_f and stop. Then, commence a rotation until either a collision is observed by a collision detector, or a timeout is reached. If the timeout is reached, then either d'_f is too large and the corner completed the rotation, or the timeout occurred too soon for a collision to occur.

In the event of either failure scenario, the recovery strategy is the same. Return the robot to its initial orientation, where the direction of translation is perpendicular to the targeted collision wall. Next, decrease the value of d'_f by an incremental amount, $\Delta d'_f$, and move so that the robot is a total distance of d'_f from the wall. Now, repeat the test of d'_f. This recovery step can be repeated until a collision occurs. At this point, an upper bound on d_f has been determined, and the robot can proceed to place an upper bound on d_r by traversing to the opposite side of the test harness and repeating this test sequence.

If a collision occurs during the first d'_f test phase, then return the robot to the starting orientation and increase d'_f by $\Delta d'_f$. Move the robot to the new distance of d'_f from the test harness wall, and repeat the test. Repetition continues until a timeout instead of a collision. At this point, an upper bound on the value of d_f has been determined and the robot can repeat an analogous procedure for d_r on the opposite end of the test harness.

Once upper bounds on d_f and d_r have been determined, the values of b, l_f, and l_r can be obtained through equations 3.24, 3.16, and 3.12.

In the proposed wheelbase method, there is not an angle, analogous to d in the other configuration routines, which the robot can use as a reference for measuring angular velocity. As a result, the wheelbase configuration routine is more involved than the proposed wheel radius and chassis length routines. In the first two methods, the unknown being measured was a time interval; in the wheelbase configuration, the unknown being observed is a distance. Despite the increased number of steps involved, Chapter 5 demonstrates that the proposed wheelbase configuration method adequately allows the robot to measure its width and location of its wheelbase.

Chapter 4

Experimental Setup and Execution

The system described and tested in this chapter demonstrates that the method presented in Chapter 3 is a practical approach to solving the self-configuration problem for microbot-class robots; that the system configures the wheel size, chassis dimensions, and wheelbase offset parameters of CARMEN each to an acceptable degree. The level of acceptability is measured by a direct comparison of self-configured results to user-measured values, as well as through a standardized method of comparing odometric error.

The following sections describe the steps taken to implement the three components of the configuration system and the tests performed in order to demonstrate the system's functionality. The chapter begins with an overview of the materials and preparation that are required. Following the setup is a description of the functionality experiments that must be conducted on each configuration component. The chapter concludes with the description of an end-to-end test which demonstrates the results of the configuration system being used in the execution of a simple robot task.

4.1 Materials

Not many materials are needed to implement the self-configuration method. A microbot, a computer with a serial port and an installation of CARMEN, an ORCBoard, and enough material to construct a four sided rectangular test harness are the necessary materials.

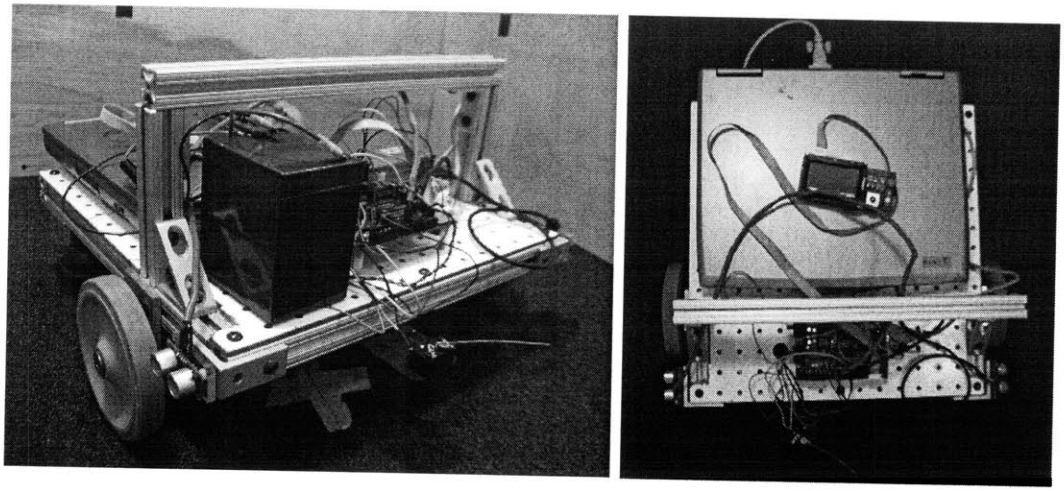


Figure 4-1: The RSS Microbot. Visible components are the sonar sensors, bump sensor, laptop, ORCBoard, crossbar, wheels, and battery.

4.1.1 Assembly of Test Harness

The method described in Chapter 3 makes the assumption that the parameters of the configuration environment are known and provided to the system before configuration takes place. Without the presence of the known physical test harness, the automated configuration process is much more difficult to solve. The physical test harness, as described in Section 3.2, can be constructed from any even surfaces which can be arranged in a way that bounds a rectangular region on the floor. This system's harness is constructed using available pieces of wood from the RSS laboratory course. By overlapping wooden pieces that are 1.22 m long by 0.39 m high, it is possible to assemble a 1.22 m by 1.00 m by 0.39 m test harness.

4.1.2 Self-Configuring Entity: RSS Microbot

The microbot, depicted in Figure 4-1, is a simple robot with a 2-wheel, coaxial, differential drive and 2 fixed-point wheels. The main chassis is made of a peg board that is approximately 0.38 m square. The body is surrounded by 4 beams of 8020 and it also has a crossbar that rises 0.15 m over the main body of the microbot. There is enough room on the main body for an ORCBoard, a laptop, and the types of sensors that are available to the RSS students. The only limitations on sensors are determined by the weight that the robot can support, the sensors that the ORCBoard can support, and the ports available on the laptop. Only quadrature phase encoders and the on-board current-sense functionality that is provided by the ORCBoard are used in this implementation.

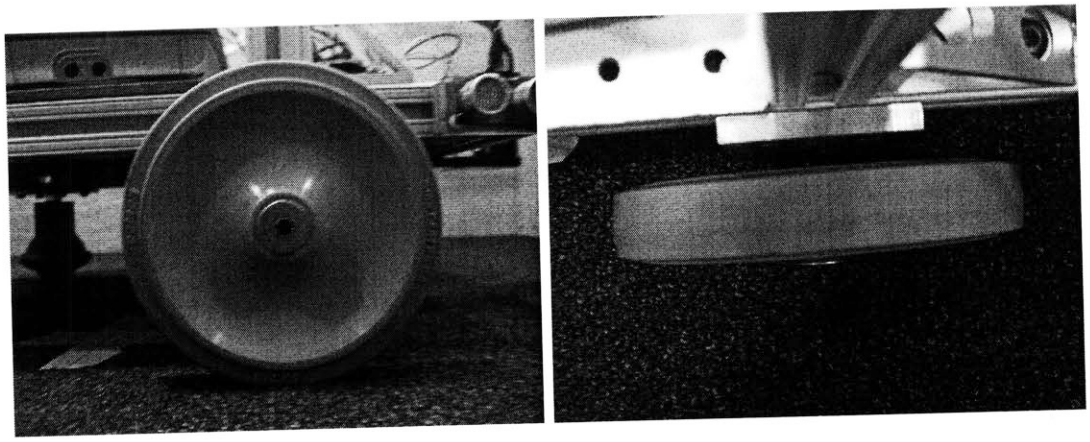


Figure 4-2: The wheels used are rubber, with a radius of 6.2 cm and a thickness of 2 cm.

Single point rotation is possible due to two fixed-point wheels. This rotational capability makes locomotion easier because translation and rotation are separated into two distinct processes. The two wheels are driven by controllable motors and are 0.0625 m in radius, allowing for dependable movement on level ground.

4.1.3 Controlling the Robot

ORCBoards are used to interface with the actuators and sensors of the microbots. The ORCBoard provides low-level control over the I/O devices and communicates with a higher-level entity, such as a robot control package. In this case, CARMEN is used to interface with the ORCBoard to allow for a higher level of abstraction and more complex controls.

Our Robot Controller Board (ORCBoard)

The ORCBoard is a robot controller that was designed and implemented by Edwin Olson, an MIT doctoral candidate at MIT, and was intended for use as the robot controller for the Mobile Autonomous Systems Laboratory (MASLab) at MIT. The board consists of FPGAs and PALs which provide control over four motor ports, several servo ports, and have the ability to interface with a variety of sensors (e.g. laser range finders, sonar sensors, bump sensors, photosensors). This self-configuration system, uses motor drivers, quadrature phase encoder ports, and the ORCBoard motor current measurement capability.

One of the limiting factors of the ORCBoard is that it sends information to a requesting entity at approximately 10 Hz. This means that updates are slow, and sample intensive

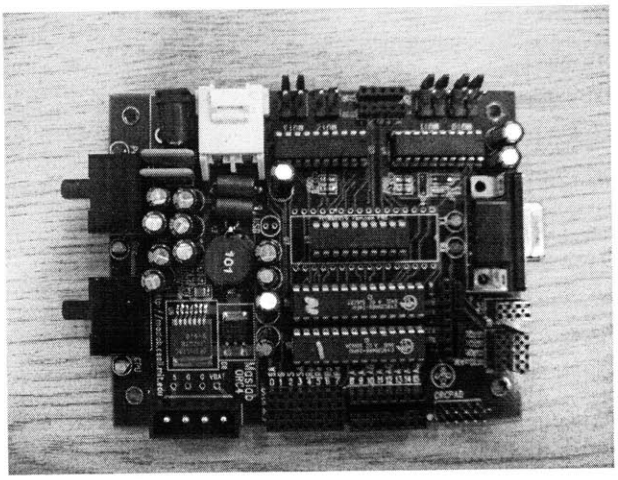


Figure 4-3: A revision 4 ORCBoard.

statistics may not produce results fast enough for the robot to issue a reaction.

CARMEN

CARMEN is the control package of choice for RSS, and already has support for the version 4 ORCBoard that is being used. The fact that the configuration system is easily integrated into the framework of RSS could allow for extensions of the system which are useful to the course.

This robot navigation toolkit is an open source, cross-robot set of applications which each run in their own process space. The CARMEN modules can share information across process space and networks through a message passing protocol, called IPC [14].

Given the test harness, robot, robot control tools, and the CARMEN Java interface, a system consisting of the methods presented in Chapter 3 can be implemented according to the description outlined in Section 4.2.

4.2 Preliminary Tasks

Before discovery of wheel radii can take place, there needs to be a mechanism through which collision detection can take place. This is a logical prerequisite because the technique for wheel discovery is based on the assumption that a collision will occur and will be detected. But, the self-configuration problem consists of pinpointing the locations of sensors and configuring the parameters of a robot control package. Therefore, it is not reasonable to

require that a particular sensor be attached in a prescribed location or orientation. However, since it is possible for the robot to be moved in a controlled manner, then there must be some form of feedback related to the movement that has occurred.

Taking notice of this, it should be possible to create a collision detector by filtering data from the odometric proprioceptive sensors. Examples of sensors within this category that are readily available on the microbot are quadrature phase encoders and the ability to sense the amount of current being drawn by the motors.

4.2.1 Implementation of a Current-based Bump Sensor

Accessing current information through CARMEN is a multi-step process. ORCBoards are equipped with current-sense capabilities on their motor ports. The values measured by these sensors are available through the ORCBoard's slave packet request API. But CARMEN must be modified to support this functionality, because its base level ORCBoard interface does not provide a mechanism for requesting this information. This modification requires: modifying the lowest level ORC library to supply access to the most recent information on current, creating a new message that will provide this information, changing the `base_main` application to retrieve the information using the ORC library and package it into the new message type, and finally adding the message to CARMEN's Java interface.

Once information regarding current is available, the remaining issues are characterizing the behavior of current samples, and implementing a filter that utilizes this information in order to detect a collision.

Modification of CARMEN

CARMEN's architecture consists of numerous processes, residing in three layers, which communicate through a message passing protocol called IPC [14]. The self-configuration system only needs access to the base layer. The base layer consists of the lowest level controller, `base_main` – which can interface with a number of different hardware types, and the interfaces to those hardware types. The interface to the ORCBoard is a driver library called `orc_lib`.

The ORCBoard is implemented by two microcontrollers, a master and slave, which communicate using an ORCBoard-specific packet communication protocol. Each microcontroller is responsible for a number of tasks, one of which is monitoring inputs. It is the

slave microcontroller which monitors the amount of current that a motor is drawing. The data for each motor consumes two bytes located at bytes 27 and 31 of the slave device's status packet. Since `orc_lib` already requests for a status update from the ORCBoard's slave device, the modification necessary consists of providing a function for retrieving the most recent current-measurement sample from `orc_lib`.

Next, an IPC message needs to be created so that `base_main` has a mechanism for publishing the new motor information. This can be done by adding a message definition to `base_message.h`. After the new message has been defined, `base_main` can be modified to publish messages containing the most recent measurement of current.

Characterization of Current

Designing an FSM which can detect collisions based on current measurements requires characterizing the sensor readings under a variety of situations. The hypothesis is that the error between a given current sample and the instantaneous mean will be small enough that an outlier will be well above the signal quality. Characterizing the current samples will give insight regarding a robust value for thresholding incoming samples.

Measurements will be taken under the following conditions: constant velocity, acceleration from a full stop, acceleration and deceleration while movie, and collision with the test harness. The velocity values used in each scenario are low, medium and high: 0.00 m/s, 0.03 m/s, 0.06 m/s, 0.09 m/s. These velocities, which range between 0.00 m/s and 0.10 m/s cover the range of velocities considered for use in the self-configuration sequence.

It is useful to create a custom graphing utility which can display and manipulate data before performing this experiment because the best method for processing the data samples, in order to detect a collision, is unknown. The functionality needed is merely the ability to plot data, display the mean and standard deviation of the data, and perform different signal processing algorithms over the sample set.

Collision Detection

The results of current-sense characterization will provide a threshold value, as well as the necessary data processing method for implementing the current-based bump sensor (CBS). Assuming that the hypothesis stated in 4.2.1 is correct, the following FSM will implement a collision detector that is based on the given sample data. The idea is that a sample received

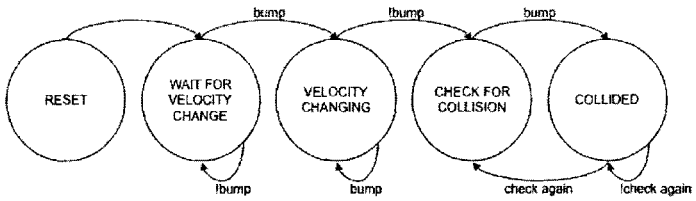


Figure 4-4: Depicted here is a state transition diagram for the current-based bump sensor.

at time t from the current sensor will be within two standard deviations of the mean that has been observed up to that point, μ_{t-1}. A collision can be viewed as an acceleration, therefore it is assumed that, under the given collision criteria, a commanded acceleration will generate a false positive.

Based on these observations, an FSM similar to the one depicted in Figure 4-4 will be able to detect collisions with a reasonable amount of precision. Every time a current sample is received, the CBS updates the values for mean and variance of the sample data, denoted by μ and σ^2 respectively, and updates its state appropriately. If a given current sample is greater than a predetermined threshold value then the FSM will report a collision and publish a message through CARMEN's IPC which reflects this observation.

The states of the FSM are as follows. On `RESET`, the FSM clears its state by setting μ and σ^2 to zero so that a new history may be collected. The machine then moves to `WAIT_VELOCITY_CHANGING`, where it waits until a peak is detected. The assumption is that this peak signifies an acceleration and not a collision. Once a peak is detected, the FSM transitions to `VELOCITY_CHANGING`, where it waits until the signal settles. This is taken to mean that the acceleration is complete. When the collision detector sees that the current data has returned to a steady state, it assumes that the next peak that is detected represents a collision. The transition from `VELOCITY_CHANGING` to `CHECK_FOR_COLLISION` denotes that the FSM is ready to detect a collision. The only difference between `CHECK_FOR_COLLISION` and `WAIT_VELOCITY_CHANGING` is that the former publishes a message which reports a collision.

Verification that the CBS functions properly will result from the following test. The robot will back up. Upon detection of a collision, it will stop. Then the robot will translate forward until it collides with the barrier in front of it, where it will stop again.

4.3 Implementation of Configuration Components

The set of procedures for parameter configuration are implemented as Finite State Machines (FSMs). The FSMs can be implemented in any order, but make the most sense when implemented in the order presented. The recommended order of implementation is wheel radius configuration, followed by chassis length, and ending with the simultaneous calculation of the wheelbase width and location. This ordering allows each test to utilize the results from previously executed configuration sequences.

4.3.1 Radius Configuration

Configuration of the wheel radius must be performed correctly because the subsequent experiments rely on this value – the chassis length computation depends on being able to travel at a specified translational velocity, and the wheelbase procedure involves moving a specified distance away from the test harness boundary.

The wheel radius configuration routine is implemented as a five state FSM, as shown in Figure 4-5. The system starts in RESET, where all of the parameters are set to their initial values. When the system has been started, state transitions are triggered by CARMEN messages. From the RESET state, the configuration routine makes a guess of the wheel radius, r'_w, and sets the wheel diameter parameter of CARMEN to be the appropriate value ($2r'_w$). Once this radius guess has been made, the robot is commanded to translate across the known distance, d, at the commanded velocity v', until a collision occurs. This experiment phase occurs during the DETERMINE-WHEEL-RADIUS state. After a collision is observed, the system updates the r'_w parameter with the newly calculated radius value, and sets the appropriate CARMEN parameter to this value. A sketch of the robot performing the DETERMINE-WHEEL-RADIUS state is shown in Figure 4-6. More detail regarding the radius configuration routine can be found in Section 3.3.1.

Theoretically, the parameters used for d, the initial r'_w, and v' do not matter. Experiments which test this theory and show the limitations of the system are described in Section 4.4.1, and the results are presented in Chapter 5.

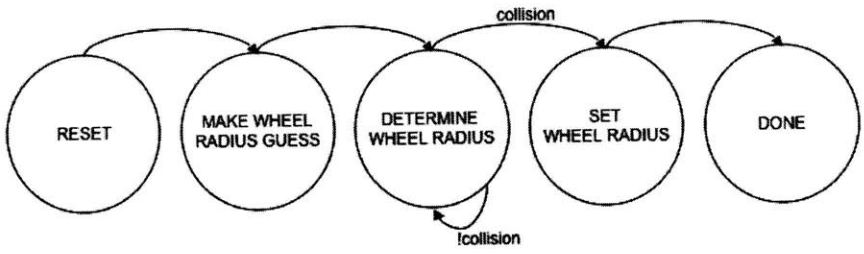


Figure 4-5: The wheel radius configuration experiment is implemented as a five state FSM. A guess of the radius length is made during the MAKE-WHEEL-RADIUS-GUESS state, and movement from the pre-specified location takes place during the DETERMINE-WHEEL-RADIUS state. The actual radius value is calculated by the amount of time that elapses between starting translation, from this point, and colliding with the harness wall. When the collision occurs, the FSM enters the SET-WHEEL-RADIUS state and calculates the wheel radius. Following this, the state machine enters the DONE state. At any time, the system can be sent to the RESET state, and transitions occur whenever a CARMEN message is received.

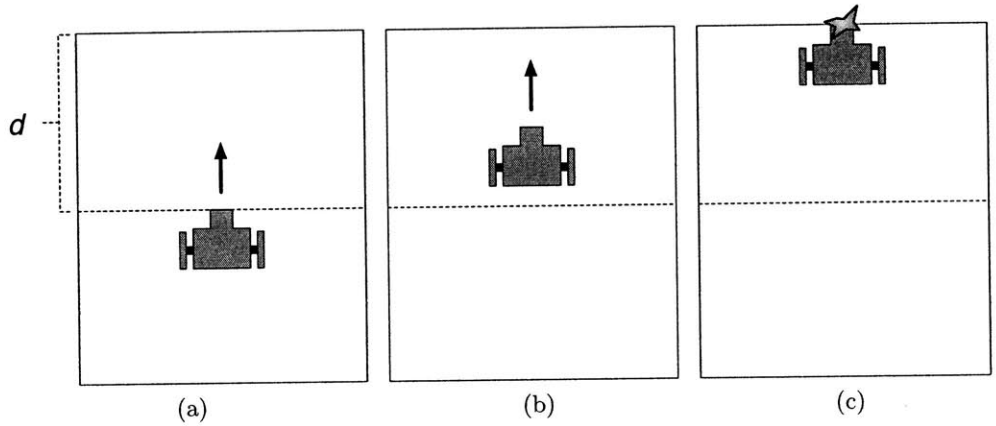


Figure 4-6: The wheel radius configuration routine is the first sequence that is executed by the self-configuration system. This configuration step yields a measurement of the wheel radius, which in turn allows the robot to translate at specified velocities. The robot, as it begins the DETERMINE-WHEEL-RADIUS step of the FSM, is shown in (a). The robot, as it waits for a collision is shown in (b). Finally, the robot collides with the test harness and can calculate the wheel radius. This last step is shown in (c).

4.3.2 Chassis Length Configuration

The chassis length configuration routine consists of one crucial step: translate across a known distance until a collision occurs. The other four states merely provide a nice package so that higher-level entities can easily interface with this FSM. These five states are shown in Figure 4-7.

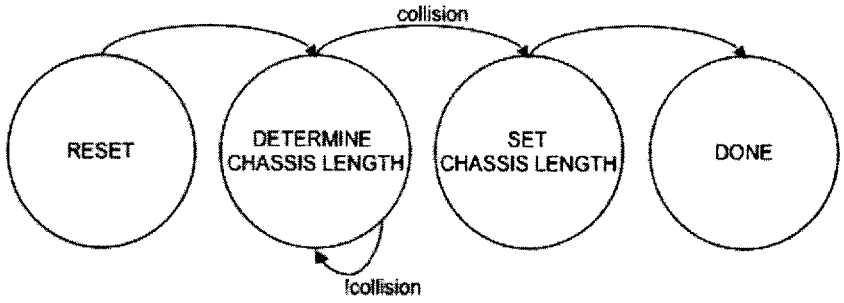


Figure 4-7: The chassis length is determined by traversing the test harness at a known velocity, and measuring the amount of time it takes to get from one side to the other. This is implemented by a four state FSM which starts in RESET. The state machine then proceeds to the DETERMINE-CHASSIS-LENGTH state, where it measures the amount of time necessary to cross the test harness at the commanded velocity v. On collision, the FSM enters the SET-CHASSIS-LENGTH state, where the chassis length is calculated. The procedure is then complete, and the state machine translates to the DONE state.

This configuration routine can be repeated as many times as necessary, but the low-level FSM will only execute one trial. If a repetition is desired, simply change the direction of the commanded velocity, v, reset the FSM, and let it execute until a collision occurs on the opposite wall.

The FSM starts in RESET and transitions whenever a new CARMEN message is received. Message subscriptions include CBS messages and odometry messages. When the configuration routine enters the DETERMINE-CHASSIS-LENGTH state, the robot is commanded to translate at velocity v until a collision occurs. On observing a collision, the FSM stops the robot and calculates the length of the robot chassis. The chassis length calculation occurs during the SET-CHASSIS-LENGTH state. At this point, the procedure is done and the FSM waits in the DONE state, doing nothing, unless it is reset. An illustration of the robot carrying out these actions can be found in Figure 4-8.

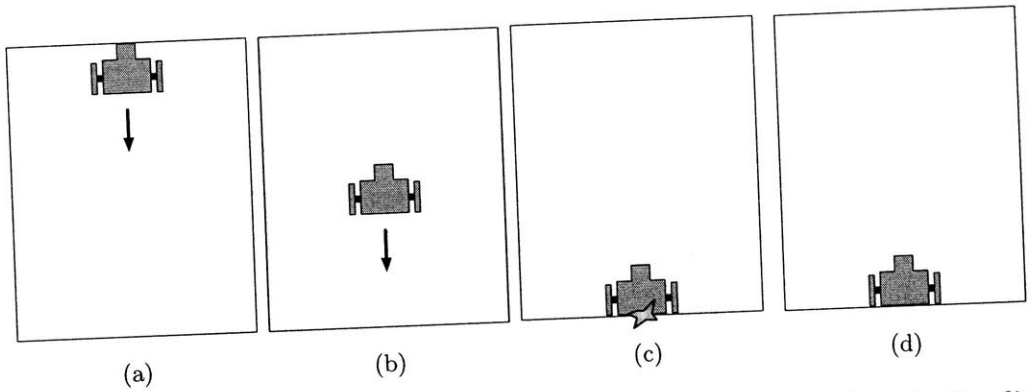


Figure 4-8: The chassis length configuration routine takes place after the wheel radius has been configured. Starting from the position where the wheel radius configuration ended, as shown in (a), the robot begins to move across the test harness. The robot continues to translate until it collides with the test harness. The translation and collision are depicted in (b) and (c), respectively. Once the collision has been detected, the robot can assess the length of its chassis, and is in position to either exit, or repeat the configuration sequence. This end state is shown in (d).

4.3.3 Wheelbase Configuration

Determining the wheelbase characteristics depends on knowing the length of the chassis, as well as being able to translate a specified distance. These values can be obtained by running the chassis length and wheel radius configuration sequences, which are discussed in sections 4.3.1 and 4.3.2 respectively.

The wheelbase width is a calculation that is based on the values of d_f and d_r, as defined in Chapter 3. A twenty three state FSM carries out this task. A high-level diagram depicting the five primary states of the FSM is depicted in Figure 4-9. The details of the two macro-states, DETERMINE-d_r and DETERMINE-d_f, are described in further detail after the high-level description of the implementation. A descriptive diagram of these two states is provided in Figure 4-10, with a sequence of illustrations corresponding to the DETERMINE-d_r state can be found in Figure 4-11.

The wheelbase configuration routine can begin by discovering either d_f or d_r, and obtain the same result. The following description assumes that d_r is configured first.

When the system is started, it is in the RESET state. An assumption is made that the rear of the robot is currently against one of the harness walls. From the RESET state, the configuration routine enters a set of states which determine the length of d_r. These states are represented by the DETERMINE-d_r state in Figure 4-9, and are drawn explicitly in

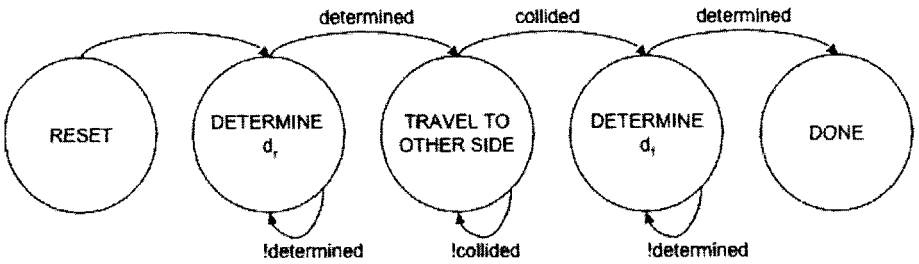


Figure 4-9: This is a simplified state transition diagram for the wheelbase configuration FSM. The two states, DETERMINE-d_r and DETERMINE-d_f, are macro states which consist of eight minor states. The FSM starts in RESET and then proceeds to approximate d_r by incrementally moving away from the wall. The configuration routine remains in the DETERMINE-d_r state until d_r has been determined. The FSM then traverses the harness and repeats the process for d_f. Once d_f has been determined, the configuration sequence is complete, and the FSM transitions to the DONE state.

Figure 4-10.

The ten states within DETERMINE-d_r are shown in Figure 4-10, as well as illustrated in Figure 4-11. An initial guess of d_r is made before entering this macro state. The sequence is straight forward, but requires multiple states to implement. The idea is to increasingly move farther away from the wall of the harness on each iteration. During each iteration, rotate to see if a collision occurs. If so, then d'_r is too small, and the sequence is repeated with a larger d'_r. Repetition continues until a time out occurs. The timeout denotes when the robot is far enough away from the wall of the test harness to complete a ninety degree rotation.

The first state after reset is the MOVE-FORWARD-INCREMENTALLY state. This and MOVE-FORWARD-INCREMENTALLY-2 implement the parts of the algorithm where the robot moves away from the wall by a distance of d'_r. MOVE-FORWARD-INCREMENTALLY is depicted in figures 4-11a and 4-11b.

Once the FSM has moved the robot d'_r away from the wall, it enters the ROTATE-TO-COLLIDE state. Here the robot is commanded to rotate until it either experiences a collision, or times out. The ROTATE-TO-COLLIDE state is depicted in Figure 4-11c.

If the state machine times out during this first rotation, then a recovery strategy is employed. The rotation is undone by rotating in the opposite direction for the same amount of time as the ROTATE-TO-COLLIDE state lasted, and the value of d'_r is decremented instead of incremented.

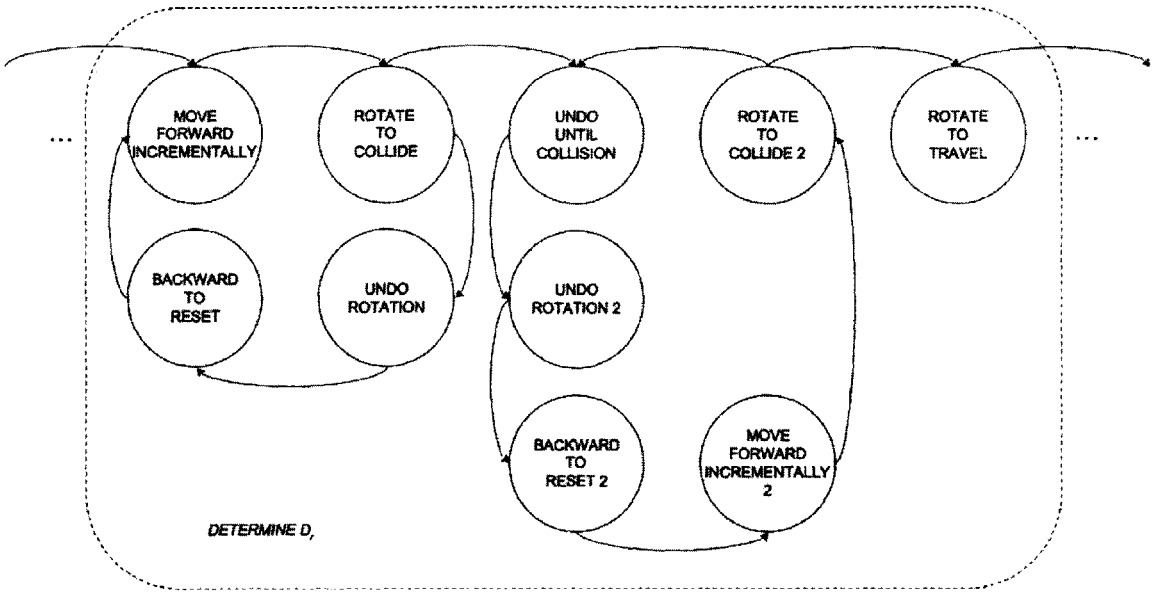


Figure 4-10: This is the state transition diagram that is encompassed by the macro state, DETERMINE-d_r. The FSM implements a procedure which experimentally determines a value for d_r by incrementally moving away from the wall and testing whether or not the robot can rotate without collision.

The recovery rotation is performed during the UNDO-ROTATION state. Next, rather than just moving backward by the amount that d'_r was decremented by, the implementation causes the robot to move all the way back to the harness wall. This translation is performed during the BACKWARD-TO-RESET state and is an attempt to decrease the amount of translation error that has been incurred. Once the robot has been returned to its starting position, the FSM reenters the MOVE-FORWARD-INCREMENTALLY state and repeats the test sequence. The first test of d'_r and the recovery strategy are represented by the leftmost loop of states in Figure 4-10.

If, however, the ROTATE-TO-COLLIDE state does not time out and a collision is detected, as is the case in Figure 4-11d, then the configuration sequence proceeds to the next iteration of the algorithm. The wheelbase configuration algorithm continues by reversing the rotation that it just executed.

In order to minimize any errors due to asymmetry in the motors, the process of returning the robot to its original orientation requires two states. During the first state, UNDO-UNTIL-COLLISION, the FSM rotates the robot in the opposite direction until another collision is experienced. The assumption is that this collision is the other rear corner of the robot chassis colliding with the harness wall. Figures 4-11e and 4-11f illustrate this

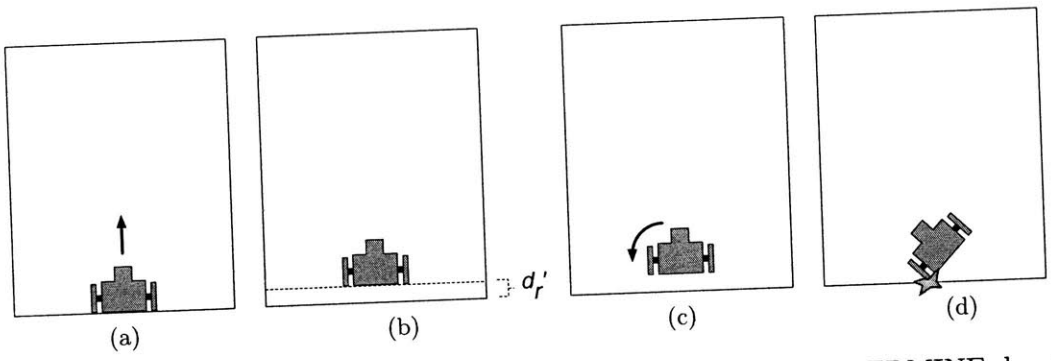


Figure 4-11: This figure illustrates the robot's actions as the DETERMINE-d_r state of the FSM is executed. Figures (a)–(d) show the robot moving forward a distance equal to the rear wheelbase offset guess, d_r'. The robot then proceeds to rotate until a collision is detected. Figures (e)–(j) illustrate the steps taken to return the robot to its initial position. Once the robot has reset itself, it can check to see whether the incremented value, $d_r' + d_{inc}$, is an adequate guess for d_r. Figures (k)–(p) depict this second trial. The result is a timeout and the robot resets itself to travel across the test harness to determine d_f.

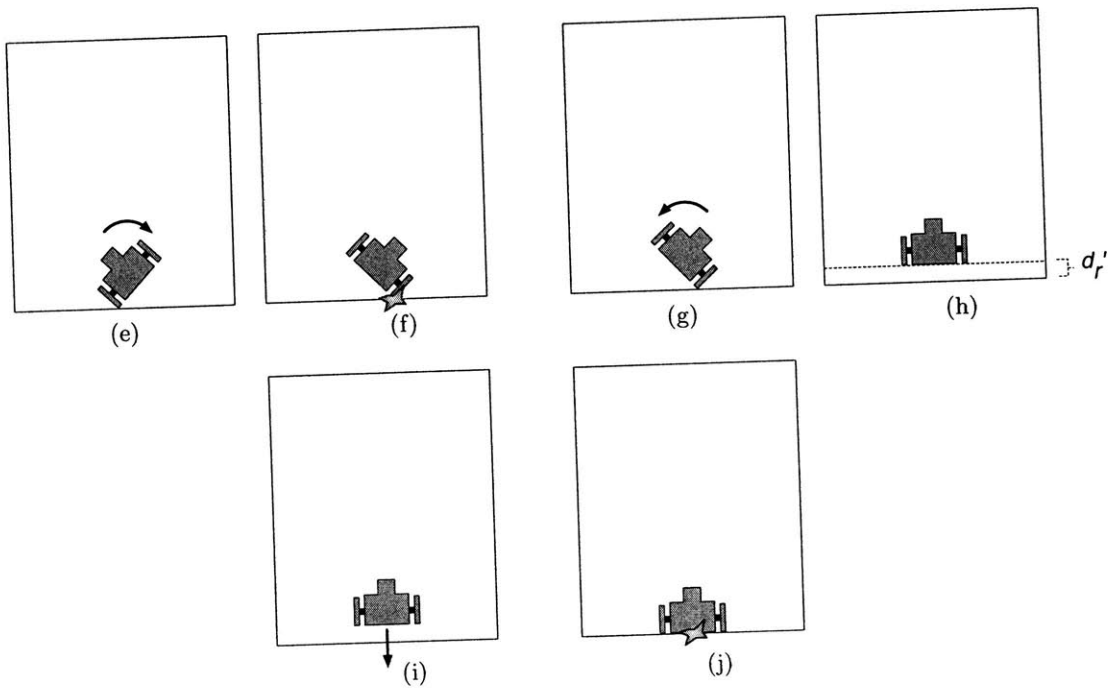


Figure 4-11: Sketch of the Execution of DETERMINE-d_r (continued)

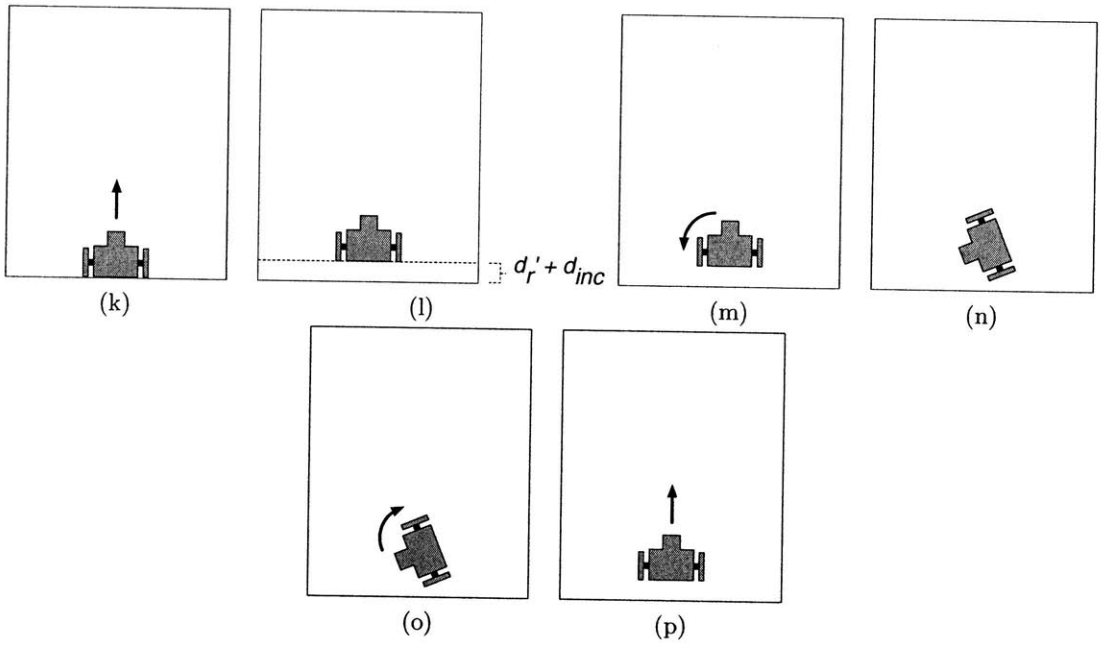


Figure 4-11: Sketch of the Execution of DETERMINE-d_r (continued)

operation.

Next, the robot rotates in the same direction as during the ROTATE-TO-COLLIDE state. This second rotation is shown in Figure 4-11g. The FSM remains in this state until the average of the reset time required for each the ROTATE-TO-COLLIDE and the UNDO-UNTIL-COLLISION states to execute, has transpired. This time interval is the average between half of the duration of the UNDO-UNTIL-COLLISION state and all of the duration of the ROTATE-TO-COLLIDE state. By using the average time, it is more likely that the robot will reset itself to the orientation it had before any rotation began, as is the case in Figure 4-11h. This second step for resetting the robot's orientation, which utilizes the average time to collision, occurs during the UNDO-ROTATION-2 state.

After the robot has returned to its starting orientation, it is commanded to back up until it collides with the test harness wall. This action occurs during the BACKWARD-TO-RESET-2 state, and is intended to reduce the amount of translation error that has been incurred. The illustrations corresponding to this action can be found in Figures 4-11i and 4-11j. When this state is completed, the value of d_r' is incremented and the procedure transitions to the MOVE-FORWARD-INCREMENTALLY-2 state.

At this point in the procedure, the incremental testing sequence is repeated. Now, the goal is to find the next occasion for the state to time out. The illustrative figure shows this

second sequence in figures 4-11k–4-11n.

If a collision is observed, the process for updating d_r' repeats by entering the UNDO-UNTIL-COLLISION state. On the other hand, if a collision is not observed before the timeout, then it is assumed that the robot was able to rotate past the point where its chassis would collide with the test harness boundary. This timeout is the situation illustrated by Figure 4-11n.

Following this second case, the configuration routine enters the ROTATE-TO-TRAVEL state, where it undoes the rotation that was performed in ROTATE-TO-COLLIDE-2. This resetting rotation is shown in Figure 4-11o. When the robot has returned to its original orientation, then the macro state is complete and the FSM can proceed to travel across the test harness. Figure 4-11p depicts the robot as it begins to travel across the test harness.

Referring back to Figure 4-9, traveling across the test harness to measure d_f is implemented by the TRAVEL-TO-OTHER-SIDE state. Once the robot collides with the opposing wall, the wheelbase configuration process enters the DETERMINE-d_f state. This state only differs from DETERMINE-d_r in that all of the translation and rotation directions are reversed.

Once the configuration routine has completed the DETERMINE-d_f macro state, the FSM has determined the values of d_r and d_f, and it enters the DONE state, signalling that the configuration sequence is complete.

4.4 Configuration Experiments

Once the different configuration components have been implemented, it is possible to perform tests which demonstrate the limits of the system. In the case of each parameter, the following tests show the operating region of the different configuration routines. While these results are specific to the microbot used in testing, the results can be used as a guideline for choosing appropriate initial values for future or similar configuration system implementations.

4.4.1 Wheel Radius Configuration

The experiments presented in this subsection involve varying the initial radius guess, the commanded translational velocity, and the distance that is being traveled and observing the

resulting value of the wheel radius configuration routine. The graphs resulting from these experiments will show the range of initial values for which the configuration routine gives optimal results.

Varying the Initial Radius Guess

One of the parameters which can potentially affect the outcome of this configuration routine is the initial radius guess, r'_w. The outcome of this experiment will demonstrate that the initial guess of the radius does not matter. The only anticipated situations where the radius parameter could affect the outcome is where the chosen value pushes the physical limits of the underlying robot hardware.

In this experiment, the robot is started at a distance d from the boundary of the test harness with a variety of initial wheel radius guesses. The value of the commanded velocity, v', is set to 0.03 m/s and the travel distance d is 0.61 m for the duration of the experiment. The value of the initial r'_w is varied from 0.02 m to 0.25 m at 0.01 m intervals. The actual wheel radius, 0.0625 m, does not fall into this set of values, so it is analyzed in addition to the other values. Each radius experiment is performed three times to create a local average value. Between each trial, the robot is manually returned to the starting location and setup for the next experiment.

Varying the Starting Location

In order to show the importance of the initial robot placement, trials using a manual restart are performed using a variety of values for d. The results of this experiment should show that the computed value is larger if the robot is started closer to the wall, than if it is started farther away from the wall.

The value of v' is set to 0.03 m/s and the initial value of r'_w is one eighth of the test harness length: 0.1525 m. The values of d will range from 0.05 m to 0.80 m at 0.05 m intervals, and three trials will be run at each value of d.

Varying the Commanded Translational Velocity

The initial parameter representing the commanded velocity, v', does not matter except for physical limitations. These limitations will be shown to be negligible. In order to demonstrate that the chosen v' can be arbitrary, trials are executed at different velocities

while holding all other variables constant. Specifically, these other variables are the values r'_w and d.

For this experiment, the initial r'_w is 0.1525 m, and d is 0.61 m. Three trials will be performed at each velocity setting, and the values of v' will range from 0.01 m/s to 0.1 m/s at intervals of 0.01 m/s.

4.4.2 Chassis Length Configuration

The parameters used for chassis length configuration are wheel radius and translational velocity. Values which can be used for each of these two parameters can be determined by performing the following experiments. Results will show that given that the wheel radius configuration routine works, the chassis length configuration will yield values close to the user-measured value when carried out in the specified velocity range. Furthermore, the configuration yields better chassis length results through repetition.

Dependence on Wheel Radius

The success of the chassis length configuration routine depends on the accuracy of the configured wheel radius. In order to demonstrate this relationship, the chassis length configuration process can be performed using different wheel estimates. The results will most likely show a linear dependence of the configured chassis length on the configured wheel radius.

This experiment is conducted by placing the robot in the starting position for chassis length configuration. Next, trials are conducted with different values for r_w. The values range from 0.055 m to 0.075 m at 0.0025 m intervals. The commanded velocity, v, is 0.03 m/s throughout the experiment. Each trial lasts for four traversals of the test harness.

Independence of Velocity

The result of the chassis length configuration is independent of the chosen translational velocity, v. This has practical limitations, such as robot movement may become jerky if v is too small or may become dangerous if v is too large; but it should be possible to show that the velocity has a minimal effect on the resulting chassis calculation.

This experiment begins by placing the robot in the starting position for chassis length configuration. The velocity, v, will range from 0.01 m/s to 0.10 m/s at 0.01 m/s intervals.

Three trials are run at each value of v, and the wheel radius parameter is set to the user-measured value: 0.0625 m.

Repetition for Convergence

Since the chassis length configuration routine is repeatable, it can be shown to converge on a chassis length after several iterations. In order to demonstrate this, two sets of experiments will be performed. One set will use the user-measured wheel radius, and the other set will use a self-configured wheel radius. Each set of experiments consists of running three sets of the chassis length configuration routine, consisting of 21 repetitions each.

4.4.3 Wheelbase Configuration

The wheelbase configuration routine is dependent on the results of both the wheel radius and chassis length configurations. The results of the experiments under their respective sections will show that the wheel radius and chassis length will consistently be close to the user-measured values. Therefore, it is not necessary to vary those values in order to show the appropriate values to use for the wheelbase configuration routine.

However, the wheelbase configuration still depends on two parameters which depend on the system implementation. These are the initial wheelbase guess and the commanded rotational velocity. Appropriate implementation values for these parameters will be determined by running the experiments described below.

Varying the Initial Wheelbase Guess

While the theory of the wheelbase configuration routine works under all conditions, the practical realization may not. Parameters that are either too large or too small with respect to the actual wheelbase may cause inaccurate measurements.

One of the parameters which may affect the resulting wheelbase configuration is the initial guess for the wheelbase width, b'. If b' is too large with respect to the actual wheelbase width, b, then it is possible that the resulting rotation will be at a velocity that is much greater than intended. A similar issue exists for when b' is too small – rotation becomes spasmodic. Under these assumptions, the result of varying the initial b' will be poor results for large values and small values, but reasonable results within a midrange of values.

In order to test this property, the experiment is setup with the user-measured wheel radius and chassis length, leaving the initial value of b' free to vary from 0.50 m to 0.02 m at intervals of 0.02 m. Each experiment will consist of three trials in order to build up a local average at each point.

Varying the Commanded Rotational Velocity

Using different commanded velocities should be equivalent to changing the initial wheelbase guess. An experiment which can test the effect of different rotational velocities on the resulting self-configured wheelbase measurement can be performed by using different velocities over a set of initial wheelbase guesses.

The effects of the commanded rotational velocity on the resulting wheelbase measurement are demonstrated as follows. Wheelbase measurements that result from the wheelbase configuration method will be recorded. The configuration routine will be executed using an initial b' that ranges from 0.195 m to 0.780 m at intervals of 0.0975 m, and a commanded rotational velocity that ranges from 0.01 m/s to 0.09 m/s at intervals of 0.02 m/s.

4.4.4 End-to-End Test

In order to evaluate the utility of the self-configuration method, it is necessary to use a satisfactory method for comparing results. Since the sensors that are being configured are odometric in nature, it follows that the UMBmark, described in Section 2.3, is a suitable method for comparing the performance of the self-configured parameters with to the performance of the user-measured parameters.

The UMBmark measures the maximum amount of systematic error incurred under typical robot movement using a given set of configuration parameters. Systematic error is measured by the UMBmark by commanding the robot to translate in a 4 meter by 4 meter square in both the clockwise and counterclockwise directions. During the test, the robot keeps track of its position internally. When the robot has completed a given circuit, the distance between the starting point and the final location is measured. Once all trials have been completed, the error between the final internal and actual locations is calculated for both the clockwise and counterclockwise sets of data. Two centers of gravity are then calculated using the two data sets. The center of gravity that is geometrically farthest from the origin is used as the maximum systematic error for the robot.

In the UMBmark paper [3], the goal is to determine corrections that can be made to the parameters which a given robot system is currently using. These corrections are easier to make if the points are clustered closely together than if the points are scattered.

Overview of Software Implementation

The UMBmark can be implemented as a six state finite state machine that uses counters to keep track of the number of rotations and translations that have occurred. A state transition diagram for such an FSM is shown in Figure 4-12.

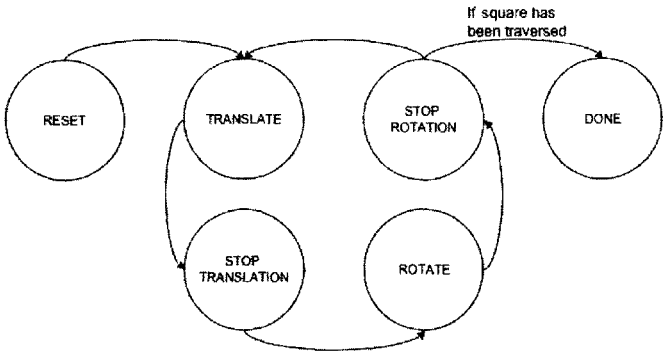


Figure 4-12: The UMBmark can be implemented as a six state FSM as shown here. Counters are used to keep track of the number of translations and rotations that have occurred. When the test is complete, the FSM exits.

The RESET state initializes the benchmark FSM by storing the starting location. Once this has occurred, the test can continue by entering the TRANSLATE state. During each step of the TRANSLATE state, the change in x and y of the CARMEN messages is calculated and added to a vector representing the total distance traveled. Once the magnitude of the travel vector reaches 4 meters, as required by the UMBmark, the FSM increments the number of translations that have been performed, and then transitions to the STOP-TRANSLATION state. If the total distance traveled is not 4 meters, the FSM remains in the TRANSLATE state.

The FSM remains in the STOP-TRANSLATION state until the robot comes to a stop. At this point, the FSM enters the ROTATE state. In this state, the robot is commanded to rotate in the specified direction until the summation of the change in theta between CARMEN messages sums to $\frac{\pi}{2}$ radians. At this point, the FSM enters the STOP-ROTATION

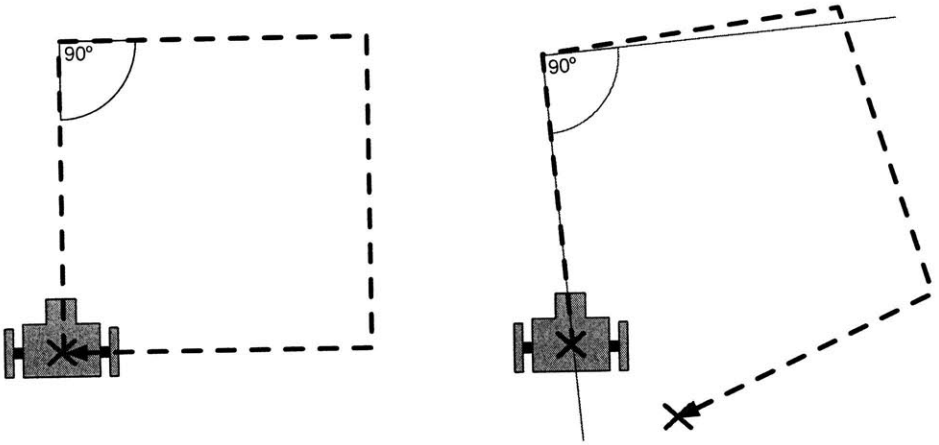


Figure 4-13: The UMBmark FSM, depicted in Figure 4-12, has an intended and actual effect. The ideal case is shown in (a). In this instance, all of the lengths are four meters, and the turns are ninety degrees. In the real world case, shown in (b), the translated distances are not straight lines that are four meters in length, and the angles are not of the prescribed magnitude.

state, where rotation stops. If enough translations and rotations have been executed to complete the 4 meter circuit, then the FSM enters the DONE state, signaling that the UMBmark is complete and that the operator may now measure the distance from the starting location to the final position of the robot. If the test is not yet complete, then the FSM returns to the TRANSLATE state to complete another leg of the square. Figure 4-13 shows the result of running the UMBmark in an ideal world (Figure 4-13a) and a realistic world (Figure 4-13b). Essentially, having the robot stop where it started is preferred, but in general, this is not the case.

Executing the UMBmark

One trial of the UMBmark consists of having the robot complete one circuit around the 4 meter square. Completing the UMBmark requires performing a total of ten trials consisting of five clockwise and counterclockwise trials.

In order to ensure accurate measurements, It is important to minimize the amount of human error that occurs during each trial. These errors include the ability to place the

robot in the same starting location on each trial, as well as accurately measuring the x, y distance between the starting and final locations of the robot. In order to make the measurements as accurate as possible, two perpendicular walls as well as tile lines are used as reference points.

The point of reference on the robot frame is the center of the wheelbase. In order to be consistent between measurements, a systematic method is used to mark this reference point on the ground. The method consists of marking the ground next to the robot's wheels with tape, moving the robot out of the way, and then marking the halfway point between the two tape-markers. The halfway point is then also marked with two pieces of tape: one piece is along the wheel axis, and the other piece is perpendicular. Figure 4-14 illustrates the taping scheme used for marking the location of a wheel's axle. The method for marking the point of rotation is shown in Figure 4-15.

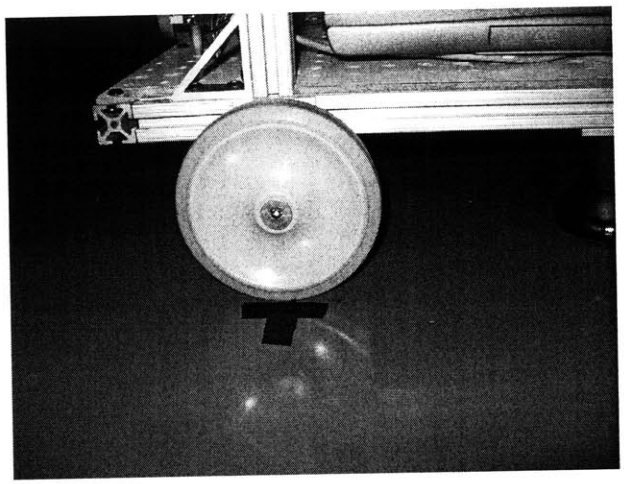


Figure 4-14: The location of the axle of the microbot is marked on the ground as shown in this image. Two strips of tape are used next to each wheel. One piece is inline with the wheel axis and the second piece is placed perpendicular to this orientation, forming a 'T'.

The four pieces of tape that comprise the two wheelbase markers serve a second purpose. By using this taping strategy to mark the starting location of the UMBmark trials, it becomes easier to consistently align the robot on each trial. The robot must be placed so that the wheels are along the tape that is parallel to them, and so that the axles are aligned with the tape that is perpendicular to the direction of translation.

Before the start of each trial, the bolts of the robot are tightened in order to ensure that results remain consistent between trials.

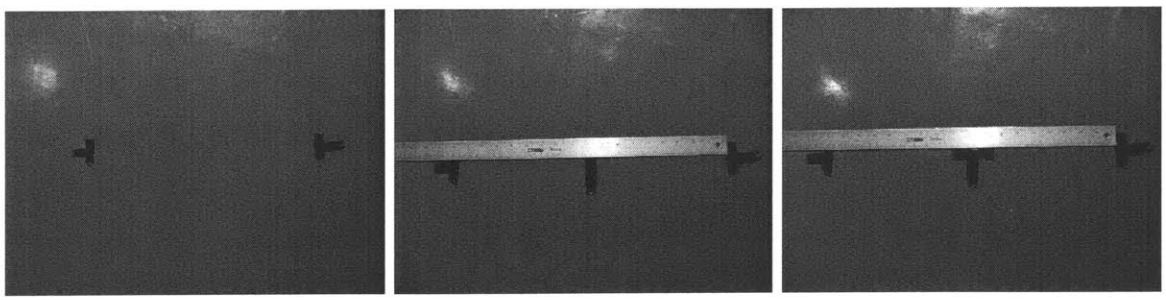


Figure 4-15: The point of rotation is marked using the first two markers. This is accomplished by placing a measuring stick between the two wheelbase markers, and marking the halfway point with a piece of tape that is perpendicular to the wheelbase. A second piece of tape is then placed along the direction of the wheelbase. It is from this midpoint marker that error measurements are made.

After the starting location has been marked with tape, two lines of tape, one leading in the direction that the robot is to travel and the other in the direction that the robot is to return from, are laid out to serve as a reference point for measurements. This taping scheme can be seen in Figure 4-16. The idea is that once the UMBmark trial has completed and the location of the robot has been marked, the distance from the starting location can be measured in terms of x and y.

Once these initial preparations have been made, the UMBmark software can be run. When the benchmark has completed, the internal measurement, as reported by the program, is recorded and tape is laid out to mark the final location of the robot. This tape is placed in the same manner as the markers that denote the starting location of the robot.

When the reference point of the robot has been marked on the ground, the distance between this point and the starting location is carefully measured. The distance between the start and final locations is recorded in terms of x and y. First, the distance from the final location of the robot to the taped reference line is measured, giving the value of y. Next, the value of x is determined by measuring the distance from this point of intersection to the starting location.

Using the notation from the Bornstein paper [3], the return position errors can be described as

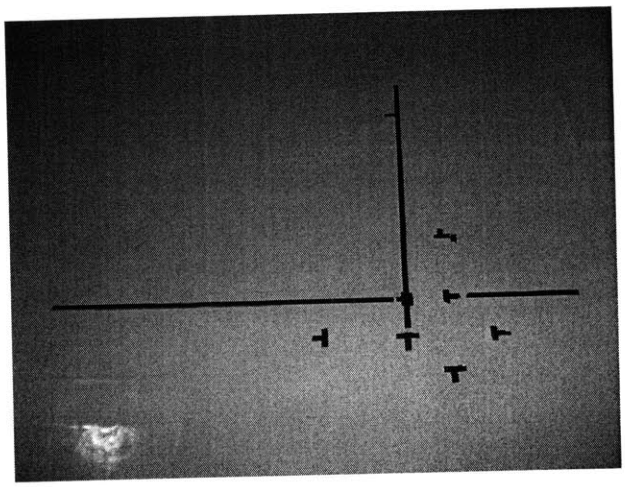


Figure 4-16: In order to reduce the human error that is incurred on each trial, a taping strategy is used to mark the starting locations for the UMBmark, as well as to serve as a reference point in measuring the absolute final position of the robot. When placing the robot for each trial, the wheels are aligned with the two outermost tape marks. As a secondary precaution, the front of the robot, when viewed from above, is aligned with a tape as well. Reference lines of tape are marked out from each starting location so that the final error measurement is more accurate. The clockwise and counterclockwise sets of starting tape, as well as the reference tape, can be seen in this figure.

$$\epsilon_x = x_{abs} - x_{calc}$$

$$\epsilon_y = y_{abs} - y_{calc}$$

where $\{\epsilon_x, \epsilon_y\}$ represent the return position error in the x and y direction, respectively; $\{x_{abs}, y_{abs}\}$ represents the human measured displacement of the robot from the origin when a given trial has been completed; and $\{x_{calc}, y_{calc}\}$ denote the calculated internal position of the robot, for the same given trial. This concept is also shown in Figure 4-17.

The UMBmark is run a total of ten times. Five of the trials are carried out with the robot traveling in the clockwise direction, and the other five are performed in the counterclockwise direction. When all trials have been completed, the center of gravity for each of the two data sets is calculated. The center of gravity which has the greatest euclidean distance from the origin is taken to be the maximum odometric error due to systematic errors.

Graphically plotting the results of the user- and self-configured points will allow for a quick evaluation of the results for each set of configuration parameters. The expectation is

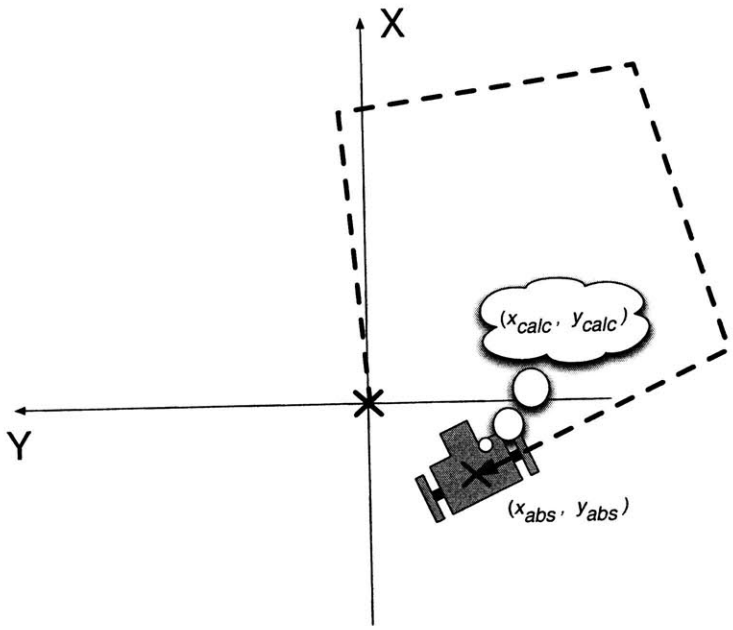


Figure 4-17: The UMBmark paper defines the error on return to be the difference between the absolute robot position and the calculated, internal robot position. This concept is shown in this figure with the calculated value being where the robot thinks it is, and the absolute value being the real-world, final position of the robot, given by the distance from the black "x" to the red "x". The black "x" marks the start position of the benchmark, and the red "x" marks the end position.

that running the UMBmark with the self-configured parameters will yield results that are comparable to, if not better than, the results of using the user-measured parameters.

Chapter 5

Results

This chapter presents the results of the self-configuration system. Beginning with the setup, the outcome of each experiment is described in detail, focusing primarily on the results and then an analysis of what the results mean and how they can be utilized.

5.1 Preliminaries

Before being able to implement the self-configuration system, it was necessary to perform the tasks that were outlined in Section 4.2. These tasks essentially consisted of the construction of the physical test harness and the implementation of the current-based bump sensor.

5.1.1 Measurement of Microbot Parameters

The alternative to using a self-configuration process is to have the robot's user manually measure the parameters of the robot. Table 5.1 contains the user-measured values of the microbot. Measurements were made using a meter stick with millimeter precision.

Table 5.1 contains some values which have been mentioned in other sections as well as two new ones. As before, r_w represents the radius of the microbot's wheels; l represents the chassis length; b represents the wheelbase width; l_f and l_r represent the distance from the wheelbase to the front and rear of the robot, respectively; d_f and d_r denote the distance by which the front and rear radii, of the robot chassis, exceed the values l_f and l_r. The two new values are b_l and b_r, which represent the distance by which the left and right wheel, respectively, jut out from the robot chassis. Since the wheelbase configuration routine does

Table 5.1: User-measured Configuration Parameters for Microbot

Parameter	Value
r_w	0.0615 m
l	0.3815 m
b	0.3795 m
l_f	0.1168 m
l_r	0.2647 m
d_f	0.1060 m
d_r	0.061 m
b_l	0.0280 m
b_r	0.0255 m

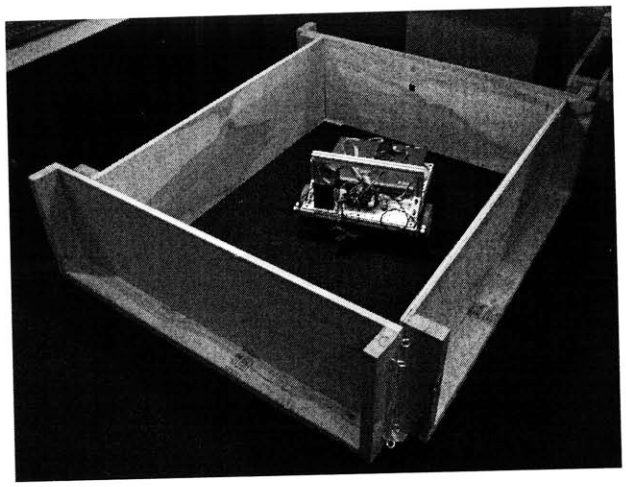


Figure 5-1: The physical test harness is easy to assemble from existing class materials. Dimensions of the harness are 1.22 m by 1.00 m and give enough room for the microbot to discover its configuration parameters while still providing enough difference between the length and width to be discernible.

not provide a mechanism for measuring these values, they are taken as given for microbot-class robots. These two values are used in both the user- and self-configured setups.

5.1.2 Construction of Physical Test Harness

Assembly of the test harness, using the maze pieces from the RSS grand challenge, resulted in the test harness that is depicted in Figure 5-1. This test harness is quick to construct and disassemble because the pieces are merely placed next to each other.

5.1.3 Implementation of Plotting Utility

The plotting utility proved to be an essential tool as it allowed for rapid prototyping of filters as well as real-time calculation of mean and variance of data signals. It is constructed using Java's Swing library and useful for live visualization of data points. Interfacing with the plotting tool is simple. The instantiating module merely provides a vector containing elements of type `double`, and the plot is updated and scaled in order to display all data points.

Two screen shots of the utility, as it displays sonar data, can be seen in Figure 5-2. The sonar samples are being generated by two co-linear sonars, that are mounted on opposite sides of the microbot, as the robot is rotating within the test harness. Implemented filters consist of identity filter, mean filter, and difference filter. The filter over the data can be changed during use. Additionally, the number of neighbors affected by the mean filter can be modified from 0 to 10, for a maximum of 21 points included in the computation of each point. A line representing the mean, as well as plus or minus one or two standard deviations, can be toggled on and off during use. This feature is helpful for determining an effective threshold for sample points.

5.1.4 Current-Based Bump Sensor

Once the current-sense functionality of the ORCBoard was completely integrated into CARMEN, it became possible to display the samples using the plotting utility described in Section 5.1.3. As predicted, both a collision and a commanded acceleration yielded similar current data. The surprise was that simple thresholding, coupled with the state machine implementation, yielded a robust result. A plot which shows a collision at 0.03 m/s compared to a commanded acceleration from 0.03 m/s to 0.06 m/s is shown in Figure 5-3. Based on the results of the characterization experiments, the current-based collision detection FSM was implemented by reporting when sample points exceeded a threshold of $\mu \pm 3.5\sigma$.

5.2 Self-Configuration

After completing the preliminary steps, the different configuration components of the system were implemented in the order of wheel radius, chassis length, wheelbase. Once a given segment had been implemented, the associated parameter experiments from Section 4.4

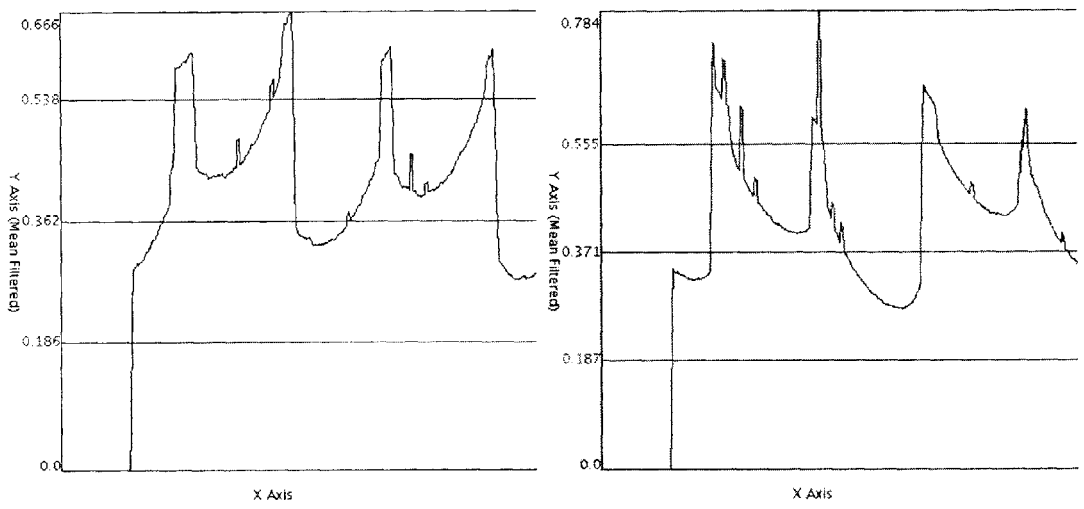


Figure 5-2: These two plots consist of readings from left and right sonar sensors, which have been passed through a mean filter that includes the 2 nearest neighbors, versus time (given in samples of approximately 10 Hz) taken while the microbot was rotating clockwise in the test harness at a rotational velocity of 0.05 radians per second. The black line across the middle represents the mean over all values within the viewing window, and the red lines are $\mu + \sigma$ and $\mu - \sigma$ respectively, where μ represents the mean and σ represents the standard deviation.

were carried out. This enabled each component of the system to be configured optimally for the subsequent configuration routine.

This section concludes with a summary of results from running the full system configuration, and the outcome of the results from the UMBmark.

5.2.1 Results of Wheel Radius Configuration

The wheel radius configuration component consists of a simple FSM which drives the robot forward until a collision is detected. The implementation of this component was successful once some initial issues were resolved. The rest of this section covers the implementation issues and their resolution, and the outcome of the experiments which were conducted on this component.

Implementation Issues

During the implementation of the wheel radius configuration component, it was discovered that CARMEN does not support on-line modification of some configuration parameters. Furthermore, the wheel size was hardcoded into the low-level hardware integration feature

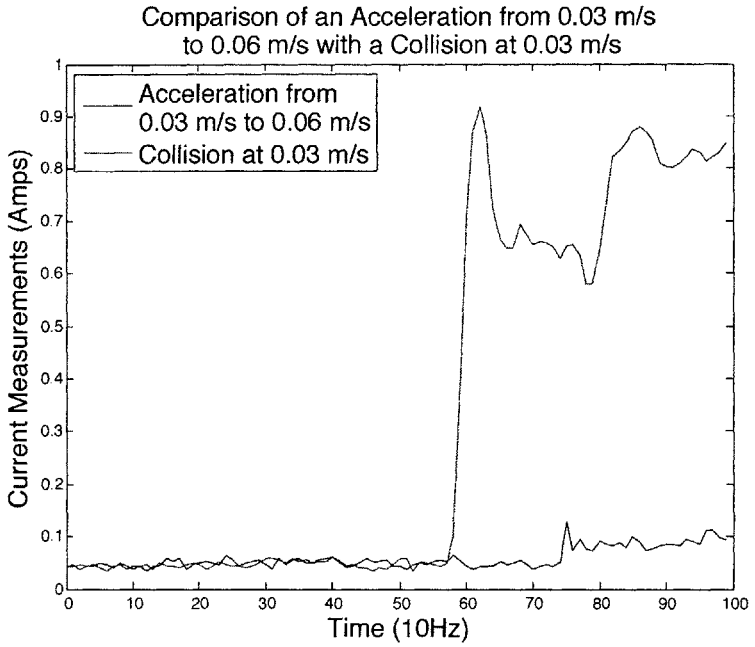


Figure 5-3: Both graphs were generated with the robot moving at 0.03 m/s. The solid blue line is a graph of an acceleration to 0.06 m/s and the solid red line marks a collision with a wall of the test harness. Both events can be seen on their respective graphs as a step. The acceleration happens around 7.5 seconds, and the collision occurs around 5.8 seconds.

of the ORCBoard driver.

It turns out that if the hardware integration feature is enabled, which is the default for RSS-CARMEN, then the wheel size that is specified through the typical parameter setting channels, either via the initialization file or by `param_daemon` messages, is ignored. This issue was resolved by putting the radius and wheelbase scaling values in the appropriate locations of the `base_main` module.

Once these changes were made to `base_main`, updates to the wheel size parameter were still not taking effect. The cause of this is that, by default, CARMEN's base module only sets values for parameters such as wheel size during the initialization sequence. In order to allow for on-line updates of the wheel size to take effect, it is necessary to alter `base_main` so that it updates the wheel size and wheelbase parameters whenever messages, indicating that the value has changed, are received.

These changes to the CARMEN modules make it possible to modify CARMEN system parameters, such as the wheel radius and wheelbase width, after system initialization has occurred and the CARMEN-based application is running.

Varying the Initial Radius Guess

The results of varying the initial radius size are shown in Figure 5-4. Each data point is the average over three trials that were performed using the corresponding initial radius value. The initial radius values range from 0.02 m to 0.25 m at 0.01 m intervals. Every trial is performed using a velocity of 0.03 m/s and starting from a distance of 0.61 m away from the targeted harness boundary.

The user-measured value of the microbot's wheel radius was 0.0615 m. When the initial radius guess of the configuration system is much smaller than the actual radius, the resulting radius measurements have more error than when the radius guess is much larger than the actual radius length. This behavior can be seen in Figure 5-4 by noting that the configured values which result from initial radius guesses closer to zero meters are farther away from the user-measured value than the data points that are closer to 0.250 m. Even so, radius guesses that are much larger than the actual radius length also have larger error than the middle-range values. This is most likely due to the jerky, and thus imprecise, movement that results from using large radius values. Overall, the resulting radius values have a linear shape that starts below the actual wheel radius and slopes upwards to values that are larger

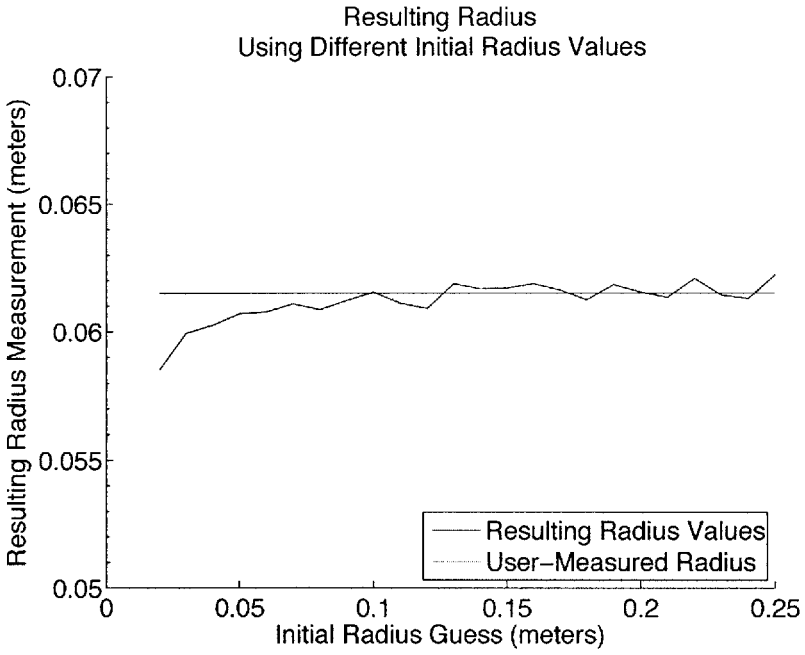


Figure 5-4: The graph shows the effect of varying the initial radius guess on the resulting radius measurement. The resulting radius values have a large amount of error when the initial wheel radius guess is much smaller than the actual wheel radius. The magnitude of the error from the user-measured value decreases as the initial radius guess is increased. The horizontal line at 0.0615 m is the user-measured radius.

than the actual wheel radius.

The maximum error of a self-configured radius value from the user-measured radius was 0.0032 m, and this occurred when using an initial radius of 0.02 m. The average error, taken over all data values was less than 0.0006 m. This means that the self-configured value is safely within 3 mm of the value that would have been measured by a human, and often within 1 mm. The average accuracy of the self-configured radius value, when compared to the user-measured value is 0.9908. Judging from these results, it seems that choosing any initial value that is between 32% and 400% of the actual wheel radius should be sufficient for a self-configuration system.

Varying the Starting Location

As the distance from the starting location to the collision wall increases from 0.05 m to 0.80 m, so does the accuracy of the resulting radius measurement. This relationship can be seen in the graph contained in Figure 5-5. The graph shows the value of d, that was used, on the x-axis and the resulting radius measurement on the y-axis. Each of the data points is the average of running three trials at the specified value of d with a commanded velocity of 0.03 m/s and an initial wheel radius of 0.1525 m.

The maximum error of a self-configured radius value, during this experiment, was 0.0106 m. This peak error occurred when running the trial that started with $d = 0.05$ m. The average error, overall, was 0.0017 m. The average accuracy of the radius values, when compared to the user-measured radius was 0.9726. Just as in the case of varying the initial radius guesses, the resulting error is small enough that a human could just as easily make the same mistake.

Additionally, the importance of the initial placement of the microbot is shown in Figure 5-6. It can be seen that there is a linear dependence on the initial user-placement of the robot. This is not a surprising result and, furthermore, shows that the configuration system functions as expected.

Varying the Commanded Translational Velocity

Corresponding to the results of varying the initial radius guess, varying the commanded translational velocity resulted in a linear graph with more error in the region with higher velocities. The results of the experiment are plotted in a graph shown in Figure 5-7. The

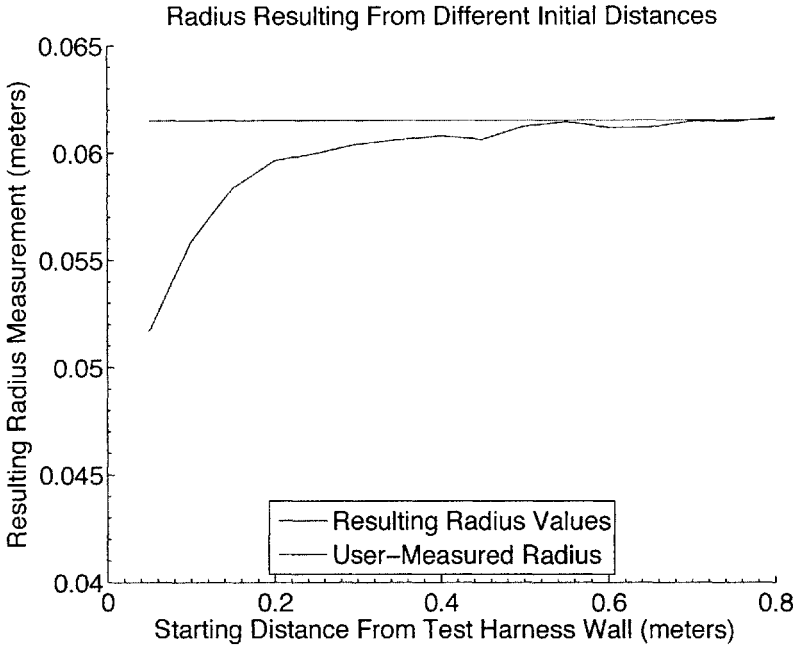


Figure 5-5: The reported wheel radius value becomes more accurate as the agreed distance, d, from the target wall is increased. The maximum error of the self-configured wheel radius from the user-measured wheel radius is 0.0106 m and occurs when d is 0.05 m. This graph shows how the resulting wheel radius value changes as d is varied from 0.05 m to 0.80 m, and compares the data to the user-measured wheel radius of 0.0615 m.

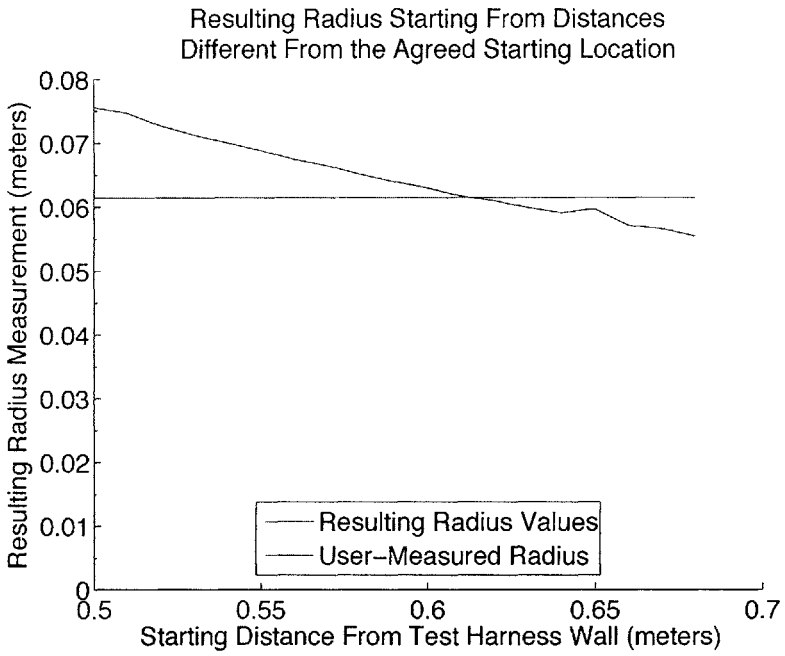


Figure 5-6: The radius measurement that results from the self-configuration routine is linearly dependent on the robot's initial placement for the configuration routine. The farther the robot is placed from from the agreed starting location, the more error is observed in the resulting wheel radius value.

data points represent the average measurement over three trials performed using the given commanded velocity. Each trial began at 0.61 m from the test harness wall and used an initial radius guess of 0.1525 m. The velocities range from 0.02 m/s to 0.10 m/s at intervals of 0.01 m/s. The translational velocity of 0.01 m/s failed to perform as the number of false collisions detected by the CBS were too numerous.

When the velocity is very low, the robot moves in a jerky manner, equivalent to having a very large radius, which can cause false collisions to be detected by the CBS. If the trial was executed to completion, however, the resulting radius measurement was very close to the user-measured value. As the commanded velocity was increased, the radius measurement remained consistent, but caused the wheel radius configuration component to report a radius that is too small in the trials involving a greater velocity.

The maximum error over all the trials occurred when the translational velocity was 0.10 m/s. This error value was 0.0014 m. The mean error value was less than 0.0005 m. The corresponding average accuracy of the radius values, when compared to the user-measured value was 0.9921. In conclusion, any translational velocity, that does not cause the robot

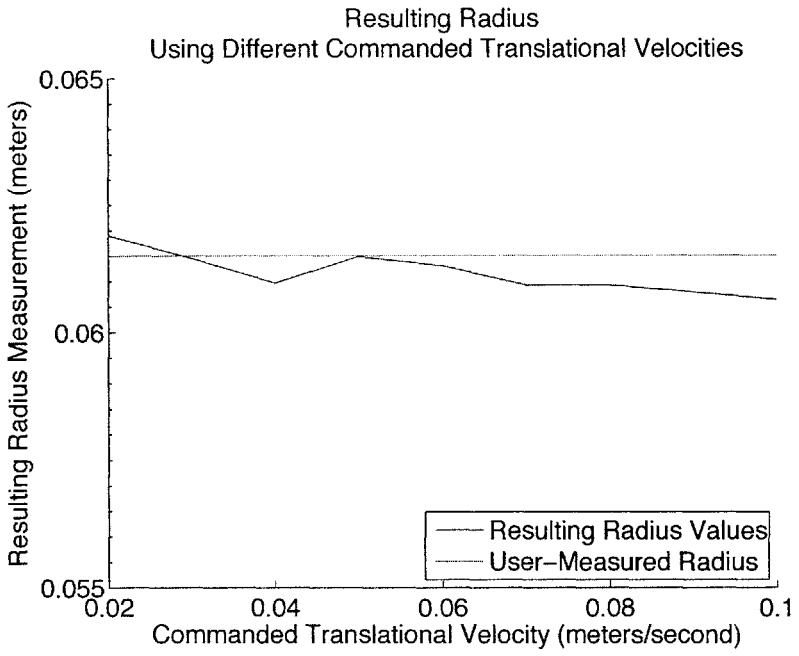


Figure 5-7: The error of the resulting radius measurement increases as the commanded translational velocity is increased. The maximum error is 0.0014 m and occurs during the trial conducted at a velocity of 0.10 m/s. The average error over all of the trials is less than 1 mm from the user-measured radius value. This graph compares the self-configured radius values that result from using a commanded translational velocity between 0.02 and 0.10 m/s to the user-measured radius value.

to exhibit spasmodic behavior, can be used to run the radius configuration step.

5.2.2 Results of Chassis Length Configuration

There were no problems encountered in implementing the chassis configuration component. The following experiments provided bounds on the operating region of the chassis length configuration component. First, it is shown how the measured wheel radius affects the resulting chassis length value. Then, the effects of using different commanded velocities are shown to have a minimal effect on the resulting chassis length measurement. Finally, the fact that this component consists of a repeatable experiment is used to converge on a chassis length over the course of multiple repetitions of the configuration routine.

Dependence on Wheel Radius

The wheel radius is varied from 0.055 m to 0.075 m at 0.0025 m intervals to demonstrate that the wheel radius that is provided to this component has a linear effect on the resulting chassis length calculation. The results of the experiment are shown in Figure 5-8. The horizontal and vertical lines are the user-measured chassis length and wheel radius, respectively. The line with negative slope is the set of chassis lengths which resulted from the corresponding wheel radius.

The graph in Figure 5-8 also shows the error that is inherent in the system. When the wheel radius used is equivalent to the user-measured wheel radius, there is a small, visible error. This error can be seen as the distance between the two points of intersection on the vertical line.

Independence of Velocity

The chassis length that is measured by the configuration component is largely independent of the velocity used for the configuration routine. Figure 5-9 shows the results of the experiment which varies the commanded translational velocity while using the user-measured wheel radius of 0.0615 m for each trial. The user-measured chassis length is 0.3815 m, and corresponds to the horizontal line in the figure.

Four trials were performed at each commanded translational velocity in order to create a local average. The error between the configuration result and the user-measured chassis length varies, but increases linearly with the magnitude of the velocity. As a result, the

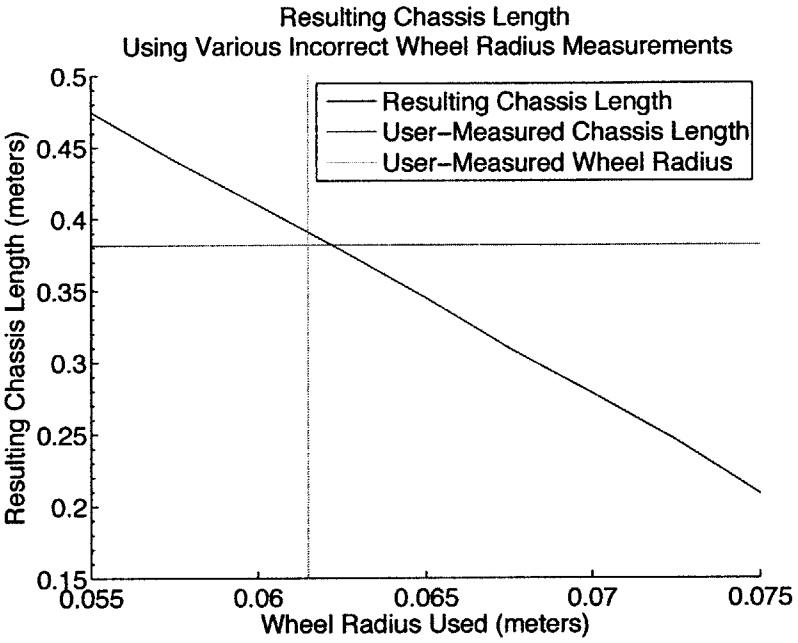


Figure 5-8: The resulting chassis length measurement is directly proportional to the previously configured wheel radius. The plot shows wheel radius values on the x-axis and the resulting chassis length calculations on the y-axis. As the wheel radius used in this experiment strays from the user-measured value, the resulting chassis length exhibits an inverse linear relation. This shows the dependence of the chassis length configuration routine on the value of the wheel radius that is being used, and shows that the configuration component is working as desired.

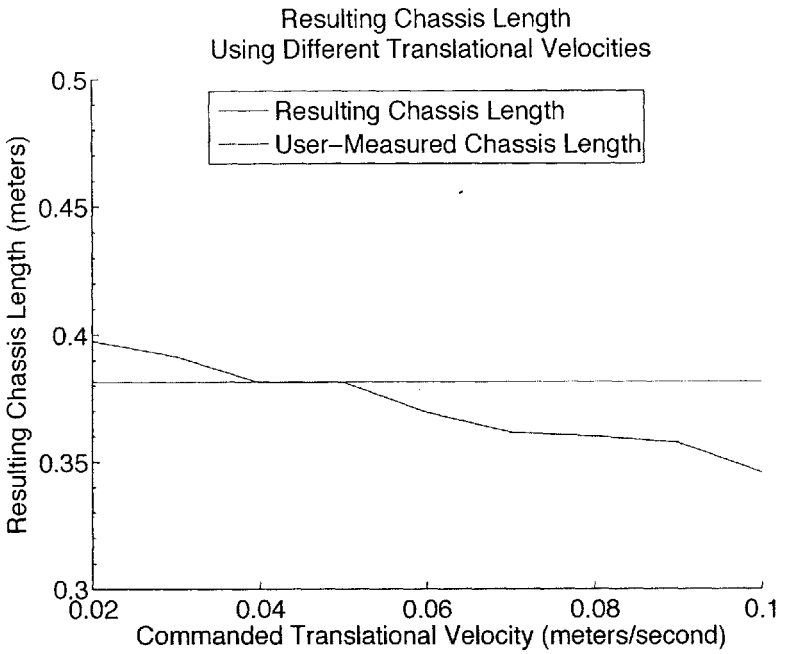


Figure 5-9: The length of the chassis is largely independent of the velocity at which the experiment is performed. Data points represent the average result of three trials that were conducted at the specified velocity using the actual wheel radius of 0.0625 m.

maximum error, 0.0547 m, occurs at the maximum tested velocity. The average error over all of the data samples is 0.0176 m. While in general the error is greater than that exhibited by the wheel radius configuration experiments, the average accuracy is 0.9540.

Repetition for Convergence

The chassis length configuration routine can be repeated in order to build up an average measurement value. Figure 5-10 consists of two graphs which each show the results of three experiments. A given trial consists of twenty one repetitions of the configuration routine.

Neither the user-measured radius nor the self-configured radius measurement yielded a chassis length calculation that matched the user-measured value. The average error of the running averages is 0.0130 m for the user-measured values and 0.0117 m for the self-configured data set. Using the cumulative values, the average accuracy of the user-measured results is 0.9659, and the average accuracy of the self-configured results is 0.9694. The accuracy values were calculated using user-measured chassis length of 0.3815 m as the accepted true value.

From this experiment, it seems that using a self-configured wheel radius gives a chassis

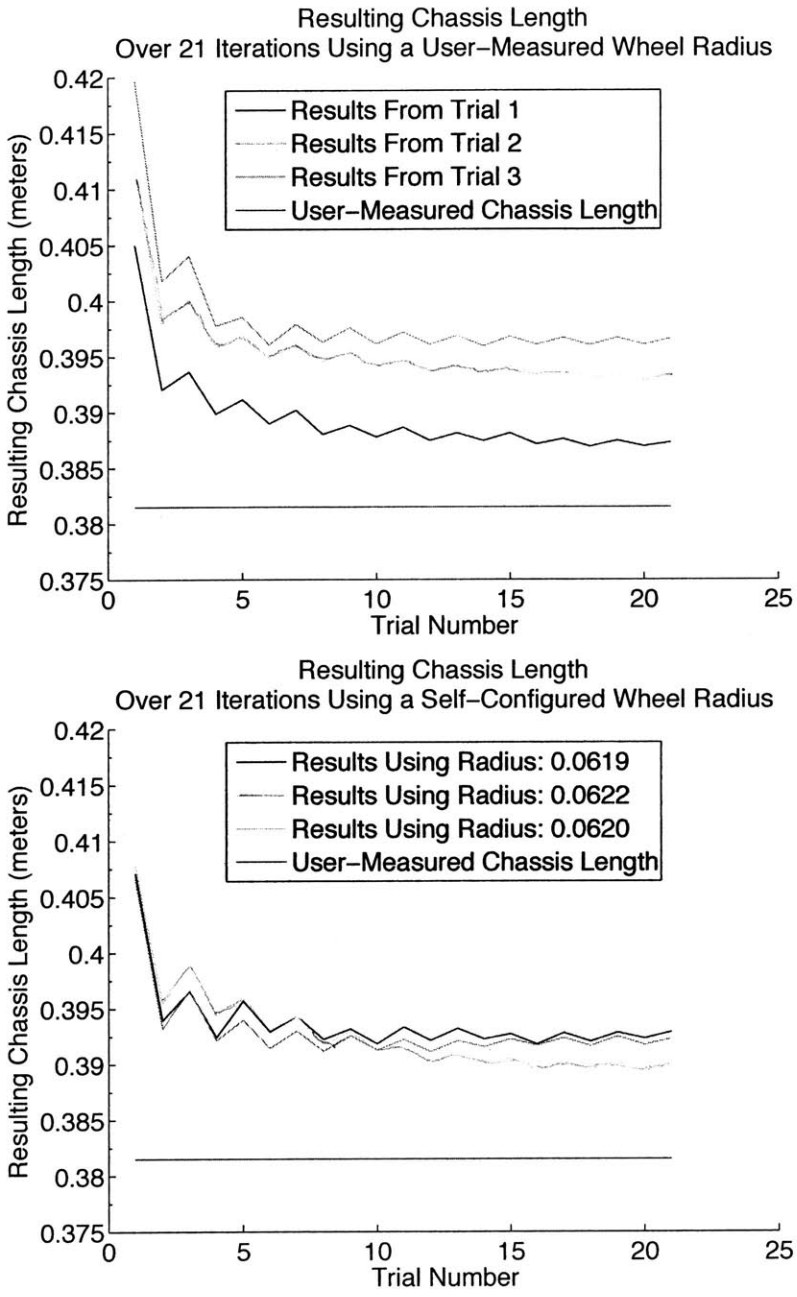


Figure 5-10: The graph depicts the average at each iteration of the chassis length configuration routine over the course of 21 harness-laps. As the number of traversals increases, the average measured chassis length approaches the user-measured value. The average accuracy, with respect to the user-measured chassis length, using the user-measured wheel radius is 0.9659, and the average accuracy using the self-configured wheel radius is 0.9694. Thus the performance under both situations is comparable.

length that is comparable to the result calculated using a user-measured wheel radius. This is not a surprising result, given that the self-configured wheel radius was very close to the user-measured value.

5.2.3 Results of Wheelbase Configuration

Wheelbase configuration gives two, coupled parts by experimentally determining the distances d_f and d_r. Together, d_f and d_r are used to calculate both the width of the wheelbase and its location with respect to the front and back of the robot chassis.

The wheelbase configuration routine consistently gives wheelbase values that are larger than the actual wheelbase. This is most likely due to the resolution of the incremental steps, given as $d_{inc} = 0.01$ m in Table 5.2, used in the configuration algorithm.

Varying the Initial Wheelbase Guess

The resulting wheelbase measurements for each of the different initial wheelbase values are shown in Figure 5-11. Each data point of the graph is the result of an average over three trials.

From the graph, the initial wheelbase guess seems to be independent of the resulting wheelbase width and body location. The wheelbase width which results from the configuration sequence tends to be larger than the user-measured wheelbase width. The average accuracy of the wheelbase width with respect to the user-measured value is 0.9255 with the maximum error being 0.0573 m.

The accuracy for the self-configured value of l_f is lower than that observed in the wheelbase value. The mean accuracy for this experiment is 0.8814. The maximum error was 0.0410 m. However, the mean accuracy for the self-configured l_r value is 0.9477, also with a maximum error of 0.0410 m.

In fact, on closer inspection the error for l_f from the user-measured value was the opposite of the error observed on l_r. This result is not surprising because l_f and l_r need to add up to the chassis length, and for the case of these experiments, the user-measured chassis length of 0.3815 m was used. In order for the two pieces to add up to the whole, any error observed on l_f must also be observed on l_r, but in the opposite direction.

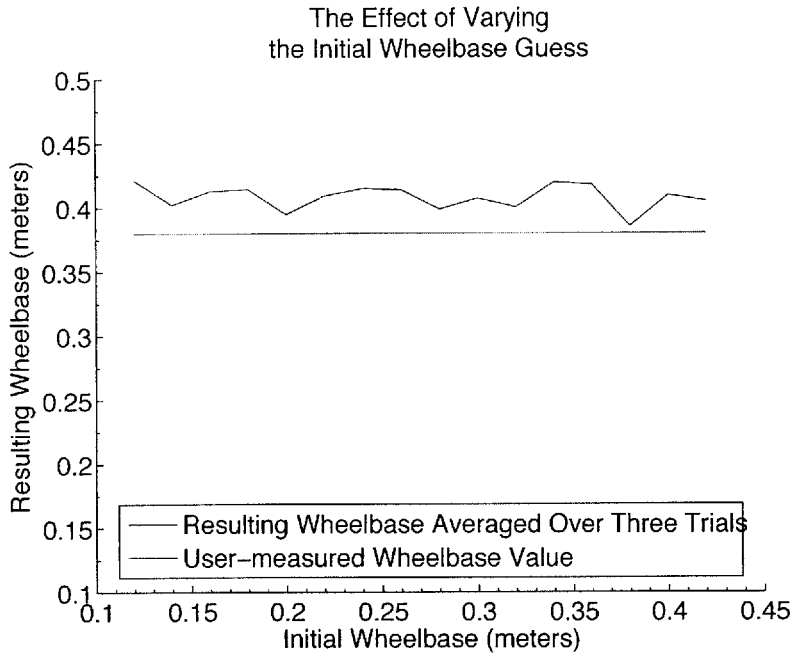


Figure 5-11: The effect of the initial wheelbase guess is similar to that of the initial wheel radius guess in Section 5.2.1.

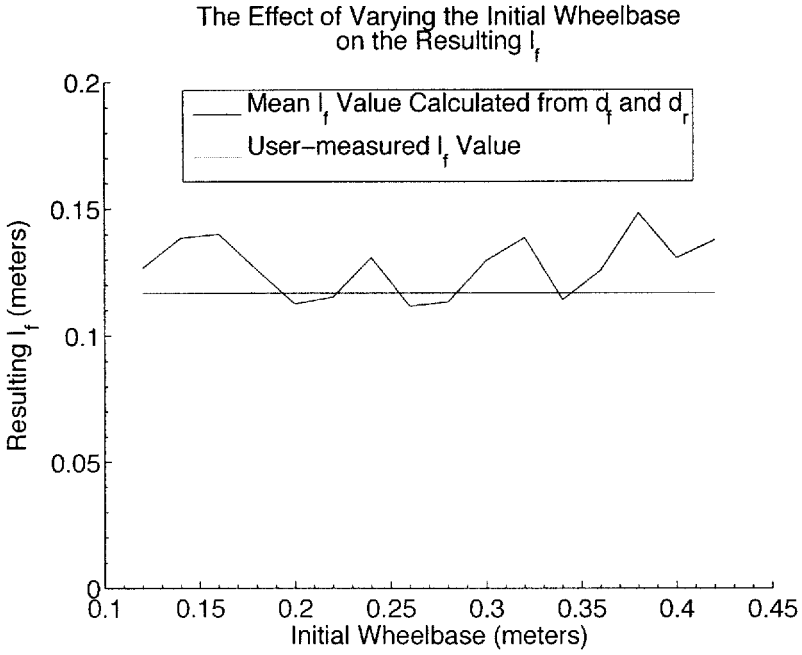


Figure 5-12: Varying the initial wheelbase guess does not have an observable effect on the resulting value of l_f. The raw data set has a mean value of 0.1274 m with a standard deviation of 0.0147. The mean accuracy, with respect to the user-measured value of 0.1168 m is 0.8814.

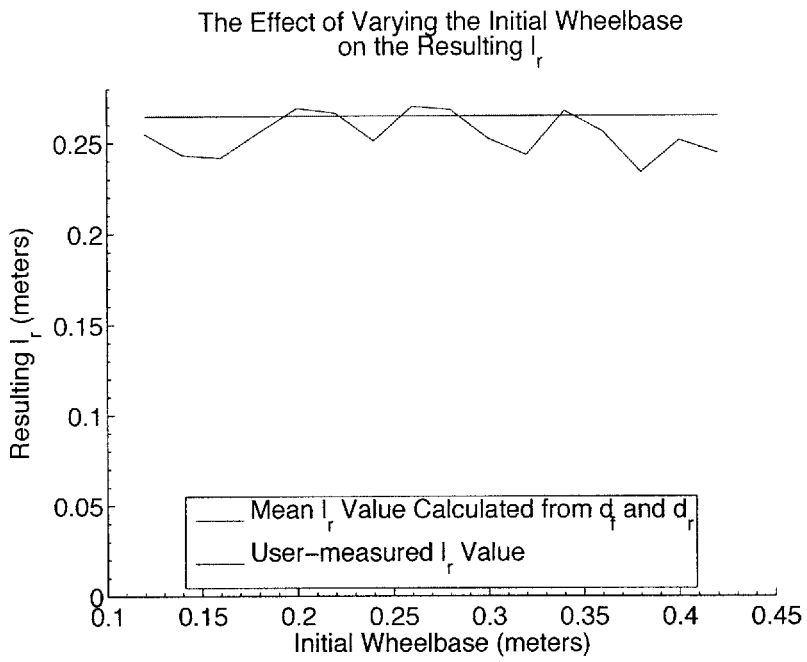


Figure 5-13: Varying the initial wheelbase guess does not have an observable effect on the resulting value of l_r. The raw data set has a mean value of 0.2541 m with a standard deviation of 0.0147. The mean accuracy, with respect to the user-measured value of 0.2647 m is 0.9477.

Varying the Commanded Rotational Velocity

The commanded rotational velocity that was used for the wheelbase configuration routine was varied from 0.02 rad/s to 0.10 rad/s at 0.02 rad/s intervals, and each set of experiments was repeated for different initial wheelbase guesses that ranged from 0.2925 m to 0.7800 m at 0.0975 m intervals, giving a total of five wheelbase values. When looking at the resulting values for a given rotational velocity, across all wheelbase values, the maximum standard deviation is 0.0148 m. Given that each data set contains values that are within 0.015 m of each other, the average across the results of a given rotational velocity can be used to produce a single point. Figures 5-14, 5-15, and 5-16 contain plots of the data resulting from this experiment.

The resulting values of the three parameters, b, l_f, and l_r are close to the corresponding user-measured values. The accuracy for the parameters, with respect to their user-measured counterparts are 0.9575, 0.8430, and 0.9307 respectively, with a standard deviation of less than 0.0040 m between samples. The substantially lower accuracy of the location values, l_f and l_r, is due to the resolution of the step used during the wheelbase configuration routine – given by d_{inc}. The values that result from the configuration sequence are independent

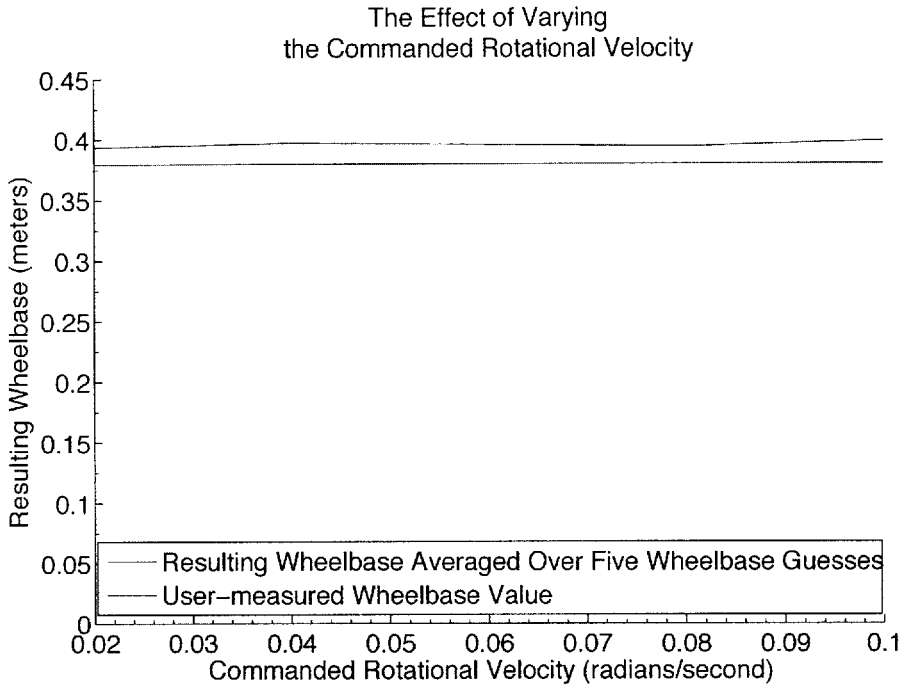


Figure 5-14: Changing the rotational velocity had no effect on the resulting wheelbase value. Each data point is calculated by averaging the wheelbase width that results from using several different initial wheelbase widths at the given commanded rotational velocity.

of the commanded rotational velocity which is used, as is apparent from the figures which contain plots of the data.

5.2.4 A Fully Self-Configured system

Although some of the results of the previous sections use self-configured parameters as their starting point, the entire configuration sequence was never explicitly carried out to completion. This subsection describes the results of running the self-configuration sequence all the way through.

The self-configuration sequence was executed using the values shown in Table 5.2. The first two variables in the table are the harness length and width, given by $l_{harness}$ and $w_{harness}$. The value, r'_w is the initial radius guess, and is $\frac{1}{8}l_{harness}$. The initial guess for the chassis length, l', is half of the harness length. The guess for wheelbase width, b', is half the harness width. The commanded translational and rotational velocities are given by v' and r' respectively. Finally, the incremental distance that the wheelbase configuration routine uses is given by d_{inc}.

In running the complete self-configuration sequence, three iterations of the chassis length

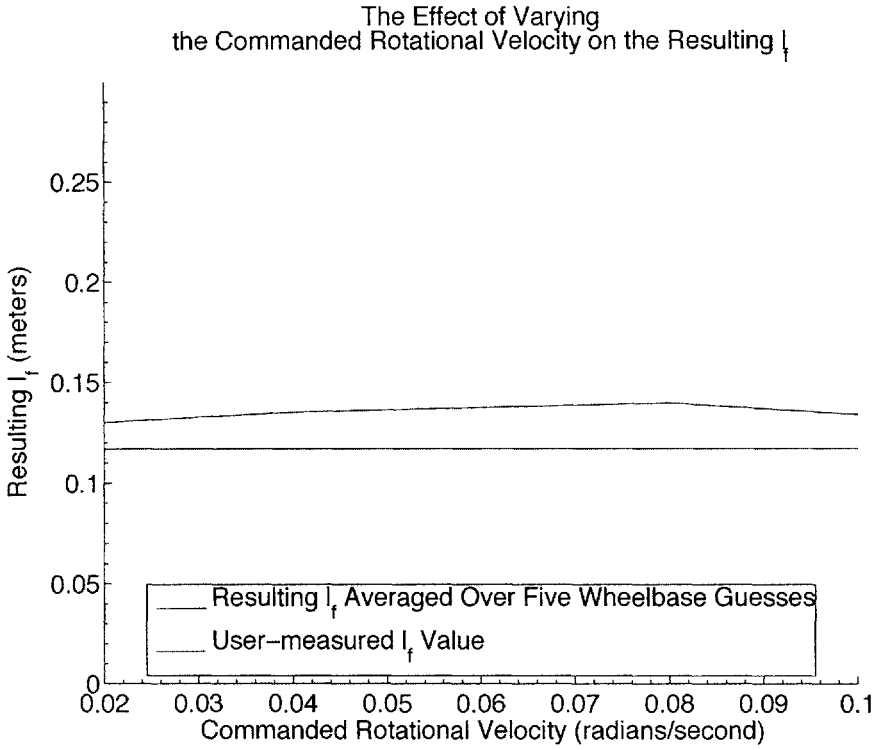


Figure 5-15: The l_f value that results from varying the commanded rotational velocity tends to be greater than the user-measured l_f value. The mean accuracy for the data used to calculate these points is 0.8430, with the standard deviation of the data set being 0.0036 m.

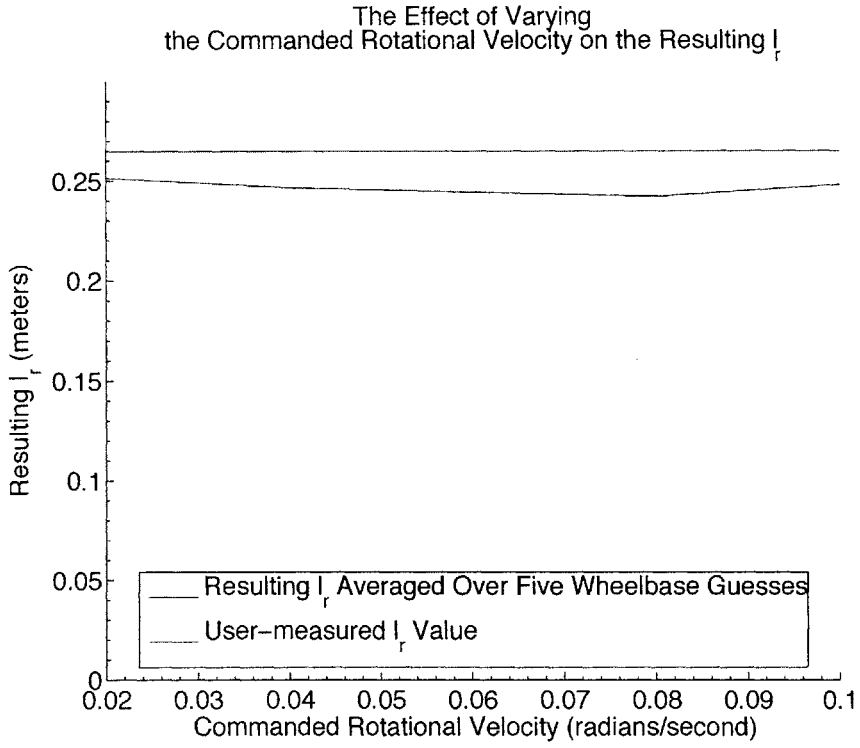


Figure 5-16: Varying the commanded rotational velocity results in l_r values that are consistently less than the user-measured l_r. The mean accuracy is higher than that achieved for l_f in that it is 0.9307, but the standard deviation over the data set remains 0.0036 m.

Table 5.2: Initial Values of Self-Configuration System

Parameter	Value
$l_{harness}$	1.2200 m
$w_{harness}$	1.0000 m
r'_w	0.1525 m
l'	0.6100 m
b'	0.5000 m
v'	0.03 m/s
r'	0.06 rad/s
d_{inc}	0.01 m

Table 5.3: Summary of Self-Configuration Results (All values, excluding accuracy, are given in meters)

Trial	r_w	l	b	d_f	d_r	l_f	l_r
1	0.0612	0.3845	0.4012	0.0994	0.0747	0.1528	0.2320
2	0.0615	0.3826	0.4002	0.0994	0.0747	0.1518	0.2307
3	0.0612	0.3820	0.3999	0.0994	0.0747	0.1516	0.2304
4	0.0610	0.3789	0.3986	0.0994	0.0747	0.1502	0.2286
5	0.0615	0.3829	0.4044	0.0994	0.0747	0.1520	0.2309
mean accuracy	0.9964	0.9955	0.9437	0.9377	0.7754	0.7014	0.8709

configuration step are used to determine the chassis length. Each trial consists of running the entire self-configuration routine is executed to completion without human intervention. Results of the configuration sequence are given in Table 5.3. The average accuracy of each parameter, with respect to the user-measured values is given in the last row of the table.

The accuracy of the wheel radius, chassis length, and wheelbase width configuration sequences with respect to the user-measured values is high. The accuracy of the wheelbase location is notably lower. The lesser degree of accuracy in wheelbase location is most likely due to the chosen resolution of d_{inc}.

The degree to which the inaccuracy of the self-configured wheelbase location affects performance is explored by the UMBmark. The results of the UMBmark are presented in Section 5.2.5.

5.2.5 Results of End-to-End Test

Running the UMBmark as the end-to-end test yielded the results which are summarized in tables 5.4 and 5.5. Following the analysis techniques outlined in the UMBmark paper [3], resulted in an $E_{max,sys} = 1.0519$ m for the user-measured parameters and $E_{max,sys} = 0.9031$

Table 5.4: Final UMBmark Positions Using User-Measured Parameters

Trial	Internal Position		Absolute Position	
cw	x	y	x	y
1	0.355	0.286	-0.018	0.046
2	0.210	0.218	-0.175	-0.163
3	0.389	0.371	-0.096	0.175
4	0.344	0.349	0.049	-0.161
5	0.284	0.222	0.003	-0.005
ccw	–	–	–	–
1	0.141	-0.177	1.307	-0.357
2	0.174	-0.134	1.106	-0.739
3	0.139	-0.151	1.111	-0.740
4	0.209	-0.210	1.010	-0.695
5	0.146	-0.227	0.976	-0.727

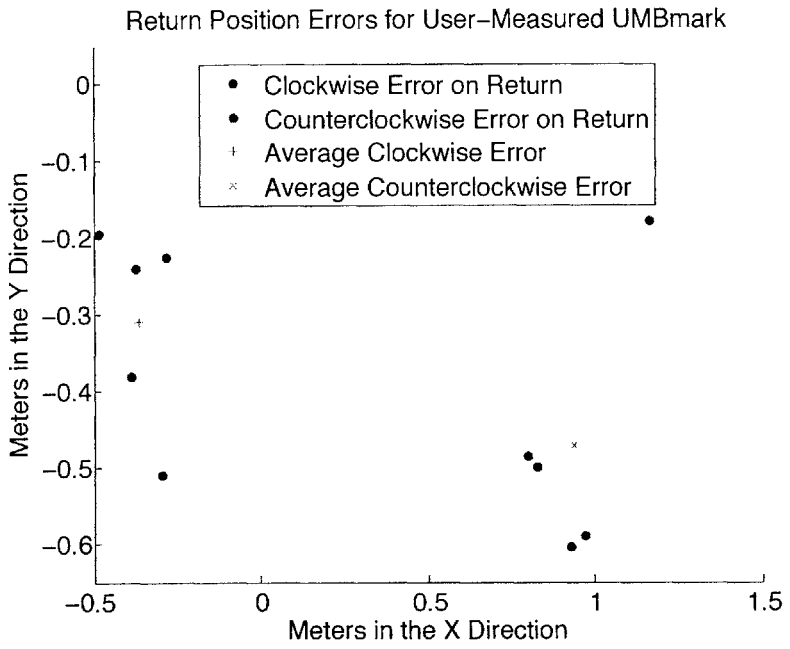
Table 5.5: Final UMBmark Positions Using Self-Configured Parameters

Trial	Internal Position		Absolute Position	
cw	x	y	x	y
1	1.039	0.678	0.917	0.702
2	0.922	0.614	0.833	0.632
3	1.056	0.774	0.926	0.766
4	1.134	0.765	0.982	0.733
5	1.034	0.702	0.978	0.731
ccw	–	–	–	–
1	0.834	-0.621	1.669	-0.891
2	0.862	-0.569	1.735	-0.838
3	0.833	-0.637	1.691	-0.906
4	0.668	-0.529	1.478	-0.846
5	0.922	-0.712	1.838	-0.990

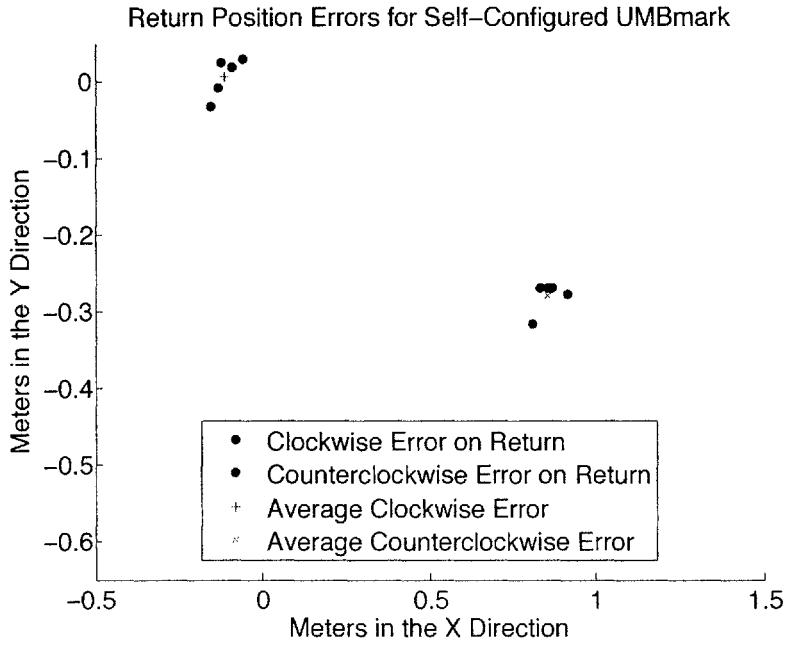
m for the self-configured parameters.

The UMBmark paper [3] summarizes the results in terms of *error on return*. Error on return is defined to be the difference between the measured, real world position, and the internal position which is calculated by the controller under test.

The resulting error on return values for the user-measured parameters, calculated from the values listed in Table 5.4, are plotted in Figure 5-17a. Similarly, the error on return for the the self-configured parameters are shown in Figure 5-17b and are calculated from the values in Table 5.5. The clockwise and counterclockwise data points, in the figures, are different colors and their centers of gravity are depicted as '+' and 'x' respectively.



(a) UMBmark results using user-configured parameters



(b) UMBmark results using self-configured parameters

Figure 5-17: These graphs show the results of running the UMBmark using the user-configured and self-configured parameters. Each graph shows the return position errors from both the clockwise and counterclockwise trials of the benchmark. The average result of each test group is also shown. The maximum systematic error for the user-configured system is $E_{max,sys} = 1.0519$ m and the equivalent calculation for the self-configured system is $E_{max,sys} = 0.9031$ m

Analysis of UMBmark Performance

The UMBmark gives a good metric for the maximum amount of systematic error that is inherent in an odometric system. It does this by focusing on the performance of a system executing a task which allows for the measurement of the error on return in both the clockwise and counterclockwise directions.

The user-measured parameters performed very well in terms of returning to the starting location with a high amount of accuracy. The maximum of the mean physical error is 1.2926 m. This error is in the counterclockwise direction. Comparing this to the equivalent maximum mean physical error of the self-configured parameters, which is 1.9056 m – also in the counterclockwise direction.

But when analyzing the results of the UMBmark using error on return, the self-configured parameters yield a much higher precision than the user-measured parameters. This is apparent from the close clustering of the points in Figure 5-17b, as compared to the dispersed nature of the points in Figure 5-17a.

The error on return metric shows that the self-configured results differed from the corresponding internal measurement in a manner which is very consistent. The consistency of these results makes it easier to correct the systematic error that results from typical robot movement.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Conclusion

This thesis investigates a possible solution for the self-configuration of a set of morphologically analogous robots using a known test environment. The proposed solution is presented alongside a software implementation of the techniques. This software system targets the discovery of parameters for the CARMEN robotics toolkit and is intended for use with microbots from *Robotics: Science and Systems* laboratory course at MIT. A series of experiments are performed on an implementation of the presented techniques in order to show that the method is a functional solution to the self-configuration problem as it relates to CARMEN and microbots.

6.1 Summary of Results

The results from Chapter 5 show that the proposed self-configuration technique works. The resulting system is a testament to this fact. The most difficult part of implementing the system was implementing the CBS. If an analogous sensor is provided by the targeted toolkit, then implementation of a self-configuration system which uses the self-configuration technique is straightforward.

The method for configuring the wheel radius yields results that are typically have 99% accuracy with respect to the value that a human operator would measure with a meter stick. The method is robust across initial wheel radius guesses that are from 32% to 400% of the actual wheel radius. The commanded translational velocity does not have an effect on the wheel radius value that results from the configuration routine. An important observation is that the longer the robot must travel before a collision, the more accurate the resulting

radius length tends to be. The robot should be placed as close to the starting position as possible to ensure a more accurate configuration.

The configuration method is able to build from the result of the self-configured wheel radius to execute the chassis length configuration with high accuracy. As the experiments in Section 5.2.2 show, middle range translational velocities tend to give more accurate results than higher velocities. The results show that the method has some inherent error. This error manifests itself in that even if the wheel radius is perfectly configured during the first phase of the configuration method, the chassis length that results is about 0.02 m too large. By repeating the experiment over a number of trials, it is possible to have the chassis length measurements converge to a value that is about 96% accurate, with respect to the user-measured value.

Using the chassis length and wheel radius which were configured in other parts of the configuration method, it is possible to run the wheelbase configuration routine to discover the width and location of the robot's wheelbase. The initial wheelbase guess and commanded rotational velocity that were used for testing this configuration component do not affect the values of the resulting wheelbase parameters. The accuracies of b, l_f, and l_r are respectively 93%, 88%, and 95%. These are the parameters which are configured with the least amount of accuracy. The cause of this is most likely the resolution that is used for the the incremental step of the configuration algorithm. In the case of this system, d_{inc} was set to 0.01 m.

When the robot parameters were configured entirely by the system, the results were fairly accurate with respect to the user-measured values. The accuracy of the wheel radius and chassis length were about 99%, and the accuracy of the wheelbase width and distance from the front and rear of the chassis were approximately 94%, 70%, and 87%.

The University of Michigan Benchmark was used as the basis for comparison of the self- and user-configured parameters. The expectation was that the self-configured parameters would perform as well as the user-configured parameters. Instead, the self-configured parameters gave more consistent results, as can be seen by the close clustering of the error-on-return points shown in Figure 5-17.

6.2 Suggestions for Future Work

The self-configuration techniques described are a starting point for many extensions and applications. The results of the self-configuration method, in its current manifestation, are that the robot's coordinate frame is known, and that the robot can translate at a specified velocity.

In the current version, each configuration technique is applied successively in order to determine each parameter value. The fact of the matter is that each control vector results in an action which is observed, in some manner, on the subsequent measurement of each sensor. A future implementation of the configuration techniques could apply this observation to its method so that parameter values are constantly updated, rather than only during a particular procedure. For example, while the robot is determining its wheel radius, it can be observing measurements on its sonar sensors. Depending on the sonar data samples, it is possible that the orientation is discovered before commencing a sonar discovery routine.

By attempting to use the sensors, and locate them in terms of body coordinates, the idea of fusing the data from all of the sensors, in order to gain more, overall information about the system becomes more of a possibility.

6.3 Closing Remarks

A method for robot self-configuration using a physical test harness has been described and implemented. The method was implemented for microbot-class robots that use the CARMEN navigation toolkit. The system carries out the method as described in Chapter 3 and successfully determines, the robot's wheel radii, chassis dimensions, and the distance from the wheelbase to the front of the robot and to the rear of the robot. The results successfully allow the robot to translate at a commanded velocity, and define the bounds of the x- and y-axis of the robot-centric coordinate system. Being able to execute a commanded translational velocity allows the robot to be controlled by client programs of the robot control toolkit. The discovery of the robot's coordinate frame provides a starting point for locating other robot sensors in terms of body coordinates. Both of these results simplify the robot toolkit configuration process, and allow an increase in the level of abstraction utilized by robotics toolkit API's.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] G. Antonelli, S. Chiaverini, and G. Fusco. A calibration method for odometry of mobile robots based on the least-squares technique: theory and experimental validation. *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, 21(5):994–1004, 2005.

[2] J. Bongard, V. Zykov, and H. Lipson. Resilient Machines Through Continuous Self-Modeling. *Science*, 314(5802):1118, 2006.

[3] J. Borenstein and L. Feng. UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots. *Ann Arbor*, 1001:48109–2110, 1995.

[4] I. Bostan, I. Lita, E. Franti, M. Dascalu, C. Moldovan, and S. Goschin. Systematic odometry errors compensation for mobile robot positioning. *CAD Systems in Microelectronics, 2003. CADSM 2003. Proceedings of the 7th International Conference. The Experience of Designing and Application of*, pages 574–576, 2003.

[5] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of [legacy, pre-1988]*, 2(1):14–23, 1986.

[6] H. Bruyninckx. Open robot control software: the orocos project. *IEEE International Conference on Robotics and Automation, 2001. Proceedings 2001 ICRA*, 3:2523–2528, 2001.

[7] K.S. Chong and L. Kleeman. Accurate odometry and error modelling for a mobile robot. *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, 4, 1997.

[8] A. Edsinger and C. Kemp. What can I control? A framework for robot self-discovery. *Proceedings of the Sixth International Conference on Epigenetic Robotics (EpiRob 2006)*, Sept. 2006.

[9] A. Martinelli and R. Siegwart. Estimating the Odometry Error of a Mobile Robot during Navigation. *In other words*, 1:3, Sept. 2003.

[10] G. Metta, G. Sandini, L. Natale, and F. Panerai. Sensorimotor interaction in a developing robot. *First International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, pages 18–19, 2000.

[11] Michael Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) toolkit. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2436–2441, 2003.

[12] L.A. Olsson, C.L. Nehaniv, and D. Polani. From unknown sensors and actuators to actions grounded in sensorimotor perceptions. *Connection Science*, 18(2):121–144, 2006.

[13] D. Pierce and BJ Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92(1):169–227, 1997.

[14] Reid Simmons. Inter-process communication. `http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html`, June 2004.

[15] Peter Soetens. Tutorial on the component interface. `http://www.orocos.org/stable/documentation/rtt/v1.2.x/doc-xml/orocos-ocl-intro.html`, 2006.

[16] Richard. T. Vaughan, Brian P. Gerkey, and Andrew Howard. On device abstractions for portable, reusable robot code. *IEEE/RSJ International Conference on Intelligent Robot Systems, 2003. (IROS 2003). Proceedings.*, 3:2421–2427, 27-31 Oct. 2003.

[17] S. Vinoski. CORBA: integrating diverse applications within distributedheterogeneous environments. *Communications Magazine, IEEE*, 35(2):46–55, 1997.

[18] Juyang Weng. Developmental robotics: Theory and experiments. *International Journal of Humanoid Robotics*, 1(2):199–236, 2004.

[19] V. Zykov, J. Bongard, and H. Lipson. Evolving Dynamic Gaits on a Physical Robot. *Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO*, 4, 2004.