LOCATING SPARSE RESOURCES IN UNMAPPED TERRAIN WITH A COLLECTIVE ROBOTIC SYSTEM USING EXPLORATION STRATEGIES INSPIRED BY PLANTS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Daniel K. Schrader

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Eric Matson, Chair

Department of Computer and Information Technology

Dr. Byung-Cheol Min

Department of Computer and Information Technology

Dr. J. Eric Dietz

Department of Computer and Information Technology

Dr. Julia M. Rayz

Department of Computer and Information Technology

Approved by:

Dr. Kathryne A. Newton

Head of the Graduate Program

Dedicated to my two best friends. They were always so happy to see me, no matter when I got home, and they patiently awaited their new, improved life. Thanks for sticking with me, ladies.

ACKNOWLEDGMENTS

I would like to gratefully acknowledge my committee for their guidance and assistance.

I would also like to thank all of the folks that helped me develop and test this system, good weather or bad. I could not have done it without all of you.

A sincere thank you to my family for supporting me, encouraging me, and providing a childhood and a family environment to enable endeavors such as this.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABBREVIATIONS

NASA	National Aeronautics and Space Administration
ESA	European Space Agency
SBC	single board computer
ANN	Artificial neural network
LCS	Learning classifier system
RSSI	Received signal strength indicator
LQI	Link quality indicator
RPi	Raspberry Pi

ABSTRACT

Schrader, Daniel K. Ph.D., Purdue University, December 2018. Locating Sparse Resources in Unmapped Terrain with a Collective Robotic System Using Exploration Strategies Inspired by Plants. Major Professor: Eric Matson.

Wherever we go, we need resources. Finding those resources in unmapped areas is an ever-present challenge. Nature provides many examples of systems that manage to find the resources they need for growth, despite having little to no information about their environment. Emulating the resource-finding strategies of animals and insects has been, and continues to be, attempted in robotic systems, to varying degrees of success. However, borrowing strategies from plants is much less explored. This dissertation explores an attempt at distilling the resource-hunting methods of plant roots into a collective robotic system.

Utilizing low-power computing and wireless communication, the robots attempt to locate "resources" (radio beacons, in this case) in an unmapped area. They work collectively via extending and branching from each other. The results of this experiment show limited success, with the limitations primarily stemming from the wireless communication. Nonetheless, it is shown that a collective robotic system, emulating plant roots, can feasibly locate resources that display a gradient, with no map of the environment.

CHAPTER 1. INTRODUCTION

We are explorers. Throughout history, we have pushed the boundaries of our known world further and further out. Technology has always driven, or at least enabled, exploration that was previously not possible. From hand-carved canoes all the way to planetary rovers, we rely on technology to help us explore our environment. As we move further out into space, we need new technologies that will assist us and keep us safe. Whether for asteroid mining or the preparation of a manned settlement, locating resources is key to our efforts. Those resources may be anything from water to metals to particular locations/topographies or even valuable data.

There are myriad methods and ideas regarding the finding of resources, but the natural world provides us with examples of success in this endeavor. Nearly every biological organism must actively seek out resources to survive, but if we are to translate these organisms' success into robotic systems, we must take into account our ability to create a synthetic approximation of different organisms. Lions are very good at finding and collecting resources, but they use sensors, processing, and actuators that are, for the moment, beyond our reach. What about insects? They are some of the most successful organisms in the history of Earth, and we can create synthetic approximations of them (though not nearly as small). So far, so good, but we can go a step further. Plants are arguably the most successful organisms in our environment. Despite very limited abilities in sensing, processing, and actuation, plants manage to locate resources around them with sufficient efficiency to grow and reproduce.

Sending a robotic system into space requires that the system possesses extremely high reliability. The more complex a system is, the more difficult it is to ensure reliability. Plants are much simpler than lions, for example, and at least not

more complex than insects. There is another important property of plants and many insects (and some animals) that is relevant to space exploration, which is that they are collective systems. Collective systems, or swarms, have the ability to be very robust, due to the lack of a single point of failure. A plant is not a singular object, but rather a collective made up of leaves, stems, stalk, roots, etc. The root system, in particular, is a collective of individual roots with no apparent central authority. When one root fails, it does not necessarily cause the entire system to fail. Emulating the plant's method of locating resources in a robotic system will likely impart onto that system many of the advantages that plants enjoy.

Developing real, full-scale, collective robotic systems is a very challenging task, in no small part due to the monetary cost of these systems. Since plants demonstrate remarkable success with such limited resources at their disposal, they lend themselves nicely to robotic emulation. Simple sensors, low-performance actuation, and relatively little computation mean that individual robots are reasonably simple and inexpensive, which makes a full-scale system feasible.

1.1 Research Question

How completely can a collective robotic system, based on resource-finding strategies of plants, locate specific resources in a completely unknown and unmapped area?

1.2 Significance

As we continue to push past the boundaries of our known world, we need technologies that will augment, enable, and protect us and our machines in unknown, unmapped places. We cannot possibly carry everything with us that we might need as we explore further out, so we need to be able to locate resources on asteroids, planets, and whatever is available, even when it is difficult or impossible for humans to do so. NASA states on their official website:

> Robotic exploration continues to deliver profound answers about our Universe by visiting far-off destinations, providing reconnaissance and collecting scientific data. When combining both human and robotic exploration methods we will use technology and our senses to increase our ability to observe, adapt, and uncover new knowledge (Wiles, 2013).

Robotics holds enormous promise for the future of space exploration, but we will need the benefits of collective systems as we move forward. Developing full-scale collective systems is a very challenging task, but one that nature has repeatedly solved. "Arguably, the complexity, flexibility and adaptation demonstrated by natural collective systems is unmatched by any man-made system" (Simes et al., 2011). Therefore, it is logical to borrow strategies from one of the natural world's most successful collective organisms, plants, to help us push the boundaries of our knowledge even further.

1.3 Purpose/Scope

I will develop a collective robotic system that explores an unmapped area in search of nutrients, which will be represented by controlled quantities, such as radio beacons. This system will grow "roots," much like a plant does when searching for nutrients. The exploration decisions will be made individually by the roots, rather than a central authority, which will be called the nexus. The primary responsibility of the nexus will be to dispense resources (i.e. additional robots) as the "roots" grow. Since the system will lack the physical connections that roots have, it will use directional wireless communication to pass messages and perform limited localization.

1.4 Assumptions

- No global coordinates (via GPS or otherwise) are available

- Test area has no major physical or radio obstacles
- Nutrients display concentration gradients (Simes et al., 2011)
- Nutrients do not fade with time
- Environmental factors do not change quickly enough to significantly influence a test
- Nutrients are static

1.5 Limitations

- Number of available robots

1.6 Delimitations

- Transport of nutrients
- Significant map generation
- Obstacle avoidance
- Any study, beyond simulation, of significant scalability of the system
- Branches of more than two children

1.7 Definitions

Nutrients: resources that are being searched for

Stigmergy: an indirect, mediated mechanism of coordination between actions in which a perceived effect of an action stimulates the performance of a subsequent action (Heylighen, 2011)

Apex: the tip of a root (Simes et al., 2011)

Node: mobile robot

Nexus: device that serves as data collector and place of initial node deployment

1.8 Summary

This chapter provided the purpose/scope, significance, research question, assumptions, limitations, delimitations, definitions, and other background information for the research project. The next chapter provides a review of the literature relevant to this dissertation.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

The literature review is split into several major domains. The first discusses collective robotics and associated topics in a theoretical, sometimes even philosophical, context. The second deals with the notion of plants as swarms, borrowing methodologies from plants to inspire robotics, and some exploratory simulation work that has been done in this area. The third presents some references for various components of the proposed system, such as neural networks and wireless technology.

2.1 With respect to collective systems, what is emergence?

"Emergence is an effect or event where the cause is not immediately visible or apparent" (Fromm, 2005). More computationally defined, emergence is "the incompressibility of a simulation process" (Huneman, 2008). Getting a bit more specific, Goldstein says, "Emergence...is the arising of novel and coherent structures, patterns, and properties during the process of self-organization in complex systems" (Goldstein, 1999). In that statement, Goldstein refers to two concepts that need to be defined and contextualized. The first of those two, self-organization, is discussed in a later section of this paper. The second concept is that of a complex system. A thorough dive into complex systems (Bar-Yam, 1997, 1998) or complexity theory is not within the scope of this paper, but a concise description is helpful.

2.1.1 Complex systems

Bar-Yam described the study of complex systems (Bar-Yam, 1997) by saying, "As a discipline, complex systems is a new field of science studying how parts of a

system and their relationships give rise to the collective behaviors of the system, and how the system interrelates with its environment." Giving context to that definition, Bar-Yam discussed the fundamental scientific principle that everything is made of parts. Traditionally, when a scientist wants to know how something works, (s)he begins by examining its parts. Those parts are made up of their own parts, which are made up of even more parts, and so goes the rabbit hole. Studying parts has given us much of our modern understanding of the world, but the dissection of parts alone is not sufficient to tackle increasingly complex problems. For that, we must study the relationships between parts. Unlike parts themselves, the relationships between parts are not consistent between different systems. This nature of relationships gives rise to complexity theory and the study of complex systems.

Returning to Goldstein's definition of emergence (Goldstein, 1999), it is reasonable to say that emergence is the result of the relationships between a system's parts and environment. Therefore, the study of emergence is, in a sense, the application and classification of complexity theory. However, this paper is not explicitly focused on complexity theory. Rather, the focus is on emergence within a specific type of complex system: collective robotics.

2.1.2 Collective robotic systems

Collective robotic systems consist of many interacting individuals (i.e. robots) that, through their interactions with each other, are able to achieve behaviors, functionality, and structures that are not achievable by any lone individual (Haasdijk et al., 2013). Collective systems, in general, are all around us in the form of viruses, microscopic particles, social insects and animals, transportation systems, and computer networks, just to name a few. The principles governing those systems and the phenomena those systems produce are the same for collective robotic systems. Emergence is what makes collective robotics interesting and challenging, so an ontology (of a sort), introduced in the next section, is necessary.

2.2 What are the primary types of emergence?

There is quite a bit of overlap between defining emergence and categorizing its types (i.e., the first and second questions in the title). At the beginning of this paper, I gave several definitions of emergence that I feel are clear and concise, but other definitions and descriptions are discussed in this section. Much like the topic of artificial intelligence, emergence takes a different form with each debate or discussion.

2.2.1 Weak vs. strong

Kernbach made a distinction between two overarching types of emergence: weak and strong (Haasdijk et al., 2013).

> *Weak emergence* describes new properties arising in systems as a result of the interactions between collective agents. Emergence, in this case, is part of the model, which describes a system's behavior.
>
> *Strong emergence* is a type of emergence in which the emergent property is irreducible to its individual agents.

Strong emergence often spawns philosophical discussion, such as in (Laughlin & Leggett, 2005; Todd, 1934). In the Introduction of (Todd, 1934), Blitz summarized the progression of the philosophy of biological evolution and the nature of the natural world, which are good examples of strong emergence, by Kernbach's definition. Fromm also refers to the emergence of life (and culture) as clear examples of strong emergence (Fromm, 2005). Laughlin, too, discussed strong emergence, but rather than biological evolution, the focus is on physics and the understanding of physical laws through the study of emergence (Laughlin & Leggett, 2005).

As interesting as strong emergence is, it does have a big drawback: predictability. The lack of predictability makes designing (or attempting to design) strong emergence rather problematic. "The general problem of designing [strong]

emergence is that we cannot say in advance which emergent behavior will be generated by the chosen rules" (Haasdijk et al., 2013). Possessing the property of being unpredictable means there are few or no analytic methods that give much insight into the emergent system's behavior, leaving engineers with simulation:

> When common knowledge of a collective system is not a subset of group-related knowledge, there is no other way of predicting the result of top-down and bottom-up approaches other than to simulate it with the full complexity of individual agents and their interactions (Haasdijk et al., 2013).

> A true emergent phenomenon is one for which the optimal means of prediction is simulation (Suki, Bates, & Frey, 2011).

2.2.2 Non-emergent and controllable-emergent behavior

Weak emergence is best clarified by the concept of controllable-emergent behavior, where the designer has explicit control over a system's degrees of freedom (Haasdijk et al., 2013). Ronald and Sipper provided good examples (an indoor mail delivery robot and a garbage collecting robot) and comparison between non-emergent and controllable-emergent behaviors (E. M. Ronald & Sipper, 2001). The mail delivery robot is described as a line follower with a very carefully engineered path, illustrating non-emergent behavior. Ronald and Sipper assigned such behavior to the realm of classical engineering, where the goal is *unsurprise*. In contrast, the garbage collecting robot is designed through evolutionary robotics, where the degrees of freedom are controlled and the outcome is, in a sense, intentionally designed. However, the engineer did not explicitly program the robot's behavior. "Yes, the evolved robot works (surprise), but it is in some oxymoronic sense *expected* surprise: as though you were planning your own surprise birthday party" (E. M. Ronald & Sipper, 2001). Fromm offers an alternative and potentially

Table 2.1.

Fromm's classification of emergence (Fromm, 2005).

Type	Name	Frequency	Predictability
I	Nominal	abundant	predictable
II	Weak	frequent	predictable in principle
III	Multiple	common-unusual	not predictable (or chaotic)
IV	Strong	rare	not predictable in principle

more familiar example of weak emergence: shoals of fish, where the presence of the shoal/swarm influences each individual's motions (Fromm, 2005).

Non-emergent (i.e. classical) collective behavior (Kornienko, Kornienko, & Priese, 2004; E. M. Ronald & Sipper, 2001; E. M. A. Ronald & Sipper, 2000) has the advantages of being stable and predictable, even when responding to irregularities, but it suffers from limited adaptability (Haasdijk et al., 2013). Fromm refers to non-emergent, or even very weakly-emergent, behavior as "nominal/intentional" (Fromm, 2005) (see Table 2.1). Controllable-emergent behavior (Kornienko, Kornienko, & Levi, 2004) fills in the middle of the scale (with strong emergence at the far end, although Fromm defines a level of emergence between weak and strong (see Table 2.1)), providing more adaptability than non-emergent systems, but at the cost of stability and predictability.

Nominal emergence

Nominal emergence, as defined by Fromm (Fromm, 2005) and Bedau (Bedau, 2002), is the weakest form of emergence. Fromm put forth the example of a classically-designed machine (e.g. a mechanical clock) to illustrate nominal emergence. "The function of a machine is an emergent property of its components," and "a specific and fixed role is assigned to each part, and this role does not change

in the course of time. The behavior of each part is always the same, it is independent of the other parts' states, the global state of the system and the environment" (Fromm, 2005). Ronald and Sipper's mail delivery robot (E. M. Ronald & Sipper, 2001) fits nicely into Fromm's idea of nominal emergence. Bedau offers a similar, albeit a bit more abstract, definition by saying that "nominal emergence is simply this notion of a macro property that is the kind of property that cannot be a micro property" (Bedau, 2002).

2.2.3 Beginnings of an ontology

There are, as mentioned previously, many ways to describe and understand emergence. Weak and strong are certainly a useful start, but such a general level of distinction leaves something to be desired. Much of the literature on emergence indicates that enforcing strict, deep categories is either irrelevant or premature, so my answer to the question "What are the primary types of emergence?" will not simply list finer and finer categories. Rather, I will introduce several of the concepts of emergence from the introductory chapter of (Haasdijk et al., 2013), as well as other sources.

Combinatorial

Huneman defined combinatorial emergence as focusing on "whole-parts relationships" and stated that "emergence is often considered to be the problem of understanding properties of the wholes that are irreducible to properties of the parts" (Huneman, 2008). However, Huneman argued that the combinatorial approach, despite its popularity (Bechtel & Richardson, 1992; Bedau, 2002; O'Connor, 1994; Silberstein, 2002), is not a very useful way to think about emergence. Properties of the whole, according to Huneman's approach, are almost always novel when compared to the properties of the parts (e.g. mass and volume).

Therefore, the combinatorial approach does not provide much insight into emergence. Rather, "focusing on what is an emergent process, rather than on the emergence of properties" (Huneman, 2008) lends more clarity and utility to the concept of emergence.

Descriptive/Explanatory

Fromm borrowed from the Cambridge Dictionary of Philosophy (Audi, 1999) to define descriptive and explanatory emergence (Fromm, 2005).

> *Descriptive emergence* means "there are properties of 'wholes' (or more complex situations) that cannot be defined through the properties of the 'parts' (or simpler situations)."

By this definition, descriptive emergence is simply a rephrasing of combinatorial emergence. Since combinatorial (descriptive) emergence is arguably the dominant view, overlap is expected. Explanatory emergence, however, is more akin to Huneman's computational philosophy of focusing on processes, instead of properties (Huneman, 2008).

> *Explanatory emergence* means the laws of the more complex situations in the system are not deducible by way of any composition laws or laws of coexistence from the laws of the simpler or simplest situations.

Surprising

Ronald and Sipper took a novel approach to defining emergence by invoking the concept of surprise (E. M. Ronald & Sipper, 2001; E. M. A. Ronald & Sipper, 2000). They provided three criteria for establishing that behavior is emergent:

1. *Design.* The system has been constructed by the designer by describing *local* elementary interactions between components (e.g., artificial creatures and elements of the environment) in a language L1.

2. *Observation.* The observer is *fully aware* of the design, but describes *global* behavior and properties of the running system, over a period of time, using a language L2.

3. *Surprise.* The language of design L1 and the language of observation L2 are distinct, and the causal link between the elementary interactions programmed in L1 and the behaviors observed in L2 is *non-obvious* to the observer who therefore experiences surprise. In other words, there is a cognitive dissonance between the observers mental image of the systems design stated in L1 and his contemporaneous observation of the systems behavior stated in L2.

Using an observer's surprise to gauge whether or not behavior is emergent is vague but intuitive. Ronald and Sipper provide some granulation to the notion of surprise (unsurprise, unsurprising surprise, and surprising surprise (E. M. A. Ronald & Sipper, 2000)), but the definitions are still qualitative.

...and more

There are nearly as many definitions and explanations of emergence as there are people writing about it. Some authors have attempted to corral the widely varying analyses (Audi, 1999; Fromm, 2005; Goldstein, 1999; Haasdijk et al., 2013; Silberstein, 2002), which is helpful, but highlights the lack of agreement within the field. Huneman rejected the common foundation of analyzing emergence by properties and takes a computational approach to identifying emergence (Huneman, 2008). Bar-Yam and Darley tried to lay out mathematically rigorous definitions of (strong) emergence (Bar-Yam, 1997, 1998, 2003, 2004a, 2004b; Suki et al., 2011).

Wimsatt came at the problem of emergence through philosophy, defining emergence as the lack of aggregativity. Put another way, Wimsatt definee emergence "by figuring what conditions should be met for the system property *not* to be emergent (i.e, for it to be a 'mere aggregate' of its parts properties)" (Wimsatt, 2000). This is all to illustrate that defining, categorizing, describing, and understanding emergence is neither easy nor agreed upon. Table 2.2 summarizes my analysis and relevant opinions on some of the works discussed so far.

2.3 Inevitability

I'll start by addressing the question of inevitability of self-development and self-repair in collective robotic systems. In short, no, it is not inevitable. One can easily imagine a collective system that either diverges from self-developmental/repair behaviors, or is designed with enough constraints so as to eliminate any significant emergence. The rest of this paper will focus on self-development and self-repair, in general, and whether or not they are necessary.

2.4 Adaptability

Adaptation and self-adaptation are core features of collective systems (CS). Levi and Kernbach defined adaptability (Levi & Kernbach, 2010):

> **Adaptability** is defined in terms of a triple relation: *environmental changes* $\rightarrow$ *system's response* $\rightarrow$ *environmental reaction.* In general, adaptability is the ability of a collective system to achieve desired environmental reactions in accordance with a priori defined criteria by changing its own structure, functionality, or behavior.

Unsurprisingly, there is a spectrum of adaptability in CS. Table 2.3 summarizes the overarching classes of adaptability, according to Kernbach. Self-developmental

Table 2.2.

Summary of discussed works on emergence.

Source	Philosophy	Strength	Weakness	Utility
(Bar-Yam, 1997, 1998, 2003, 2004a, 2004b)	Math formalism	Quantitative, measurable	Must formally define constraints	Med.
(Bedau, 2002)	Downward causation	Makes weak emergence traceable	Arguably violates causal laws	Low to med.
(Suki et al., 2011)	Analytical decomposition	Quantitative, measurable	Relies on full simulation	High
(Fromm, 2005)	Categorization	Flexible, fairly encompassing	Qualitative, general	Med. to high
(Goldstein, 1999; O'Connor, 1994)	Historical context	Thorough	Non-technical	High
(Haasdijk et al., 2013)	Survey of collective robotics	Thorough, directly relevant	None	High
(Huneman, 2008)	Process (not property) analysis	Scientifically rigorous	Relies on computability	High
(Kornienko, Kornienko, & Levi, 2004)	Top-down rule generation	Controllable, compact results	Computationally complex and demanding	High
(E. M. Ronald & Sipper, 2001; E. M. A. Ronald & Sipper, 2000)	Levels of surprise	Intuitive, simple	Qualitative, subjective	Low
(Silberstein, 2002)	Critique of reductionism	Well-researched, thoughtful	Very...academic	Med.
(Wimsatt, 1997, 2000)	Lack of aggregativity	Novel, useful in categorization	High (philosophical) barrier to entry	Low to med.

Table 2.3.

Adaptivity classes of collective systems (Haasdijk et al., 2013).

Type of CS	Comment
CS with fixed interactions	Environmental fluctuations can be foreseen and absorbed by external mechanisms; cooperative behavior includes some adaptive mechanisms but is mostly predefined (Colestock, 2005).
Tunable, reconfigurable CS	More developmental degrees of freedom, adaptivity is achieved in different ways, from parameter changing, feedback-based mechanisms (Astrom, 1987), and adaptive [self-organization] (Vaario, 1994) to fully reconfigurable systems. A multitude of learning mechanisms can be applied (Domingos, 2002).
Self-developmental CS	Systems capable of structural changes and with changeable reward/feedback mechanisms have bound and unbound cases.

collective systems are listed as the most adaptable, which is logical, and also makes clear that adaptability is a prerequisite of self-development.

"Adaptability is closely related to environmental changes, the ability of a system to react to these changes, and the capability of the designer to forecast the reaction of the environment to the system's response" (Haasdijk et al., 2013). Table 2.5 describes a few types of environmental changes and corresponding examples, both with the ability to be forecast and without.

For an adaptive collective system to respond to any environmental change, it must have some means of detecting that change. Kernbach separated the capability to detect changes into three categories: model reference based, self-tuning based, and concept based, described in Table 2.4 (Haasdijk et al., 2013). Once a system has detected a change in the environment, it needs some mechanism(s) to react to that change. Table 2.6 lays out three types of adaptive mechanisms, with self-organized mechanisms being the most adaptive. However, Table 2.6 does not make a distinction between structural and functional self-organization (SO), but

Table 2.4.

Capabilities to detect changes in CS (Haasdijk et al., 2013).

Type	Comment
Model reference-based	Widely used scheme in, for example, adaptive control (Chalam, 1987), machine learning (Domingos, 2002), artificial evolutionary systems (Nolfi & Floreano, 2004), and many other areas, where the detection of changes represents a error between model and system ("plant" in control theory). A multiplicity of **feedback, reward, fitness based** mechanisms (Astrom, 1987; Fogel, 2006) originate from this approach.
Self-tuning-based	Very popular approach (see, e.g., (Åström, 1980), the first ideas are described by (Kalman, 1958)). It consists of a parameter estimator, a design calculation, and a regulator with adjustable parameters, the idea being to select "a design for a known plant parameter and to apply it to an unknown plant parameter using recursively estimated values of these parameters" (Chalam, 1987).
Concept-based	Self-developmental systems with a high degree of plasticity cannot use model- or tuning-based detection mechanisms; their mechanisms of detection are not plastic enough. Instead, the so-called self-concept-based approach has been proposed (first in human psychology (Maslow & Lowery, 1968; McLean, Pasupathi, & Pals, 2007)).

Table 2.5.

Four types of environmental changes in robotic applications and examples of cases both forecast and not forecast (Levi & Kernbach, 2010).

Environmental changes leading to:	Examples: forecast	Examples: not forecast
Appearance of new situations	Installation of industrial robots in a new workshop	Work in previously unexplored environment (e.g., landing on Mars)
Changed functionality	Changing a type of locomotion (e.g., from wheeled to legged), when changing a terrain type	Search-and-rescue scenario when robots encounter unknown obstacles
Modified behavioral response	Gravitational perturbation of flying object in space and finding new control laws for engines	Disturbed control of legged locomotion for obstacles of random geometry
Optimization of parameters	Changing of day/night light and adapting intensity of additional light	Adapting locomotive parameters for randomly moving obstacles

Table 2.6.

Main adaptive mechanisms in collective systems (Haasdijk et al., 2013).

Type	Comment
Parameter-based adaptive mechanisms	Traditional for control theory (see, e.g., (Narenda & Annaswamy, 1989)); the system is controlled through control parameters by modifying the values, the controlled system responds by changing its behavior. A multitude of possible variations exists: when a system is known, its analytical model can be used for control purposes; when the environment is simple, it is incorporated into the analytical model; when a system is unknown (the black box approach), different feedback mechanisms can be used for control purposes. Different ways of adapting the system are the focus of unsupervised, reward-based learning approaches. The parameter-based adaptive mechanisms are very efficient but have several essential drawbacks related to low flexibility.
Modularity-based adaptive mechanisms	The system consists of modules that can be dynamically linked to each other. The linkage can be of binary as well as fuzzy character. Examples of such systems are artificial neural networks (ANNs) (Fausett, 1994), genetic programming (GP) (Koza, 1992), reconfigurable robotics (Shen et al., 2006), and others. The modular structure has several particular issues, for example, granularity of modules–how large are changes of the transfer function created by relinking only one elementary module.
Self-organized adaptive mechanisms	Self-organizing systems consist of many interacting elements with a relative high degree of autonomy (Haken, 2004). The transfer function of such systems is "generated" dynamically through interactions. Self-organized adaptive mechanisms introduce feedback directly into the interactions among elements (Alvarez-Ramírez, 1993; Basso, Evangelisti, Genesio, & Tesi, 1998).

structural SO is, in theory, even more adaptable than functional SO (see my paper answering question 1).

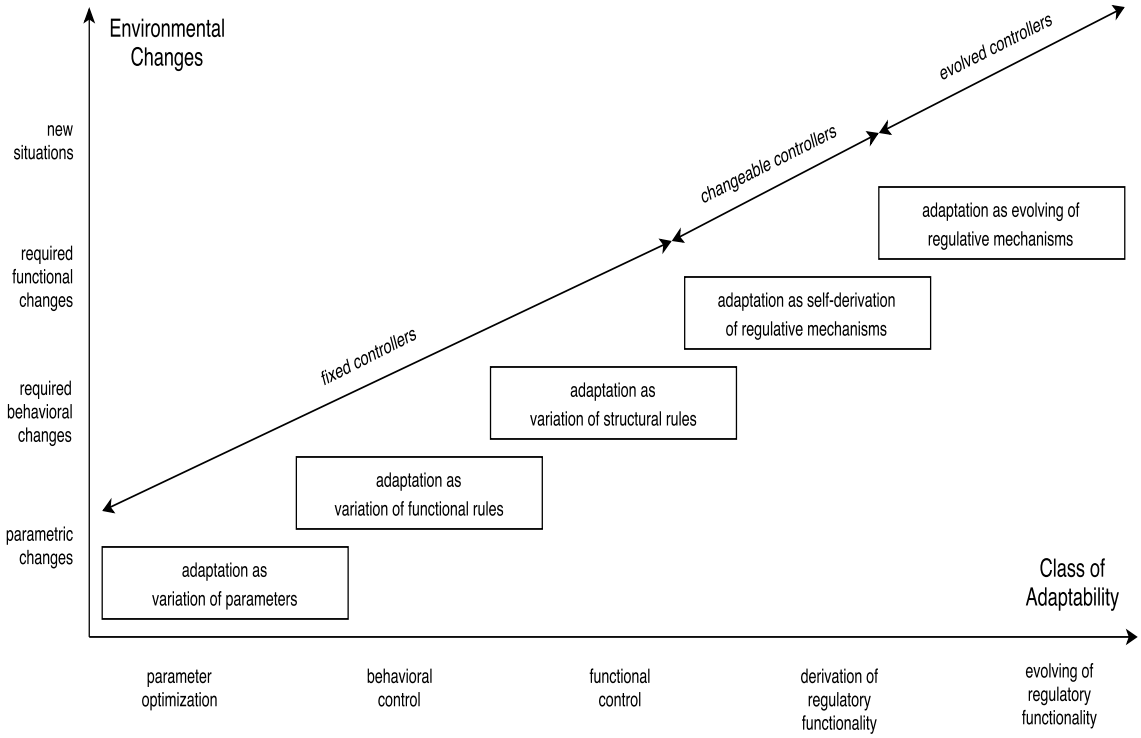

Figure 2.1. Different mechanisms of functional and structural adaptation in collective systems (Levi & Kernbach, 2010).

2.4.1 Domains of study

Adaptation is a generic concept, but within the engineering community, there are three domains that deal heavily with adaptability: control theory, artificial intelligence, and software-intensive systems (Haasdijk et al., 2013). Of the three, control theory's study of adaptability dates back the furthest. In 1959, Whitaker applied adaptive controls to aircraft and spacecraft (Whitaker, 1959). Two decades later, Egardt wrote about the stability of adaptive control (Egardt, 1979). Adaptive control theory continued to be studied in the 1980s (Anderson, Bitmead, Johnson, Riedle, & Parks, 1989; Narenda & Annaswamy, 1989), 1990s (Kristic, Kanellakopoulos, & Kokotovic, 1995), and is still an active area of study (Wilson, 2001).

Artificial intelligence (AI) researchers approach adaptation a bit differently than control theorists. Where adaptive control theory tends to address the "fixed controllers" in Figure 2.1, artificial intelligence research deals more with the "changeable controllers" and perhaps even the "evolved controllers." Beer, in the book "Intelligence as Adaptive Behaviour," claimed that traditional AI is not very flexible or adaptable. To remedy this, Beer discussed the modeling of an artificial insect's nervous system (i.e. a collective system made up of many neurons), in the context of adaptive behavior. Beer is not alone in looking to nature for inspiration (Kernbach, Thenius, et al., 2009). The adaptability of biological organisms is a popular topic in the field of AI (Floreano & Mattiussi, 2008). Keijzer challenged the notion of decision-making being inherently intelligent, and instead discusses adaptability in terms of cognition (or "active thinking") (Keijzer, 2003). The works discussed here are only a small sample of the debate about AI's embodiment and dependence on adaptation. A full literature review on this topic would be extensive.

Software-intensive and distributed (collective) systems, particularly in the context of business, rely on ever-increasing adaptability to keep up with ever-increasing complexity (SAP, 2005). However, unlike the large research topic of

collective robotic systems, distributed business systems focus more narrowly on adaptation in terms of scalability, self-optimization, self-protection, context awareness, and software reliability issues (Cheng et al., 2009; Haasdijk et al., 2013).

2.4.2 Self-adaptation

Modern literature, particularly in evolutionary computation, often makes a distinction between adaptation and self-adaptation (Beyer, 1996). Bäck defined dynamic parameter control, adaptive parameter control, and self-adaptive parameter control as all distinct from each other (Bäck, 2001). "Adaptive," according to Bäck, has the intuitive meaning of feedback-based regulative mechanisms, whereas "self-adaptive" describes mechanisms that can change regulative structures (Haasdijk et al., 2013). Figure 2.2 illustrates the relationships between adaptation and self-adaptation (and self-development, discussed in the next section).

2.5 Self-developmental systems

The body of research about developmental systems is widespread and multi-disciplinary. Evolutionary robotics (Nolfi & Floreano, 2004) has its roots in developmental systems research and often attempts to mimic biological evolution. The self-* features (e.g., self-monitoring and self-repair), as Kernbach referred to them, can emerge from the developmental approach and are related to adaptability, evolveability, behavioral emergence, and the control of long-term development (Haasdijk et al., 2013).

The study of self-developmental robotics (Oudeyer, 2004) may have originated in neuroscience, artificial neural networks, and evolutionary systems (Haasdijk et al., 2013). Rather than focusing on cognition and ontogenetic development (i.e. developmental systems research), self-development addresses issues like self-exploration, self-supervision, self-learning, and more (Lungarella, Metta, Pfeifer, & Sandini, 2003).

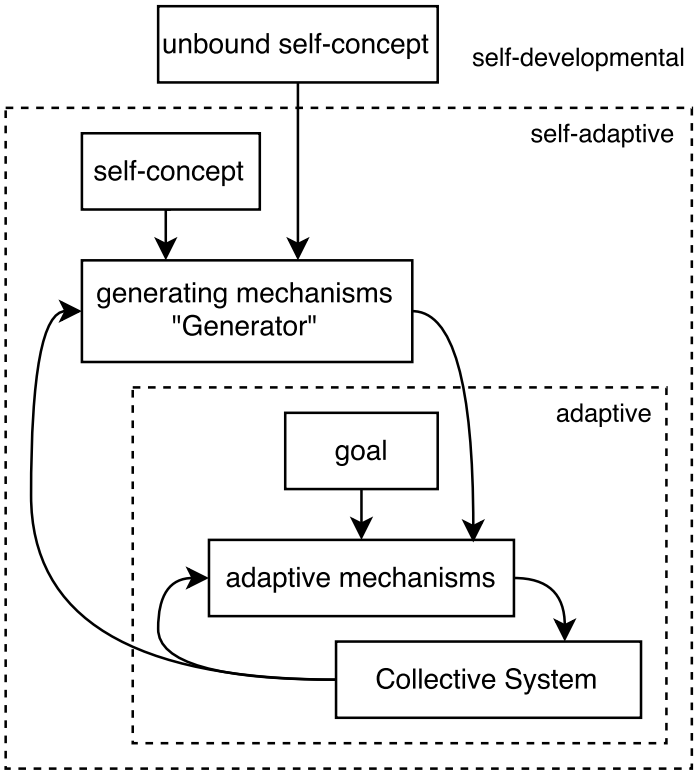

Figure 2.2. Generating relation between adaptive and self-adaptive mechanisms (Levi & Kernbach, 2010).

2.5.1 Open-ended evolution and unbounded self-development

> Open-ended evolution is characterized by a continued ability to invent new properties–so far only the evolution of life on earth (data partly from fossil records) and human technology (data from patents) have been shown to generate adaptive novelty in an open-ended manner (Rasmussen et al., 2004).

Kernbach talked about "unbounded self-development" as a robotic version of open-ended evolution. Both concepts deal with growing complexity (potentially indefinitely growing (Ruiz-Mirazo, Umerez, & Moreno, 2008)), but unbounded self-development also includes the ideas of embodiment, links between structures and functions, energy and homeostasis, and others. "When driving forces of adaptive processes are mostly bound, expressed by reward or fitness, the self-concept may include driving forces which are of unbounded character. In this way, self-development does not necessarily imply any evolutionary progress, but a progress driven by the unbounded force of self-concept" (Haasdijk et al., 2013). The idea of self-concept comes from psychology, which defines it as "conscious beliefs about the self that are descriptive or evaluative" (Kernis & Goldman, 2003; Markus & Wurf, 1987; McLean et al., 2007). In robotics, self-concept is a bit different (Kernbach, Levi, Meister, Schlachter, & Kernbach, 2009):

> To explain the idea of a self-concept and a structural generator, we consider the case when locomotion should have a specific form, such as a symmetric movement of legs, segmented (like by insects) construction of body, or there are imposed constraints or a priory desired properties. The self-concept contains in a compressed form a description of these constraints/properties.

Standish experimented with open-ended evolution via the artificial life simulators called Tierra and Avida (Standish, 2002). Although tempered by the

acknowledgment that longer simulation runs might provide different results, Standish concluded that an organism can undergo self-development without appreciably increasing in complexity. This work, along with (Spector, Klein, & Feinstein, 2007) and others, led Kernbach to ask (Haasdijk et al., 2013):

- Which processes can generate complexity?
- How should we control long-term unbounded development?

Which processes can generate complexity?

John von Neumann's well-known concept of cellular automata directly addresses the question of how complexity is generated in collective systems:

> ...synthesis of automata can proceed in such a manner that each automaton will produce other automata which are more complex and of higher potentialities than itself (von Neumann, 1966).

However, as Standish discovered (Standish, 2002) and as Kouptsov explained (Kouptsov, 2008), a system's complexity is independent of its size–"the self-similar structural production does not increase complexity" (Haasdijk et al., 2013). This conclusion leads to the notion that "structural production rules parameterized by random (environmental) values may lead to infinite growth of complexity and diversity and are candidates for the unbounded self-concept" (Haasdijk et al., 2013).

Before moving on, I feel it is important to make clear the difference between complexity and complication. Complicated systems are (ideally) explicitly and thoroughly designed (e.g., the Space Shuttle). They rely on expertise and careful division of tasks. Complex systems, on the other hand, are typically much less defined and controllable. They often have many moving parts, so to speak, and are theoretically capable of significantly greater degrees of adaptation (and perhaps even self-development) than complicated systems.

How should we control long-term unbounded self-development?

The controlling of self-development processes in artificial systems is both challenging and crucial, due to their ability to change their own structure and regulative/functional mechanisms. "Artificial adaptive systems with a high degree of plasticity (Levi & Kernbach, 2010) demonstrate a developmental drift" (Haasdijk et al., 2013). Developmental drift is the divergence of the system's properties and behavior from the system's goal, caused by the very independence and autonomy that enables self-development. The consequences of large-scale developmental drift, if carried far enough, lead to complex and self-aware artificial systems that may have different goals than we do (e.g., the "Terminator" scenario, in the extreme case).

Ever increasing complexity and unbounded self-development/concept in collective robotic systems could lead to emergent phenomena that are nearly impossible to trace back to a definable cause: artificial societies and cultures (Winfield & Griffiths, 2010), bio-technical hybrid systems (Novellino et al., 2007), animal-robot mixed societies (Caprari, Colot, Siegwart, Halloy, & Deneubourg, 2005), and synthetic biology (Alterovitz, Muso, & Ramoni, 2009), for example (Haasdijk et al., 2013). Currently, we do not have a good answer to the problem of controlling unbounded self-development, but this is an active area of research. For example, Stepney et al. address the engineering of emergence (and therefore, self-development) in the context of molecular nanotechnology via the Theory Underpinning Nanotech Assemblers project (Stepney, Polack, & Turner, 2006), but the study of this subject is still in its youth.

2.5.2 Developmental plasticity of collective systems

Developmental plasticity, like other concepts in collective robotic systems, originally comes from biology. It means different things to different disciplines of biology, but for example, neuroscience defines developmental plasticity as changes in

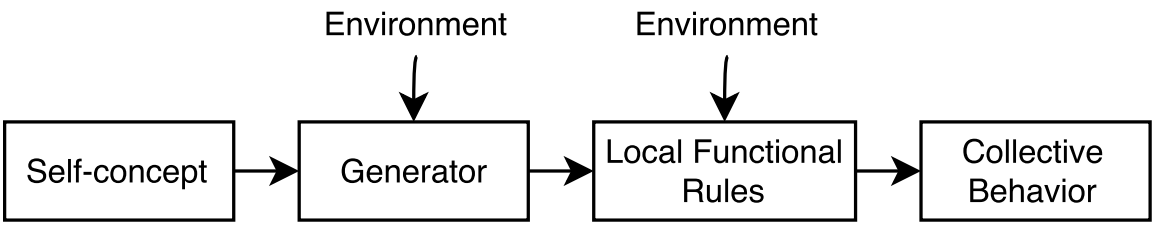

Figure 2.3. Sketch of the self-developmental approach in the functional case [of developmental plasticity] (Haasdijk et al., 2013).

neural connections, due to environmental stimuli (Foehring & Lorenzon, 1999). Of course, since robots do not yet possess any decent facsimile of a biological brain, developmental plasticity in (collective) robotics means something different. "The developmental plasticity of collective systems expresses a degree of flexibility and changeability of different regulative, functional, structural, or homeostatic components during the runtime" (Haasdijk et al., 2013; Kernbach, 2008).

Self-development and self-adaptation in collective robotic systems rely on developmental plasticity; it is the underlying enabler of self-driven change. Collective robotics breaks developmental plasticity into two categories: functional and structural. In the functional case, macroscopic properties are related to macroscopic behavior, like flocking, foraging, and aggregation (Haasdijk et al., 2013; Sahin, 2005). In the structural case, macroscopic properties correlate with common structures, which the robots aggregate to and act collectively to share resources (Haasdijk et al., 2013). Since function relies on structure, changing the structure also changes the functionality (and behavior) (Kernbach, 2008).

2.6 Is self-development/repair necessary in collective robotic systems?

The short answer, much like the beginning of this paper, is no. One could design a collective robotic system that had no ability for self-development or

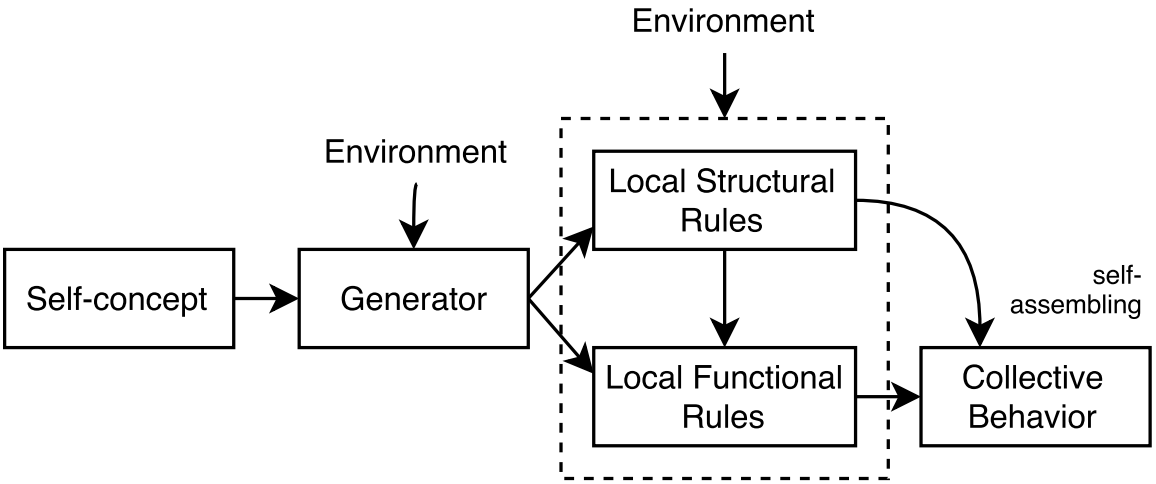

Figure 2.4. Sketch of the self-developmental approach in the structural case [of developmental plasticity] (Haasdijk et al., 2013).

self-repair. In fact, such a task would likely be trivially easy when compared to the challenge of designing useful emergence.

However, the short answer is not very interesting. Such a question needs some context and/or constraints. The primary advantage of collective robotics over "traditional" robotics (i.e., a small number of explicitly controlled machines) is its flexibility, adaptability, and self-* features. The reason collective robotic systems are studied so much is that we want to create artificial systems that can do useful work without a lot of "baby-sitting." Take, for example, a hypothetical mission to explore a distant planet. Would we want to send our best humanoid robot? No, of course not. Such systems are the opposite of robust and adaptable. We would (should) send a collective system that is designed to exhibit emergent self-development and certainly self-repair, given its remote and unpredictable environment. So is self-development/repair necessary in collective robotic systems? For the interesting ones, yes.

2.7 Biological collective systems (swarms)

The natural world is full of examples of collective systems/swarms. When engineers look to nature for inspiration about swarms, animals and insects usually take center stage. From the human perspective, this makes sense, since we can readily observe animals and insects. Animal and insect individuals tend to be large enough to see with the naked eye, and they operate in the same magnitude of time as we do.

Honeybees (Kernbach, Thenius, et al., 2009; Schmickl et al., 2013; Seeley, Kirk Visscher, & Passino, 2006; Szopek, Schmickl, Thenius, Radspieler, & Crailsheim, 2013) and ants (Meng & Jin, 2011) are popular choices when engineers study insect swarms, likely due to the accessibility to and observability of these species and their dramatic swarm behaviors. Moving up in the food chain, birds and fish get a lot of attention for the same reasons (Couzin, 2009; Couzin, Krause,

James, Ruxton, & Franks, 2002; Giardina, 2008). Nature, however, has more than insects and animals to offer in the way of collective systems.

2.7.1 Plants as swarms

Considering swarms only as "large groups of simple autonomous agents interacting locally" (Simes et al., 2011) opens the door to all levels of biology being studied as swarms, even all the way down to bacteria (Atkinson & Williams, 2009). Plants fall firmly within such a definition of a swarm. Simoes et al. phrase this idea nicely (Simes et al., 2011):

> All directional growth decisions and a majority of environmental sensing are made in the apex [(the tip of the root)]. [...] Since there is no anatomic evidence for a central sensing and decision unit and considering the rather low computational capacity of a plant cell (compared to neuronal systems of animals, for example), it appears meaningful to consider the apex as a simple autonomous unit taking decisions on own account (Baluska, Mancuso, Volkmann, & Barlow, 2004). [...] When looking at the root as a collective, growth patterns are not chaotic, but seem to follow a higher order, and emerge as a result of the individual decision-making of the apexes.

The literature covering plants as swarms and as inspiration for robotics is quite thin, but some works do exist. Brabazon et al. discussed several plant-inspired computational algorithms (Brabazon, ONeill, & McGarraghy, 2015), although not specifically in the context of robotics. Baluka et al. even went so far as to propose that the tips of plant roots act as a primitive, decentralized brain (Baluska et al., 2004). Some research attempts to create a synthetic copy of the physical structure of a plant root, although this is experimental, at best (Mazzolai et al., 2011; Tonazzini, Popova, Mattioli, & Mazzolai, 2012; Wooten, 2016).

European Space Agency study

When we change the focus to looking at plant roots as simply inspiration for a collective robotic system, rather than a blueprint, many more options become available. Of primary interest is a report released by the European Space Agency (ESA) that borrows exploration and nutrient-finding strategies from plants to explore unknown terrain with a collective robotic system (Simes et al., 2011). The ESA report discussed the biological factors (such as in (Poovaiah & Reedy, 1995)), motivations, and justifications (Allen et al., 2007; Doussan, Pagès, & Pierret, 2009), and presents a simulation of plant root growth (see Figure 2.5). The authors of the report also provided the beginnings of a framework for a robotic sensor web (Delin et al., 2005) that is based on their understanding of plant roots (see Figure 2.6), which provides significant motivation for this dissertation.

Some aspects of plant roots do not transfer directly to a robotic implementation, such as soil stigmergy (see Figure 2.7) and the scale of branching. Simoes et al. dealt with the limited branching by choosing to use additional robots only for branching and not for elongation, which will not work outside simulation (due to finite wireless radio range). The lack of soil stigmergy was dealt with by simply keeping each simulated robot up to date with a difference map, which again is a very non-trivial problem in a real robotic system.

The ESA study directly ties the nutrient-finding strategies of plants to a robotic system for space exploration. Their methodologies are thorough enough to provide a foundation for the proposed research but need considerable adaptation and extension to be implemented in a real robotic system. Since the ESA report so heavily inspired this research, I contacted the report's authors to ask about extending their work. Six out of the seven authors responded with excitement and offers of assistance, and also asked to be kept abreast of any major progress.

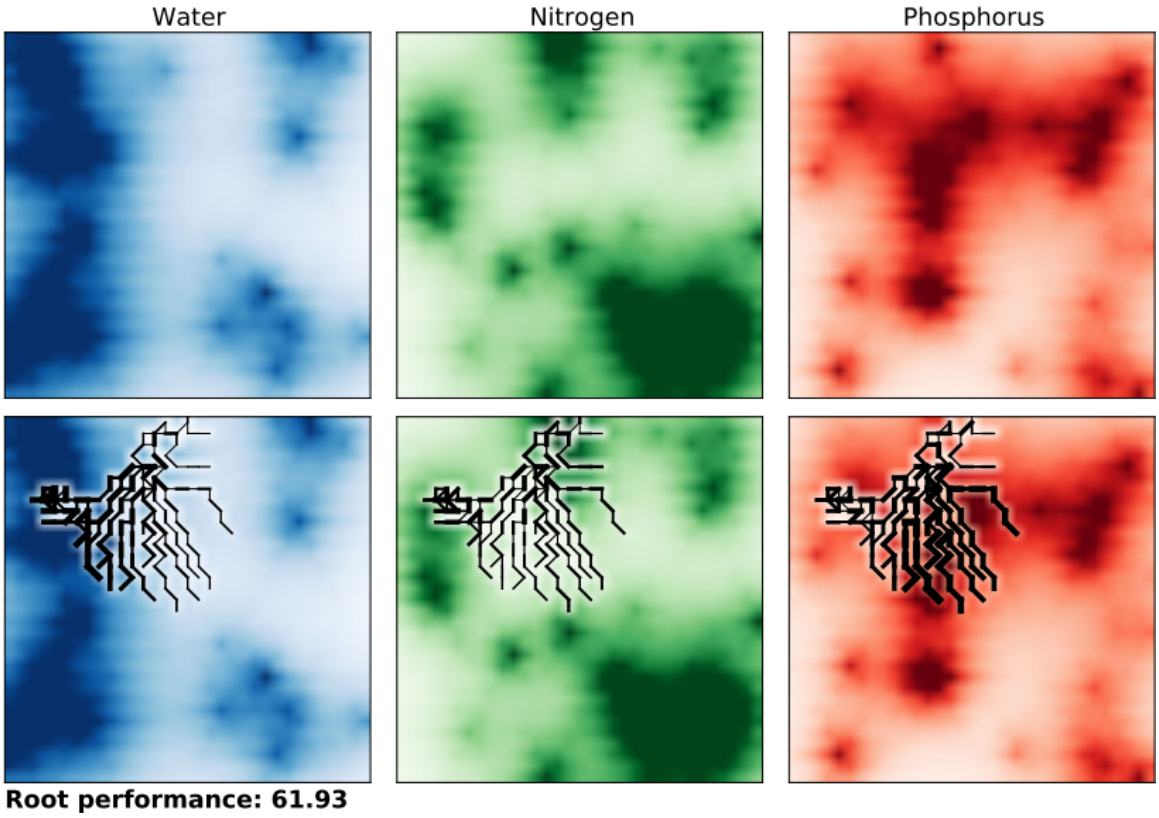

Figure 2.5. A root growth simulation at times t = 0 and t = 200. The root extracted (58.27, 53.95, 73.56) out of a soil initially holding (555.59, 542.37, 499.89), a fraction of (0.10, 0.10, 0.15) (Simes et al., 2011).

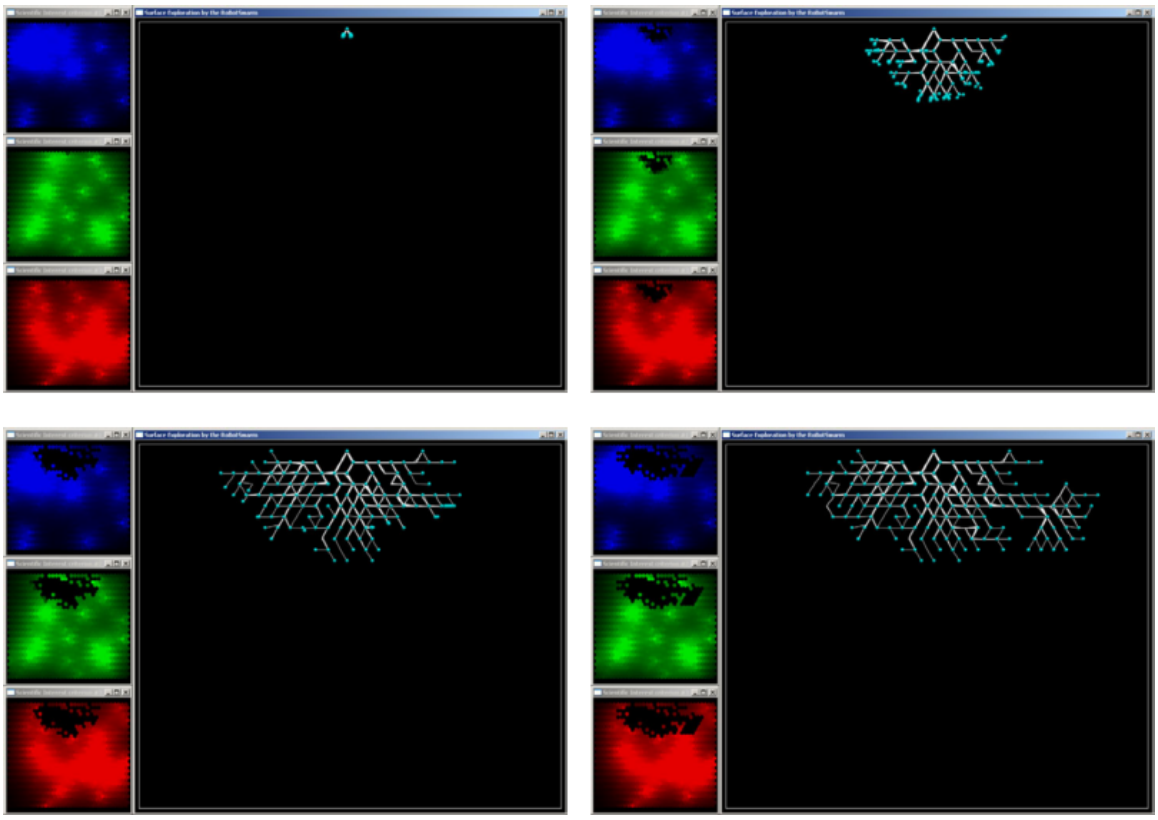

Figure 2.6. Sensor Web deployment on scenario A (256 robots) (Simes et al., 2011).

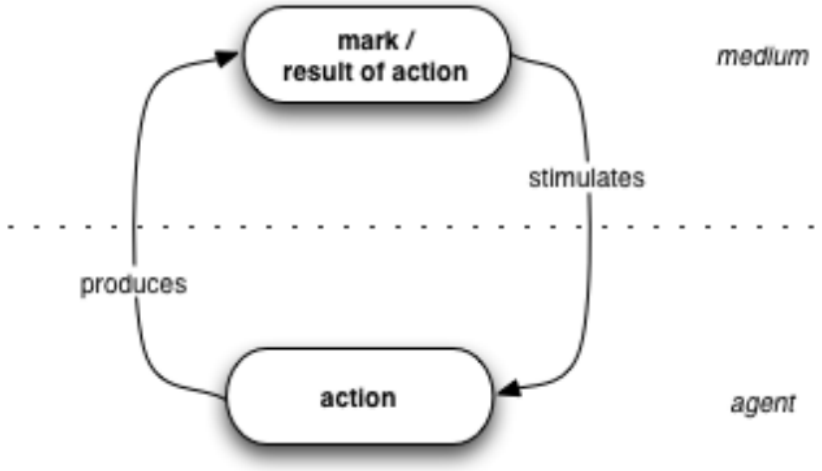

Figure 2.7. Basic components of stigmergy (Heylighen, 2011).

2.7.2 Natural inspiration

Why do we look to nature for guidance when studying collective systems? The simple answer is that nature has it figured out, at least much more than we do. Honeybees create nurturing hives, ants create thriving colonies, fish survive longer in schools, plants grow in nearly every Earthly environment, and so on. Biological collectives display remarkable success, so we attempt to reverse engineer that success. With regard to swarm robotics, specifically, Garnier stated that "robots require a complete specification," which means a formal, mathematical description must be created that at least approximately quantifies the link between individuals and the group (Meng & Jin, 2011). An explicit, mathematical description enables iteration and improvement (e.g., Figure 2.8), which increases both the performance of the engineered system and our understanding of the biological system. As George Bernard Shaw said, "Imitation is not just the sincerest form of flattery - it's the sincerest form of learning."

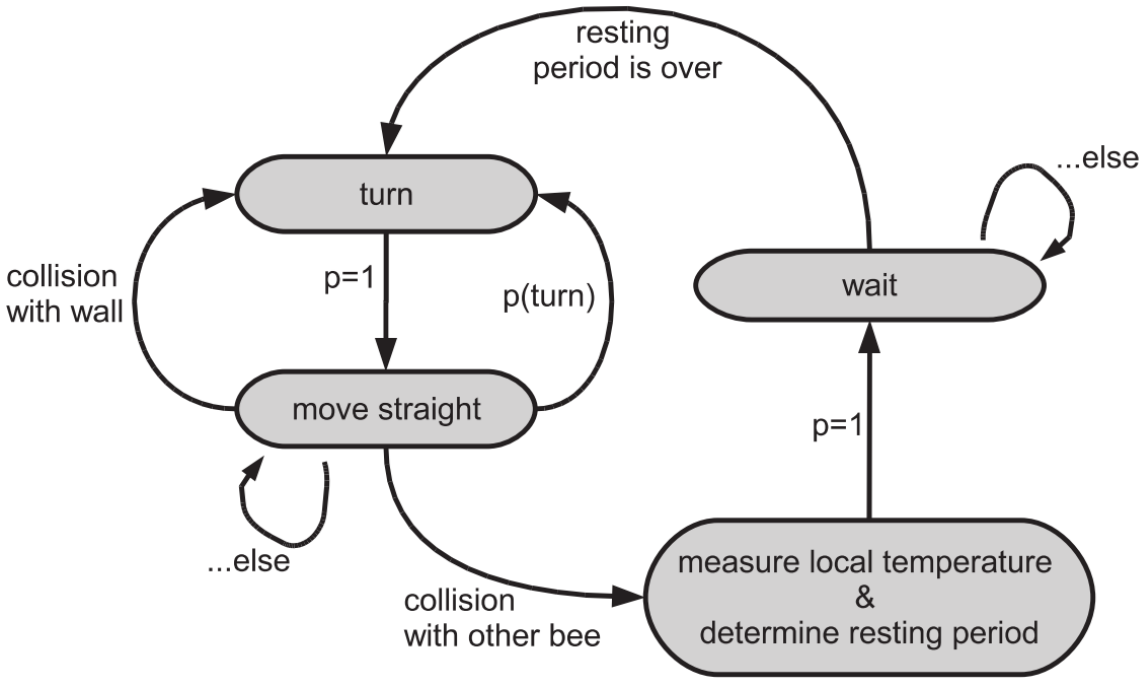

Figure 2.8. Finite state automaton that can describe the observed bee behavior (Kernbach, Thenius, et al., 2009).

CHAPTER 3. STARS OF THE SHOW (ROBOTS)

3.1 Brief Introduction to the Experiments

Before talking about the robots, it is important to provide some context about the tests they execute. Their objective is to, via methods inspired by plant roots seeking nutrients, locate radio beacons in an area, but without specific knowledge of the environment. The robots work as a team and must be able to communicate, as well as determine the directional bias of specific radio signals. The environments are not strictly controlled, such as in a laboratory, but are assumed to be flat and free of obstacles (e.g., parking lot). The experiments and methodologies are discussed further in the coming chapters, but with this minimal explanation, I can give the basic requirements for the robots.

3.2 Requirements

- Less than $1000 in parts per robot
- Small enough to be easily carried and transported
- Capable of communicating with each other over short distances (<100 meters)
- Capable of tracking the source of a radio transmission
- Enough energy storage to last for roughly a full day (about 8 hours) of operation
- Field-repairable

3.2.1 Build vs. Buy

The dilemma of building versus buying machines like this is one that many engineers deal with. There are many ready-to-run robotic platforms available today, but after searching, none of the available platforms met all of the requirements listed above. Finding a machine that could be made, without undue effort, to do what I needed it to do was not a problem. Finding that machine for under $1000 was not possible at the time.

The decision to build my own robots for this experiment was a big commitment. Designing, acquiring, and building the robots took several months. Working out all of the obvious hardware bugs took another considerable chunk of time.

3.3 Design

3.3.1 Wireless Communication

Plant roots have physical connection to each other and to the main body. Since that is not very feasible in the case of mobile robots, a wireless communication technology had to be employed. The key considerations when choosing which technology to use were cost, power consumption, and existing mesh networking capabilities. Mesh networking is required because there is not an omnipresent central connection point. The robots must be able to operate without a direct connection to the coordinator, so they need to relay each other's messages. Creating such a networking protocol is outside the scope of this research.

Several wireless technologies that meet at least some of these requirements are readily available. The easiest to acquire and use looked to be Digi's ZigBee (specifically, XBee) radios. These radio modules are inexpensive and low-power when compared to common wireless protocols like IEEE 802.11. ZigBee is designed to be a stand-alone wireless network for "internet of things" devices and has a fairly

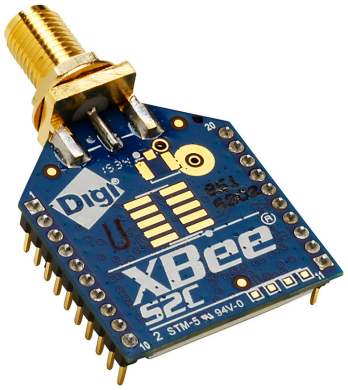

Figure 3.1. XBee radio module.

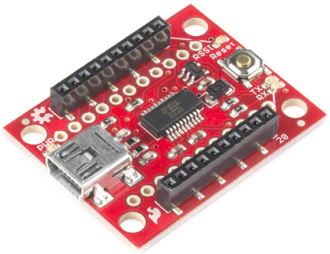

Figure 3.2. SparkFun XBee Explorer USB.

mature mesh-like protocol. The low-power XBee radios operate at 2.4 GHz, which made it fairly easy to find antennae. Coupled with a USB board from SparkFun, these radios fit nicely into the design.

Each robot carries two XBee radios. One is attached to an omnidirectional antenna and is used for communication with the coordinator and with other robots. The second is attached to a directional patch antenna and is primarily used for direction finding (although it does do limited communication). The directional antenna needed to be as light as possible, since it is mounted relatively high on a mast, and have as narrow of a beam width as was reasonably achievable to maximize the precision of the directionality.

Figure 3.3. L-Com 2.4GHz directional patch antenna.

After working with these radios, it is apparent that they are not well-suited for this application. They were certainly not designed for it, and despite fulfilling the initial requirements, the ZigBee protocol caused, or at least exacerbated, problems later in development. I will discuss this more in Chapter 5.

Why not use two directional antennae?

Two directional antennae on a mobile robot have been proven to provide connectivity and enhanced range over omnidirectional antennae (Min, Matson, & Jung, 2016). The choice to use an omnidirectional antenna as the primary communication antenna is due to practicality during testing. There are additional benefits to using only one directional antenna:

- Omnidirectional antennae tend to be less expensive than directional antennae
- Each directional antenna requires a rotation mechanism, which increases the cost and complexity of each robot

Figure 3.4. 6S 6.6Ah lithium polymer battery from HobbyKing.

- Using an omnidirectional antenna for primary communication enables the option of using the robot itself as the rotation mechanism for the directional antenna

3.3.2 Electrical

The design of the electrical system is intended to be as easily assembled/repaired as possible, to allow for field repairs. The primary components are the battery, voltage regulators, DC motor controller, stepper motor controller, Arduino Micro, current sensor, fuse, power switch, and the voltage divider. The battery is a 6 cell lithium polymer (LiPo) that operates at 22.2 volts (nominally). I chose this relatively high voltage to make it easy to use step-down voltage regulation for the 12V and 5V components. The batteries have a capacity of 6.6 Amp-hours.

There are three voltage regulators on-board. The 5V regulator is to power the Raspberry Pi 3 (discussed below). The two 12V regulators provide power to the stepper motor and the drive motors. I chose to give the stepper motor its own smaller 12V regulator to avoid loss of power if all the drive motors stalled. The drive

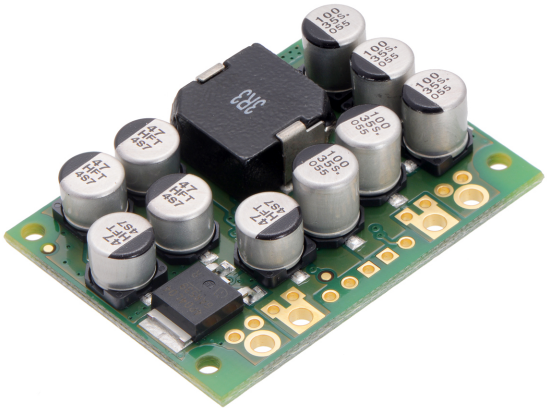

Figure 3.5. 12V step-down voltage regulator for drive motors.

Figure 3.6. 12V step-down voltage regulator for stepper motor.

motors have a stall current of 5 A, which would more than consume the large 12V regulator's capacity of 15 A. If this happened, the stepper motor would lose power, which is not desirable. Therefore, the stepper motor has its own voltage regulator.

The drive motors are controlled by a two-channel brushed DC motor controller. To reduce the cost of each robot, I used the two channels to control all four wheels by utilizing a skid steer scheme. The wheels (motors) on each side of the robot are connected together, effectively making the robot steer like a tank. The motor controller is a RoboClaw 2x7A, meaning each channel can only pull a

Figure 3.7. 5V step-down voltage regulator for Raspberry Pi.

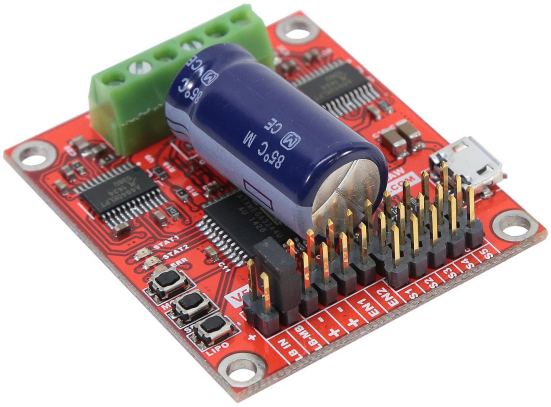

Figure 3.8. Controller for drive motors.

constant 7 A. The combined stall current for two motors is 10 A, which exceeds the controller's capacity, but I considered the risk of damage to be minimal. None of the planned test scenarios involved situations where the motors might stall, and the current draw of two combined motors during operation is less than half of the controller's stated capacity.

The robots' only means of direction-finding is the directional 2.4 GHz antenna. In order to search in all directions, that antenna needs to rotate, which is the job of the stepper motor. The stepper motor controller is capable of delivering 2

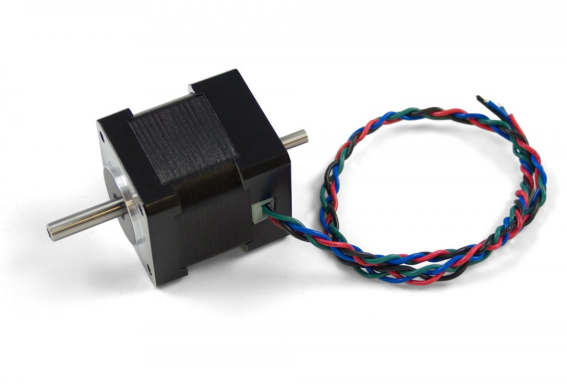

Figure 3.9. Stepper motor to rotate the directional antenna.

A per coil, but the stepper motor is rated for 1 A per coil, giving plenty of extra capacity. Although the controller is capable of micro-stepping, I use full steps. There are 200 steps per revolution, or 1.8°per step. Since the antenna has a beam width of 30°, micro-stepping is overkill.

Interfacing with the stepper motor controller, current sensor, GPS receiver, and the voltage divider for the battery is the responsibility of the Arduino Micro. I chose to handle these tasks with an Arduino, rather than directly with the Raspberry Pi, because the software libraries are more available for Arduino. The Arduino is powered via USB to the Raspberry Pi, and the 5V output of the Arduino provides power to the current sensor, stepper motor driver, and GPS receiver.

The original plan was to measure voltage and current concurrently, thus providing the data necessary to calculate power usage. The current sensor was chosen to provide as much resolution as possible within the amperage range the robots pull. The output of the sensor is an analog voltage which is connected to one of the Arduino's analog pins, enabling rapid reading of current consumption to make the power calculation as accurate as possible. As development of the system progressed, it became clear that the amount of ZigBee network traffic was a choke point, so the current sensor readings were not sent. Still wanting to have some measure of power consumption, I had two options. First, I could store the current

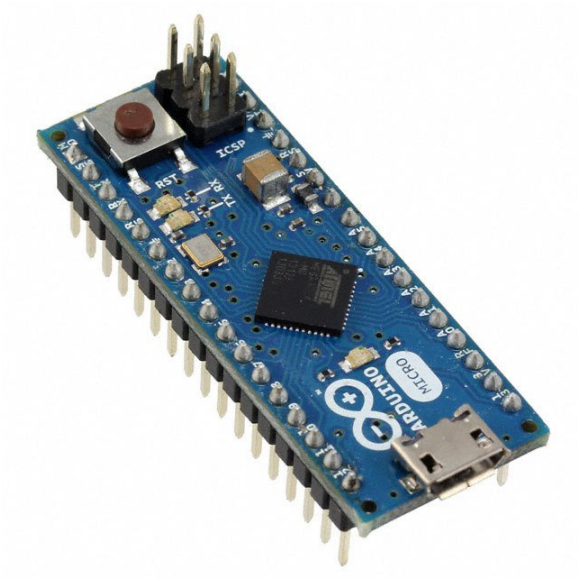

Figure 3.10. Arduino Micro.

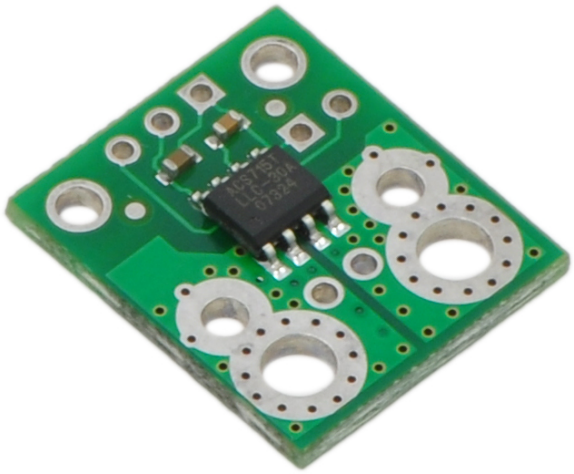

Figure 3.11. Current sensor.

and voltage measurements on the Raspberry Pi and extract them after a test. Second, I could measure the voltage of the battery after every test and, combined with a characterization of the battery, calculate the power consumed during that test. For the sake of simplicity and time, I chose the second option.

The aforementioned voltage divider consists of a fixed resistor and a potentiometer, with an analog Arduino pin connected between them. Equation 3.1 describes a voltage divider.

$$V_{\text{out}} = V_{\text{in}} * \frac{R_2}{R_2 + R_1} \tag{3.1}$$

Knowing that a 6S LiPo battery has a fully-charged voltage of 25.2V gives us V_{out}. The corresponding V_{in} should be 5V. Selecting a common 3.9kΩ resistor, plugging these values into Equation 3.1, and solving for R_2:

$$R_2 \approx 965\Omega \tag{3.2}$$

A potentiometer with a maximum resistance of 1kΩ would probably work, but to avoid any issues associated with the manufacturing vagueries of inexpensive potentiometers, I chose to go up to 2kΩ.

Since the Arduino can not handle the battery's voltage directly, the potentiometer is adjusted such that when the battery is fully charged, the Arduino reads 5V (or an analog-to-digital converter output of 1023) on its pin. The ATmega32U4 at the heart of the Arduino has 10-bit analog to digital converters. Combined with the maximum 5V rating of the ATmega32U4's pins, the resolution of the ADC $= 5V/2^{10} \approx 0.0049V$.

The selected resistances give a voltage divider ratio of $R_1/R_2 = 4.04$. To map the ADC values to battery voltages, multiply each voltage "chunk" by the voltage represented by the ADC values:

$$V = (value_{ADC} * resolution_{ADC} * 1) + (value_{ADC} * resolution_{ADC} * 4.04) \tag{3.3}$$

Equation 3.3 shows that a battery voltage of 20V, for example, corresponds to an ADC value of 810.

A slow-burn fuse rated for 20A was chosen to roughly correspond to the maximum power draw of all the voltage regulators combined, while allowing for transient spikes. The power switch is a matter of convenience and is a simple single pole, single throw (SPST) toggle switch.

One more thing...

Each robot carries a GPS receiver. This receiver is not used for localization or navigation. It was intended to be used simply to log each robot's position during tests to facilitate nice-looking diagrams. Due to time constraints, I abandoned this effort.

3.3.3 Computation

The primary piece of computational equipment is the Raspberry Pi 3 (RPi), with the Arduino Micro providing limited support, as described previously. The RPi is not doing any particularly heavy numerical computation (although it is doing some arithmetic, finding mean and median, etc.). Rather, it is running multiple Python threads (via Robot Operating System, discussed later) that manage asynchronous input and output, constituting a sort of state machine.

3.3.4 Mechanical/Physical

Selecting a chassis to carry all of the discussed equipment was an important part of the design process. Initially, I considered constructing a custom chassis, but eliminated this option as it became too much of a time/resource sink. None of the components are particularly heavy or hazardous, so a relatively simple plastic chassis made by ServoCity fit the bill. Out of the ten robots I built, eight are ServoCity's Scout and two are the Warden. They function the same, but I had to mix the chassis designs due to supply issues. The electronic components are bolted

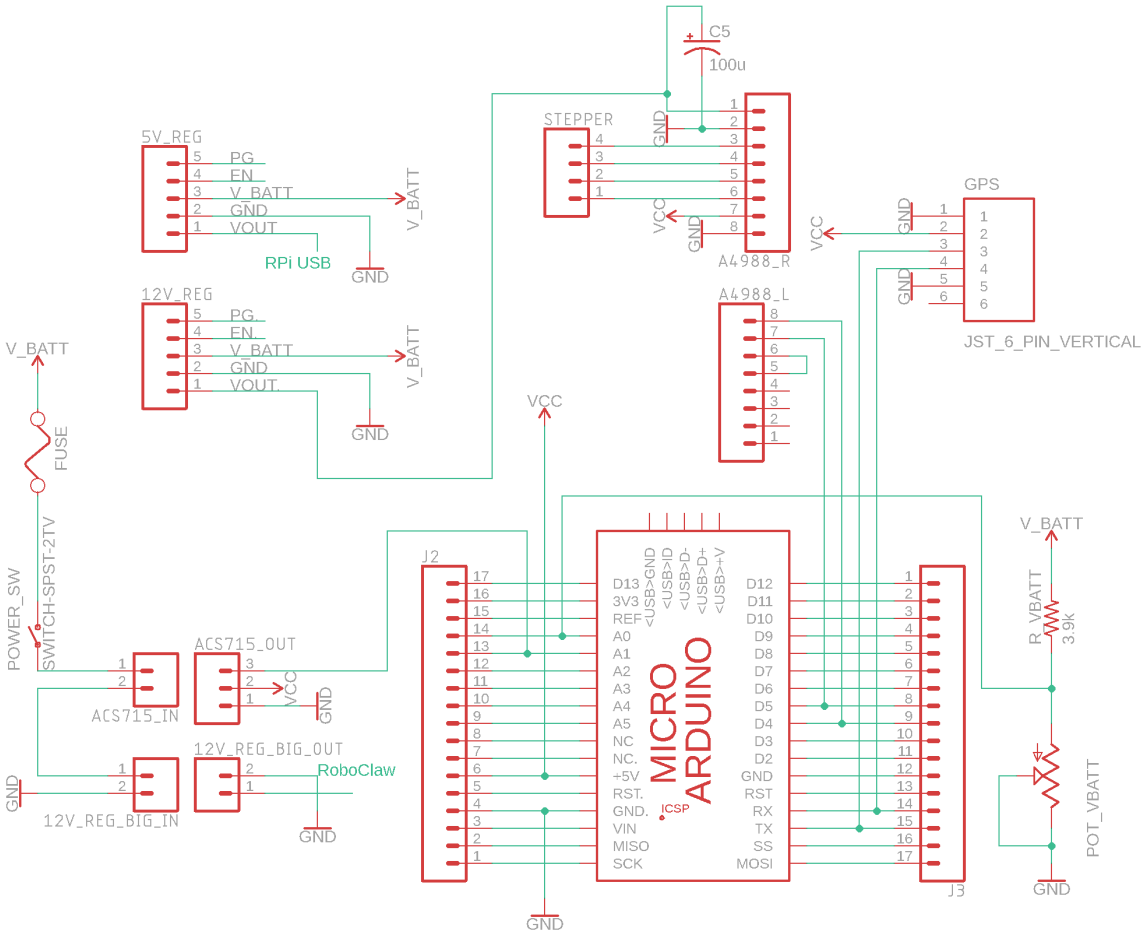

Figure 3.12. Electrical schematic for the robot.

Figure 3.13. Raspberry Pi 3 Model B.

to the chassis (on top of standoffs). The battery is contained by either bolts (on the Warden) or a 3D printed cage (on the Scout).

One of the reasons I chose ServoCity's chassis kits was the compatibility with their Actobotics aluminum structural components. Actobotics allowed me to construct a solid, rotating mast to mount the directional antenna and stepper motor to (and various other components, in the case of the Warden). After a tricky trial and error process, I had a reliable, reproducable mast design.

Figure 3.14. ServoCity Scout robot chassis.

Table 3.1.: Bill of materials for the robots. Does not include wires, cables, screws, bolts, nuts, etc.

Component	**Vendor**	**Dollars**	**Quantity**
Scout chassis kit	ServoCity	169.99	8
Warden chassis kit	ServoCity	199.99	2
RoboClaw 2x7A motor controller	ServoCity	69.99	10

Component	Vendor	Dollars	Quantity
XBee radio module	DigiKey	18.19	20
XBee Explorer USB	SparkFun	24.99	20
Raspberry Pi 3	Amazon	38.00	10
MicroSD memory card	Amazon	5.99	10
6S LiPo battery	HobbyKing	42.30	10
Stepper motor (35STH36)	Phidgets	16.00	10
A4988 Stepper Motor Driver Carrier	Pololu	7.49	10
8dBi Omnidirectional Antenna	Amazon	7.99	10
14dBi Flat Panel Antenna (HG2414P)	L-Com	57.96	10
12V, 15A Step-Down Regulator	Pololu	39.95	10
12V, 2.2A Step-Down Regulator	Pololu	9.95	10
5V, 2.5A Step-Down Regulator	Pololu	8.95	10
Arduino Micro	DigiKey	20.63	10
XT90 pigtail connector	Amazon	7.99	10
2kΩ trim potentiometer	DigiKey	0.78	10
3.9kΩ resistor	DigiKey	0.10	10
100uF capacitor	DigiKey	0.36	10
ACS715 Current Sensor Carrier	Pololu	9.95	10

Component	Vendor	Dollars	Quantity
NEMA 14 stepper motor mount	ServoCity	6.99	10
0.77" pattern set screw hub	ServoCity	4.99	10
Side Tapped Pattern Mount D	ServoCity	5.99	30
90°angle bracket	ServoCity	4.99	40
12" aluminum channel	ServoCity	9.99	16
10.5" aluminum channel	ServoCity	8.99	2
3"x1.5" pattern plate	ServoCity	1.59	20
4.5"x1.5" pattern plate	ServoCity	1.99	10
Fuse holder	Amazon	2.98	10
Glass tube fuse	Amazon	0.79	20
Power switch (SPST toggle)	Amazon	11.99	10
Screw terminal (4-pin)	Pololu	2.25	10
Screw terminal (2-pin)	Pololu	2.25	70
Perma-Proto breadboard pack	AdaFruit	12.50	7

3.4 Construction

Deciding on the design and components was only part of the battle. Constructing the first robot was a tricky task. I had to figure out component layout, wiring, develop a workflow, and work out hardware bugs before moving on to

Figure 3.15. ServoCity Warden robot chassis with some Actobotics aluminum structural components.

building the other robots. The primary considerations during this process were structural soundness and field repairability, since the robots would have to be transported to and from, as well as be repaired in, the field. Minimizing the number of wires traversing between the inside and top chassis layers was important, since these all have to be disconnected to remove the top for repairs/adjustments. The fuse, for example, would be easiest to replace if it was placed on top of the robot. However, that meant two more thick wires going to the top, so I placed it on the

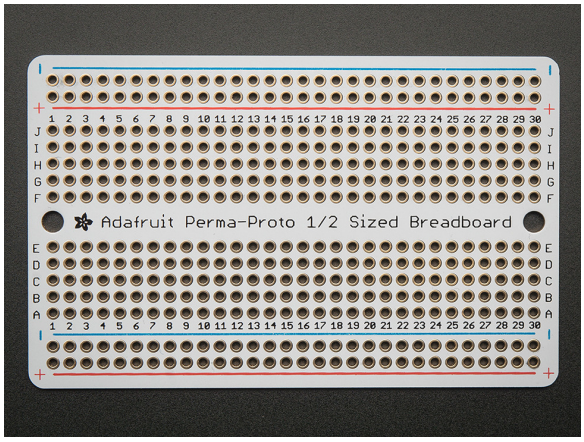

Figure 3.16. AdaFruit Perma-Proto Board.

edge of the bottom chassis plate. This placement allows easy replacement and does not cause unnecessary wire runs.

Securing each component to the chassis required the use of plastic standoffs and nuts/bolts of varying sizes, which meant drilling dozens of holes. The next step was constructing and soldering the proto-board circuitry; a tedious, slow, delicate, occasionally injurious process. All structural hardware, both for the chassis and the Actobotics components, was installed with a threadlocker compound so I wouldn't have to worry about anything jiggling loose during testing.

3.5 Beacons

None of this effort would amount to much if the robots had nothing to find. Radio beacons serve as the "resource" that the system is seeking. Using radios, specifically the same type of radios onboard the robots, means no additional sensors are needed. Additionally, and perhaps more importantly, the use of radio beacons provides a natural gradient for the robots to follow (like chemicals in the soil being sought by plant roots). Early attempts at establishing a gradient for the robots to sense and follow included sound and color, but these were quickly disqualified. Sonic beacons provide a natural gradient, but would require test conditions that would be

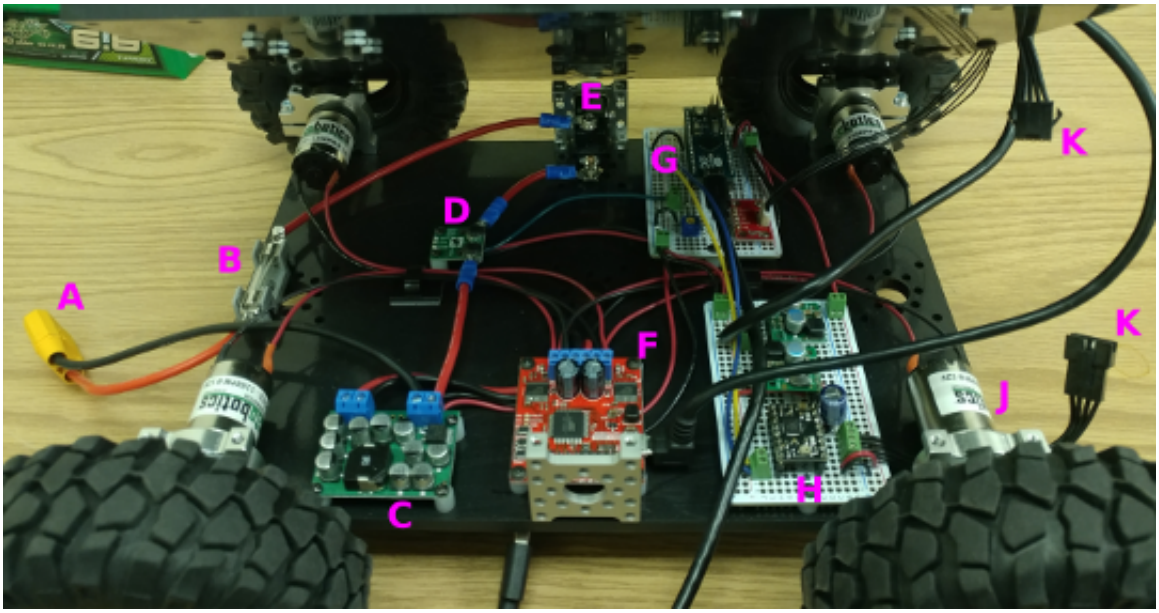

Figure 3.17. (A) Battery connector. (B) Fuse. (C) 12V regulator for drive motors. (D) Current sensor. (E) Power switch. (F) Drive motor controller. (G) Arduino Micro. (H) Stepper motor driver. (J) Drive motor. (K) Stepper motor connector.

Table 3.2.

Bill of materials for the beacons.

Component	**Vendor**	**Dollars**	**Quantity**
Arduino Uno	SparkFun	24.95	10
XBee Arduino shield	SparkFun	14.95	10
XBee module	DigiKey	18.19	10
AA battery holder	Amazon	4.39	10
2.4GHz 8dBi Omnidirectional Antenna	Amazon	7.99	10

difficult to ensure. Color is less prone to environmental interference (although not immune), but it does not provide a natural gradient. The idea of repeatedly creating color gradients of some sort (in loosely controlled conditions) is rather unappealing, so radio beacons were the winner.

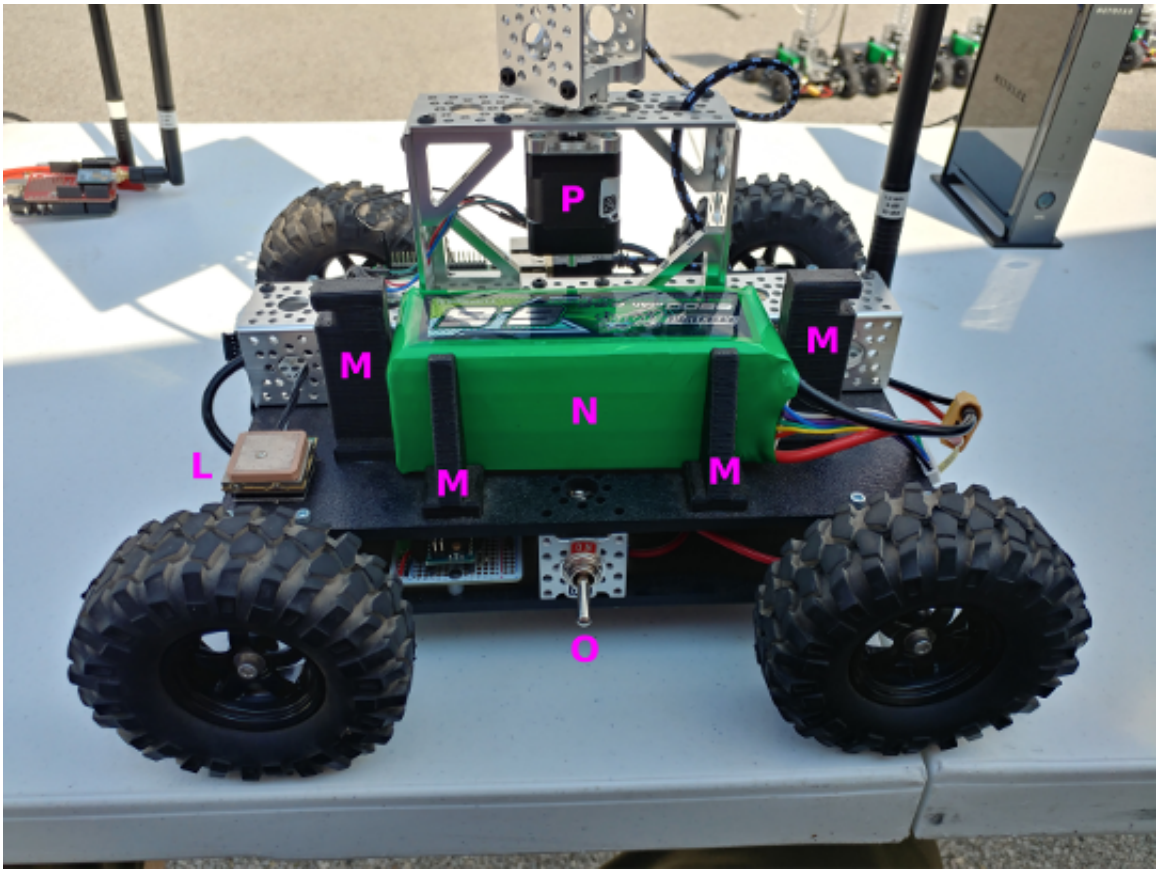

Figure 3.18. (L) GPS receiver. (M) Battery cage. (N) Battery. (O) Power switch. (P) Stepper motor.

3.6 Software and firmware

The software and firmware running the system are split into four components: beacons, each robot's Arduino, each robot's RPi, and the Nexus. Much, but not all, of the roadmap for the software development was based on intuition and revision through testing. There were two key components that I pre-determined: the overall architecture for the robots' Raspberry Pis and for the nexus. The robots were conceived as state machines, of a sort. In early revisions of the system, this manifested as one giant, multi-threaded Python script, then moved to each state being its own Python script with state and other relevant data being

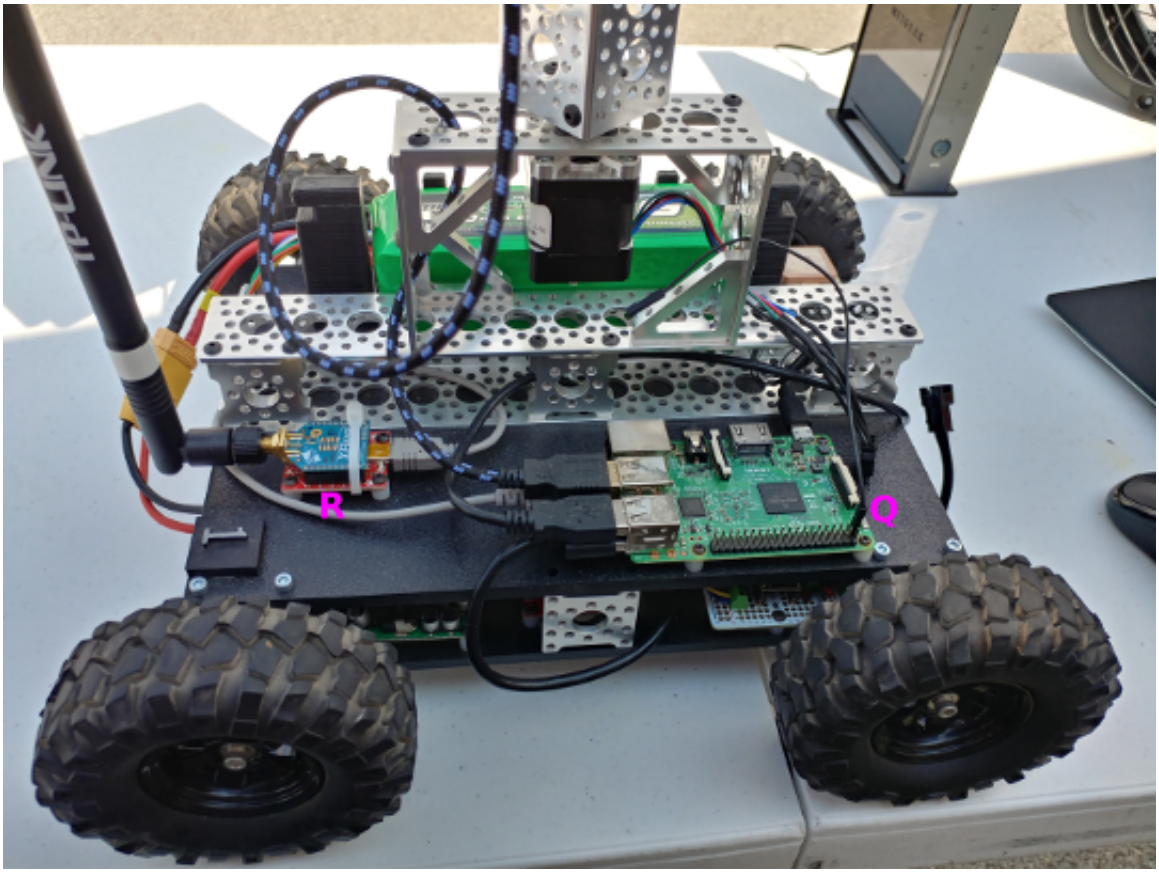

Figure 3.19. (Q) Raspberry Pi 3. (R) XBee radio used for communication.

passed between scripts. However, this approach proved nothing but problematic, even after several complete rewrites. I eventually switched over to Robot Operating System onboard the robots, which is discussed in Chapter 4. The nexus is still essentially what I intended it to be from the beginning. It acts as a point of deployment for the robots, monitors the system, and dispenses resources when requested, similar to some of the functions of the main body of a plant.

My intuition was aided by, or anchored to, a somewhat top-down design methodology within each "piece" of software. Accordingly, I determined what needed to be accomplished, processed, output, etc., then built functions and scripts to assemble a code structure for that purpose. As these "pieces" came together, system functionality coalesced. Figure 3.30 looks chaotic, but it is a good

illustration of the result of this design methodology. Each ROS node (oval) is its own Python script, with messages between ROS nodes represented as arrows. Per my original architecture for the robots' software, each state has its own ROS node. As development continued, I added supporting ROS nodes and messages when I thought it logical to separate functionality.

3.6.1 Robots

Arduino Micro

The general idea of this code is to interface with the GPS receiver, current sensor, battery voltage divider, and stepper motor driver. Additionally, it communicates this data to the RPi upon request. Figures 3.22 and 3.23 illustrate the robot's firmware.

Raspberry Pi

Each robot's RPi is using Robot Operating System (ROS) to manage its software by running ROS locally. ROS is not used to communicate between robots or to the Nexus. Its use onboard the robots arose from the difficulty of managing an asynchronous, multi-threaded software system (which is discussed more in Chapter 4). An overview of the robot's software is provided in Figures 3.24, 3.25, 3.26, 3.27, 3.28, 3.29, and 3.30.

3.6.2 Beacons

The XBee radio beacons broadcast a simple packet every n seconds to enable the robots to find them. When a beacon is found, it is sent a message to stop broadcasting. Once the "resource" is located, the system should not waste energy finding that same "resource" again. This disabling of a beacon is not directly communicated to the other robots. Instead, the disabling of the beacon

communicates its status through stigmergy, like the depletion of chemicals in plant root-bearing soil. Figure 3.31 illustrates the beacon's operation.

3.6.3 Nexus

The Nexus is essentially a collection of ROS nodes running on a computer connected to an XBee radio that is in "Coordinator" mode. The Nexus has a few roles to play, but does not actively make navigation decisions for the robots. The robots do that for themselves. The Nexus initiates the system by checking which robots are present and healthy, dispensing parameters to them, deploying them, and managing which requests get approved when there are not enough idle robots to accommodate all the requests. Figures 3.32, 3.33, 3.34, and 3.35 describes the Nexus's behavior.

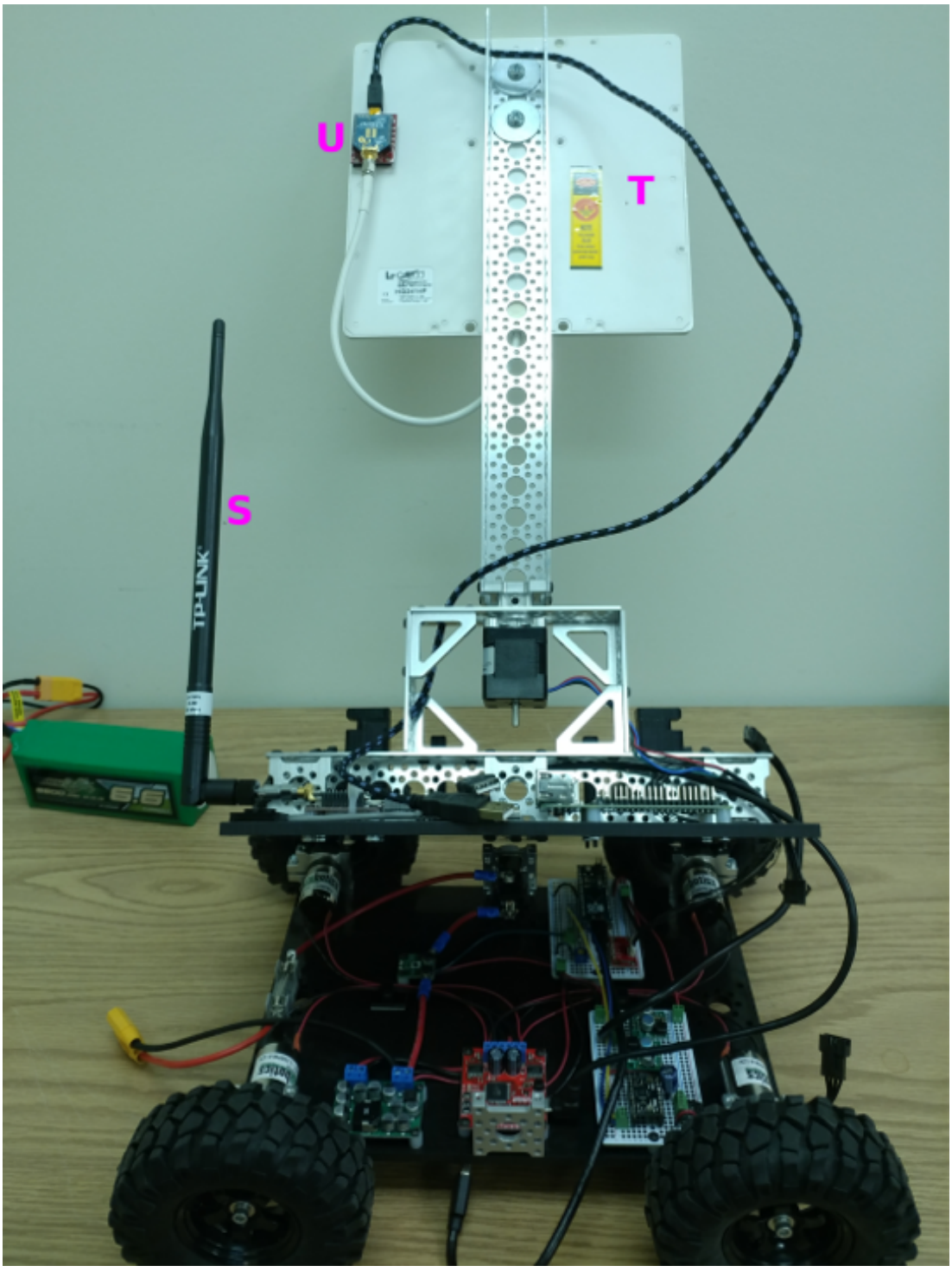

Figure 3.20. (S) Omnidirectional antenna. (T) Directional patch antenna. (U) XBee radio used primarily for direction-finding.

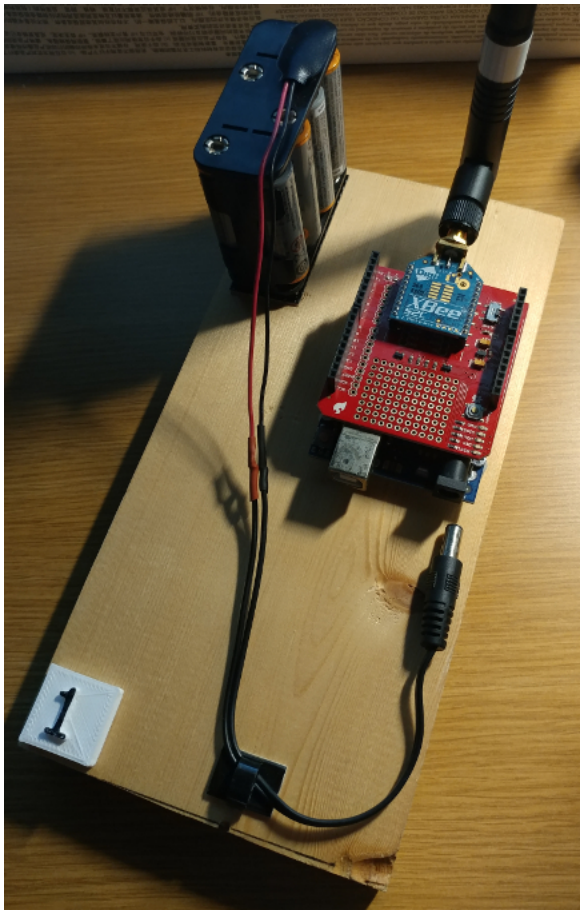

Figure 3.21. The beacons consist of a battery pack, an Arduino Uno, an XBee shield, and the XBee radio.

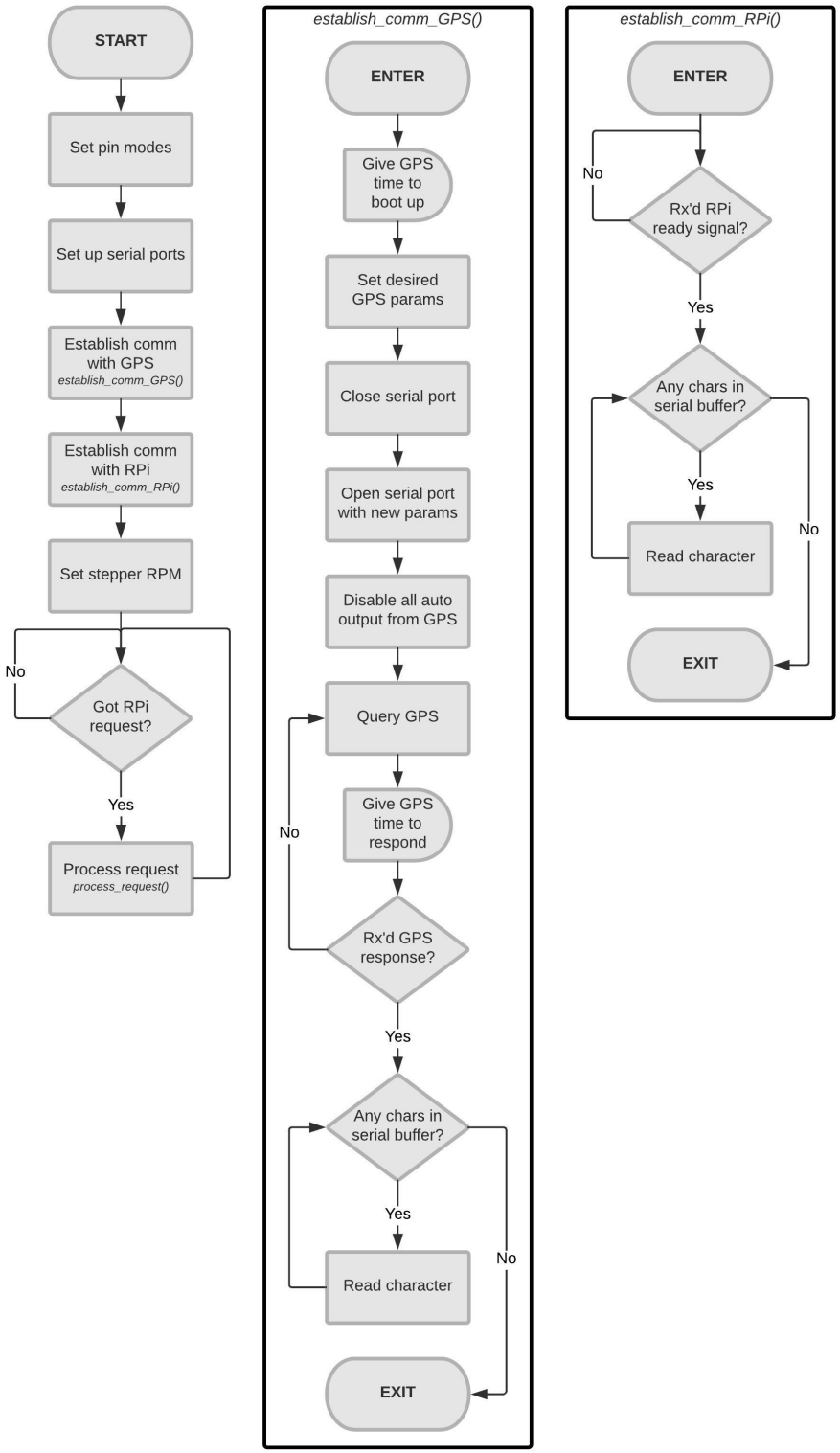

Figure 3.22. First part of the flowchart of the firmware for the robot's Arduino Micro.

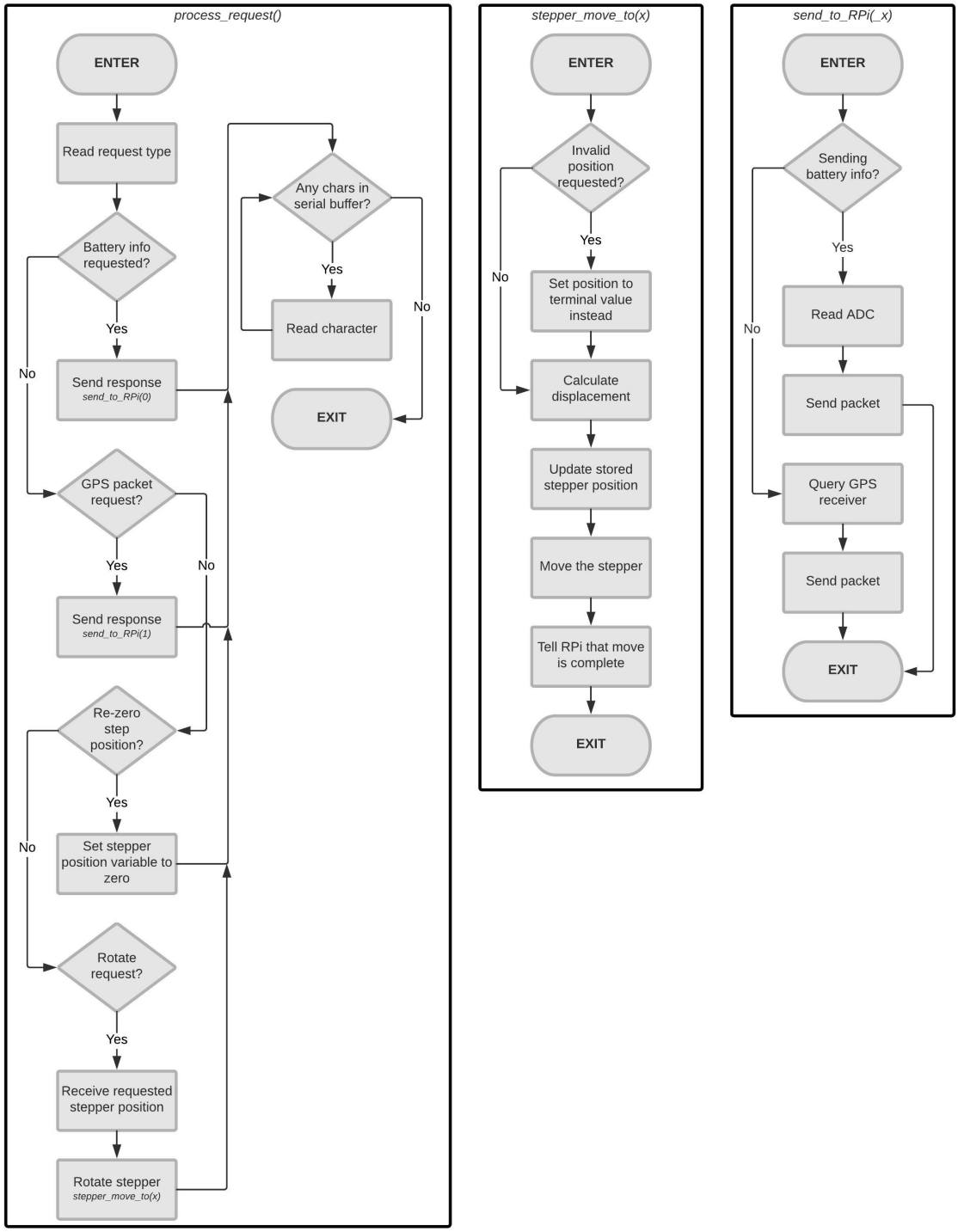

Figure 3.23. Second part of the flowchart of the firmware for the robot's Arduino Micro.

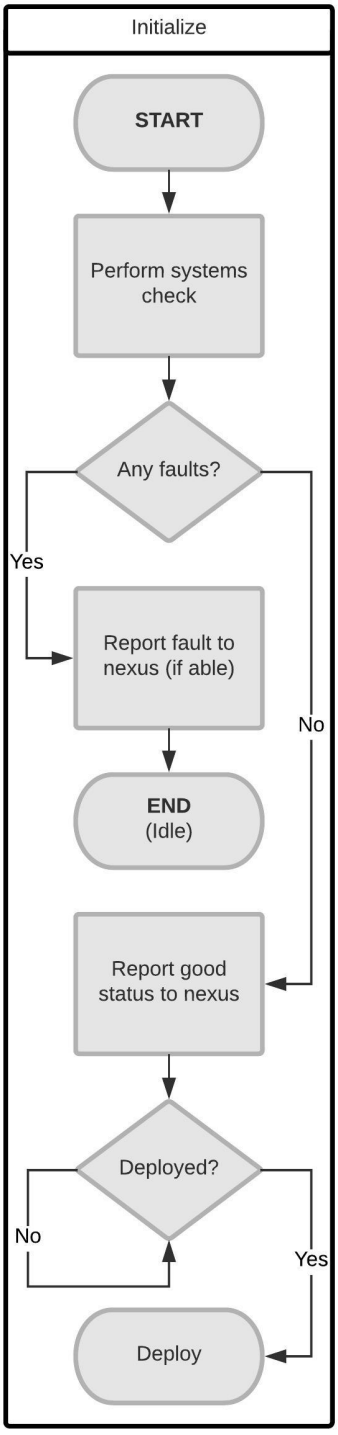

Figure 3.24. Flowchart of the software on each robot's Raspberry Pi.

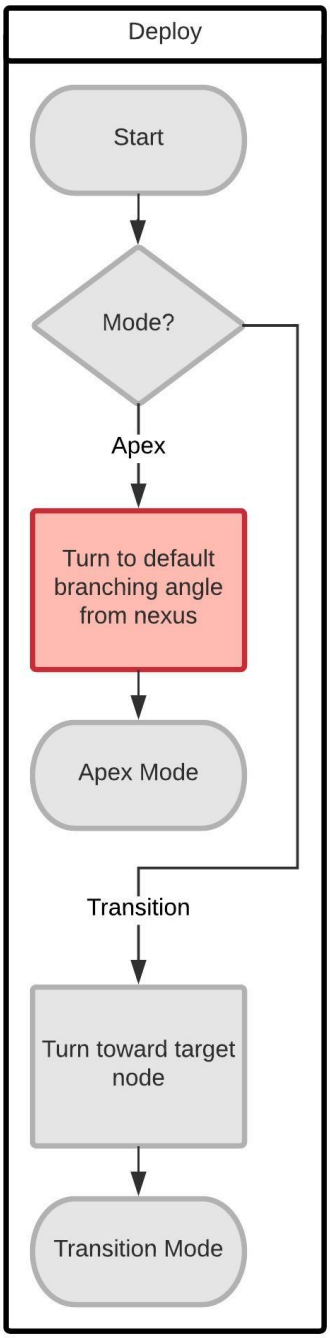

Figure 3.25. Flowchart of the software on each robot's Raspberry Pi. Red indicates something that was in the initial design but was removed.

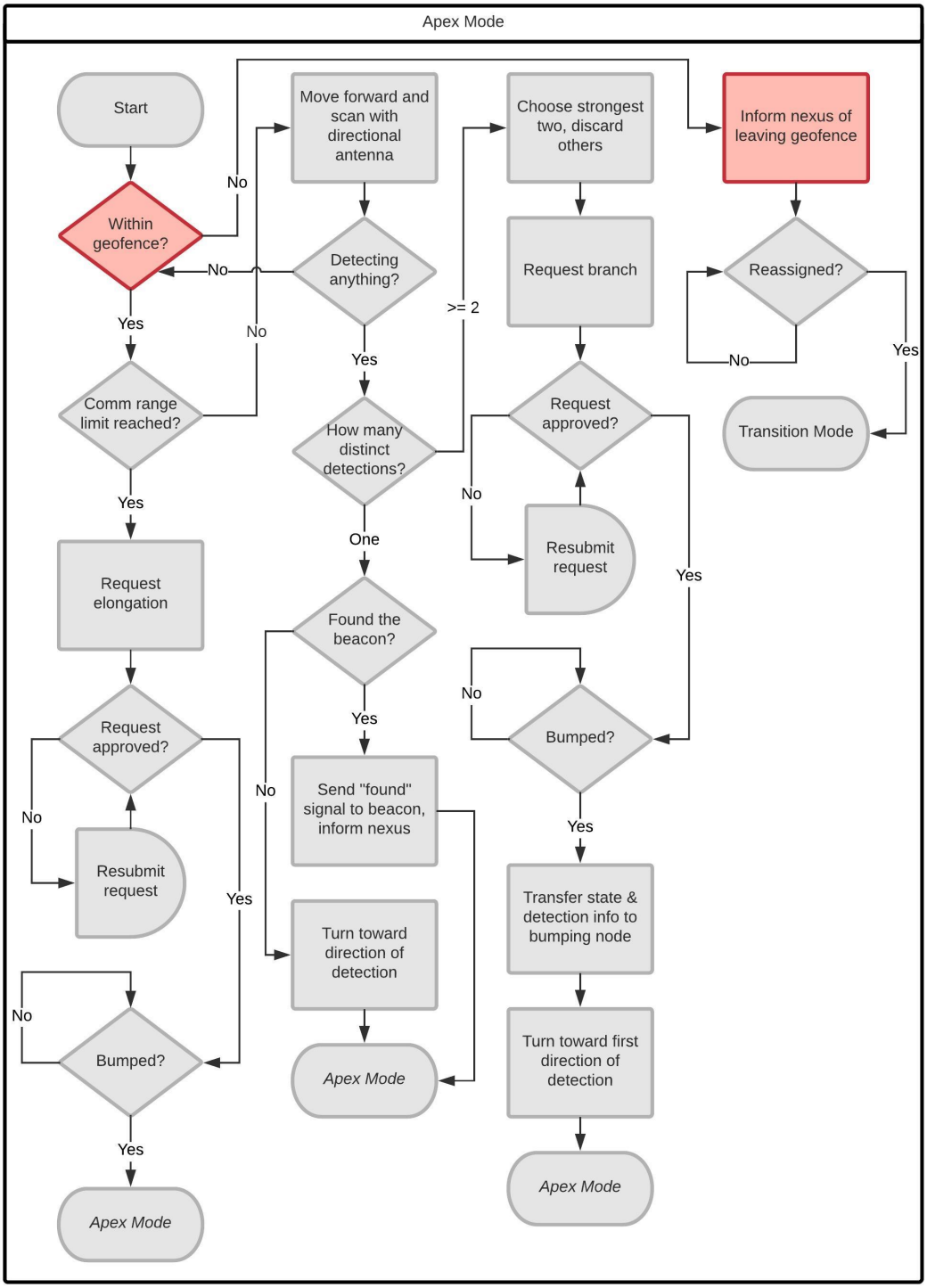

Figure 3.26. Flowchart of the software on each robot's Raspberry Pi. Red indicates something that was in the initial design but was removed.

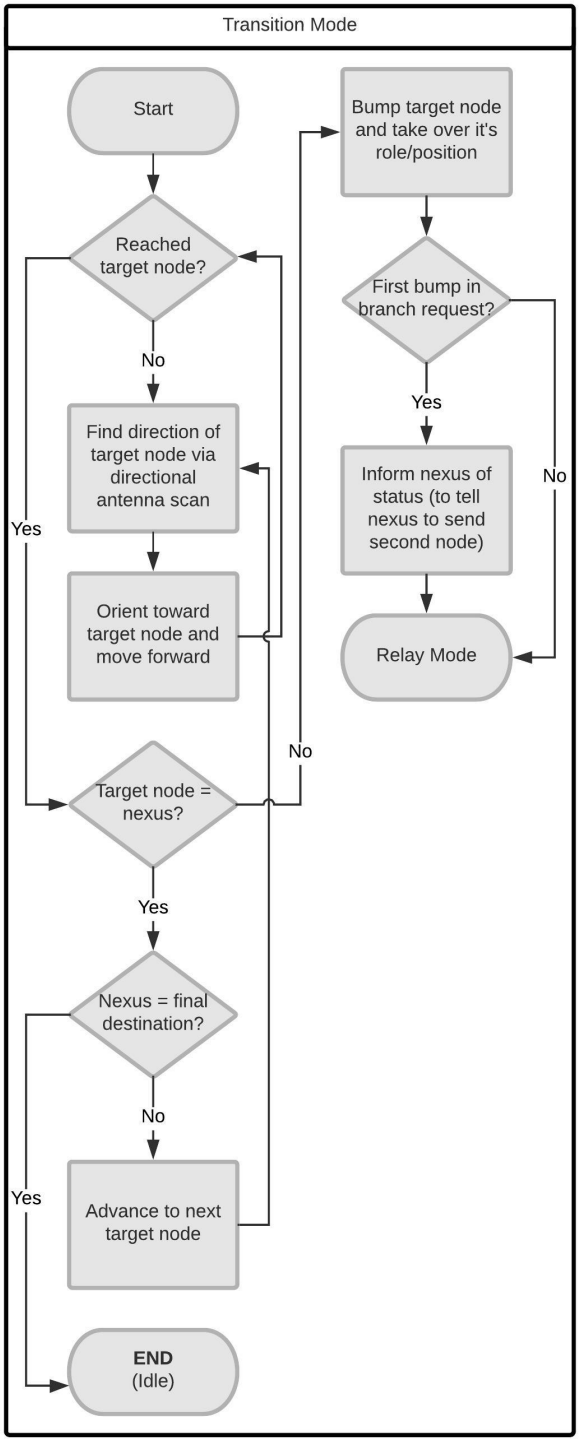

Figure 3.27. Flowchart of the software on each robot's Raspberry Pi.

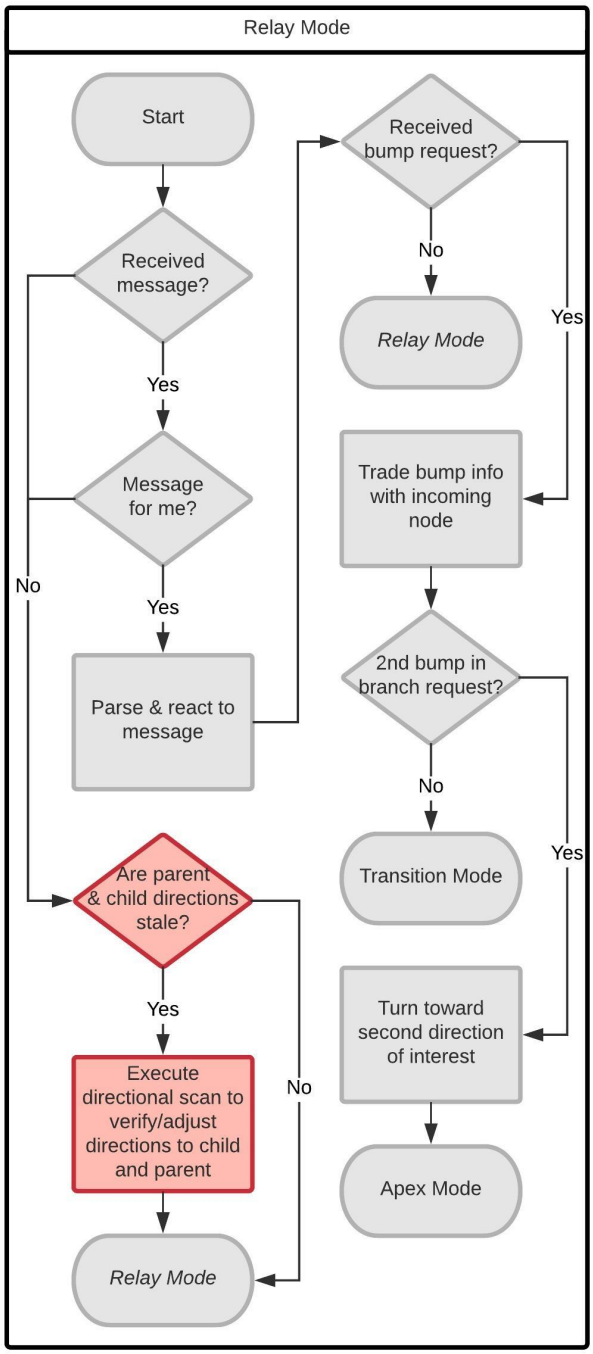

Figure 3.28. Flowchart of the software on each robot's Raspberry Pi. Red indicates something that was in the initial design but was removed.

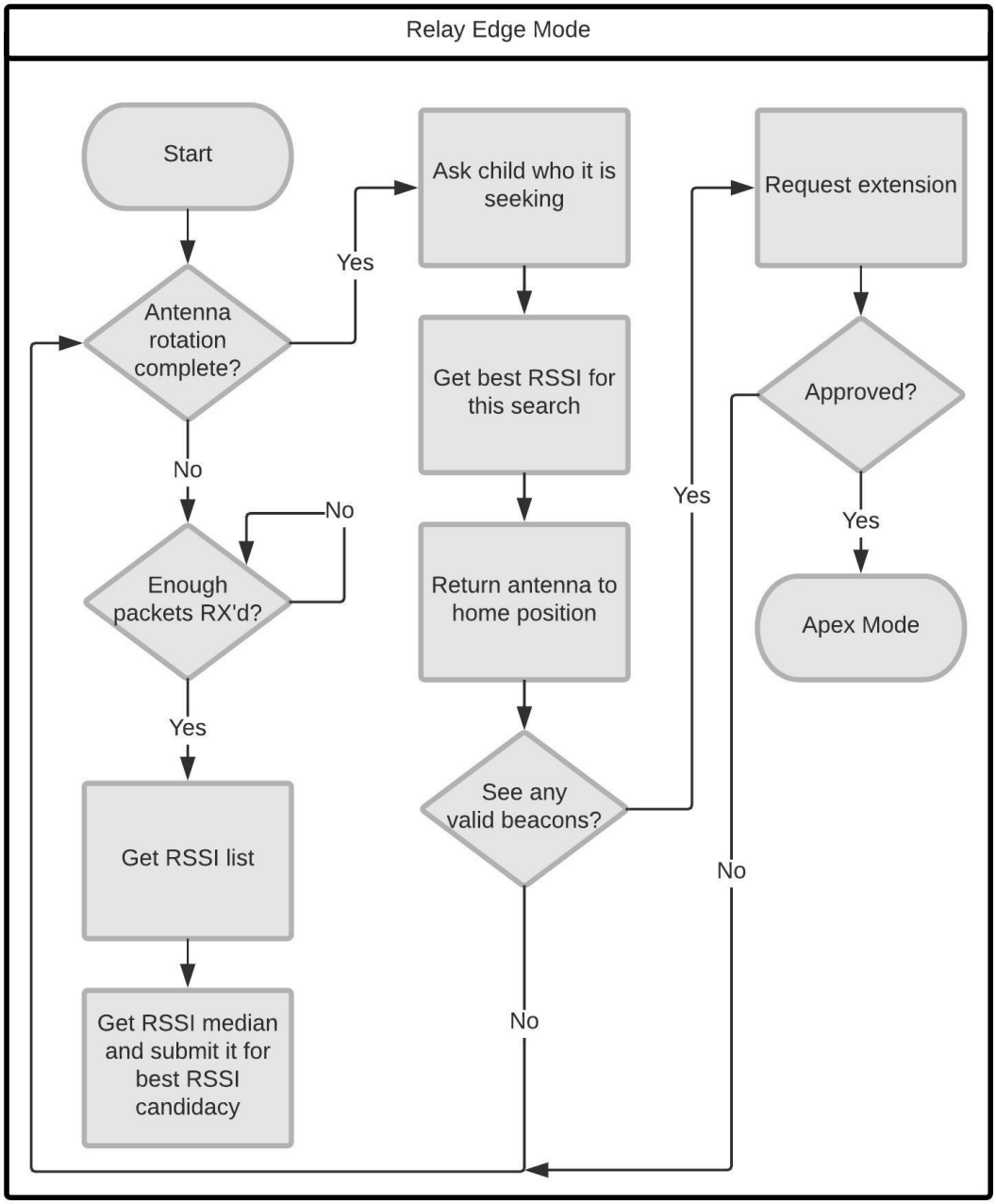

Figure 3.29. Flowchart of the software on each robot's Raspberry Pi.

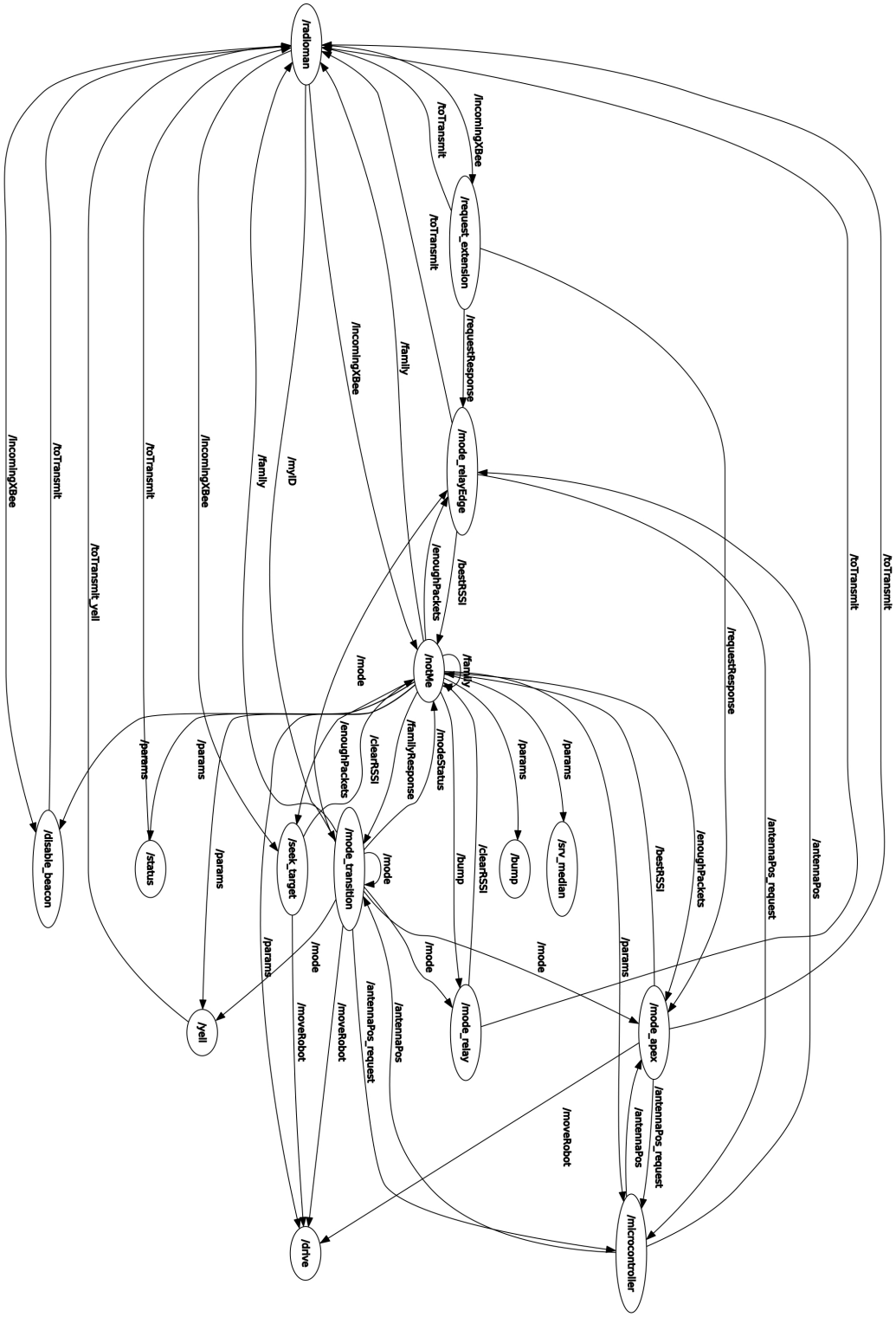

Figure 3.30. Graph of ROS architecture on each robot's Raspberry Pi.

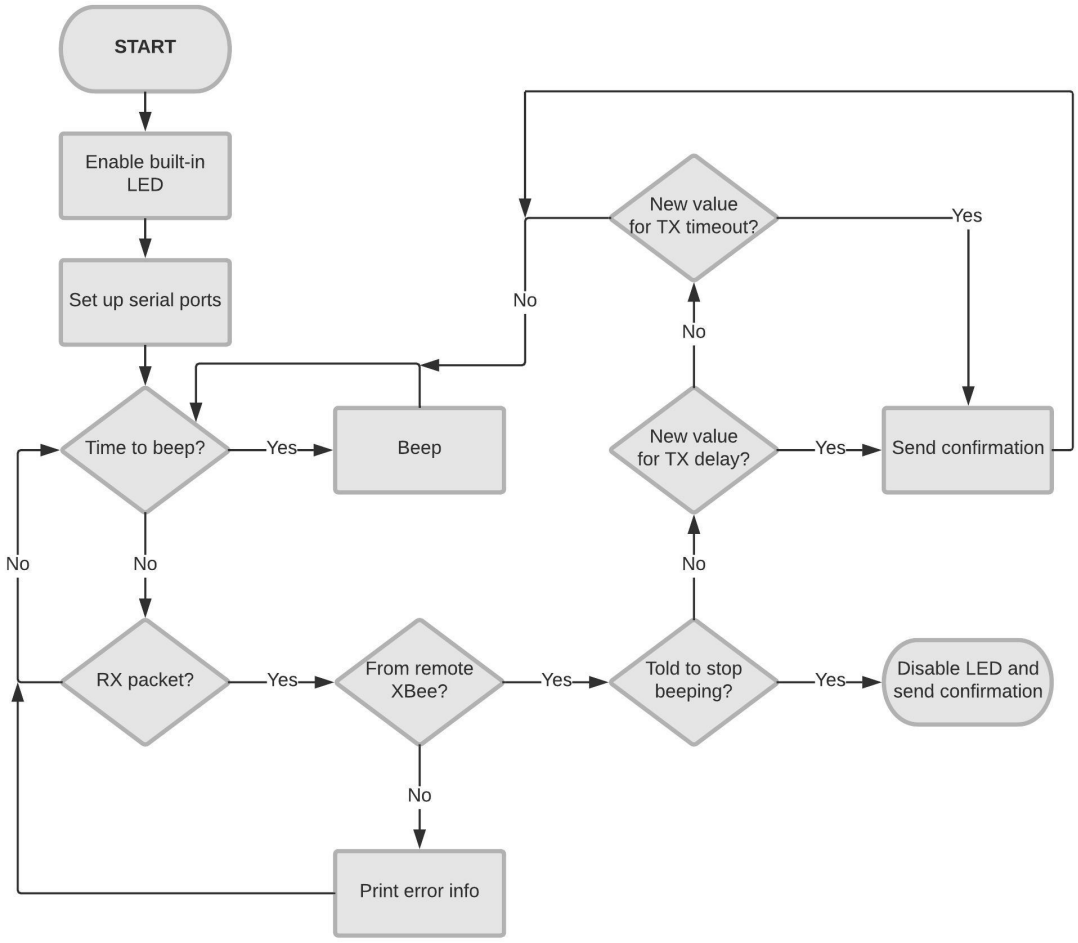

Figure 3.31. Flowchart of the beacon's firmware.

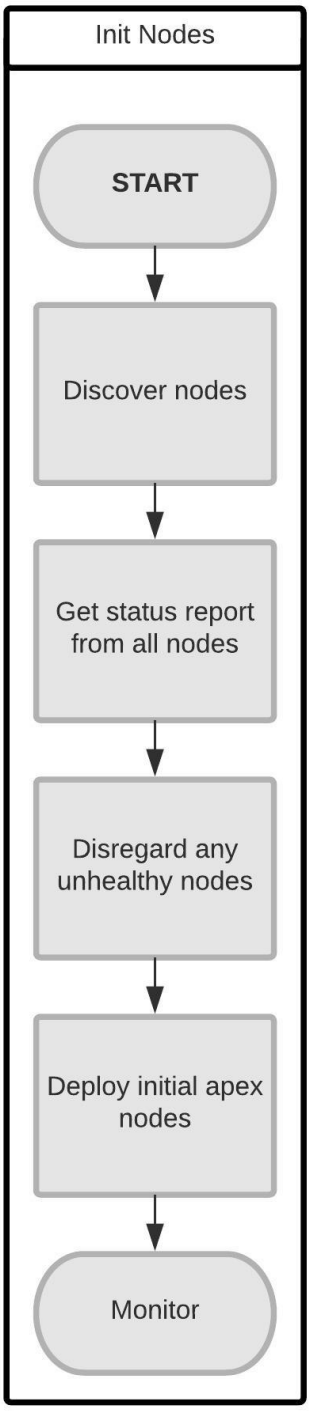

Figure 3.32. Flowchart of the Nexus's process for initializing nodes (robots).

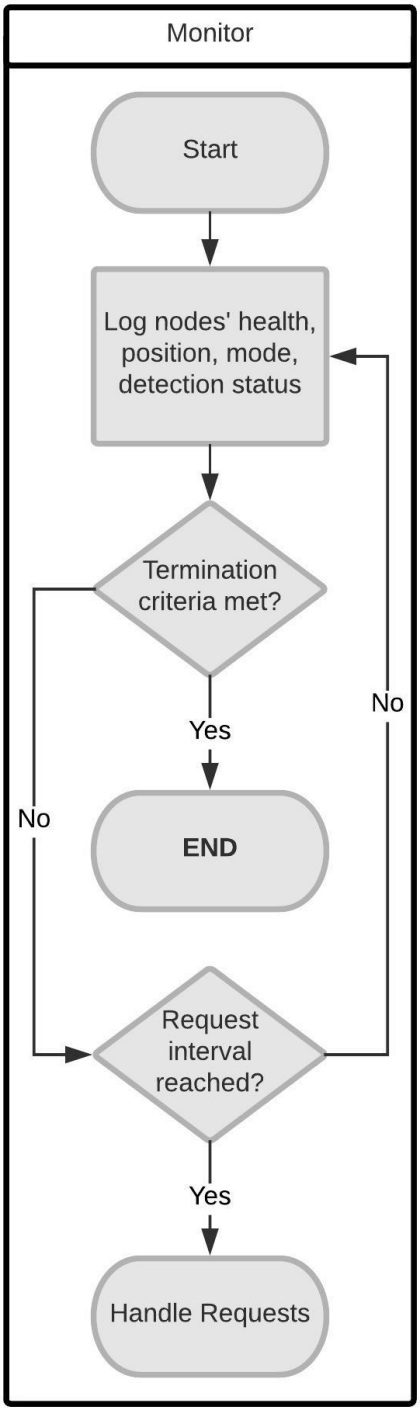

Figure 3.33. Flowchart of the Nexus's process for monitoring nodes (robots).

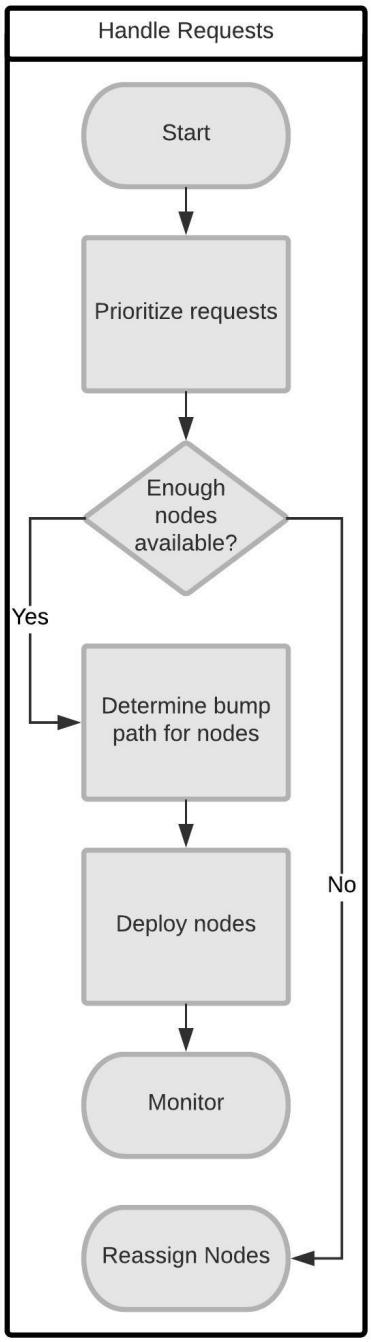

Figure 3.34. Flowchart of the Nexus's process for handling extension requests from nodes (robots).

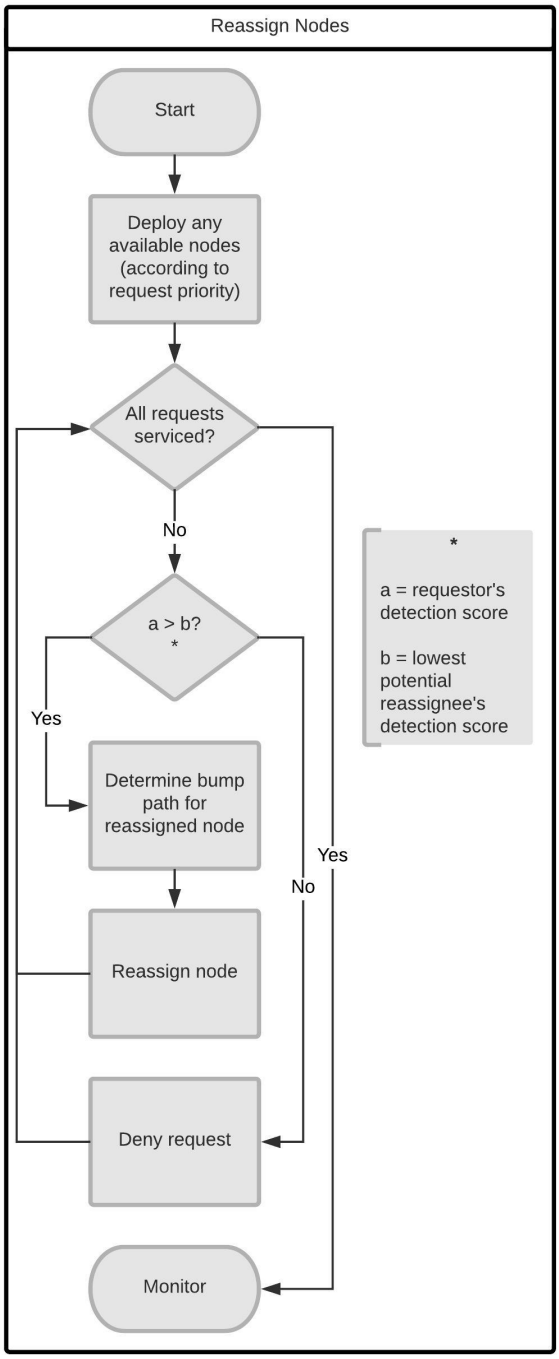

Figure 3.35. Flowchart of the Nexus's process for reassigning deployed nodes (robots) to accommodate new requests.

CHAPTER 4. (ADVENTURES IN) METHODOLOGY AND TESTING

Adapting the nutrient-finding strategies of plants to a robotic system requires many carefully-assembled pieces, such as developing the robots themselves, emulating the communication that physically-connected roots have, emulating the decisions and behavior of root apices, addressing the lack of inherent stigmergy, and dealing with a scarcity of root-extending resources (i.e., robots) that plants do not share to the same degree. This study is inspired by a 2011 report released by the European Space Agency (ESA) (Simes et al., 2011) that studied the nutrient-finding methods of plants and how those methods might be applied to a robotic system.

4.1 Similarities to and differences from plant roots

Simply stated, the goal of the robotic system is to locate "nutrients" (represented by radio beacons) in an unmapped environment. The robots are placed in the unmapped environment and spread out to explore the environment in search of nutrients. As discussed in the previous chapters, plant roots provide inspiration for this study, due to their success in a very limiting environment (soil). The primary features being borrowed from plant roots are:

- Mostly decentralized processing and decision-making
- Exploratory actuation consisting only of elongation and branching
- Limited environmental sensing
- Following nutrient gradients
- Indirect communication with other parts of the root system via stigmergy (i.e. radio "nutrient" beacons being turned off once found)

However, a full and direct transfer of plant roots into a robotic system is not yet feasible, so some adaptation and substitution is required:

- Rather than moving through the ground, the robots move on top of it.
- Root apices are physically connected to the rest of the plant system. Since pulling tethers around is not very practical, the robots use wireless communication to attempt to emulate the physical connection.
- Plant roots follow chemical gradients in the soil, which would be difficult to control for testing purposes. Instead of chemicals, radio beacons are used to represent nutrients, due to the ease of detection and inherent gradient they provide.
- When a root system creates a new branch that does not find nutrients, the resources used to create that branch are not directly reclaimed. In the robotic system, unsuccessful branches are eventually reclaimed and reassigned, due to the severely constrained resources for growth, compared to a plant root.

4.2 Operational Overview

I will use an example to illustrate the intended operation of the system. When the nexus is activated, nodes 1 and 2 will be in apex mode and will start moving away from the nexus (Figure 4.1(a)). Apex mode (represented by triangles in the figures) means the node is searching for nutrients and is capable of making elongation and branching decisions. Elongation occurs when an apex node wants to continue moving in one direction, but has reached the end of its wireless range. Branching occurs when an apex node finds two directions of interest. There are sure to be scenarios in which an apex node finds more than two directions of interest, but for the sake of this study, branching will be limited to two children.

When node 1 reaches the end of its wireless range, it communicates an elongation request to the nexus. The nexus then releases node 3, which tracks

toward node 1 via the onboard directional antenna. When node 3 reaches node 1, node 3 "bumps" node 1 and replaces it, which results in node 1 moving away from node 3 in the elongation direction (Figure 4.1(b)). This action is analogous to a plant root growing. Node 3 is now in relay edge mode. The possible modes that each node can be in are described in Section 4.2.1.

Node 2 decides to branch, so the nexus sends the first of two additional nodes toward node 2. Using the same bumping procedure as before, node 4 replaces node 2 (Figure 4.1(c)). Then, to complete the branching, node 5 replaces node 4 (Figure 4.1(d)). Finally, node 2 requests elongation, so node 6 is released from the nexus, bumping node 5, which then bumps node 2 and sends it in the direction of elongation (Figure 4.1(e)). Notice that only the outermost nodes are in apex mode, with nodes 3, 5, and 6 acting as relays.

The nexus has now deployed all the nodes, but the system does not just sit still. Node 4 requests elongation, but since there are no available nodes, the nexus determines if any of the apices are not supplying enough nutrients (i.e. not detecting a strong enough signal). In this case, node 2 is not meeting the threshold, so it gets reassigned to elongate node 4 (Figure 4.2). This process continues until one of the termination criteria is met.

4.2.1 Modes

Each node is capable of being in four modes: apex, relay edge, relay, or transition. Apex mode is active when a node is the outermost node in its branch, or put another way, when the node does not have any children. In Figure 4.1(e), nodes 1, 2, and 4 are in apex mode. Being an apex means being an explorer, so an apex node's directional antenna is actively scanning for nutrients (recall that nutrients are represented by radio beacons in this study). Being an apex also means making decisions about when and where to elongate and branch.

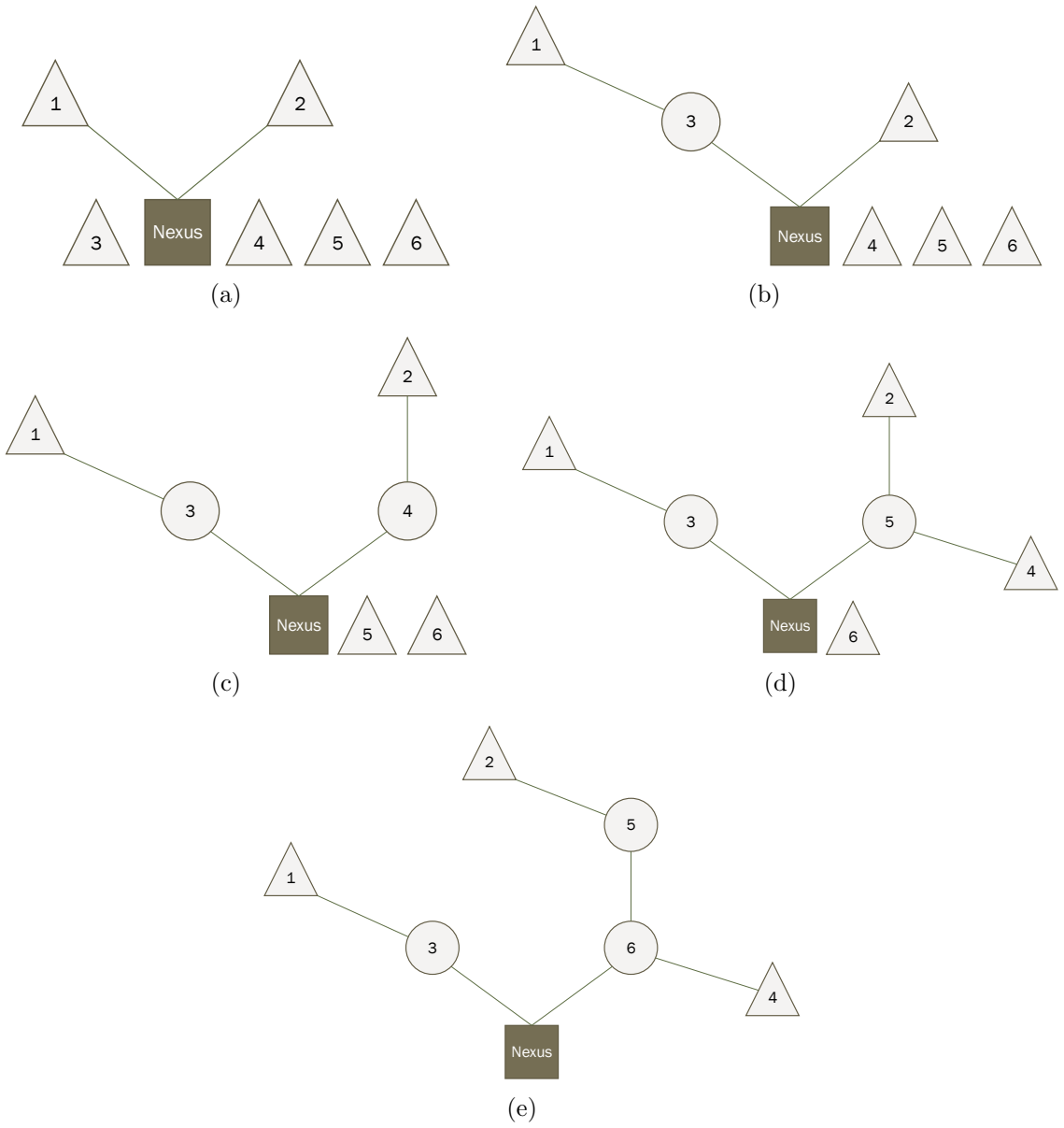

Figure 4.1. Example deployment scenario.

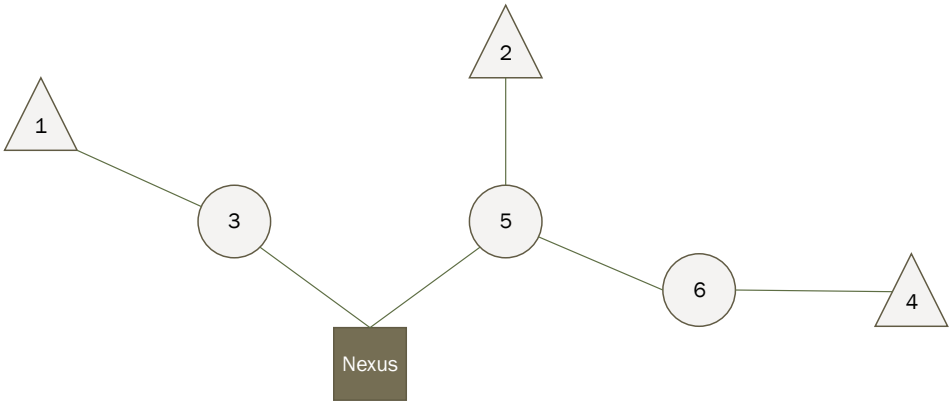

Figure 4.2. Node reassignment in example deployment scenario.

```
initialization;
while signal strength is below bump threshold do
    scan for direction of target node;
    if direction is not current heading then
        turn to face direction of target node;
    end
    move forward;
    measure signal strength;
end
stop moving;
exit Transition mode;
adopt role of target node;
```

Algorithm 1: Bumping

Relay mode accomplishes what its name suggests. When a node is in relay mode, it acts as a wireless relay between its parent and child/children, enabling the growth of the system. In Figure 4.1(e), nodes 3, 5, and 6 are in relay mode. The nexus is always in relay mode, albeit an enhanced relay (known as the ZigBee Coordinator).

Relay edge mode is similar to relay mode, but is active when a relay node only has one child. In this mode, the node will scan for any signals of interest (except the one being sought by its child). If any are located, the node will request an extension in an attempt to track this signal. The node will not, however, move until it is bumped by another node, because it is acting as a relay for its child. Moving could affect its connection to its child and/or parent.

Transition mode is active when a node is moving toward another node that it is going to bump. A node in transition mode is using its directional antenna to track toward the node to be bumped, stopping when the received signal strength is high enough to indicate a "collision." At this point, the transitioning node becomes a relay and the bumped node enters into transition or apex mode.

4.2.2 Bumping

As implied in the previous sections, bumping is the term I am using to indicate a node "colliding" with another node. Collision, in this context, does not mean a physical collision, but rather a received wireless signal strength that is sufficient to consider the two nodes to be collocated. The bumping threshold will be fairly imprecise, due to the noisy nature of these types of wireless signals, but will have to suffice in the absence of any other proximity sensors.

The alternative to bumping is for nodes to go around each other, but reliably navigating around another node with only wireless signal strength as a guide would be very difficult. I could put additional sensors on the nodes to allow for more reliable navigation around each other, but more sensors means more cost and more complexity, both of which I am attempting to minimize.

Result: Newly-arriving node replaces existing node.
initialization;
while *signal strength is below bump threshold* **do**
    scan for direction of target node;
    if *direction is not current heading* **then**
        turn to face direction of target node;
    end
    move forward;
    measure signal strength;
end
stop moving;
exit Transition mode;
adopt role of target node;

Algorithm 2: Bumping

4.2.3 Elongation

A node makes the decision to elongate (i.e. request an extension) when it reaches the end of its wireless range from its parent. For an apex node, this may or may not involve detection of a signal of interest. For a node in relay edge mode, elongation is only requested if a signal of interest (other than the signal being sought by its child) is detected. This situation could also be called branching.

Result: Elongation of a branch.
initialization;
if *wireless range reached (or moving prohibited)* **then**
    request elongation;
    while *request awaiting approval* **do**
        if *request timeout reached* **then**
            repeat request;
        end
    end
    wait for bump;
    transfer role to newly-arrived node;
    resume tracking toward direction of interest;
end

Algorithm 3: Elongation

4.2.4 Signal Strength as Nutrient Flow

In the ESA report that inspired this study (Simes et al., 2011), the authors suggest treating data flow from the nodes as analogous to nutrient flow in a plant root. The purpose of this would be to implicitly determine the amount of scientifically interesting data being produced by each apex, thus enabling a more direct transfer of the plant analogy to the robotic system. In my robotic system, this nutrient flow would be represented by the signal strength of the radio

"nutrient" beacons being detected by apex nodes and reported to the nexus. If the nodes were detecting a larger variety of nutrients, adding this layer of abstraction may prove useful in the design and interpretation of the system. However, since the nodes in this study are only detecting one nutrient, I will leave the application of this analogy to the reader.

4.2.5 Simultaneous Requests

The nexus is not directly controlling the actions of the nodes, so it is possible that the nexus will receive nearly simultaneous requests for elongation. Ideally, the nexus would have perfect information about all requests that will be made during the course of a test. Since that is not the case, the nexus services requests at a regular interval, rather than immediately upon receipt. This allows the nexus to do limited optimization of the requests it has at any given time.

4.3 Test Locations

Finding a location to test this system proved to be a bit of a challenge. Even at small scale, there are several characteristics that a location must have. When combined, those characteristics eliminate most locations. The primary considerations were unrestricted (mostly, at least) access, no foot or vehicle traffic, as large as reasonably possible, not subject to excessive radio interference/reflection, flat, and free of obstacles. That effectively leaves remote, unused parking lots as the only viable option, but those are few and far between.

Throughout development of the system, I did not have regular access to such a location, so I had to bounce around to different locations, most of which were pretty far from ideal. These locations included the top level of a parking garage (Figure 4.3), an unused planting bed (during the winter) at a greenhouse (Figure 4.4), an empty church sanctuary (Figure 4.5), the gravel parking area of a rugby field (Figure 4.6), and several school parking lots after hours (Figures 4.7, 4.8, and 4.9).

Figure 4.3. Testing location: the top level of a parking garage.

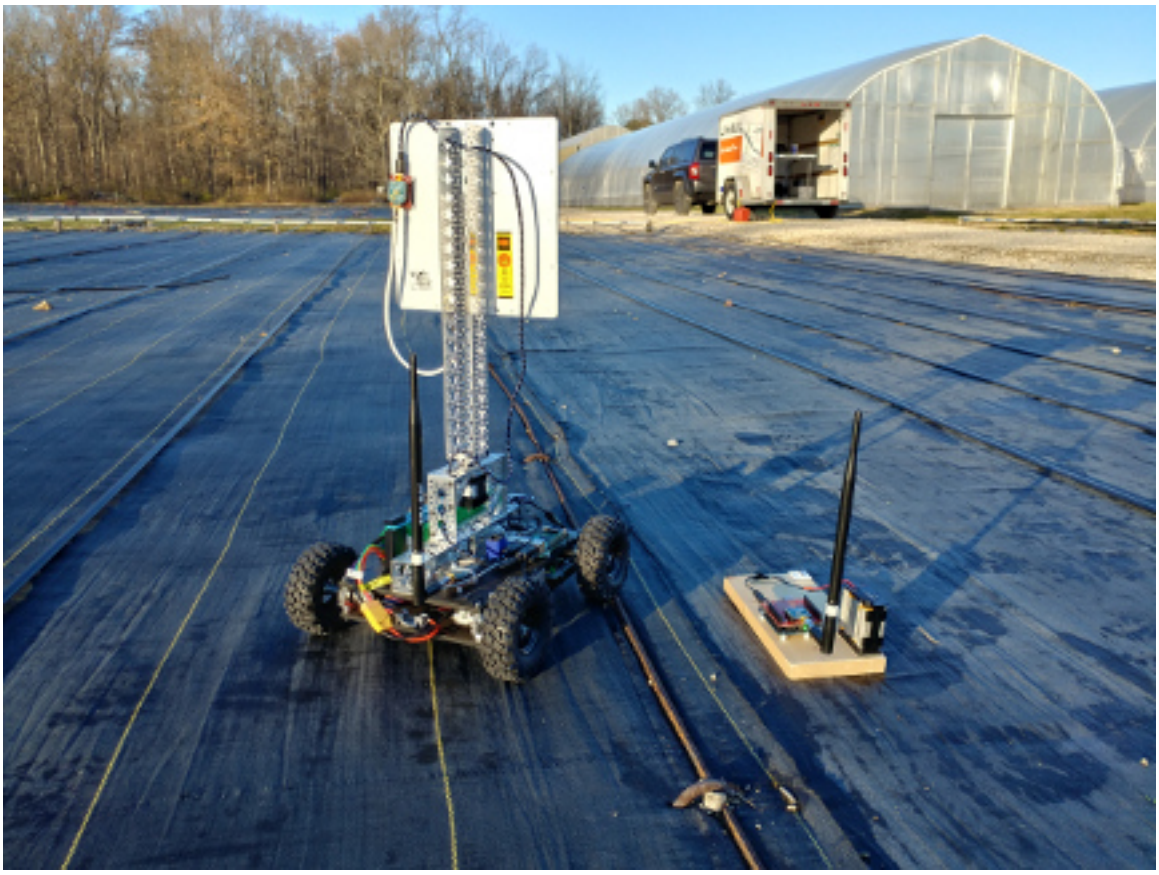

Figure 4.4. Testing location: unused planting bed of a greenhouse during the winter.

Figure 4.5. Testing location: an empty church sanctuary.

Figure 4.6. Testing location: the parking area of a rugby field.

Figure 4.7. Testing location: a school parking lot after hours.

Figure 4.8. Testing location: a school parking lot after hours.

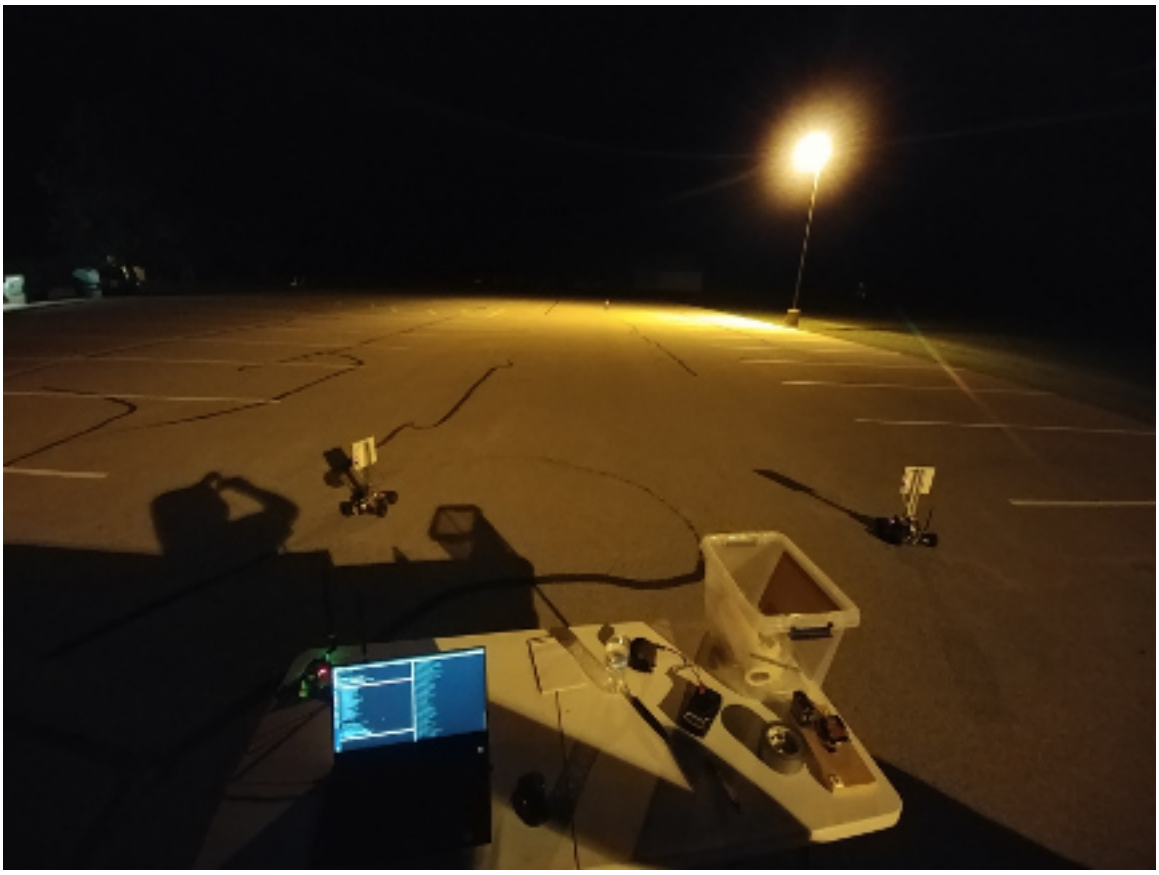

Figure 4.9. Testing location: a school parking lot after hours.

4.4 Variables

The independent variables for this study are:

- Number of nodes that deploy at initialization
- Time to wait for initial deployment to complete
- Number of nodes participating
- Time to wait for confirmation of parameters being received by nodes
- Signal strength required of node's best detection to avoid reassignment
- Interval for servicing new requests
- Number of beacons available
- Maximum duration of trial
- RPM of stepper motor for the directional antenna
- Duration of turn interval while seeking a specific signal
- Interval for sending broadcast "beep" messages
- Timeout for "clear to send" for the XBees
- Duration of driving forward while searching
- Speed of backward wheel spin
- Speed of forward wheel spin
- Ratio of right/left wheel spin speeds during a turn
- Ignore GPS status (for in-lab testing)
- Voltage of batteries to be considered depleted

- Maximum number of times the robot will rotate when searching for a signal
- Difference in RSSI values that is required to consider a value an outlier
- Number of packets required for a search step to complete
- Timeout for waiting to receive required number of packets
- Number of stepper motor steps per search step
- Signal strength required to bump
- Signal strength required to consider a detection to be valid
- Signal strength required to disable a beacon
- Sensitivity gain for matching RSSI values
- Signal strength at which the robot is at maximum range from its parent
- Interval for sending status messages to the Nexus

The dependent variables for this study are:

- Ratio of located nutrients to available nutrients
- Energy required to locate particular ratios of available nutrients

4.5 Measure of Success

The only major, meaningful measure of success is showing that a low-cost, plant-inspired, robotic exploration system is feasible.

4.6 Simulation, or lack thereof

Rather than following directly in the footsteps of the authors of (Simes et al., 2011), I chose not to use neural networks, at least for this first phase of the study.

Instead, the control scheme is more akin to a traditional state machine. I did not see how simulation of such a system would further my efforts to gauge the feasibility of the system.

4.7 An abridged version of the road so far

Projects like this one tend to be difficult and fraught with problems. This project was no exception. Getting the system to where it is now has been a long road that has somewhat diverted from the original plan.

The Python code that runs on each robot's RPi started out as a single Python script. Since a robot does not know when it will receive messages, the software must be able to cope with asynchronous input. The default method of handling such a situation is to use threading, which Python has standard libraries for. As the complexity grew, this architecture became unwieldy and increasingly difficult to predict. Long days and nights of attempting to corral this situation (such as shown in Figures 4.10 and 4.11) resulted in several complete rewrites of the robots' software, but to no avail.

Confusing and frustrating behavior from the robots continued to the point where I questioned the directionality of the directional antennas. No antenna acts like a laser, even high-quality antennas, as evidenced in Figure 4.12. I tried to shield the back and sides of the antenna with aluminum foil, even extending that foil into a box in front of the antenna (see Figures 4.13, 4.14, and 4.15), but none of this helped and it proved to be too much for the stepper motor to control.

Taking things a step further, I sought out antenna configurations that were favored for their controllable directionality. The two configurations I chose were the "cantenna" and the parabolic dish. The problem is that these types of antennas tend to be heavy and bulky, by this project's standards. The fastest and least expensive solution was to design and build my own antennas, using CAD software and consumer-grade 3D printing. Figures 4.16 and 4.17 show the CAD models for

Figure 4.10. Developing and testing the system out of a trailer.

Figure 4.11. Developing and testing the system out of a trailer.

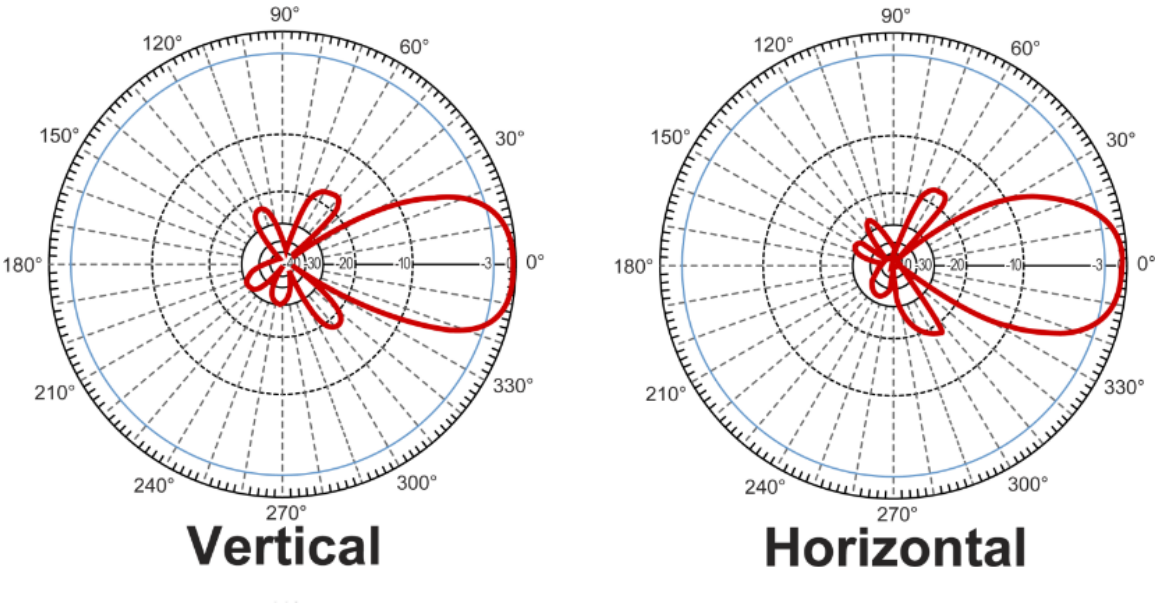

Figure 4.12. RF antenna gain pattern of L-Com HG2414P antenna, per the manufacturer's datasheet.

the frames of these antennas. Figures 4.18 and 4.19 show the antennas mounted to a robot for testing. To keep the antennas as lightweight and wind-resistant as possible, I printed the frames out of PLA plastic and glued aluminum screen (the kind used for screen doors) into them. This combination was as radio-opaque (for 2.4GHz, at least) and wind-transparent as I could quickly create. The parabolic dish antenna is only a section of a parabola, because I am not concerned with the vertical performance of the antenna. Using only a section made the antenna lighter and easier to construct.

Testing revealed that my custom-built antennas were not any better than L-Com's panel antenna. In fact, they were a bit less directional. Figure 4.20 shows that the difference in signal strength (i.e. the bias) at different angles to the source was the strongest for the original panel antenna.

Satisfied that the performance of the antenna was as good as could be reasonably expected, I concluded that the problematic behavior was due to

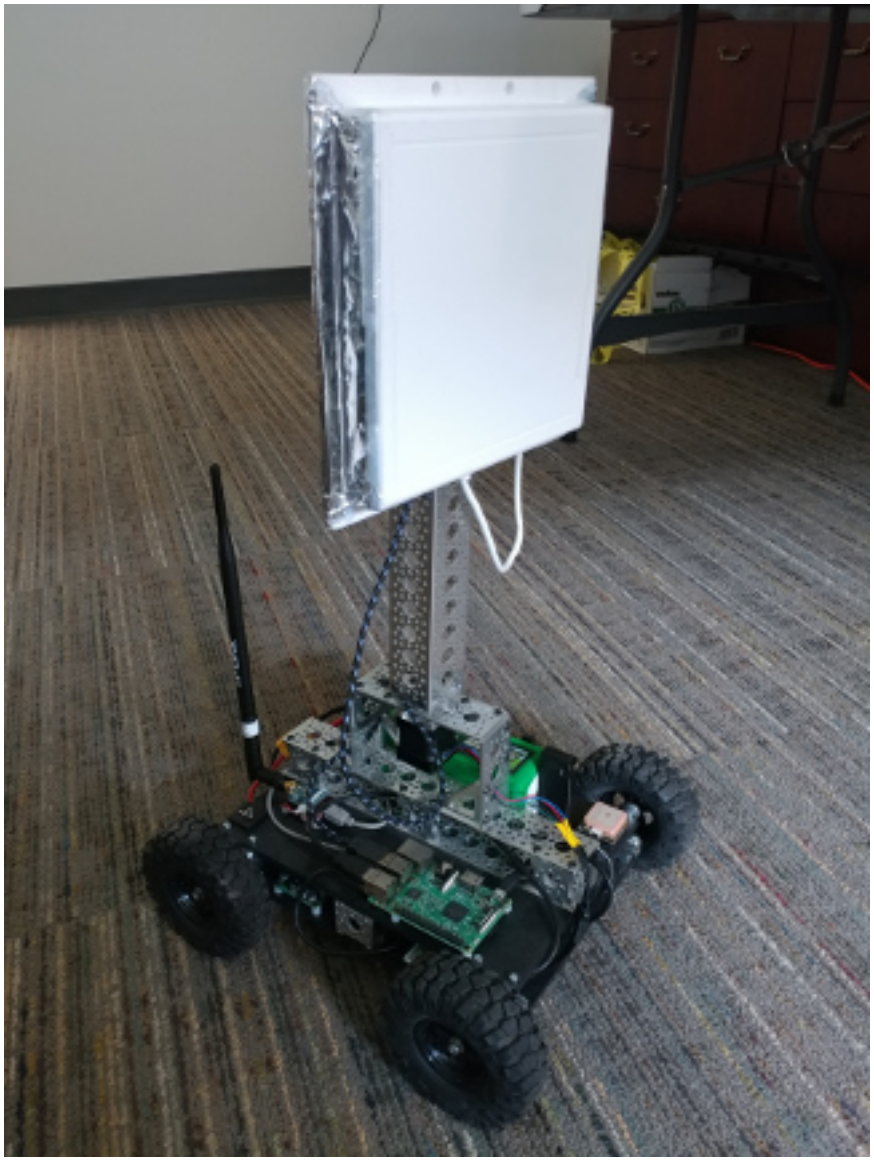

Figure 4.13. Unsuccessful attempt to attenuate unwanted RF radiation with aluminum foil.

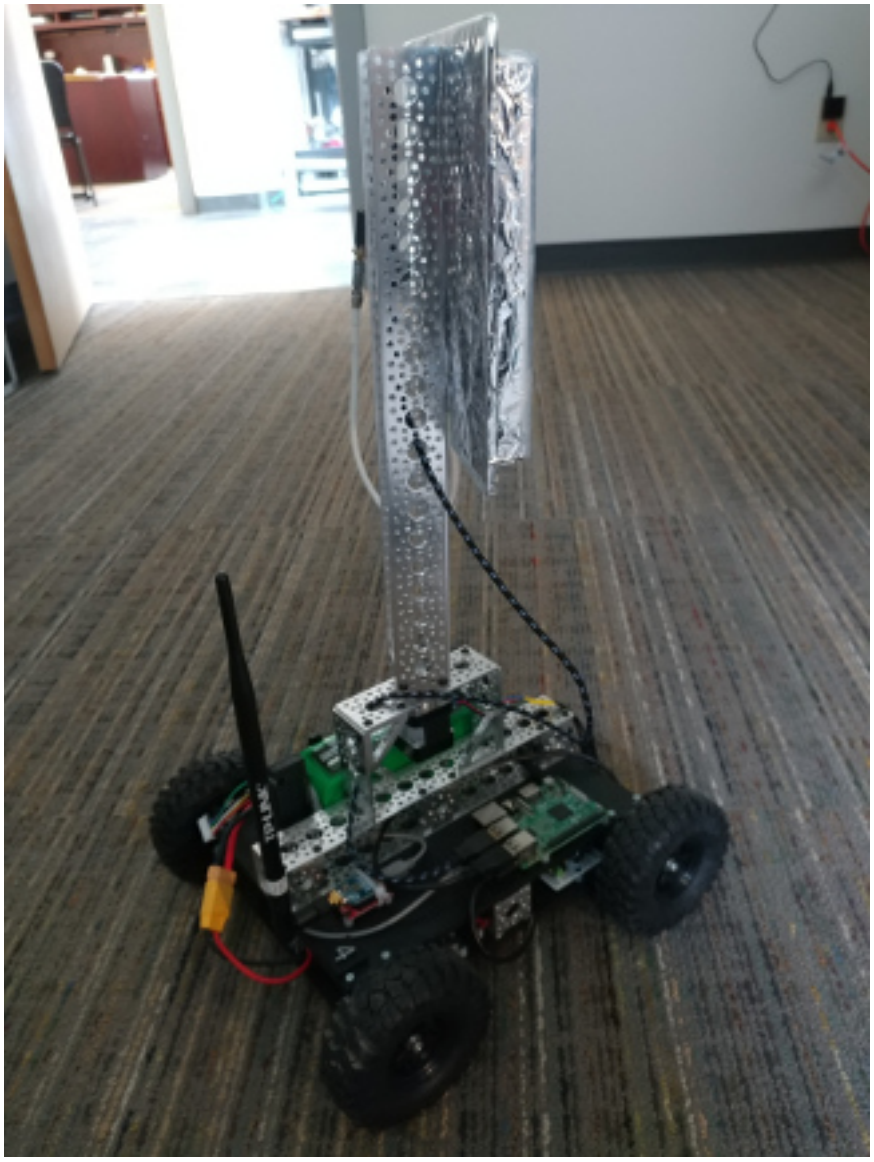

Figure 4.14. Unsuccessful attempt to attenuate unwanted RF radiation with aluminum foil.

Figure 4.15. Unsuccessful attempt to attenuate unwanted RF radiation with aluminum foil.

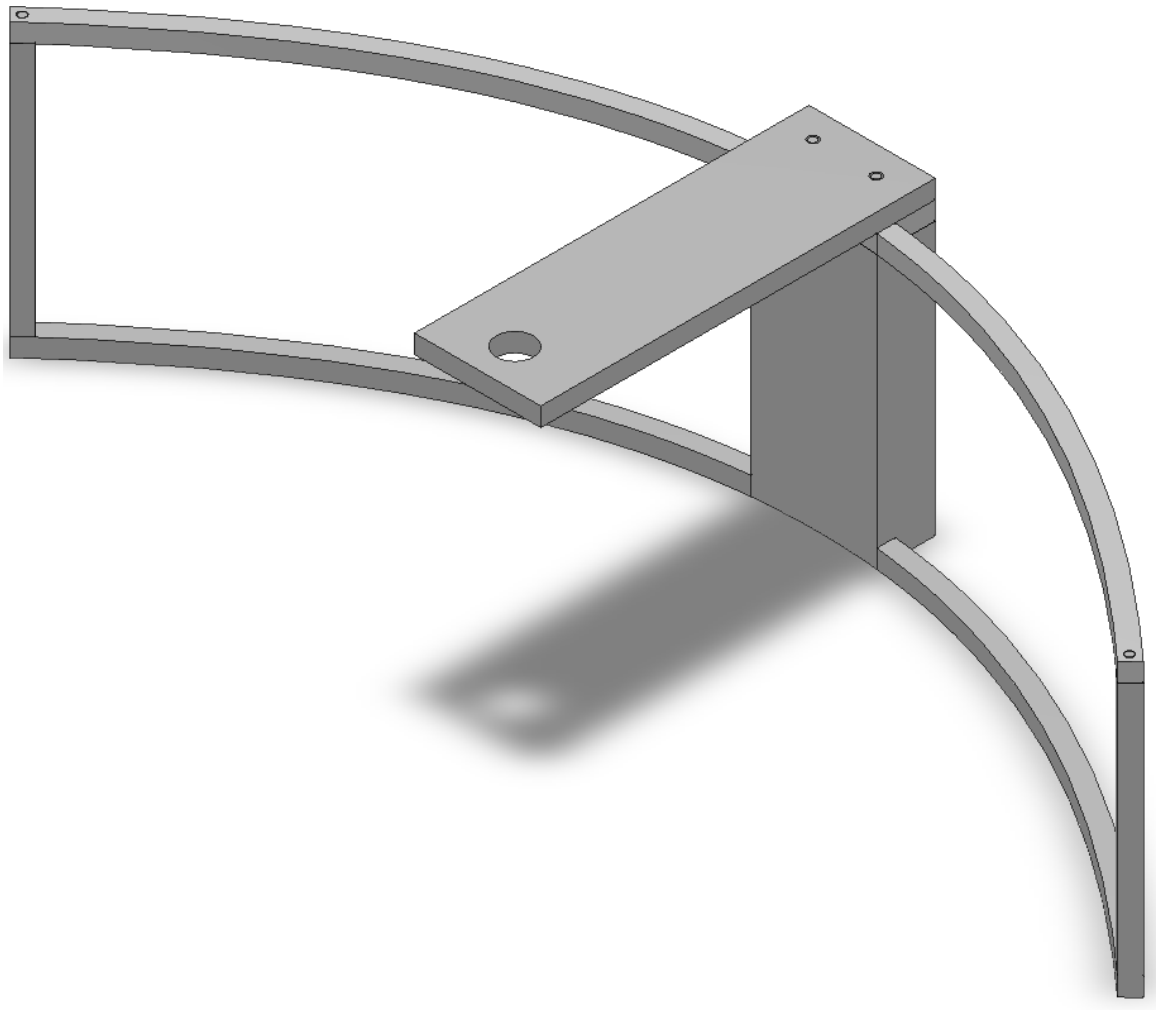

Figure 4.16. CAD model of my do-it-yourself parabolic dish section frame.

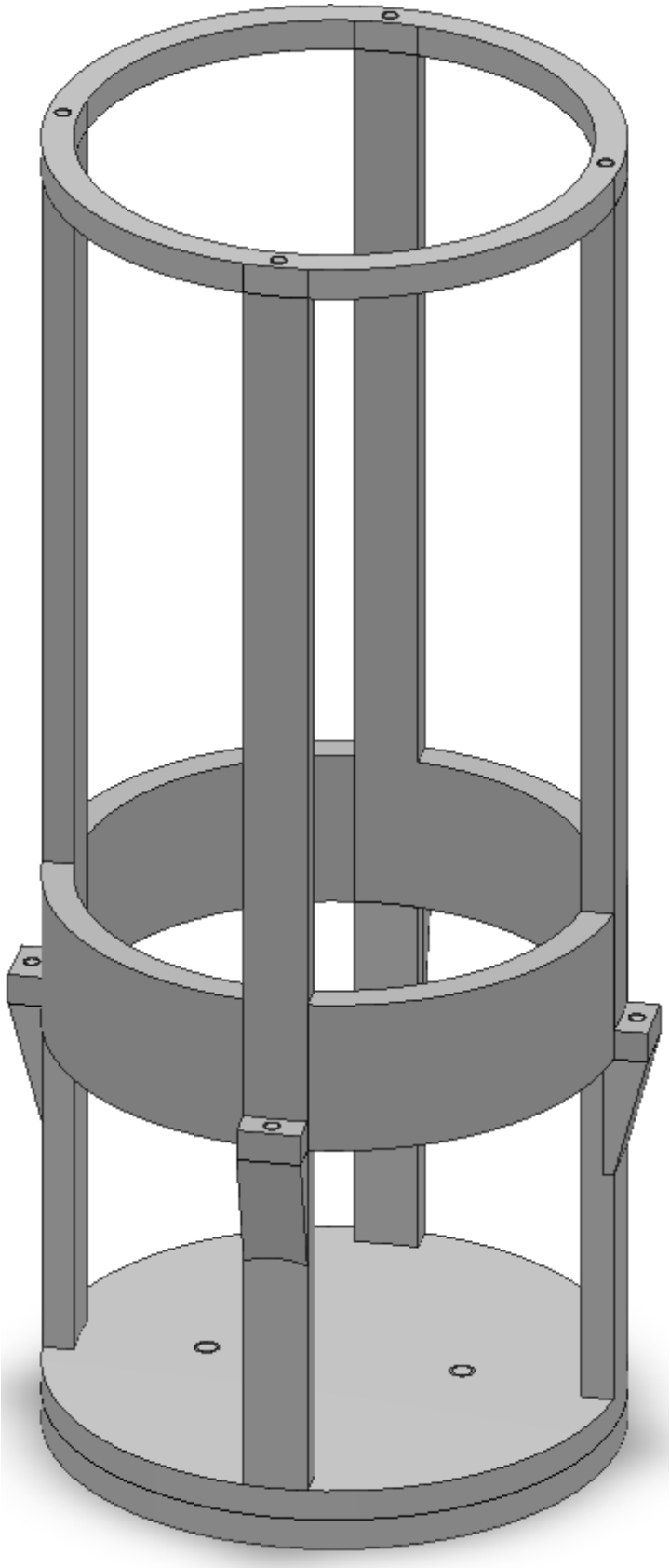

Figure 4.17. CAD model of my do-it-yourself "cantenna" frame.

Figure 4.18. Custom-made parabolic dish section antenna.

Figure 4.19. Custom-made "cantenna".

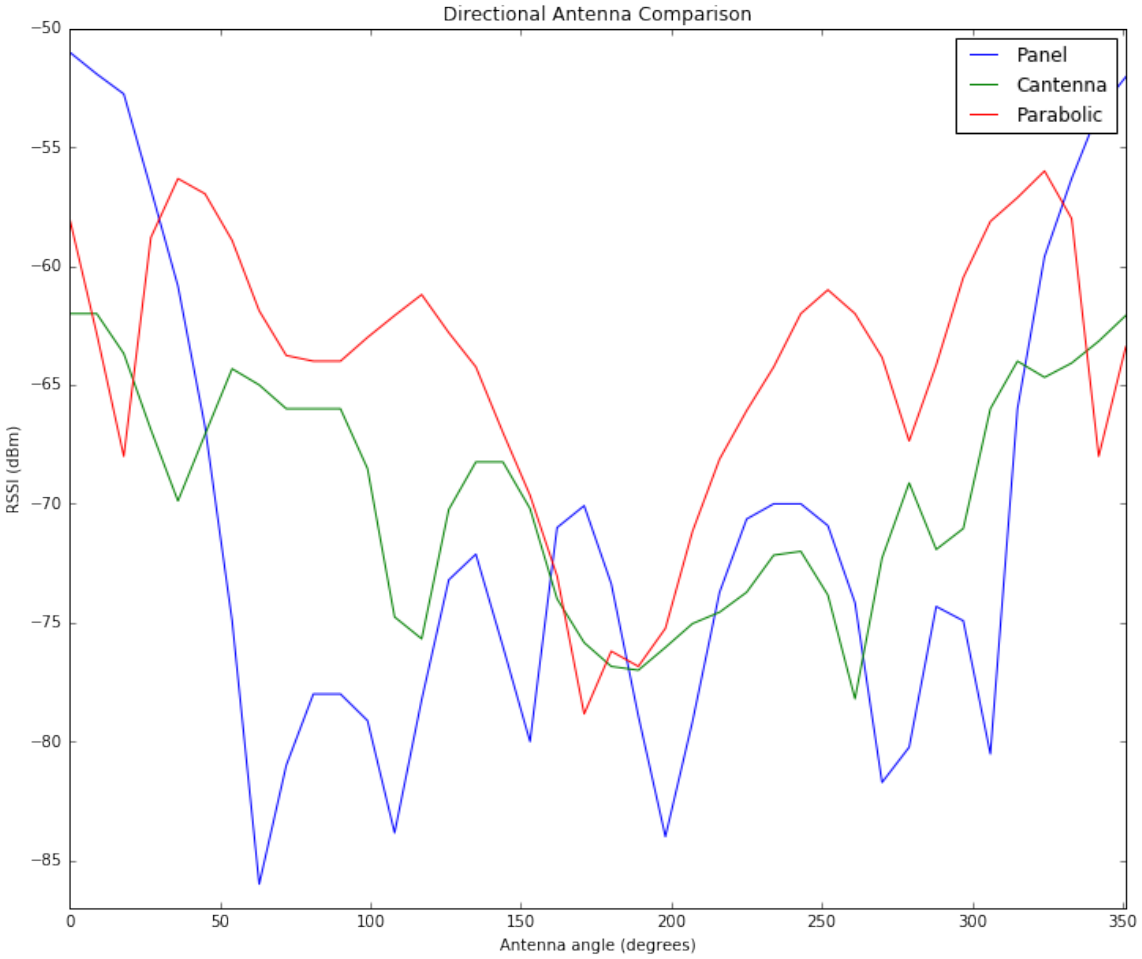

Figure 4.20. Comparison of directional bias of three types of directional antennas.

code/algorithm issues, despite the many months of effort already spent on that avenue.

I had previously chosen not to use Robot Operating System (ROS) on the robots because I did not want to figure out how to get ROS to communicate through the XBees. Revisiting this decision, though, introduced the idea of using ROS locally to manage the many threads and asynchronous input that the system requires. ROS is a mature, stable software ecosystem that is quickly becoming a standard tool for anyone working with robotics. Assuming at least some of the issues I had been dealing with were due to my multi-threaded architecture, I completely rewrote the robot's software again, this time using ROS (see Figure 3.30 for an illustration).

The move to ROS onboard the robots was a good one. The robots were behaving more correctly, more reliably, and more predictably. However, this iteration of the software had its share of bugs and problems, too. Despite working much better than previous versions, the system was still very far from perfect. Nonetheless, development and testing moved forward.

4.8 Practical odds and ends

Moving from a lab environment to the field requires some forethought and extra steps. Mostly for the benefit of anyone who happens to read this looking for some tips from experience, I'll briefly talk about the key odds and ends I needed to be effective in the field.

4.8.1 Connecting to the robots for development and debugging

The ability to write code and monitor debugging output in the field is vital for a project like this. To enable those things, I set up an inexpensive WiFi router for the robots and gave each one of them a static IP address. Along with a Virtual

Network Computing (VNC) server running on the RPi, this setup allowed me to connect to and develop directly each robot while in the field.

4.8.2 Shutting down and rebooting the robots

Properly shutting down or rebooting the RPi is important to avoid data corruption. Connecting to each robot via SSH or VNC to shut them down every time is cumbersome. Additionally, the WiFi router needed to be disabled during testing so as to not interfere with the XBees running on 2.4GHz (which it very much did). My solution was to run a small Python script in the background that monitored two of the RPi's general purpose I/O pins, one for shutting down and one for rebooting. I then connected a jumper wire to the 3.3V pin on the RPi, briefly connecting it to one of the two monitored pins to initiate the desired action.

4.8.3 Deploying code changes to all robots

In most situations, all the robots need to be running the same code, since their roles throughout the trial are not predetermined. Copying and pasting code via SSH or VNC is inconvenient and error-prone, so I needed a better solution. I set up the robot's code within a Git repository, which provides a mechanism for code deployment. However, the robots need a Git server to pull from, so when I was actively developing/debugging, I used one robot as a development machine. Once I was satisfied with a code change, I would commit it to the Git repository and use Git's "serve" tool. This temporarily runs a Git server for the onboard repository, which the other robots can pull from, thus deploying the code to all the robots quickly and reliably.

4.8.4 Running software automatically

None of these tools and techniques are worth anything if I have to connect to each robot to manually initiate them. They all need to run when the RPi boots up. To accomplish this, I wrote a Bash script that starts the VNC server, runs the shutdown/reboot script in the background, and attempts a Git "pull" to grab any new code that might be available from the development robot. Once those tasks are done, the Bash script runs the ROS launch file, which starts everything up and waits for deployment.

4.9 Testing procedure

Once at the point of attempting to collect performance data, it is very helpful to have a written checklist, since small mistakes can render entire trials invalid. I had several versions of checklists, such as those for comparing the performance of different antenna designs, testing the wireless range in different environments, etc. The most important checklist is the one used for full-scale data collection. The steps involved in such a list vary widely between different projects and situations, but my procedure was a text document that I filled in for each trial:

1. ROS bag name:
2. Check voltage of beacon batteries
3. Place beacons in the field and turn them on
4. Record polar coordinates of each beacon (from Nexus)
    - Beacon 1:
    - Beacon 2:
    - Beacon 3:
    - Beacon 4:

- Beacon 5:
- Beacon 6:
- Beacon 7:
- Beacon 8:
- Beacon 9:
- Beacon 10:

5. Clear/store any ROS bags that already exist
6. Record robot battery voltages
    - Robot 1:
    - Robot 2:
    - Robot 3:
    - Robot 4:
    - Robot 5:
    - Robot 6:
    - Robot 7:
    - Robot 8:
    - Robot 9:
    - Robot 10:
7. Set parameters in ROS launch file
8. Turn on all participating robots (wait 60 seconds for them to boot)
9. Start ROS monitoring on Nexus
10. Run ROS launch file on Nexus
11. Record robot-to-beacon distance when beacons are found

- Beacon 1:
- Beacon 2:
- Beacon 3:
- Beacon 4:
- Beacon 5:
- Beacon 6:
- Beacon 7:
- Beacon 8:
- Beacon 9:
- Beacon 10:

12. At termination: rename and store ROS bag
13. At termination: record robot battery voltages
    - Robot 1:
    - Robot 2:
    - Robot 3:
    - Robot 4:
    - Robot 5:
    - Robot 6:
    - Robot 7:
    - Robot 8:
    - Robot 9:
    - Robot 10:
14. Notes:

Recording the battery voltage for each robot at the start and end of a trial is necessary because I disabled battery reporting during development, as part of the effort to simplify troubleshooting. Checking the voltages serves two purposes: first, it indicates when batteries are depleted, and second, it allows for rough calculations of the consumed power (after characterizing the battery's performance, since all robots carried identical batteries).

Getting the GPS coordinates of the Nexus and the polar coordinates of the beacons is not strictly necessary. It was done in this case to enable simple illustrations of the test scenarios/environments.

CHAPTER 5. RESULTS AND DISCUSSION

When the system was performing the best it ever had, I ran some trials under varying conditions. Of course, there were thousands of tests under all manner of conditions, but those were in the service of development and not representative of the system as a whole. The example scenario given in 4.2 would be great to see happen in the field, but that is not quite what happened. I will discuss the trials in the order that they occurred and attempt to elucidate some of the decisions and results.

Before testing in each environment, I performed some basic calibration of the system. This mostly consisted of getting a rough idea of the signal strengths correlated to distances (as most of the available parameters remained unchanged after they had been satisfactorily determined during development). For example, to calibrate the required signal strength for extension requests, I measured the RSSI that a node received from the nexus as it moved away. When it was at the desired distance, I noted the RSSI. The same process applied to determining the required signal strength to disable a beacon or bump another node. While this procedure was far from perfect, it gave me a rough starting point for experimentation.

The basic model, if you will, behind the experiments is a simplified version of plant roots growing from a seed (albeit not in every direction). Expanding out from this starting point, the robots should "hunt" for the beacons, utilizing each other to expand beyond their singular range. Plant roots do something similar, although they are continuous, rather than discrete.

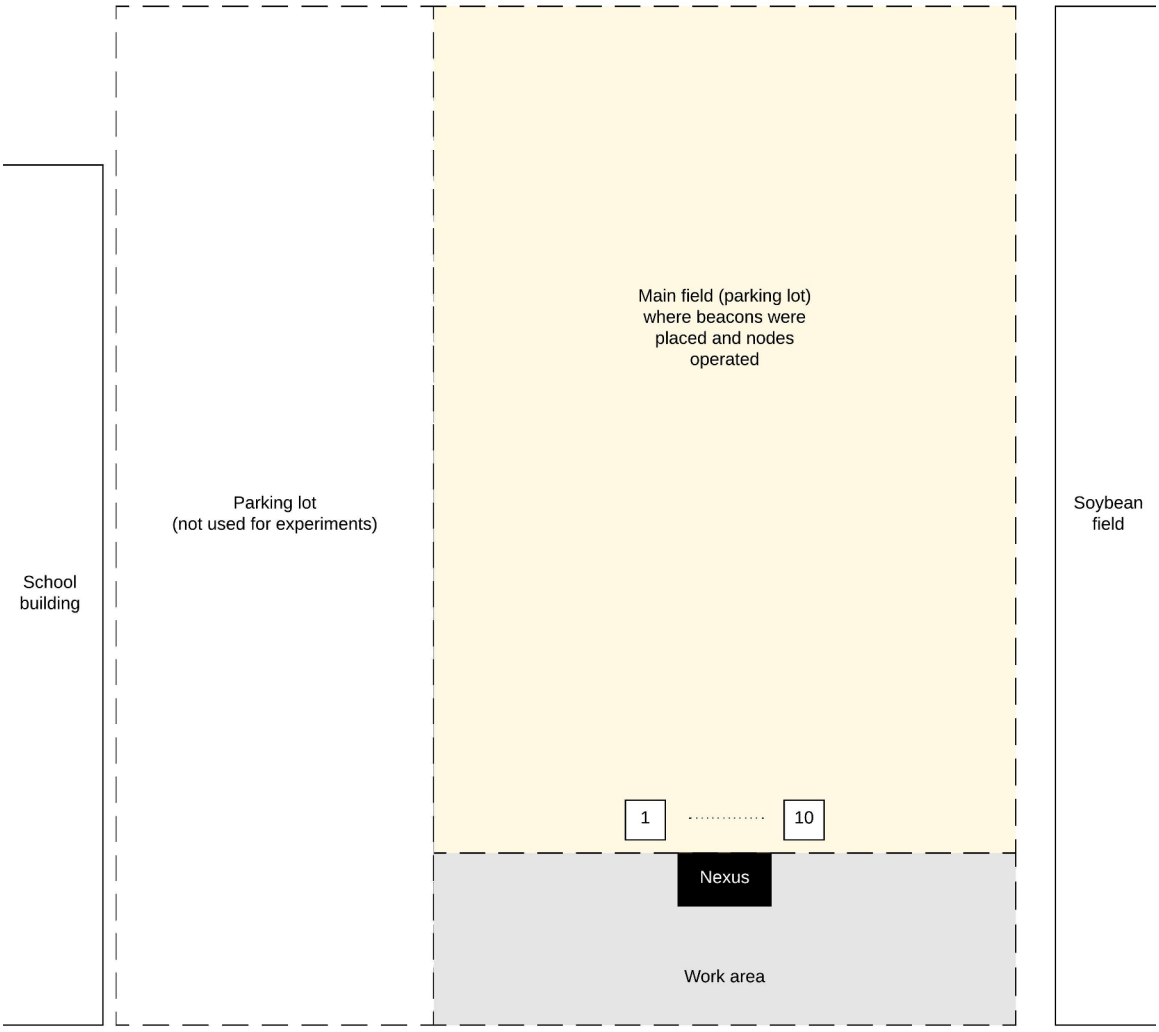

Figure 5.1. Layout of the testing area during experimentation (and some development). The robots started in a line directly in front of the nexus.

One more thing...

There are 10 total beacons, but I did not deploy all the beacons for each test, mostly due to space constraints. You may notice that the beacons change between trials, but this was simply due to operational convenience. All the beacons are identical.

Table 5.1.

Results for Trial 1.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
1	1	48 ft @ 75°	not found
	5	73 ft @ 65°	71 ft
	9	34 ft @ 105°	30 ft
	10	66 ft @ 100°	not found

Table 5.2.

Results for Trial 2.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
2	1	48 ft @ 75°	not found
	5	73 ft @ 65°	45 ft
	9	34 ft @ 105°	15 ft
	10	66 ft @ 100°	not found

5.1 Trial 1

Beacons 5 and 9 were found almost immediately, probably indicating that the signal strength required to "find" a beacon was too low.

5.2 Trial 2

Robot 5 seemed to struggle to find signals to seek. It drove away from the beacons (and behind the Nexus) for a few minutes, but then corrected and started moving toward the beacons. Since the environment was not very controlled, this could have been the result of reflections off of cars, buildings, etc. Robot 2 did not drive in the wrong direction, but it fruitlessly searched in circles.

Table 5.3.

Results for Trial 3.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
3	1	48 ft @ 75°	not found
	5	73 ft @ 65°	45 ft
	9	34 ft @ 105°	not found
	10	66 ft @ 100°	not found

5.3 Trial 3

The conditions for Trial 3 were the same as for Trial 2, including the beacon locations. Robot 3 drove within a few feet of a beacon, but did not disable it. However, only Beacon 5 was found at a distance of 45 feet. Robot 3 requested an extension, which was assigned to Robot 10. After an initial search, Robot 10 bumped Robot 3 without moving. Other extensions requests and bumps started happening so quickly that I lost track of what was going on. I let the system try to figure things out for awhile, but eventually terminated the trial, due to the confusion.

5.4 Trial 4

It seemed that the environment was perhaps too noisy and the robots were unable to make clear distinctions between signals, so I moved the beacons farther out. Despite being significantly farther away, Beacon 10 was found quite quickly at a distance of 83 feet. The signal strength required to disable a beacon was fairly high, due to what happened with Robot 3 in Trial 3, so this was surprising. Once that happened, the deployed robots searched for awhile, but couldn't reliably find any signals to track.

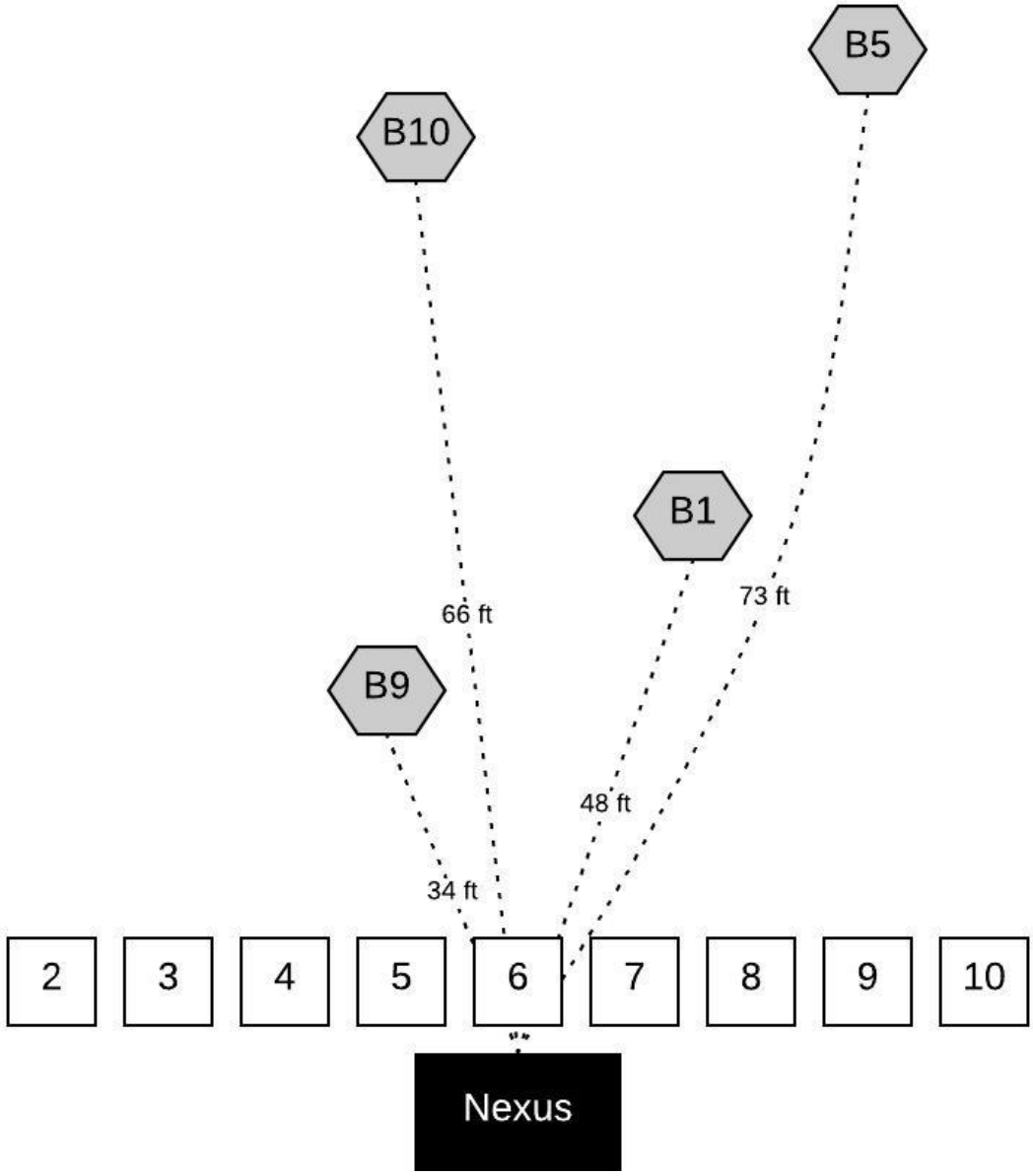

Figure 5.2. Arrangement at the start of trial 3. Squares are robots (nodes) and hexagons are beacons.

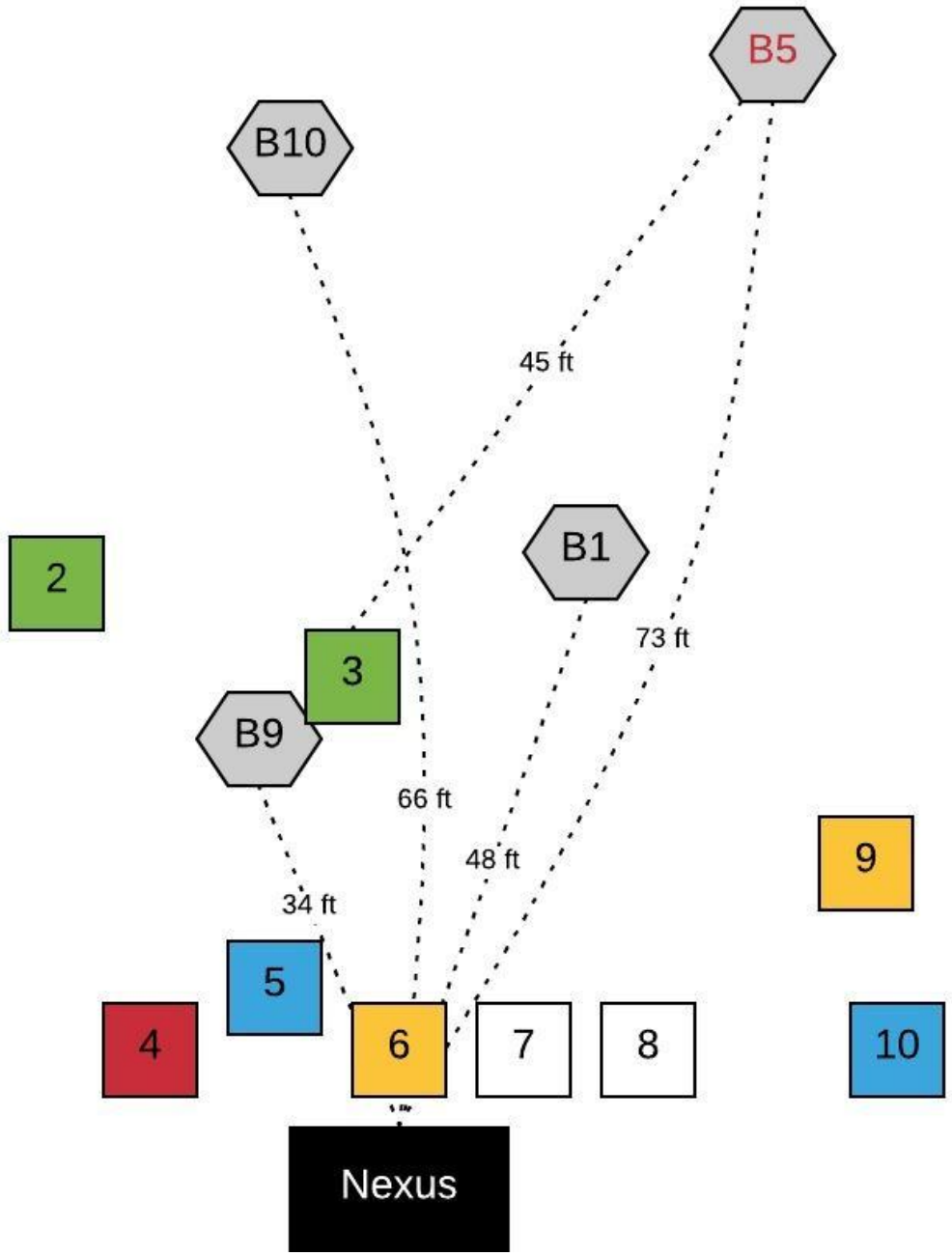

Figure 5.3. Arrangement at the end of trial 3. Squares are robots (nodes) and hexagons are beacons. For the nodes, white = idle, green = apex, yellow = transition, blue = relay edge, red = relay. For the beacons, red text means that beacon was found.

Table 5.4.

Results for Trial 4.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
4	1	91 ft @ 120°	not found
	5	177 ft @ 95°	not found
	9	178 ft @ 70°	not found
	10	86 ft @ 75°	83 ft

Table 5.5.

Results for Trial 5.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
5	1	48 ft @ 75°	not found
	5	73 ft @ 65°	not found
	9	34 ft @ 105°	not found
	10	66 ft @ 100°	not found

5.5 Trial 5

Trying again did not make much of a difference. In fact, no beacons were found this time, and Robot 2 ended up requesting an extension when it was to the side of the Nexus, rather than out in the field of beacons.

5.6 Trials 6 and 7

Some amount of failure and poor performance was expected, so I continued to give the system chances. Trial 6 was the first trial of a new day, so the beacons moved a bit. Similar results were seen in these trials. Beacon 9 was found in Trial 6, but at a distance of 100 feet. Beacon 5 was found in Trial 7, at a distance of 65 feet.

Table 5.6.

Results for Trial 6.

Trial	Beacon	Polar coords. from Nexus	Distance when found
6	1	69 ft @ 75°	not found
	4	148 ft @ 80°	not found
	5	68 ft @ 110°	not found
	9	116 ft @ 100°	100 ft

Table 5.7.

Results for Trial 7.

Trial	Beacon	Polar coords. from Nexus	Distance when found
7	1	69 ft @ 75°	not found
	4	148 ft @ 80°	not found
	5	68 ft @ 110°	65 ft
	9	116 ft @ 100°	not found

The behavior of the robots seemed to indicate that they were still struggling to discern signals.

5.7 Trial 8

During development of the system, it was often run with only one beacon and two or three robots. In an attempt to figure out what was happening, I went back to such an arrangement. A single beacon (Beacon 9) was placed at 116 ft @ 100 degrees from the Nexus and three robots were activated. One of the robots was even started about one third of the distance to the beacon, but this resulted in that robot searching many times, then driving toward the Nexus (away from the beacon).

Table 5.8.

Results for Trial 8.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
8	9	116 ft @ 100°	not found

Table 5.9.

Results for Trial 9.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
9	9	200 ft @ 90°	not found (barely)

5.8 Trial 9

At this point, I was wondering if the environment was somehow attenuating the radio signals, giving the robots nothing to track. I reduced the deployment to a single beacon and a single robot, set the power of the XBees to maximum, and put the beacon straight ahead at about 200 feet. Additionally, the beacon was set on top of a car to free up more of the Fresnel zone. All of this resulted in the robot tracking straight toward the beacon and getting within 6 feet of it.

5.9 Trial 10

The beacon (and the car) were moved to about 180 ft @ 75 degrees and the same system configuration was run. The purpose was to be more convinced that the success of the last trial was not a fluke. With the same result as in Trial 9, it seems that the robots are able to track a signal quite precisely. Even better, these trials demonstrated the self-correcting nature of the algorithm. The robots do not drive very straight most of the time, so they end up turning away from the signal they are tracking, but nothing in the algorithm is aware that this happened. Since the robots stop to scan 360 degrees with the directional antenna, then turn around in a circle

with the antenna facing forward to try to find the same signal, they end up realigning themselves toward the signal with each search iteration.

The beacons' proximity to the ground appears to have been causing the issues. Too much of the Fresnel zone was obscured, severely attenuating the signal. Worse, the pavement obstruction was probably also scrambling the signal.

Table 5.10.

Results for Trial 10.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
10	9	180 ft @ 75°	not found (barely)

5.10 Trial 11

Returning to full-scale testing, the beacons were placed a few feet off the ground. Despite being able to track the beacons more effectively, the robots still struggled to track each other. Beacon 9 was found at a distance of 40 feet, but after that, Robot 4 requested an extension without leaving the start line. Robot 10 was assigned to that request, but because the directional antennas were to the side of each other (since Robot 10 had not yet moved either), Robot 10 struggled to find Robot 4's signal. Also, the robots were not elevated like the beacons now were, so I suspect they were dealing with highly-obstructed Fresnel zones. Robot 10 eventually completed its search, but it immediately bumped Robot 4 without moving. The robots did, however, exhibit the desired behavior of requesting, bumping, and changing state. Robot 10 went into relay edge mode and began searching for additional beacon signals. After seeing one, it requested an extension, which is what it is supposed to do. Robot 7 began seeking it.

Robot 4 moved forward a bit, but quickly requested another extension, which Robot 5 was assigned to. Robot 2 then requested an extension while still too close to the Nexus, which was assigned to Robot 6. Then Robot 5 immediately requested

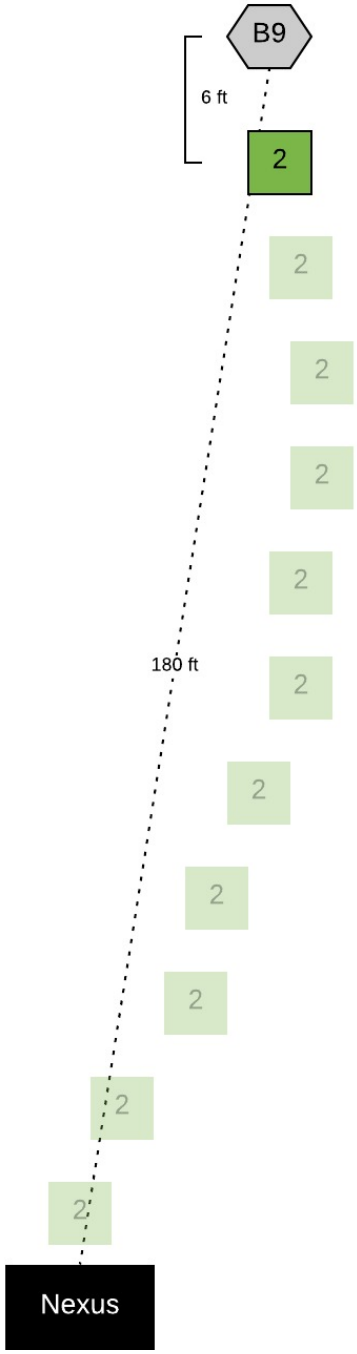

Figure 5.4. Trial 10, when one robot sought one (elevated) beacon. Each intermediate location is where the robot stopped to initiate a new search.

its own extension, which Robot 8 started working on. Once Robot 8 bumped Robot 5, 5 went into relay edge mode and requested an extension to chase a beacon (again, correct behavior). A short time later, Robot 4 requested another extension, but since all the robots were deployed and all of the apices were chasing promising signals, the request was denied.

As I've described, the scene was a bit chaotic. However, the robots did successfully execute the desired actions, despite not moving much.

Table 5.11.

Results for Trial 11.

Trial	**Beacon**	**Polar coords. from Nexus**	**Distance when found**
11	1	188 ft @ 78°	not found
	3	59 ft @ 95°	not found
	9	96 ft @ 65°	40 ft
	10	133 ft @ 85°	not found

5.11 Trial 12

Things got a bit hectic in Trial 11, so I ran the same setup again. This time, the two robots that initially deployed into apex mode pressed forward. They ended up circling the same beacon and even running into it, but failed to disable it. No extension requests were made.

5.12 Trial 13

The setup for this trial was the same as for the last two trials, with the exception of slightly widening the margin of acceptability for considering two signals to be the same. Changing this parameter did, in fact, make the robots track the beacons more loosely, occasionally even seeming to send them directly away from

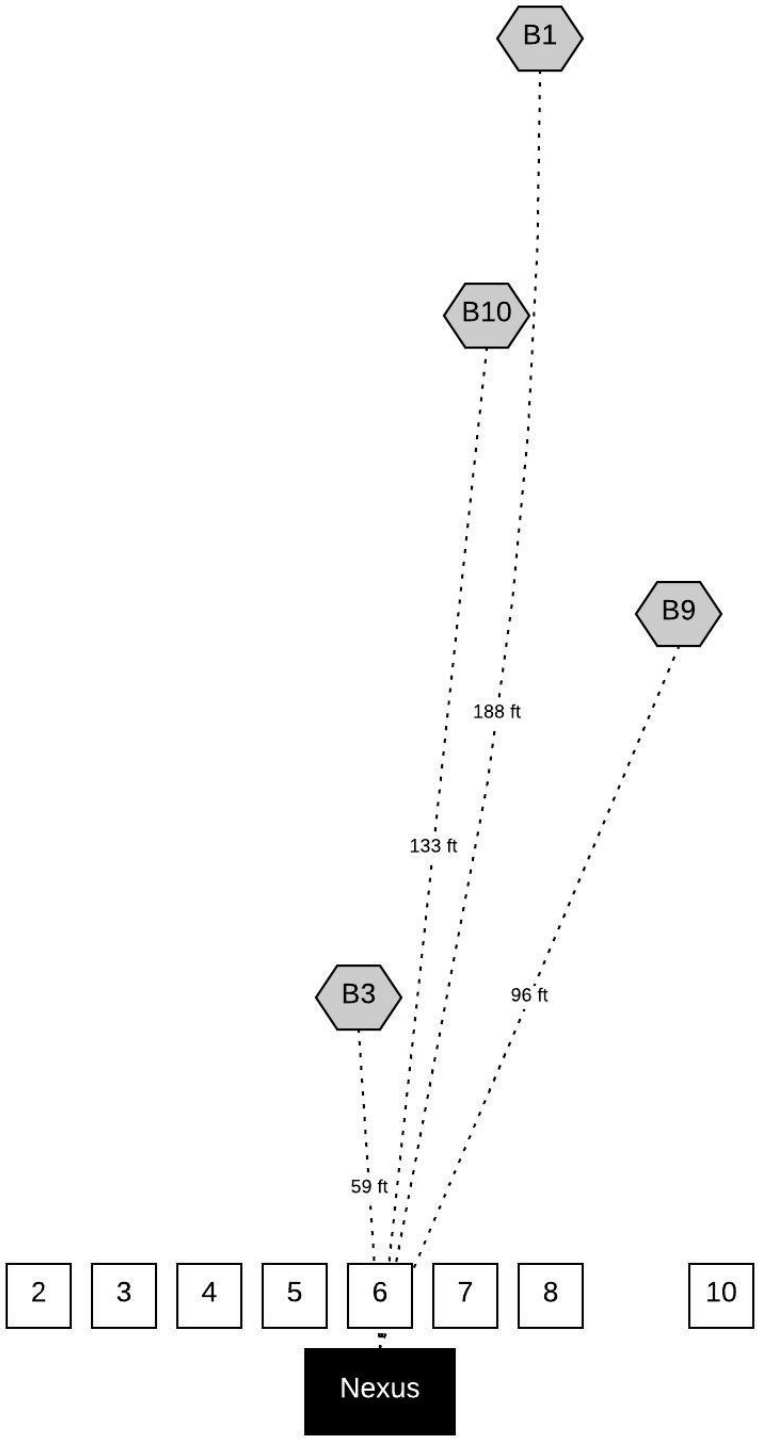

Figure 5.5. Arrangement at the start of trial 11. Squares are robots (nodes) and hexagons are beacons.

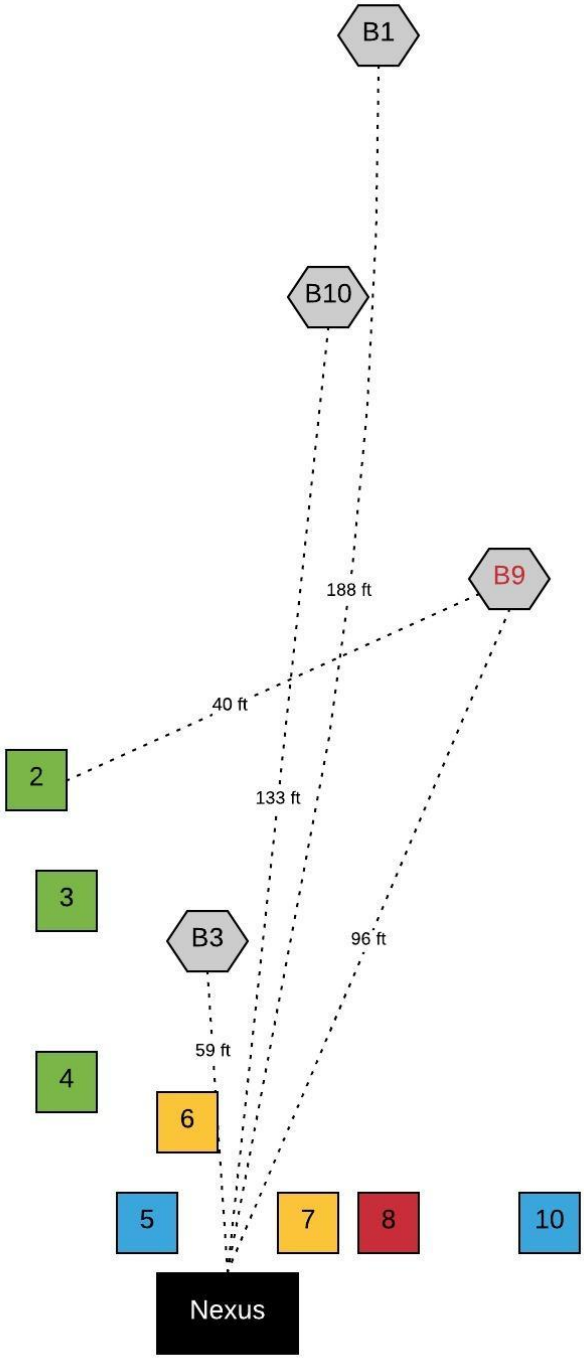

Figure 5.6. Arrangement at the end of trial 11. Squares are robots (nodes) and hexagons are beacons. For the nodes, white = idle, green = apex, yellow = transition, blue = relay edge, red = relay. For the beacons, red text means that beacon was found.

Table 5.12.

Results for Trial 12.

Trial	Beacon	Polar coords. from Nexus	Distance when found
12	1	188 ft @ 78°	not found
	3	59 ft @ 95°	not found (barely)
	9	96 ft @ 65°	not found
	10	133 ft @ 85°	not found

the beacons (perhaps due to seeing a similar enough signal on the back lobe of the antenna's radiation pattern). The robots got a bit farther away from the Nexus before requesting extensions, but that was inconsistent. Another effect of loosening the margin was that the beacons were disabled more easily, sometimes at significantly different distances.

The robots have a tendency to run into each other, because, at times, they are tracking the same signals. This is analogous to plant roots tangling around each other when seeking nutrient concentrations. If the nodes told each other who they were seeking, instead of relying on stigmergy, they would probably not run into each other as much.

Table 5.13.

Results for Trial 13.

Trial	Beacon	Polar coords. from Nexus	Distance when found
13	1	188 ft @ 78°	not found
	3	59 ft @ 95°	25 ft
	9	96 ft @ 65°	not found
	10	133 ft @ 85°	60 ft

5.13 Trials 14 through 18

Several more trials resulted in several more similar outcomes. The robots track the beacons fairly well, but struggle to track each other. When they do manage to interact, though, the structure of those interactions is good. Figures 5.7 through 5.17 show the chronological steps of Trial 17.

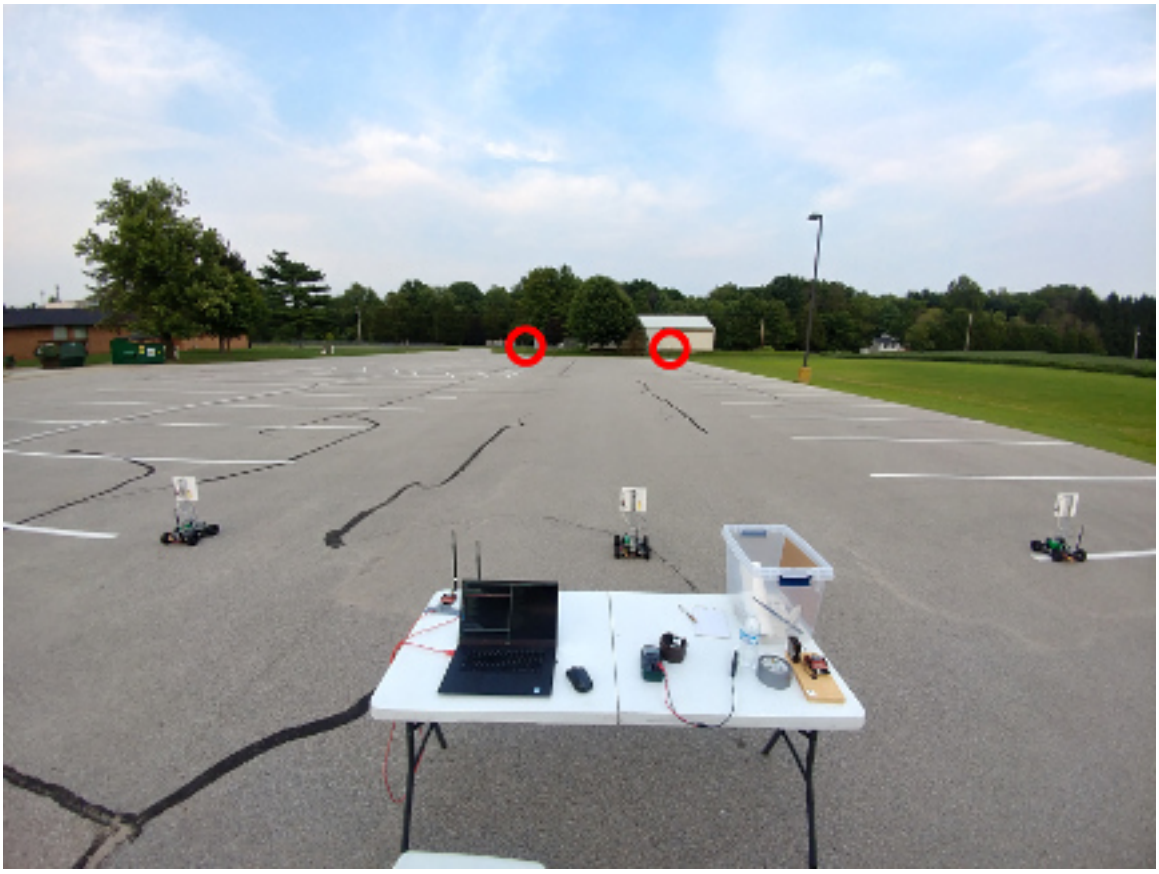

Figure 5.7. Trial 17: step 1. The beacons are difficult to see, so they are highlighted by red circles.

5.14 Discussion

Starting with the positive, the concept of a plant root-inspired, resource-hunting multi-robot system seems plausible. Despite the difficulties faced

Figure 5.8. Trial 17: step 2. The highlighted robot has completed one search iteration and has moved one step toward the beacon.

in this project, there were enough successes to believe that, with more time and resources, the system could be made much more robust. Automated resource finding will be important as we push further out into space. We will need the ability to harvest resources from the places around us, since leaving Earth with everything required for the mission will become increasingly impractical. While the system shown in this document is very far from fulfilling such a role, it does represent a possible starting point.

Following a gradient has been demonstrated to be practical, albeit sometimes problematic for this project. Using the received signal strength indicator (RSSI) for this purpose is not reliable, but there are more sophisticated indicators, such as the

Figure 5.9. Trial 17: step 3. The highlighted robot continues to track toward the beacon.

link quality indicator (LQI), that could be evaluated. The robots could also be constructed with their antennas much higher off the ground, or perhaps with multiple directional antennas to enable differential signal identification (like our ears helping us determine a sound's direction). Yet another option would be to use other types of sensors, such as sound, light/laser, etc, that may be more predictable. Of course, the "resource" that is being sought does not need to be radio signals, since those were chosen for practical reasons in this project. Following chemical gradients, airborne or not, is another option. Perhaps tracking magnetic fields would be more practical. The purpose of this experiment was not to focus on wireless communication, but rather to show that the plant's method of following gradients

Figure 5.10. Trial 17: step 4. The highlighted robot continues to track toward the beacon.

away from a starting point is transferable to robots. The robots did, in fact, follow gradients and execute a state machine that approximates plant roots, and they did this with low-power computing and inexpensive, imprecise sensing and actuation.

I expected these trials to be successful, at least partially, because I saw individual behaviors work correctly in the lab and during controlled testing. My intuition, augmented by an understanding of the fundamentals of how plant roots operate, also told me that the basic notions upon which this project is based are valid. Nonetheless, the complexity and fragility of a prototype system like this are not to be underestimated.

Figure 5.11. Trial 17: step 5. The highlighted robot continues to track toward the beacon.

One of the key lessons taken from this project was the difficulty and unreliability of using low-power (digital) RF communication for all sensing and communication. The effectiveness of the radios was dramatically affected by environmental interference, such as a Wi-Fi access point being enabled or proximity to the ground. The sensing and communication also became less reliable as more robots were activated, since this added to the ZigBee network traffic. I had to continue to slow the search process and allow for longer timeouts as more robots and beacons were added. ZigBee, in particular, is not designed for such a dynamic application. It is intended for mostly static applications where power consumption is more important than responsiveness and synchronization. For example, the XBee

Figure 5.12. Trial 17: step 6. The highlighted robot continues to track toward the beacon.

radios do not necessarily push received packets out of the serial port immediately. Sometimes, several packets are buffered and spit out all at once, which does not lend itself to an application that requires predictable timing. For example, I often noticed that one step of a search would time out and move to the next step, but the XBee would then dump multiple messages very quickly. This was understood by the RPi to be a signal coming from a direction that it may not have actually come from. Another negative effect of this behavior affected the reading of RSSI. The Xbees do not include an RSSI value in the data they send to the serial port. Instead, when the RPi received a message from the XBee, the RPi had to make a request for the RSSI of the last packet. If there is too much latency in this process happening for

Figure 5.13. Trial 17: step 7. The highlighted robot continues to track toward the beacon.

every message, the RSSI value reported to the RPi will misrepresent some of the messages it received in a burst, which produces inaccuracies in matching and searching for signals. Using gradients of radio frequency energy as a mechanism for tracking is not impractical, in principle. However, using such a dramatically discretized form of measuring that energy does not suit this application. Overall, I do not believe ZigBee is a good choice for this system.

Another key take-a-way is the difficulty of developing and managing what amounts to a state machine of this complexity. While it is, of course, possible to do so, it would be a major effort to make such a system robust and reliable. Alternative system architectures/control methods should be explored, such as

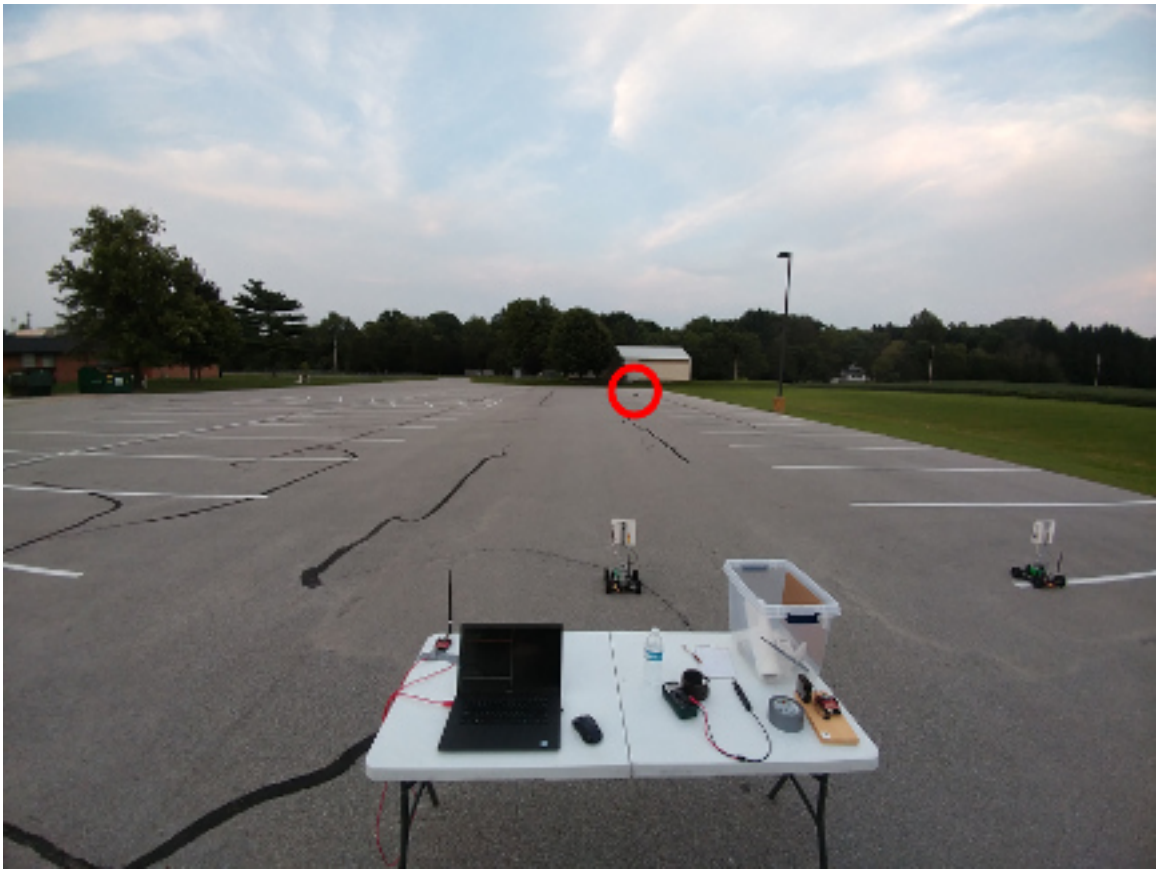

Figure 5.14. Trial 17: step 8. The highlighted robot continues to track toward the beacon.

neural networks (as proposed by the authors of (Simes et al., 2011)). Aside from dealing with the problems associated with wireless communication, convincing the many-faceted state machine to behave as I intended was the biggest challenge.

The purpose of the robots built for this experiment was not to be anywhere near a final product. That being said, care still needs to be taken when designing a prototype or single-purpose machine. The hardware and mechanics of the robots performed satisfactorily, for the most part, but there are two notable components that made testing and development more problematic. The stepper motor that rotated the directional antenna was strong enough to hold the antenna in place in moderate winds, but it was not able to hold the antenna when the robot hit a bump

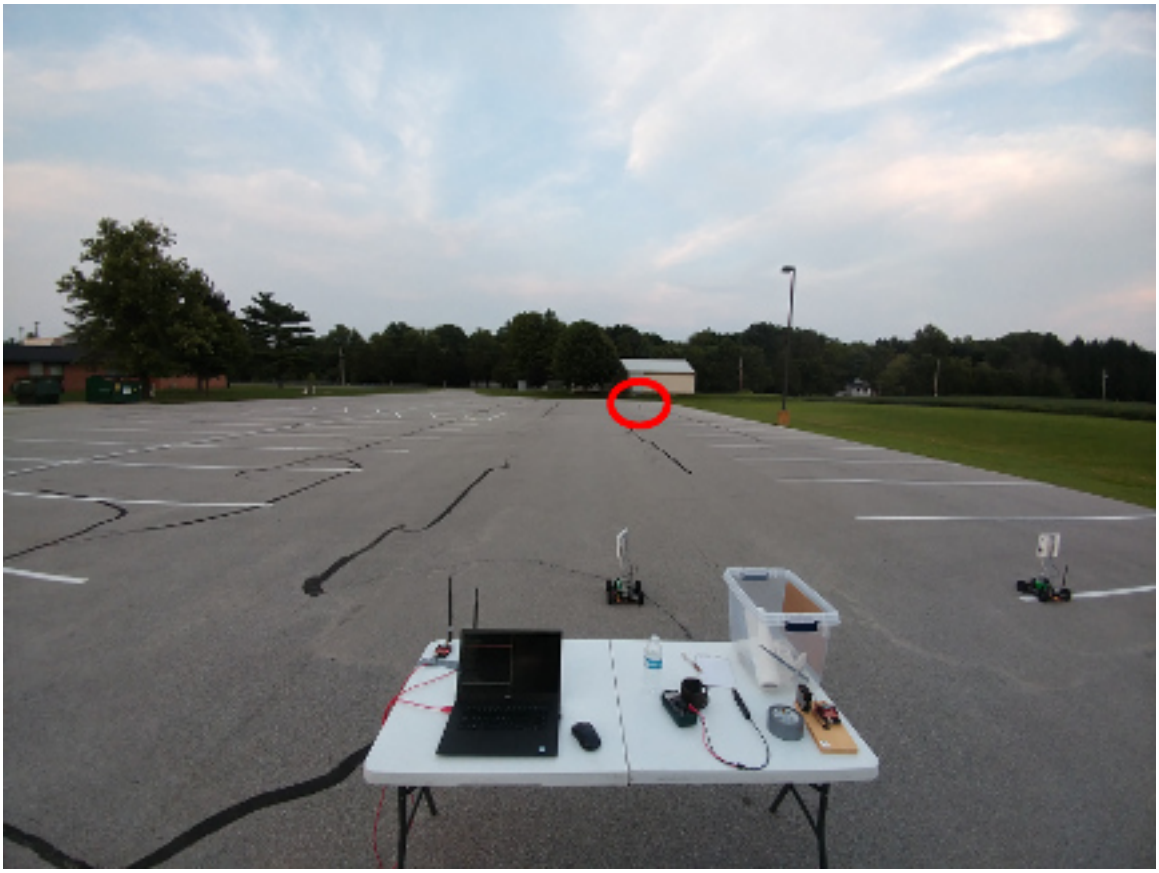

Figure 5.15. Trial 17: step 9. The highlighted robot continues to track toward the beacon.

or shuddered during a turn. A stronger motor, or perhaps a different design, could help with this. The other major issue was the robots' method of turning, combined with the motor controller. Skid steer is a simple turning mechanism, but with the friction between the tires and asphalt, combined with the somewhat top-heavy weight distribution, the robots often shook when turning. Additionally, the motor controllers did not reliably start and stop the drive motors, resulting in robots occasionally turning excessively, turning awkwardly, or not turning at all.

Returning to the metric of success mentioned in Section 4.5, I interpret these results as meeting the success criteria. It is not pretty or elegant, but seeing the system work as well as it did shows the beginnings of plausibility. The robots

Figure 5.16. Trial 17: step 10. The more distant highlighted robot has requested an extension, which the closer highlighted robot has been assigned to (notice its antenna in mid-rotation).

showed the ability to successfully track toward the beacons and "gather" the resources (i.e. disable the beacons). They also detected a weakening connection to the rest of the system and requested resources to continue exploration, which the rest of the system responded to.

5.15 Conclusion

My conclusion is that it is feasible and potentially useful to explore an environment with a multi-robot system, using strategies adapted from plant roots.

Figure 5.17. Trial 17: step 11. The more distant highlighted robot was bumped by the closer highlighted robot. The closer robot has now called for its own extension, which the remaining robot is in the process of executing.

The outcome of this particular effort was not a complete success, but it did show that the concept can work. Even continuing development of the state machine approach could produce good, consistent results, but other architectures may prove more effective.

CHAPTER 6. MOVING FORWARD

After moving through the initial phase of development and testing of this system, I have identified a handful of recommended tasks for continuing this research. These tasks are discussed in no particular order, and this list is far from complete. It is simply intended as a starting point.

6.1 Verify Omni-directional Antenna Radiation Pattern

The frequency, severity, and unpredictability of problems with seeking signals was enough to justify verifying the radiation pattern of the omni-directional antennas. Ideally, the radiation pattern would be radially symmetrical (perpendicular to the ground plane, at least). If this is not the case, the orientation of the antennas, which is difficult to discern at a glance, would have an undesired effect on the robots' ability to seek each other and the beacons.

Having access to an appropriately instrumented RF anechoic chamber would be great, but in the likely event that you do not, getting a rough idea of the antenna's 2-D radiation pattern could be done with a bit of leg work. In the quietest and least reflective RF environment you can find, measure the signal strength at as many points as possible along widening perimeters around the transmitter, such as in Figure 6.1. This data could then be represented as a heat map, contour map, etc. to provide a visualization of the radiation pattern. Of course, this method is far from perfect and rather imprecise, but would illuminate any glaring differences from the ideal radiation pattern.

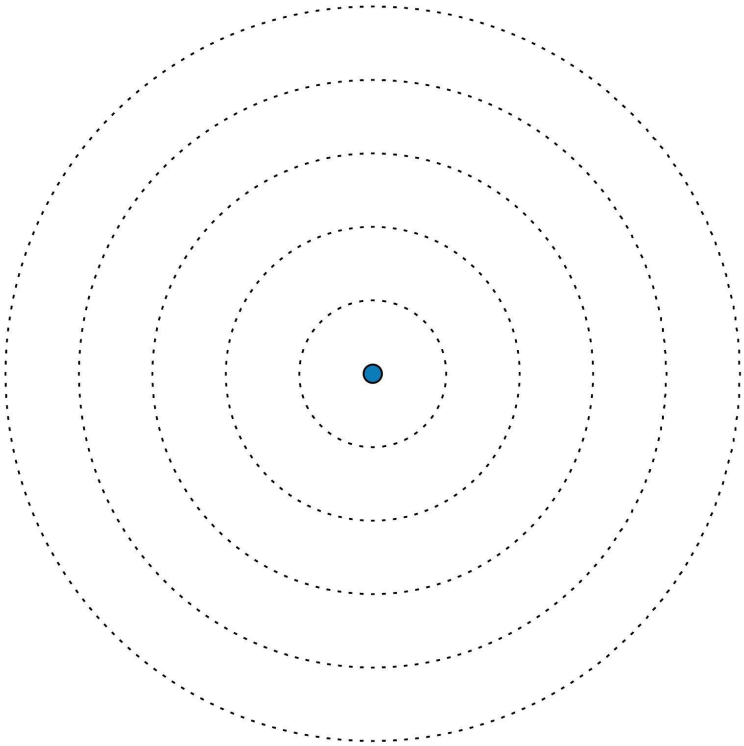

Figure 6.1. Sample of in-plane paths along which to measure the signal strength from the omni-directional antenna (represented by the blue dot), to verify that there is not significant asymmetry in the radiation pattern.

6.2 More Thorough Bench Testing

With careful planning, more testing could be done in the lab before moving the system into the field environment. Bench (lab) testing is helpful because it allows you to better control environmental variables. Testing in the field makes it very difficult, often impossible, to isolate variables. With a system as complex as this, isolating variables/behaviors is essential to making progress. Observing the following behaviors in a controlled environment would be prudent, before moving into the field, to avoid wasting time and effort. Most of these were observed in the lab during the course of this research, but not all. Those that weren't were due to lack of necessary equipment or time to circumvent the need for such equipment.

- Deployment (by the nexus)
- Seeing signals of interest and recognizing/storing the best
- Turning to move toward the most interesting signal (by recognizing what appears to be the same signal again)
- Moving toward that signal, then initiating another search
- Disabling beacons if they have been "found"
- Recognizing the need for an extension and making the request
- Stably waiting for that extension to be fulfilled
- Bumping (and the associated information exchange between nodes)
- Seeking another node by the same process as that for a beacon
- Creating a branch via a relay edge node requesting a second extension to explore a second signal of interest
- Receiving a bump route and reacting accordingly, including when being reassigned

With those basic behaviors working independently, the work moves to ensuring that they work together and initiate each other, where appropriate. As I mentioned before, those behaviors were (mostly) independently verified in the lab, but field testing did not consistently display those behaviors. One key factor that I've identified is the amount of signal interference due to proximity to the ground (i.e. obstruction of the Fresnel zone). It would be helpful to quantify, in a controlled environment, the scale and nature of the Fresnel zone obstruction. This data would inform not only the choice of testing environment, but may highlight the need for physical and/or code changes to the robots.

As the system matured and became more complex, reliable procedures for verifying the above listed behaviors were increasingly elusive. This was due to the increasing difficulty of creating in-situ conditions that isolated the desired behavior, while still providing the structure and data that the system expected to have during that behavior. While I attempted to allow for this while developing the software, more modularity and isolation of code is necessary. Two complementary approaches come to mind. The first is having a debugging software "switch" that, when enabled, retrieves real-time operational data and state information from a controlled input (e.g. files, intentionally-generated signals, etc.), rather than from the "natural" sources. This would require carefully-designed input mechanism(s), but based on my experience, would save time over having to operate by trial and error in the field. The second, complementary, approach is to employ a simulation environment. Simulation would not only provide the ability to control, with fine granularity, the inputs to the system. It would also allow for much more rapid iteration, due to the time compression of the simulation.

With such a development infrastructure in place, it would be feasible to implement some additional functionality. Some suggested improvements, behaviors, and measures to further develop and verify in a lab environment are:

- More detailed reporting to the nexus of what a node is seeing and/or seeking (for monitoring and debugging)

- Better measure of distance by using more sophisticated metrics of signal quality (rather than simply the signal strength)
- Direct association of signal values with directional antenna orientation

6.3 Granularity of Field Data Collection

During and after field testing, it was difficult to decode the intermediate and transient states of the robots. A more detailed picture of the system in operation would aid in development and debugging. It would also make clearer the analogy to the biological plant root methodology. Modifying the nodes to regularly report the following data to the nexus would be helpful:

- Parent
- Child(ren)
- Strength of best detected signal
- Relative direction of best detected signal
- Name of beacon or node being sought
- Current action

The nexus could then record and display, perhaps graphically, this information to the user. The analogy to plant roots could be made visual by drawing a conceptual picture of the system as it evolves, similar to 2.6. Having such a record of experiments would enable a "play-by-play" for analysis, as well as a more detailed correlation to the functions and actions of plant roots.

CHAPTER 7. SUMMARY

We are going to need autonomous systems to help us locate resources in extraterrestrial and/or unexplored environments. One way we might do this is with multi-robot systems that explore their environment in a way analogous to plant roots. Plants are very successful organisms, despite possessing low levels of sensing, computing, and actuation. Transferring these properties into a machine means that machine can be relatively simple and inexpensive.

A report by the European Space Agency (ESA) (Simes et al., 2011) explored this idea through simulation and concluded that it was promising. This document describes an effort to build and deploy a real system of robots, inspired by the ESA report, that executed a proof-of-concept experiment. The "nutrients" that the robots look for are radio beacons, which they attempts to locate with directional antennas. Similar to the elongation and branching behaviors of plant roots, the robotic system extends and explores via robots acting as relays.

There were and still are many challenges to accomplishing such a task. Using off-the-shelf ZigBee wireless devices, for example, turns out to be rather inappropriate for this application. Creating a control architecture that reliably executes the desired actions is another significant challenge (one that was not completely overcome). Despite the difficulties, this experiment showed that the concept of a robotic plant root analog is feasible outside of simulation and deserves further exploration.

LIST OF REFERENCES

LIST OF REFERENCES

Allen, M. F., Vargas, R., Graham, E. a., Swenson, W., Hamilton, M., Taggart, M., ... Estrin, D. (2007, 11). Soil Sensor Technology: Life within a Pixel. *BioScience*, *57*(10), 859. Retrieved from http://dx.doi.org/10.1641/B571008 doi: 10.1641/B571008

Alterovitz, G., Muso, T., & Ramoni, M. F. (2009, 1). The challenges of informatics in synthetic biology: Frombiomolecular networks to artificial organisms. *Briefings in Bioinformatics*, *11*(1), 80–95. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2810114&tool=pmcentrez&rendertype=abstract doi: 10.1093/bib/bbp054

Alvarez-Ramírez, J. (1993, 10). Using nonlinear saturated feedback to control chaos: The H??non map. *Physical Review E*, *48*(4), 3165–3167. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/9960954 doi: 10.1103/PhysRevE.48.3165

Anderson, B. D. O., Bitmead, R. R., Johnson, C. R., Riedle, B. D., & Parks, R. P. C. (1989). *Stability of Adaptive Systems : Passivity and Averaging Analysis * Linear Control Systems a Computer-aided Approach** (Vol. 25) (No. 1). MIT press.

Astrom, K. (1987, 2). Adaptive feedback control. *Proceedings of the IEEE*, *75*(2), 185–217. Retrieved from http://ieeexplore.ieee.org/document/1457988/ doi: 10.1109/PROC.1987.13721

Åström, K. J. (1980). Design principles for self-tuning regulators. In H. Unbehauen (Ed.), *Methods and applications in adaptive control* (pp. 1–20). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://www.springerlink.com/index/10.1007/BFb0003250 doi: 10.1007/BFb0003250

Atkinson, S., & Williams, P. (2009). Quorum sensing and social networking in the microbial world. *Journal of the Royal Society, Interface / the Royal Society*, *6*(40), 959–78. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2827448&tool=pmcentrez&rendertype=abstract doi: 10.1098/rsif.2009.0203

Audi, R. (1999). *The Cambridge Dictionary of Philosophy* (Vol. 584) (No. 2). Cambridge University Press. Retrieved from http://www.amazon.com/dp/0521637228 doi: 10.1198/tech.2007.s467

Bäck, T. (2001, 6). Introduction to the Special Issue: Self-Adaptation. *Evolutionary Computation*, *9*(2), iii-iv. Retrieved from http://www.mitpressjournals.org/doi/10.1162/106365601750190361 doi: 10.1162/106365601750190361

Baluska, F., Mancuso, S., Volkmann, D., & Barlow, P. (2004). Root apices as plant command centres: the unique 'brain-like'status of the root apex transition zone. *Biologia (Bratisl.)*, *59*(Suppl. 13), 7–19. Retrieved from http://www.izmb.uni-bonn.de/baluska/pdf/articles/58.pdf

Bar-Yam, Y. (1997). *General Features of Complex Systems* (Vol. I) (No. 1). EOLSS UNESCO Publishers.

Bar-Yam, Y. (1998). *Dynamics of Complex Systems* (Vol. 12) (No. 4). Cambridge, MA: Perseus Press. Retrieved from http://link.aip.org/link/CPHYE2/v12/i4/p336/s1&Agg=doi doi: 10.1063/1.168724

Bar-Yam, Y. (2003). When systems engineering fails-toward complex systems engineering. In *Smc'03 conference proceedings. 2003 ieee international conference on systems, man and cybernetics. conference theme - system security and assurance (cat. no.03ch37483)* (Vol. 2, pp. 2021–2028). IEEE. Retrieved from http://ieeexplore.ieee.org/document/1244709/ doi: 10.1109/ICSMC.2003.1244709

Bar-Yam, Y. (2004a, 7). A mathematical theory of strong emergence using multiscale variety. *Complexity*, *9*(6), 15–24. Retrieved from http://doi.wiley.com/10.1002/cplx.20029 doi: 10.1002/cplx.20029

Bar-Yam, Y. (2004b, 3). Multiscale variety in complex systems. *Complexity*, *9*(4), 37–45. Retrieved from http://doi.wiley.com/10.1002/cplx.20014 doi: 10.1002/cplx.20014

Basso, M., Evangelisti, A., Genesio, R., & Tesi, A. (1998). On Bifurcation Control in Time Delay Feedback Systems. *International Journal of Bifurcation and Chaos*, *08*(04), 713–721. Retrieved from http://www.worldscientific.com/doi/abs/10.1142/S0218127498000504 doi: 10.1142/S0218127498000504

Bechtel, W., & Richardson, R. C. (1992). Emergent phenomena and complex systems. *Emergence or reduction*, 257–288.

Bedau, M. A. (2002). Downward Causation and the Autonomy of Weak Emergence. *Principia*, *6*(1), 5–50. Retrieved from https://periodicos.ufsc.br/index.php/principia/article/view/17003%5Cnhttp://philpapers.org/rec/BEDDCA doi: 10.7551/mitpress/9780262026215.003.0010

Beyer, H.-G. (1996). Toward a Theory of Evolution Strategies: Self-Adaptation. *Evolutionary {C}omputation*, *3*(3), 311–347. Retrieved from http://www.mitpressjournals.org/doi/abs/10.1162/evco.1995.3.3.311

Brabazon, A., ONeill, M., & McGarraghy, S. (2015). Plant-inspired algorithms. In *Natural computing series* (Vol. 28, pp. 455–477). Springer Berlin Heidelberg.

Retrieved from http://link.springer.com/10.1007/978-3-662-43631-8_25 doi: 10.1007/978-3-662-43631-8{_}25

Caprari, G., Colot, A., Siegwart, R., Halloy, J., & Deneubourg, J. L. (2005). Building mixed societies of animals and robots. *IEEE Robotics & Automation Magazine*, *12*(2), 58–65.

Chalam, Y. (1987). *Adaptive Control Systems: Techniques and Applications*. CRC Press. Retrieved from https://www.crcpress.com/Adaptive-Control-Systems-Techniques-and-Applications/Chalam/9780824776503

Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., ... Whittle, J. (2009). *Software Engineering for Self-Adaptive Systems: A Research Roadmap* (B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, & J. Magee, Eds.). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://link.springer.com/10.1007/978-3-642-02161-9_1 doi: 10.1007/978-3-642-02161-9{_}1

Colestock, H. (2005). *Industrial robotics: selection, design, and maintenance*. McGraw-Hill. Retrieved from https://books.google.com/books?id=YgJUAAAAMAAJ

Couzin, I. D. (2009). Collective cognition in animal groups. *Trends in Cognitive Sciences*, *13*(1), 36–43. Retrieved from http://www.sciencedirect.com/science/article/pii/S1364661308002520 doi: 10.1016/j.tics.2008.10.002

Couzin, I. D., Krause, J., James, R., Ruxton, G. D., & Franks, N. R. (2002). Collective memory and spatial sorting in animal groups. *Journal of Theoretical Biology*, *218*(1), 1–11. Retrieved from http://www.sciencedirect.com/science/article/pii/S0022519302930651 doi: 10.1006/jtbi.2002.3065

Delin, K. A., Jackson, S. P., Johnson, D. W., Burleigh, S. C., Richard, R., Mcauley, J. M., ... Baker, V. R. (2005). Environmental Studies with the Sensor Web: Principles and Practice. *Sensors (Basel, Switzerland)*, *5*(2), 103–117.

Domingos, P. (2002). *Machine learning*. McGraw-Hill. Retrieved from http://portal.acm.org/citation.cfm?id=778311

Doussan, C., Pagès, L., & Pierret, A. (2009). Soil exploration and resource acquisition by plant roots: An architectural and modelling point of view. *Sustainable Agriculture*, *23*(5-6), 583–600. Retrieved from https://hal.archives-ouvertes.fr/hal-00886193 doi: 10.1007/978-90-481-2666-8{_}36

Egardt, B. (1979). Stability of Adaptive Controllers. *Springer-Verlag Berlin Heidelberg New York*, 145. doi: 10.1007/BFb0005037

Fausett, L. (1994). *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications* (L. Fausett, Ed.). Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Floreano, D., & Mattiussi, C. (2008). *Bio-inspired artificial intelligence*. The MIT Press. Retrieved from https://mitpress.mit.edu/books/bio-inspired-artificial-intelligence doi: 10.1007/s10710-010-9104-3

Foehring, R. C., & Lorenzon, N. M. (1999, 3). Neuromodulation, development and synaptic plasticity. *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale*, *53*(1), 45–61. Retrieved from http://doi.apa.org/getdoi.cfm?doi=10.1037/h0087299 doi: 10.1037/h0087299

Fogel, D. B. (2006). *Evolutionary Computationn Toward a New Philosophy of Machine Intelligence* (3rd ed.). Wiley-IEEE Press. Retrieved from http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471669512.html doi: 10.1002/0471749214

Fromm, J. (2005, 6). *Types and forms of emergence.* Retrieved from http://arxiv.org/abs/nlin/0506028 doi: citeulike-article-id:262942

Giardina, I. (2008, 8). Collective behavior in animal groups: Theoretical models and empirical studies. *HFSP Journal*, *2*(4), 205–219. Retrieved from http://www.tandfonline.com/doi/abs/10.2976/1.2961038 doi: 10.2976/1.2961038

Goldstein, J. (1999, 3). Emergence as a Construct: History and Issues. *Emergence*, *1*(1), 49–72. Retrieved from http://www.tandfonline.com/doi/abs/10.1207/s15327000em0101_4 doi: 10.1207/s15327000em0101{_}4

Haasdijk, E., Eiben, a. E., & Winfield, A. F. T. (2013). *Individual, Social and Evolutionary Adaptation in Collective Systems*. Pan Stanford. Retrieved from http://eprints.uwe.ac.uk/20223/1/Haasdijk_etal_HCR_Collective_Social_Learning_finaldraft.pdf%5Cnhttp://www.panstanford.com/books/9789814316422.html doi: 10.4032/9789814364119

Haken, H. (2004). *Synergetics introduction and advanced topics* (1st ed.). Springer-Verlag Berlin Heidelberg. Retrieved from http://www.springer.com/us/book/9783540408246 doi: 10.1007/978-3-662-10184-1

Heylighen, F. (2011). *Stigmergy as a generic mechanism for coordination : definition , varieties and aspects.* doi: 10.1016/j.cogsys.2015.12.007

Huneman, P. (2008). Emergence Made Ontological? Computational versus Combinatorial Approaches. *Philosophy of Science*, *75*(5), 595–607. Retrieved from http://www.jstor.org/stable/10.1086/596777 doi: 10.1086/596777

Kalman, R. E. (1958). Design of a self-optimizing control system. *Trans. of the ASME*, *80*, 468–478.

Keijzer, F. (2003, 12). Making decisions does not suffice for minimal cognition. *Adaptive Behavior*, *11*(4), 266–269. Retrieved from http://adb.sagepub.com/content/11/4/266.extract doi: 10.1177/1059712303114006

Kernbach, S. (2008). *Structural self-organization in multi-agents and multi-robotic systems*. Logos Verlag Berlin GmbH.

Kernbach, S., Levi, P., Meister, E., Schlachter, F., & Kernbach, O. (2009). Towards self-adaptation of robot organisms with a high developmental plasticity. In *Computation world: Future computing, service computation, adaptive, content, cognitive, patterns, computationworld 2009* (pp. 180–187). doi: 10.1109/ComputationWorld.2009.11

Kernbach, S., Thenius, R., Kernbach, O., & Schmickl, T. (2009, 6). Re-embodiment of honeybee aggregation behavior in an artificial micro-robotic system. *Adaptive Behavior*, *17*(3), 237–259. Retrieved from http://adb.sagepub.com/content/17/3/237.short doi: 10.1177/1059712309104966

Kernis, M. H., & Goldman, B. M. (2003). Stability and variability in self-concept and self-esteem. In M. Leary & J. Tangney (Eds.), *Handbook of self and identity* (pp. 106–127). Guilford Press.

Kornienko, S., Kornienko, O., & Levi, P. (2004). Generation of desired emergent behavior in swarm of micro-robots. In *Proceedings of the 16th european conference on ai (ecai 2004)* (Vol. 16, p. 239). Valencia, Spain.

Kornienko, S., Kornienko, O., & Priese, J. (2004, 8). Application of multi-agent planning to the assignment problem. *Computers in Industry*, *54*(3), 273–290. Retrieved from http://dl.acm.org/citation.cfm?id=1017163.1017167 doi: 10.1016/j.compind.2003.11.002

Kouptsov, K. L. (2008). Production-rule complexity of recursive structures. In A. A. Minai & Y. Bar-Yam (Eds.), *Unifying themes in complex systems iv* (pp. 149–157). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://link.springer.com/10.1007/978-3-540-73849-7_17 doi: 10.1007/978-3-540-73849-7{_}17

Koza, J. R. (1992). *Genetic Progamming: On the Programming of Computers by Means of Natural Selection*. The MIT Press. Retrieved from https://mitpress.mit.edu/books/genetic-programming

Kristic, M., Kanellakopoulos, I., & Kokotovic, P. (1995). *Nonlinear and adaptive control design* (1st ed.). New York, NY, USA: John Wiley & Sons, Inc.

Laughlin, R. B., & Leggett, A. (2005). *A Different Universe: Reinventing Physics from the Bottom Down* (Vol. 58) (No. 10). Basic Books. Retrieved from http://scitation.aip.org/content/aip/magazine/physicstoday/article/58/10/10.1063/1.2138425%5Cnhttp://link.aip.org/link/PHTOAD/v58/i10/p77/s1&Agg=doi doi: 10.1063/1.2138425

Levi, P., & Kernbach, S. (2010). *Symbiotic Multi-Robot Organisms* (Vol. 7). Springer-Verlag Berlin Heidelberg. Retrieved from http://link.springer.com/10.1007/978-3-642-11692-6 doi: 10.1007/978-3-642-11692-6

Lungarella, M., Metta, G., Pfeifer, R., & Sandini, G. (2003). Developmental robotics: a survey. *Connection Science*, *15*(4), 151–190. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/09540090310001655110 doi: 10.1080/09540090310001655110

Markus, H., & Wurf, E. (1987). The dynamic self-soncept: A social psychological perspective. *Annual Review of Psychology*, *38*(1), 299–337. Retrieved from http://www.annualreviews.org/doi/abs/10.1146/annurev.ps.38.020187.001503 doi: 10.1146/annurev.ps.38.020187.001503

Maslow, A., & Lowery, R. (1968). *Toward a Psychology of Being*. Hoboken, NJ: Wiley. Retrieved from http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471293091.html

Mazzolai, B., Mondini, A., Corradi, P., Laschi, C., Mattoli, V., Sinibaldi, E., & Dario, P. (2011, 4). A miniaturized mechatronic system inspired by plant roots for soil exploration. *IEEE/ASME Transactions on Mechatronics*, *16*(2), 201–212. doi: 10.1109/TMECH.2009.2038997

McLean, K. C., Pasupathi, M., & Pals, J. L. (2007, 8). Selves creating stories creating Selves: A process model of self-development. *Personality and Social Psychology Review*, *11*(3), 262–278. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/18453464 doi: 10.1177/1088868307301034

Meng, Y., & Jin, Y. (2011). *Bio-Inspired Self-Organizing Robotic Systems* (Vol. 355; Y. Meng & Y. Jin, Eds.). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://www.springerlink.com/index/10.1007/978-3-642-20760-0 doi: 10.1007/978-3-642-20760-0

Min, B. C., Matson, E. T., & Jung, J. W. (2016, 5). Active Antenna Tracking System with Directional Antennas for Enhancing Wireless Communication Capabilities of a Networked Robotic System. *Journal of Field Robotics*, *33*(3), 391–406. Retrieved from http://doi.wiley.com/10.1002/rob.21602 doi: 10.1002/rob.21602

Narenda, K. S., & Annaswamy, A. M. (1989). *Stable Adaptive Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Nolfi, S., & Floreano, D. (2004). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. The MIT Press. Retrieved from https://mitpress.mit.edu/books/evolutionary-robotics

Novellino, A., D'Angelo, P., Cozzi, L., Chiappalone, M., Sanguineti, V., & Martinoia, S. (2007, 1). Connecting neurons to a mobile robot: An in vitro bidirectional neural interface. *Computational Intelligence and Neuroscience*, *2007*, 2. Retrieved from http://dx.doi.org/10.1155/2007/12725 doi: 10.1155/2007/12725

O'Connor, T. (1994). Emergent properties. *American Philosophical Quarterly*, *31*(2), 91–104. Retrieved from http://philpapers.org/rec/OCOEP doi: 10.2307/20014490

Oudeyer, P.-Y. (2004). Intelligent Adaptive Curiosity: a source of Self-Development. *Science*, *117*, 127–130. Retrieved from http://cogprints.org/4144/ doi: 10.1.1.58.3374

Poovaiah, B. W., & Reedy, a. S. N. (1995). *Plant Roots: The Hidden Half* (Vol. 56) (No. 4). CRC Press. Retrieved from http://books.google.com/books?hl=ja&lr=&id=C7daGqhy1N8C&pgis=1 doi: 10.1080/08854300308428351

Rasmussen, S., Chen, L., Deamer, D., Krakauer, D. C., Packard, N. H., Stadler, P. F., & Bedau, M. A. (2004, 2). Transitions from Nonliving to Living Matter. *Science*, *303*(5660), 963–965. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/14963315 doi: 10.1126/science.1093669

Ronald, E. M., & Sipper, M. (2001). Surprise versus unsurprise: Implications of emergence in robotics. *Robotics and Autonomous Systems*, *37*(1), 19–24. Retrieved from https://www.semanticscholar.org/paper/Surprise-versus-unsurprise-Implications-of-Ronald-Sipper/a98aa0f005ce43a15e8e9ae7305d270eb34b5248 doi: 10.1016/S0921-8890(01)00149-X

Ronald, E. M. A., & Sipper, M. (2000). Engineering, emergent engineering, and artificial life: Unsurprise, unsurprising surprise, and surprising surprise. *Artificial Life*, *7*, 523–528.

Ruiz-Mirazo, K., Umerez, J., & Moreno, A. (2008). Enabling conditions for 'open-ended evolution'. *Biology and Philosophy*, *23*(1), 67–85. Retrieved from http://dx.doi.org/10.1007/s10539-007-9076-8 doi: 10.1007/s10539-007-9076-8

Sahin, E. (2005). Swarm Robotics: From Source to Inspiration to Domains of Application. In *Special issue, autonomous robots* (Vol. 3342, pp. 10–20). Springer Berlin Heidelberg. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.161.6393%5Cnhttp://link.springer.com/10.1007/b105069 doi: 10.1007/b105069

SAP. (2005). *Adaptive Manufacturing - Enabling the Lean Six Sigma Enterprise.*

Schmickl, T., Szopek, M., Bodi, M., Hahshold, S., Radspieler, G., Thenius, R., ... Kernbach, O. (2013, 9). ASSISI: Charged hot bees Shakin' in the spotlight. In *International conference on self-adaptive and self-organizing systems, saso* (pp. 259–260). IEEE. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6676514 doi: 10.1109/SASO.2013.26

Seeley, T. D., Kirk Visscher, P., & Passino, K. M. (2006). Group decision making in honey bee swarms. *American Scientist*, *94*(3), 220–229. Retrieved from https://www.americanscientist.org/issues/id.993,y.2006,no.3,page.5/postComment.aspx doi: 10.1511/2006.3.220

Shen, W. M., Krivokon, M., Chiu, H., Everist, J., Rubenstein, M., & Venkatesh, J. (2006). Multimode locomotion via superbot robots. *Proceedings - IEEE International Conference on Robotics and Automation*, *2006*(2), 2552–2557. Retrieved from http://dx.doi.org/10.1007/s10514-006-6475-7 doi: 10.1109/ROBOT.2006.1642086

Silberstein, M. (2002). Reduction, Emergence and Explanation. In P. K. Machamer & M. Silberstein (Eds.), *The blackwell guide to the philosophy of science* (pp. 80–107). Cambridge: Blackwell. doi: 10.4324/9780203133972

Simes, A. L. F., Cruz, C., Ribeiro, R. a., Correia, L., Ampatzis, C., & Izzo, D. (2011). *Path Planning Strategies Inspired by Swarm Behaviour of Plant Root Apexes* (Vol. 31; Tech. Rep. No. 0). Technical Report 09/6401 of European Space Agency, Advanced Concepts Team. Ariadna Final Report.

Spector, L., Klein, J., & Feinstein, M. (2007). Division blocks and the open-ended evolution of development, form, and behavior. In *Gecco '07: Proceedings of the 9th annual conference on genetic and evolutionary computation* (pp. 316–323). New York, NY, USA: ACM. Retrieved from http://dx.doi.org/10.1145/1276958.1277019 doi: 10.1145/1276958.1277019

Standish, R. K. (2002, 10). Open-Ended Artificial Evolution. *International Journal of Computational Intelligence and Applications*, *3*(167). Retrieved from http://arxiv.org/abs/nlin/0210027 doi: 10.1142/S1469026803000914

Stepney, S., Polack, F. A. C., & Turner, H. R. (2006). Engineering emergence. In *11th ieee international conference on engineering of complex computer systems (iceccs'06)* (p. 9 pp.-). doi: 10.1109/ICECCS.2006.1690358

Suki, B., Bates, J. H., & Frey, U. (2011). Complexity and emergent phenomena. *Comprehensive Physiology*, *1*(2), 995–1029. doi: 10.1002/cphy.c100022

Szopek, M., Schmickl, T., Thenius, R., Radspieler, G., & Crailsheim, K. (2013, 1). Dynamics of Collective Decision Making of Honeybees in Complex Temperature Fields. *PLoS ONE*, *8*(10), e76250. Retrieved from http://dx.plos.org/10.1371/journal.pone.0076250 doi: 10.1371/journal.pone.0076250

Todd, T. W. (1934). *Emergent Evolution* (1st ed., Vol. 80) (No. 133). Springer Netherlands. Retrieved from http://www.springer.com/us/book/9780792316589 doi: 10.1126/science.80.2073.271

Tonazzini, A., Popova, L., Mattioli, F., & Mazzolai, B. (2012, 6). Analysis and characterization of a robotic probe inspired by the plant root apex. In *Proceedings of the ieee ras and embs international conference on biomedical robotics and biomechatronics* (pp. 1134–1139). doi: 10.1109/BioRob.2012.6290772

Vaario, J. (1994). Modeling adaptive self-organization. In *Proceedings of artificial life iv* (pp. 313–318).

von Neumann, J. (1966). Theory of Self-Reproducing Automata. *IEEE Transactions on Neural Networks*, *5*(1), 3–14.

Whitaker, H. P. (1959). An adaptive system for control of the dynamics performances of aircraft and spacecraft. *Inst. Aeronatical Sciences*, 59–100.

Wiles, J. (2013). *Why We Explore.* Retrieved from https://www.nasa.gov/exploration/whyweexplore/why_we_explore_main.html#.WDD7cOYrJPY

Wilson, D. I. (2001). *Adaptive control*. Courier Corporation.

Wimsatt, W. C. (1997). Aggregativity: Reductive Heuristics for Finding Emergence. *Philosophy of Science*, *64*(S1), S372. Retrieved from http://www.jstor.org/stable/188418 doi: 10.1086/392615

Wimsatt, W. C. (2000). Emergence as non-aggregativity and the biases of reductionisms. *Foundations of Science*, *5*(3), 269–297. doi: 10.1023/A:1011342202830

Winfield, A., & Griffiths, F. (2010). Towards the emergence of artificial culture in collective robotic systems. In P. Levi & S. Kernbach (Eds.), *Symbiotic multi-robot organisms* (Vol. Vol.7, pp. 425–433). Berlin ; Heidelberg: Springer-Verlag. Retrieved from http://wrap.warwick.ac.uk/48446/

Wooten, M. (2016). *Novel Vine-like Continuum Robot for Environmental Exploration Applications* (Doctoral dissertation, Clemson University). Retrieved from http://tigerprints.clemson.edu/all_theses/2406