8-2006

Distributed Self-Deployment in Visual Sensor Networks

Christopher Allan Beall
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Christopher Allan Beall entitled "Distributed Self-Deployment in Visual Sensor Networks." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Hairong Qi, Major Professor

We have read this thesis and recommend its acceptance:

Donald W. Bouldin, Itamar Elhanany

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

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

To the Graduate Council:

I am submitting herewith a thesis written by Christopher Allan Beall entitled "Distributed Self-Deployment in Visual Sensor Networks". I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Hairong Qi

Major Professor

We have read this thesis
and recommend its acceptance:

Donald W. Bouldin

Itamar Elhanany

Accepted for the Council:

Anne Mayhew

Vice Chancellor and
Dean of Graduate Studies

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

Distributed Self-Deployment in Visual Sensor Networks

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Christopher Allan Beall
August 2006

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Hairong Qi. Without her constant motivation and encouragement this work would not have been possible. I also thank her for the opportunity to work in the Advanced Imaging & Collaborative Information Processing lab, and for her patience when things got busy. I would also like to thank Dr. Bouldin and Dr. Elhanany for serving on my committee.

I would like to give special thanks to my parents, for their encouragement and support in every way possible. Without their support, achieving this milestone would have been a lot more difficult.

Thanks also go to Dr. Roger Parsons, Director of the Engage Engineering Fundamentals Program, for allowing me to work as a Teaching Assistant to fund my graduate studies. I also thank the Engage team for allowing me to use their resources and tools, to help with building the Mobile Sensor Platforms.

I would also like to thank all of the members of the AICIP lab. In no particular order, Raghul, Lidan, Hongtao, Ortal, Lu, and Cheng. Special thanks go to those of you who helped me with the construction of the Mobile Sensor Platforms.

This research was sponsored in part by UT SARIF award and NSF ECS-0449309 (CAREER).

Abstract

Autonomous decision making in a variety of wireless sensor networks, and also in visual sensor networks (VSNs), specifically, has become a highly researched field in recent years. There is a wide array of applications ranging from military operations to civilian environmental monitoring. To make VSNs highly useful in any type of setting, a number of fundamental problems must be solved, such as sensor node localization, self-deployment, target recognition, etc. This presents a plethora of challenges, as low cost, low energy consumption, and excellent scalability are desired.

This thesis describes the design and implementation of a distributed self-deployment method in wireless visual sensor networks. Algorithms are developed for the implementation of both centralized and distributed self-deployment schemes, given a set of randomly placed sensor nodes. In order to self-deploy these nodes, the fundamental problem of localization must first be solved. To this end, visual structured marker detection is utilized to obtain coordinate data in reference to artificial markers, which then is used to deduct the location of a node in an absolute coordinate system. Once localization is complete, the nodes in the VSN are deployed in either centralized or distributed fashion, to pre-defined target locations. As is usually the case, in centralized mode there is a single processing node which makes the vast majority of decisions, and since this one node has knowledge of all events in the VSN, it is able to make optimal decisions, at the expense of time and scalability. The distributed mode, however, offers increased performance in regard to time and scalability, but the final deployment result may be considered sub-optimal. Software is developed for both modes of operations, and a GUI is provided as an easy control interface, which also allows for visualization of the VSN progress in the testing environment.

The algorithms are tested on an actual testbed consisting of five custom-built Mobile Sensor Platforms (MSPs). The MSPs are configured to have a camera and an ultra-sonic range sensor. The visual marker detection uses the camera, and for obstacle avoidance during motion, the sonic ranger is used. Eight markers are placed in an area measuring 4×4 meters, which is surrounded by white background.

Both algorithms are evaluated for speed and accuracy. Experimental results show that localization using the visual markers has an accuracy of about 96% in ideal lighting conditions, and the proposed self-deployment algorithms perform as desired. The MSPs suffer from some physical design limitations, such as lacking wheel encoders for reliable movement in straight lines. Experiments show that over 1 meter of travel the MSPs deviate from the path by an average of 7.5 cm in a lateral direction. Finally, the time needed for each algorithm to complete is recorded, and it is found that centralized and distributed modes require an average of 34.3 and 28.6 seconds, respectively, effectively meaning that distributed self-deployment is approximately 16.5% faster than centralized deployment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Technological advances in the capabilities of computer systems and electronics, as well as the constant decline of the cost associated with purchasing such systems, have contributed to the emergence of several new areas of research within Computer Engineering. One of the most interesting new fields is that of mobile sensor networks (MSNs) and their applications. At the most basic level, mobile sensor networks are computers or microprocessors linked by a wireless network, where each node is able to take various measurements of (or sense) its environment. Information gathered from the sensors may then be evaluated locally at each node, or it could be distributed throughout the sensor network.

There are many different ways that information can be gathered and used, and this is what makes MSNs so interesting and versatile. In fact, MSNs hold great promise to revolutionize the way that humans think about using and interacting with computer systems. These intelligent networks can be used in military, as well as in civilian settings. The military is particularly interested in using MSNs in situations that could reduce the risk to human life. For example, mobile sensor networks could be used for reconnaissance operations in particularly dangerous areas that are not well known. Currently there are already unmanned aircraft and satellite imaging systems, but there are no robust systems to explore an area directly on the ground, without human involvement. Another military application that has drawn renewed attention is that of unmanned convoys guided by mobile sensor platforms. A large number of enemy attacks are staged during the transfer of supplies; these tactics are especially

devastating because not only do they disrupt the supply train crucial to the success of any military operation, but they also inflict large amounts of damage. If mobile sensor networks guided these transactions the human cost could be greatly reduced. Some of the civilian applications are very similar. There are ideas of computerized driving on highways, that is, turning each car into a mobile sensor node that is able to sense its surroundings and drive on its own, without human interaction. This idea seems particularly attractive for long-haul trips, and once perfected could greatly reduce the number of fatal car crashes due to driver fatigue.

Another highly appealing area is security surveillance and intrusion detection. Current methods address these needs by employing fixed sensors, most commonly cameras and motion detectors, at predetermined locations within a certain zone that is to be monitored. The limitations here are obvious; the system is unable to automatically adapt to changes of the configuration of the environment and may therefore miss important events. Mobile systems, on the other hand, would be able to detect an initial event, and then track it. For example, if an intruder enters an area that is being monitored by a cluster of nodes, the cluster may dynamically reconfigure itself to better track the intruder. It could then decide to follow the individual and may even interfere with the individual, if possible.

Each of the aforementioned applications bring with them their own unique set of problems and challenges. However, when it comes to mobile sensor systems there are a few basic issues that are common to all different types of configurations. These issues include problems such as energy consumption and conservation, system integrity and security, safety as it relates to humans who may be near these potentially dangerous systems, etc. Aside from these hardware related issues, there are a number of implementation issues that also need to be addressed. To set up an effective mobile sensor network we must also address important issues such as communication between nodes, data processing algorithms, and perhaps most importantly, navigation through the environment. For an autonomous mobile sensor platform to be able to effectively navigate through its environment, it must be equipped with the sensing devices necessary to capture or perceive its surroundings, so it can then calculate its estimated position. The process of estimating a sensor node's position within a certain area, based on a global or relative coordinate system is called *localization*.

Localization is an integral part of making MSNs useful, since without knowledge of the precise location of each mobile sensor node, all of the data gathered by each node may become irrelevant or useless. For example, continuing with the theoretical scenario given in the previous paragraph: If an intruder is detected by a mobile sensor node, but the node is unable to transmit its own location along with the intrusion alert, the alert may be rendered useless.

The next logical step in autonomous robot configurations is to implement self-deployment. This essentially means that a cluster of MSNs should be able to self-deploy to pre-defined or optimal target locations once localization is complete. The final locations of MSNs can be determined on the fly by the cluster or a cluster head, or the operator may decide where they should move to. Either way, the challenge is to properly deploy each MSN, which means that accurate localization is highly critical to success. Even small errors in localization estimates can lead to completely erroneous self-deployment results.

1.1 Motivation

Localization is a relatively new area of research that is getting more and more attention as the demand for mobile sensor networks in civilian and military applications increases. The lack of robust localization techniques prevents many of these systems from really achieving their full potential. To this end, projects such as the DARPA Grand challenge [7] have made great headway in improving localization methods. While some of these projects, specifically the DARPA grand challenge have finally been successful, one major remaining hurdle is the cost of systems that work properly in practice. Some of the successful systems include devices such as GPS, laser range finders, radar systems, stereo cameras, as well inertial measurement units. All of these are too bulky and too costly to be included on small mobile sensor systems. In addition, some of these only work outdoors and do not provide very good resolution. Therefore, due to all the previously mentioned points, alternative ways of estimating the location of a node must be developed.

This thesis develops a robust method for localizing and self-deploying a cluster of MSNs in an enclosed environment. As mentioned in the previous paragraph, many

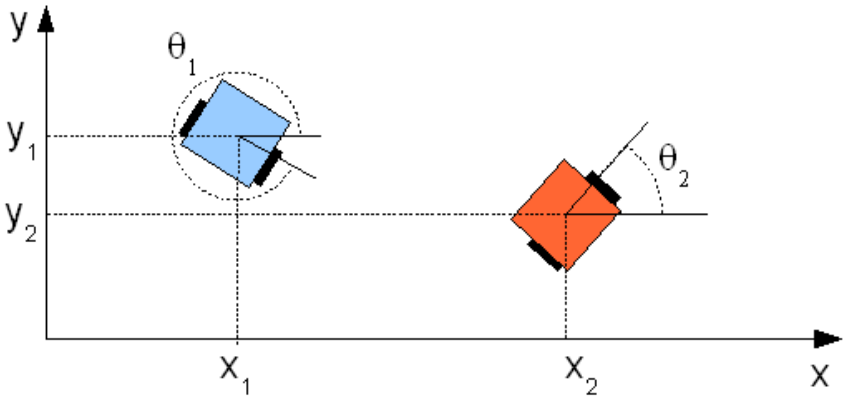


Figure 1.1: Pose of two MSPs in a global cartesian coordinate system

localization systems are too expensive to implement. Therefore, this thesis will explore ways to reduce this cost, and ultimately presents a method of localization that costs little more than printing a few sheets of paper.

1.2 An Introduction to Localization

Before we begin the discussion of different localization techniques we must explain some commonly used terminology.

- **Mobile Sensor Platform (MSP):** The "robots" used for this research will often be referred to as Mobile Sensor Platforms (or nodes) instead of simply Wireless Sensor Networks (WSNs) to accurately represent their function and purpose. This makes clear that the systems being discussed are not only wireless, but mobile. (They have motors and wheels).

- **Pose:** Pose refers to the position and the angle of each MSP within the global coordinate system, or with respect to other MSPs. We will use coordinates x_i and y_i to indicate the position of mobile sensor node i. θ_i will indicate the angle of the MSP with respect to the global coordinate system. Please consult Fig. 1.1 for further detail.

- **Relative Localization:** Relative localization refers to the localization of one mobile sensor node with respect to another or several other mobile sensor nodes. Relative localization gives no information about the node's location within a greater area, or the global coordinate system. This is the major drawback of this type of localization; knowing only the relative pose of each mobile sensor platform can make it very hard to navigate in an area. Depending on the type of application, knowing only relative coordinates may make data gathered by these nodes completely irrelevant. For example, if fire is reported by a sensor node, but the only localization information provided is that the node is fifty meters away from another node, the report is almost totally useless.

- **Absolute Localization:** The term absolute localization refers to localization done with respect to a global coordinate system. This method is preferred by far over the previously mentioned relative localization method. When a mobile sensor node's absolute position is known, we are informed precisely where in the area the node is located. To absolutely localize a mobile sensor node, means to know its absolute position. Knowing the absolute coordinates of an MSP is also preferred because it can always be used to derive relative position vectors between individual nodes.

There are many different ways of performing localization within a mobile sensor network. Each localization methodology has its own set of advantages and disadvantages. The type of sensors that are available for use on the MSP generally dictate the kind of localization methods that can be used. The environment in which the localization is performed also plays a great role in determining which localization method is ideally suited for that particular environment. For example, some localization procedures work only indoors, others work only outdoors. An optimal localization method would work in any type of environment, indoors or outdoors. Since it is so critical for MSNs to localize themselves, the next section discusses a variety of sensors that have been demonstrated to give satisfactory results for various types of localization applications.

1.3 Sensors used for Localization

There are many different types of sensors that can be used in WSNs, but when it comes to localization it is convenient to distinguish between two different classes of sensors according to their typical usage: proprioceptive and exteroceptive sensors. Proprioceptive sensors are ones that make measurements on the sensor node itself, that is, they take measurements internally, generally without relying on the environment. Since these types of sensors rely primarily on internal measurements they are generally only useful to estimate relative positions, or the angle components (pitch, roll and yaw) of the pose estimate. Exteroceptive sensors take measurements of the environment. In other words, they are externally oriented sensors. These are useful for target detection, obtaining distance information (relative or absolute), etc.

1.3.1 Proprioceptive Sensors

Wheel Encoders: Wheel encoders are used to measure the distance that a mobile sensor node has traveled. This is typically achieved by placing a type of optical odometer directly on one or several of the wheels of the mobile sensor unit. When a differential steering mechanism is employed by the MSP, placing wheel encoders on at least two wheels allows the platform to measure the degree of turning. Therefore it follows that these sensors make it very easy to calculate the distance traveled by the platform. The process of calculating an object's current position and orientation (pose) based on its previous location and its current course and speed, is called "dead reckoning". While this is a very effective method over the short range, the cumulative error when traveling for a long distance (especially with many turns) can be significant. Adding wheel slippage and poor surface contact to that, the resulting location estimate may be completely wrong. However, wheel encoders can also help to correct problems that may exist with the drive system itself. For example, if two motors in a differential steering system unintentionally run at different speeds given the same voltage, wheel encoders can be used to detect the anomaly in the rotational speed. The robot controller can then choose to correct this by issuing a command for the lagging wheel to speed up, until both wheel encoders report the

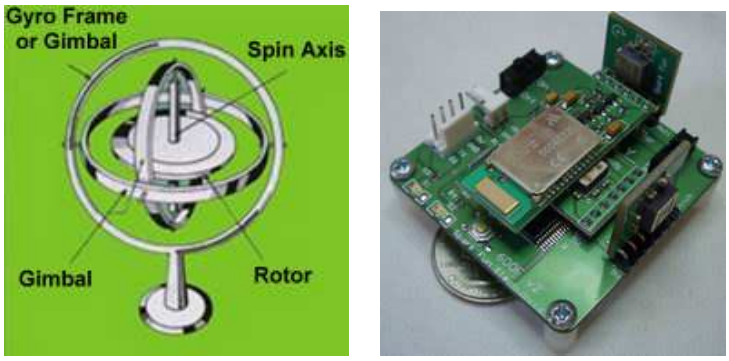
(a) Reflective encoder disk [17] (b) Custom cut slot encoder disk [3]

Figure 1.2: Two common types of wheel encoders

same rotational speed. Further, a reported speed of 0 may indicate that the wheel is stuck, or malfunctioning completely.

There are two common types of wheel encoders: Reflective disk encoders and slot encoders, examples are shown in Fig. 1.2(a), and Fig. 1.2(b) respectively. Reflective disk encoders use an IR-LED and IR-sensor to detect the presence of white or black color. The slot encoder uses an opto-coupler to detect whether a beam of light is broken or not. Counting the frequency of high and low light levels per time unit allows the sensor node to calculate precisely how fast the motor is moving.

Gyroscopes and Inertial Sensors: These devices come in all different shapes and sizes. The most basic devices are used to detect the angular orientation of an object, also called the *roll*, *pitch* and *yaw rates* [20]. A single gyroscope can measure the rotation around a single rotational axis. Therefore, to measure all possible directions of rotation, three gyroscopes are required. The general principle of operation of gyroscopes relies on conservation of angular momentum. Fig. 1.3(a) illustrates this concept. Gyroscopes are analogous in function to the part of the ear that provides humans with orientation information. Gyroscopes are relatively affordable and easy to use, so they can be incorporated into small mobile platforms. Inertial Measurement Units (IMU) consist of three gyroscopes and sometimes also compensating temperature probe. These complete units are considerably more expensive, and are



(a) Angular momentum tends to be conserved as the rotor keeps spinning in the same direction and the base may be rotated in a different direction [20]

(b) Commercially available gyroscope [8]

Figure 1.3: Gyroscope concept and example device

therefore not well suited for small MSP development. IMUs have played a large roll in autonomous vehicle challenges, such as the DARPA Grand challenge. [7]

Pressure Sensors: These sensors provide the MSP with atmospheric pressure readings. This type of sensor is useful for measuring altitude, which makes it particularly interesting for use on MSPs that are deployed from an airplane, or some other high altitude object.

1.3.2 Exteroceptive Sensors and Related Systems

Light Sensors: The most basic type of light sensor can be used to detect when an MSP gets close to a wall or some other object (as long as the object itself is not a light source, in which case the reading would be misleading). When used in sets of two or more, together with an LED, light sensors can easily be used as line tracking devices. This provides for very limited localization, and is usually not sufficient. This configuration can be used to track a line, but from the intensity reading alone it is impossible to distinguish where along the line the MSP might be located. Line tracking in coordination with wheel encoding can provide a much more

(a) A photoresistor [33] (b) IR sensor [5]

Figure 1.4: Light sensors

robust localization result. Fig. 1.4(a) shows a commonly available type of light sensor, also called a photo-resistor.

Infrared Sensors: These are just like light sensors, except that they are used to measure Infrared (IR). IR can be used to set up "beacons" with an encoded signal throughout an area, and an IR detector and decoder can be utilized to detect these beacons. Localization using this method can be difficult to set up, and is highly susceptible to interference from other IR sources, and multi-path issues. An example is shown in Fig. 1.4(b)

Visible Spectrum Cameras: These are probably one of the most common sensor types on MSP. We are often interested in transmitting digital images (still images or video), or we might be interested in performing some image processing on the captured frames to obtain some result which can be used to make some decision. There are literally hundreds of different models of cameras, with a wide range of prices and features.

Multi-Spectral Cameras: More advanced MSPs may use multi-spectral cameras for very specific applications, such as Automatic Target Recognition (ATR), target tracking, etc.

Range Sensors: The use of these types of sensors is obvious. However, as with all other varieties of sensors, there are multiple types, and each has its own advantages and disadvantages. The two cheapest kinds of sensors are IR-distance sensors and sonar range sensors. IR range sensors are good for short distance ranging. If longer distance is required, then sonar may be a better approach. Sonar rangers usually have a considerable field of view (up to about 60°). Depending on the type of the application this might be desirable or undesirable. Both of these sensor types are easy to control using a simple computer interface and standard programming language.

Compared to these very basic devices, laser range finders, on the other hand, are in a class of their own. These highly sophisticated range sensors are very expensive. In addition, they are also generally quite bulky, which makes them unsuitable for small sensor platforms. Laser rangers have been used by autonomous vehicle projects, such as DARPA [7]. This type of device can be very helpful for scene reconstruction and map building, as it is able to make precise measurements of the distance at each point in the field of view.

Global Positioning System: GPS systems rely on a satellite constellation of two dozen satellites at an altitude approximately 20,200km above the earth. The obvious advantage to implementing a localization system using GPS is the global availability. However, the normal GPS accuracy is only about 4-20 meters. Differential GPS (DGPS) can increase this accuracy substantially. DGPS makes use of local stationary GPS receivers that calculate the difference between their known position and the position received by their signal. This difference is then broadcast as an FM signal so that GPS receivers in the area can add this difference to their own position estimates. DGPS can achieve accuracy of 1-3 meters [4]. This level of accuracy is sufficient for most types of applications, but before we start putting DGPS systems on every sensor node we must consider several important drawbacks: First, GPS systems still come with a considerable price tag. This significant cost makes it impractical to put GPS on a large number of MSPs, given the assumption that the cost of each node ought to be relatively low. Secondly, even the most expensive GPS system will not work reliably indoors, as a clear view of the sky is required, and at least four of the satellites must be in view. Further, DGPS is not yet available in this area, so

only the coarser GPS data can be obtained [4]. This means that even near an open window the GPS system would not function properly. So, while GPS initially seems like the ideal system to address all our localization needs, a different method must be found.

1.4 Overview of Localization Techniques

There are many different ways to perform localization using the sensors described in Sec. 1.2. Some of these methods are more suitable for indoor localization problems than others. This section will give an overview of most previously studied localization techniques, and consider all of their advantages and disadvantages. Some of these methods are not at all feasible for small MSPs, but it is important to consider all possibilities when beginning the design of a brand new system. The most popular, and well studied methods, are discussed in this section.

1.4.1 Dead-Reckoning

This is one of the most straight-forward methods that can be used for localization in almost any type of MSNs. To determine the current position you take the previous position and add the distance traveled since that previous measurement. The distance that is being covered can be measured by a simple odometric device, such as the wheel encoders mentioned in Sec. 1.3.1.

Recall that wheel encoders measure the distance traveled by counting the number of pulses that are received (color transitions that are detected as the wheel rotates). To calculate the actual distance it is also necessary to know the size of the wheel (its radius r), and knowledge of how many lines (or slots) there are on the wheel encoder is required.

$$distance = (\text{pulses received}) \frac{\#lines}{\text{wheel encoder disk}} * 2\pi r \tag{1.1}$$

For example, suppose a MSP moves in a straight line and each wheel encoder communicates to the MSP controller that 90 pulses have been measured, the wheel encoder disk has 45 lines on it, and the circumference of a wheel is 30cm. The simple formula

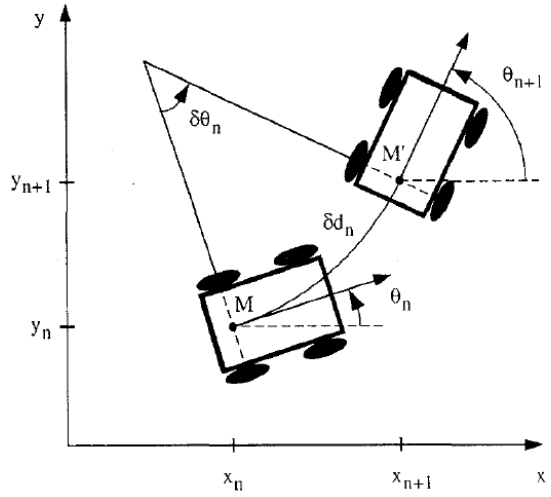


Figure 1.5: MSP moving along a circular path for dead-reckoning. The starting location is indicated as M, final position M' [32]

shown above then gives that the distance traveled is 60cm. This simple procedure makes calculating positions trivial, assuming that platforms travel strictly in the directions of the x or y axis. Of course this would rarely be the case in practice. Therefore, the equations to calculate the displacement caused by a movement along an angled or even a curved path, such as the one shown in Fig. 1.5, are given below [18, 32],

$$\delta d = \frac{\delta d_r + \delta d_l}{2} \tag{1.2}$$

$$\delta\theta = \frac{\delta d_r + \delta d_l}{B} \tag{1.3}$$

where δd_r and δd_l are the displacements of right and left wheels, respectively, and B is the distance between the two wheels. δd is the total displacement, and $\delta\theta$ is the total change in the angle of the MSP. The new x and y coordinates, and the angle θ after each move of the MSP is then given by

$$x_{n+1} = x_n + \delta d_n \cos\left(\theta_n + \frac{\delta\theta_n}{2}\right) \tag{1.4}$$

$$y_{n+1} = x_n + \delta d_n \sin\left(\theta_n + \frac{\delta\theta_n}{2}\right) \tag{1.5}$$

$$\theta_{n+1} = \theta_n + \delta\theta_n \tag{1.6}$$

It is absolutely critical to remember that these calculations are approximations! In effect, these equations assume that the MSP executes the pivot first, and then travels in a straight line. If this approximate calculation is applied to find the approximate location after moving for quite a distance, the error may be significant. One way to get around this is to partition the path for the calculation into several segments. Each of the calculations for these segments will be far more precise, and the overall result should also contain less error.

Dead-Reckoning has a few other important disadvantages besides the calculation problems discussed above [32]:

- Wheel slippage can have detrimental effects on the integrity of measurements made by the wheel encoder. Cumulative error through several consecutive steps can be significant as a result.
- The radii of the wheels have to be known, and they should be exactly the same for each wheel that has a wheel encoder attached. Differences in the radii can lead to measurement error.
- A starting point must be known. This makes dead-reckoning infeasible in randomly deployed sensor networks where none of the deployment locations are known *a priori*.

Various methods can be employed to remedy the growth of the cumulative heading error. Hardt et al. [32] study improving the dead-reckoning method by using a magnetic compass and gyroscopes to their robot in order to measure any accidental turning from the desired path. The magnetic compass can measure heading with good precision, as long as there is not a large amount of magnetic interference from electronics surrounding the MSP. To overcome this limitation, data from the gyroscope is fused with data from the compass to arrive at an even more accurate heading estimate. This has shown to eliminate the cumulative build-up of heading errors, and can greatly improve dead-reckoning accuracy overall.

1.4.2 Localization using Wireless Communication

There is a large body of research in the field of wireless localization. The aim is to exploit the fact that wireless communications equipment is already present in mobile sensor networks, and we just have to find ways to use this for our localization needs.

Hop Counting This localization technique uses the number of hops it takes to reach a node with a known location, to estimate the location of the node in question. This technique is particularly attractive, because no special equipment is required, given that the sensor network is already equipped with wireless communication capability. Hop Counting algorithms assume that the network would function in an ad-hoc way, so that each node can talk to every other node within its reach. A further assumption is that each node has relatively weak wireless transmitters (to conserve power), and therefore it is only able to reach a handful of nodes located within a certain radius close to it. Then, given that there are nodes with known locations strewn about the perimeter of the area, node locations can be estimated with ease. For example, suppose we have an area that is the size of a 10 by 10 grid, where each grid size represents the distance a wireless node can transmit on average. Now, suppose a node determines that it takes 5 hops to a node located at the top of the perimeter, and it also takes 5 hops to a node at the right side of the perimeter. It becomes clear that the node must be located approximately in the center of the 10 by 10 grid. It is important to note that these are just rough estimates, and no precise location can be assumed from those estimates. Further, no information about the heading of the MSP can be inferred. There are several ways of discovering the least amount of hops required to reach a perimeter node. A simple "Smallest Hop Count Discovery" algorithm may be as follows [16]:

1. Perimeter nodes will broadcast to their neighbors informing them that they are nodes with known location.

2. Each node, as it receives hop count information about the perimeter node, will increase the hop value by 1, and compare it against the previously known shortest hop distance to that perimeter node.

3. If this distance is shorter, or the distance was previously unknown, the new distance is retained and transmitted to its neighboring nodes.

4. This process continues until each node has a closest hop count to each perimeter node.

Since this localization methodology gives such a good map of nodes' locations in a network topology sense, it is very well suited for setting up optimal routing algorithms in wireless networks. The usefulness for physical location estimation may be limited.

Triangulation There is an entire class of localization algorithms that rely on triangulation for determining good localization estimates. Triangulation uses simple geometric relationships to calculate the position of a node. There are two ways that triangulation can be done [12]:

1. Lateration: This uses multiple distance measurements from markers or transmitters that have a known location.

2. Angulation: This uses an angle or bearing relative to points with a known separation between them.

We will first discuss several methods of measuring the distance between nodes that are used with lateration: Time of Arrival (TOA) and Received-Signal-Strength (RSS).

Time of Arrival Also called Time of Flight, this appears to be the most widely studied triangulation technique that is currently employed in wireless sensor networks. Just like the hop counting method, this particular technique is highly attractive because most WSNs are already set up to use it, that is, they have wireless transmitters and receivers. Time of Arrival distance estimation uses the time measured for an RF, or acoustic signal to travel from a (usually fixed) reference node to the node of interest.

The resulting TOA is given by the other node's transmission time added to the propagation delay. The delay for such a transmission is the distance between the two sensor nodes, divided by the velocity of the signal that is traveling between the two

nodes. The speed of RF signals is about 10^6 times greater than the speed of sound. That means it is far easier to measure propagation delays using sonar equipment, than it is using wireless signals. However, wireless TOA is so interesting because wireless is now a de facto standard in many educational and public buildings, so wireless TOA could use existing wireless infrastructure.

Wireless TOA measurements are plagued with many problems, such as multipath errors, signal loss, and noise and interference from other devices. For our TOA readings to be accurate we need to be able to overcome these problems. A good way of estimating the actual TOA under the presence of additive noise is to use the time that maximizes the cross-correlation between the received signals and the known transmitted signal [23]. We can estimate the minimum variance in the location estimate by first calculating the variance that may result from using a particular RF signal with the following equation [23]:

$$var(TOA) \geq \frac{1}{8\pi^2 B T_s F_c^2 SNR} \tag{1.7}$$

where B is the bandwidth in Hz, F_c is the center frequency, and T_s is the signal duration in seconds. Another important issue to consider is the actual measurement of the transmission time. It has been proposed to synchronize the clock on each node, and to use a pre-determined send time. The receiving node then simply takes the arrival time and subtracts the known send time from that. However, mobile sensor network clock synchronization and calibration algorithms have only achieved precisions on the order of $10\mu s$ [29]. This kind of clock skew between nodes would be acceptable for measuring the delay of acoustic signals, but this is hardly acceptable where $1\mu s$ equals about 1 foot (30cm) of distance traveled. To overcome this limitation we can instead measure round-trip times of TOA packets. This means that the time measured will be double the time it takes to travel one way, in addition to a small delay for the other node to issue the reply. This delay should be known beforehand, and then the distance traveled by the signal can be calculated easily. One disadvantage is that this technique gives no information at all about the heading of the mobile sensor node that we are trying to localize.

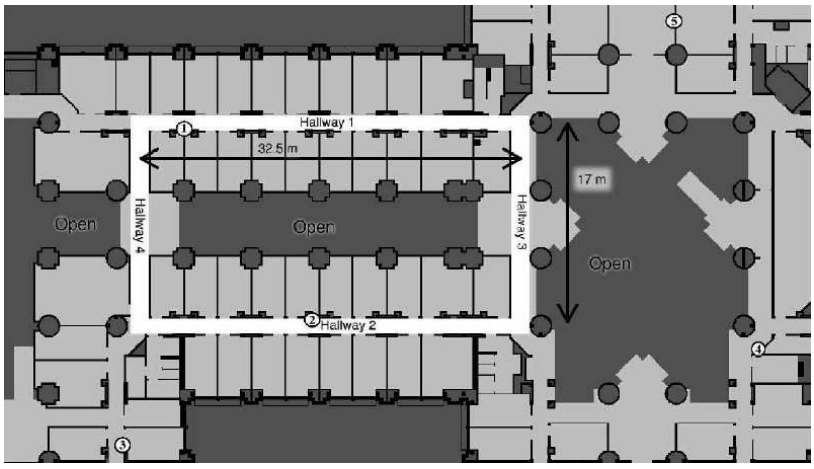


Figure 1.6: Wireless ethernet localization map. Localization was evaluated using a wireless laptop in Hallways 1 and 3 [15]

The most likely application of the TOA technique would be within 802.11b networks, operating at a frequency of 2.4Ghz. This frequency is license free around the world, so it is ideal to use for wireless sensor networks. However, this also means that there are some other electronics that can interfere with wireless communications on this frequency. For example, sources of probable interference can be 2.4 GHz phones, which are still being sold in stores around the world, microwave ovens, Bluetooth devices, and even welding equipment [15]. Ladd et al. [15] evaluated the feasibility of using Wireless Ethernet for localization, and found that they were able to get localization results within 1.5 meters of the true location 70% of the time, in an area that measures over 40×20 meters with only 5 poorly placed wireless access points. Fig. 1.6 shows a map of the testing environment, and Fig. 1.7 shows the tracking performance while walking up and down hallway 1.

This shows that rough estimates of location can be given very reliably. In fact, the FCC has already mandated that wireless phone providers develop a way to locate any phone that makes a 911 emergency call. This has spawned a variety of new startup companies which are developing various techniques to meet this requirement. Many of them rely on the basic TOA principles to do this.

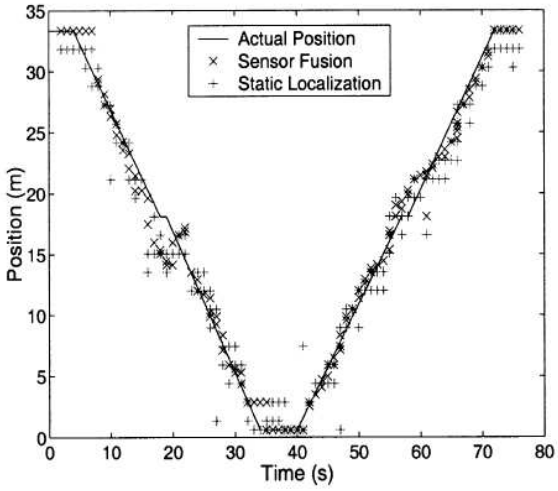


Figure 1.7: Wireless ethernet localization graph. Measured tracking performance while walking up and down hallway 1. [15]

Received Signal Strength The idea here is similar to that in TOA. Instead of measuring the delay in receiving a signal from a node, we measure the received signal strength. So, again the fixed node is asked to send a signal, and the localizing node measures the signal strength to determine the approximate distance. RSS is defined as the voltage measured by a receiver's received signal strength indicator (RSSI) circuit [23]. RSS is commonly also given as the measured power, that is, the squared magnitude of the signal strength as it is received. Just as with TOA, RSS can be used with acoustic as well as RF signals. As previously stated, acoustic signals are easier to measure because they travel far slower than RF signals. One significant advantage that RSS has over other techniques is that it can be measured during regular communication. This means that no "communication energy" is wasted on determining a node's location. While this method is also relatively easy to implement, there are many sources of error that must be taken into consideration. Multipath signals, as well as shadowing present major problems to accurately measuring the RSS [23]. Shadowing by obstructions in the environment can be unpredictable, which makes RSS most useful in open-space applications (outdoors). Moreover, since signal power decays at a rate of $\frac{1}{d^2}$, the possible error is multiplicative, whereas with TOA, it was only additive.

Another important factor to consider when setting up an RSS system is calibration of RSSI circuits. The transmitting power needs to be known, so that a perceived RSS value is correctly interpreted to mean a certain distance. This also means that RSSI circuits have to be perfectly calibrated between all nodes. If the WSN is highly heterogeneous this can quickly turn into a calibration nightmare. One possible solution to this problem would be to require each node to transmit their calibration information. Also, transmit power can diminish as batteries run out.

Angulation and Angle of Arrival This technique is more complicated than the TOA technique. Instead of measuring the time of flight distance to several nodes with fixed locations, we aim to measure the angle of arrival (AOA) at which the signal arrives. This can basically be done in two ways:

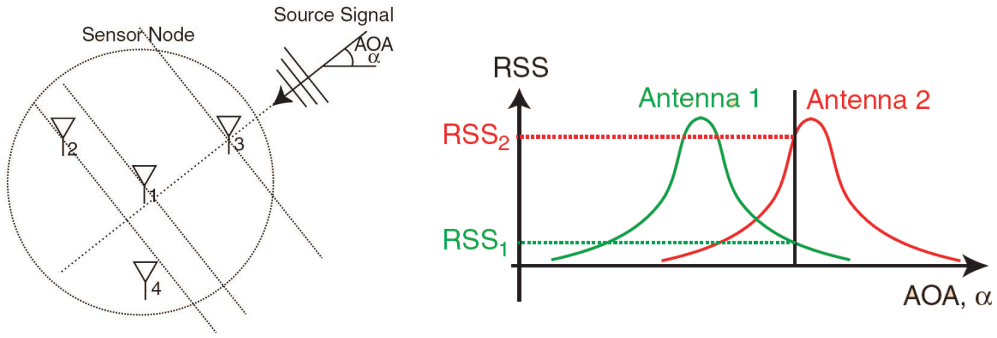
- The most popular method is to use a microphone or antenna array and to utilize array signal processing techniques to evaluate the received signals. Of course the individual sensor locations on the mobile sensor platform must be known to be able to perform these calculations. The AOA is estimated from the difference in arrival times for the received signals. Fig. 1.8(a) shows a Y-shaped sensor array [23].

- The alternative method uses RSS principles. Two (or more) directional antennas with overlapping main beams may be used to estimate the AOA based on the ratio of the RSS values received from each receiver's RSSI circuitry. Fig. 1.8(b) shows an example of using RSS ratios in estimating AOA [23].

Since both of these techniques require multiple sensors, the cost to implement this may be high. However, some sensor applications will require multiple microphones anyway, so acoustic localization using AOA may be done at no additional cost to the platform design.

1.4.3 Laser Range Scanning

Several research groups have studied the usefulness of using Laser Range Scanners for localization. Since these devices provide only range data, it is assumed that some kind



(a) AOA using TOA; a four-sensor Y-shaped configuration is shown

(b) AOA is estimated using the RSS ratio between RSS_1/RSS_2 between directional antennas on the same sensor node

Figure 1.8: The two main AOA estimation methods [23]

of range processing algorithm must be employed to perform the localization. SICK Laser range scanners, as the one shown in Fig. 1.9, are popular in mobile sensor platform applications. Several different methods have been studied:

- Sohn et al. [30] assume that the mobile sensor platform has knowledge of the global map. It will then attempt to use the scanner to build a local map of its current surroundings, and then match that to the global map to determine the global location. The authors report localization results that are 40-80% better than those that can be obtained by using conventional localization algorithms [30]. The steps are as follows:

  1. Line Segment Extraction: The laser scan data is used to find piecewise linear line segments that correspond to the wall locations surrounding the MSP. An efficient line segmentation algorithm is used
  2. Matching Line Segments: The line segments in the global coordinate system are compared to the line segments in the local coordinate system.
  3. Pose Estimation: This is done by minimizing a cost function. Line segments that contain more data points are given a higher weight, so a more accurate localization result is achieved.

- Laser Range Scanners can also be used for target recognition. By equipping several MSPs with laser range scanners, Ryde et al. were able to map out an

Figure 1.9: LMS 200-30106 laser range scanner. Manufactured by SICK [27]

area using cooperative localization techniques [25]. Shapes are needed so that each MSP can detect the presence of other MSPs. Since indoor environments contain many straight edges, a circular shape is chosen. Using this method the authors claim to have achieved 96% accuracy.

1. Once the MSPs have detected each other, mapping can begin.
2. After an area has been mapped, one of the MSPs will remain stationary, while the others move to a new location within the line of sight of the stationary node. This way the new location of the MSPs is instantly known.
3. This process repeats until the entire area is mapped.

The advantage of using this methodology is that the area is mapped as the nodes localize themselves. This can be of high importance in previously unknown areas, since this is where many of the other localization methods fall short (many other methods require some prior knowledge of the environment).

The major disadvantage that comes with any type of laser range scanning is the high cost of the equipment. Power consumption of the equipment must also be considered when evaluating the use of laser range scanners.

1.4.4 Visual Detection Systems

It is already known that there will be some kind of camera on the proposed MSP, so localization techniques that make use of this camera are very attractive, since there would be little or no added cost to implement these methods.

Since Image Processing, and to some extent Automatic Target Recognition (ATR) are relatively mature fields, there are many different approaches to detecting visual landmarks in real-time video streams taken by a camera onboard the MSP.

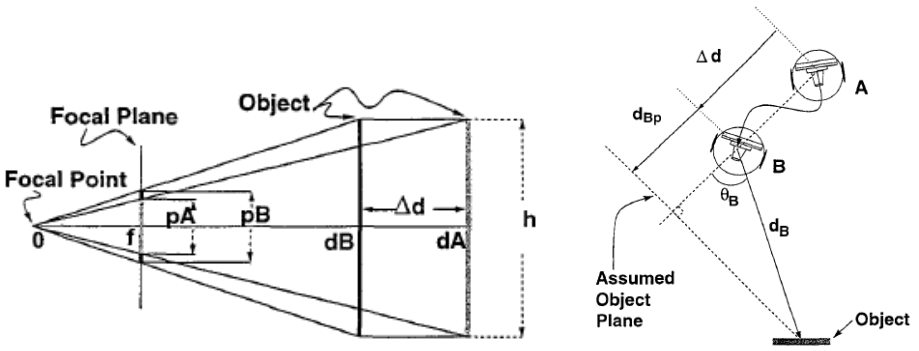
Arbitrary Landmark Detection

The most general way to perform localization based on visual image is to detect arbitrary landmarks in the field of vision. This means that no prior knowledge of the environment is needed. Several interesting landmark detection algorithms are presented below:

Sahin et al. [26] have developed an object localization module which segments the image into colored regions to be extracted as landmarks. The algorithm then is able to measure the pose in relation to one of those landmarks based on the change in size and eccentricity of the object. One of the major challenges that presents itself here is that of tracking an object through multiple frames, especially as its perceived shape may change significantly as the MSP moves around [26]. Once the object tracking problem is solved, we can apply different approaches to determining the actual location:

- **Visual Looming** This technique uses the change in retinal projection size of the object that is being watched. In Fig. 1.10, the camera is tracking an object of size h, from two distances, d_A and d_B. It does not matter whether the location of the object, or the location of the MSP is changed. With a constant focal length f, the size of the projection at the two distances will change. Given that p_A and p_B represent the size of the projections, respectively, then the distances d_A and d_B can be found using similar triangles [26],

$$d_A = -p_B \frac{\Delta d}{p_B - p_A} \tag{1.8}$$



(a) Focal point geometry

(b) Geometry of a moving sensor node

Figure 1.10: Visual looming geometry [26]

$$d_B = -p_A \frac{\Delta d}{p_B - p_A} \tag{1.9}$$

where Δd is the difference between d_A and d_B, and p_A and p_B are the widths of the field of view for objects A and B on the focal plane.

This technique only provides ranging information. Therefore, using this technique by itself is not sufficient for localization. Fig. 1.11 shows the operation of the visual looming algorithm.

- **Running Fix Triangulation** Eccentricity of the observed object is used to determine the angle from two different points of view with known separation. This is usually achieved by taking a picture of an object at one location, moving some distance around the object, and then taking another picture, as shown in Fig. 1.12.

  The distances can be calculated by

$$d_A = \Delta d \frac{\tan\theta_A}{\cos\theta_B(\tan\theta_A + \tan\theta_B)} \tag{1.10}$$

$$d_B = \Delta d \frac{\tan\theta_B}{\cos\theta_A(\tan\theta_A + \tan\theta_B)} \tag{1.11}$$

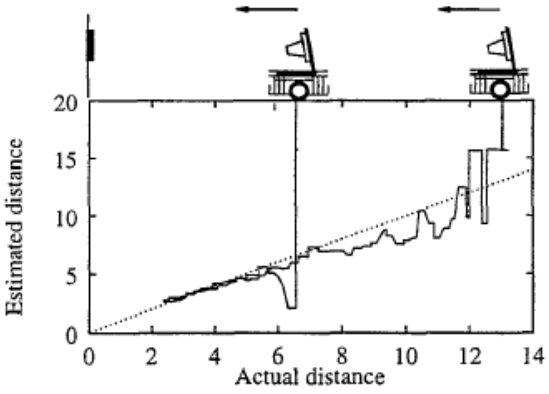


Figure 1.11: Visual looming setup and ranging result with two different starting distances [26]

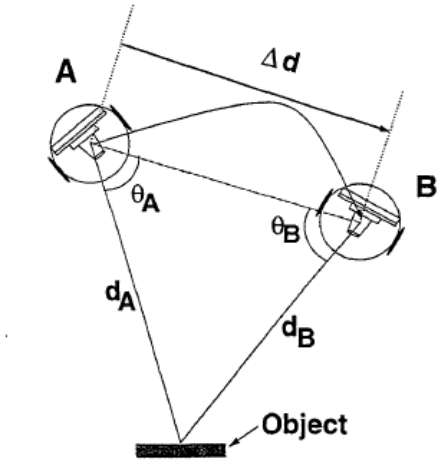


Figure 1.12: Eccentricity used with triangulation. It is here used to derive distances [26]

(a) Hallway with edges detected using Canny edge detector

(b) Extracted features

Figure 1.13: Lines and edges used as landmarks [6]

Another approach is to use lines and edges occurring in indoor environments as landmarks, and to track those. Edge detection in images is easy, so the challenging part is the tracking and labeling of different line segments. Dao et al. have demonstrated the feasibility of such a method in [6]. Fig. 1.13 shows how Canny edge detection was applied to extract features as their MSP moves down a hallway.

Structured Marker Detection

A large amount of research in visual marker detection is focused on *structured markers*, sometimes also referred to as *fiducial images* or just simply *fiducials*. Structured marker images are synthetic patterns that are well suited for pattern recognition. There are many such markers being used every day: merchandise bar codes at grocery stores, shipment tracking codes, etc. Fig. 1.13 shows a wide variety of different marker/barcode systems.

Since there are so many different systems, it may not be clear which marker system would be the most suitable for use with wireless sensor networks and localization. Several questions must be answered:

1. Which system achieves high accuracy rates in detection?
2. Which system has low processing time, suitable for low-power processors found in WSNs.
3. Which system is easy to implement in practice?

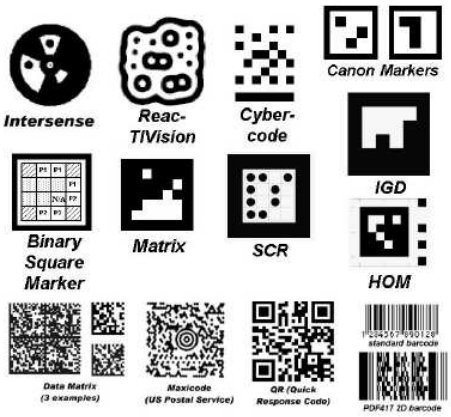


Figure 1.14: A variety of visual markers [9]

4. What is the cost of each system?

The most important question is the one of detection accuracy. There have been several papers that aim to answer that question. All of these come to the conclusion that the ideal fiducial image should be in black and white, and should have feature size large enough to be detected from a large distance. This eliminates all the conventional barcode systems shown in the last row of Fig. 1.14.

It is obvious that the marker image should be easy to segment from the captured image. This means that the fiducial should have a very high contrast in comparison to the background. To this end, all systems require that the marker be printed on a white sheet of paper in black ink. Furthermore, since these markers will be used for pose estimation, it must be relatively easy to calculate the position of the camera with respect to the marker.

Reliable pose estimation of the camera relative to the marker requires at least four non-linear points in the fiducial image and knowledge of their locations in the world coordinate system. However, it is possible to calculate pose from only three points, but the pose estimate may be ambiguous [21]. This requirement makes quadrilateral shapes ideal. The straight edges of a square shape can be used to calculate best-fit lines allowing for the corner locations to be calculated with very high accuracy. This in turn ensures that the pose estimate is of high quality.

Figure 1.15: Sample ARToolKit markers. [34]

There are several popular marker systems that consist of a black square on the outside, and some kind of marker image on the inside. Following is a discussion of four of these marker systems:

AR ToolKit The Augmented Reality ToolKit (ARToolKit) system is perhaps the most popular marker system because it is distributed under the GPL License for non commercial usage (other licensing options are also available) [2]. Augmented Reality (AR) software is primarily used to take images or video of real-world scenes, and to superimpose computer-rendered graphics on top of them. The ARToolKit uses a highly optimized detection algorithm, which makes it usable in real-time applications. Images captured by the camera are first thresholded, and then the contours are extracted. Next, sub-pixel corner detection is performed. This ensures that the most accurate estimate of edge locations is obtained. The marker decoding itself is based on a matching algorithm that compares just three of the geometry invariants with those that can be found in the pre-trained database. It is possible to create additional markers yourself, and to train the ARToolKit software to recognize them. When creating new markers it is important to pay attention to generate images which are non-similar, and which are not symmetric about the center. Such symmetry would make the pose estimation process ambiguous, and therefore such markers must be avoided. Two sample fiducials are shown in Fig. 1.15. The ARToolKit algorithm also calculates a "confidence coefficient" that is used to decide between several close matches to a pattern. First, the mean and standard deviation for the captured image and the pattern image are computed [21]. In the following equations x and y are coordinates in the images, where $I(x, y)$ is the image being inspected, and $P(x, y)$ is the marker image.

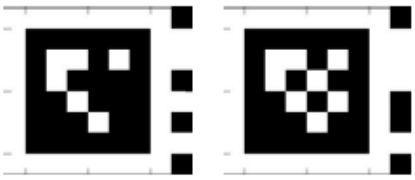


Figure 1.16: Sample HOM markers. [34]

$$\mu_I = \frac{1}{xy} \sum_x \sum_y I(x,y) \tag{1.12}$$

$$\mu_P = \frac{1}{xy} \sum_x \sum_y P(x,y) \tag{1.13}$$

$$\sigma_I = \left(\sum_x \sum_y (I(x,y) - \mu_I)^2 \right)^{1/2} \tag{1.14}$$

$$\sigma_P = \left(\sum_x \sum_y (P(x,y) - \mu_P)^2 \right)^{1/2} \tag{1.15}$$

Finally, the correlation coefficient can be calculated as

$$\rho = \frac{\sum_x \sum_y (I(x,y) - \mu_I)(P(x,y) - \mu_P)}{\sigma_I \sigma_P} \tag{1.16}$$

If the coefficient for a marker in the set is greater than a fixed threshold of 0.5, and it is the largest coefficient in the set, this marker is said to be a match.

HOM The HOM marker system has been created by Siemens AG, and is not freely available. To improve the detection accuracy of the marker, there is an additional 6bit strip to the side of the marker. While this system was originally designed to process only static images, a library to work with realtime video has been added in recent years. [34]. Fig. 1.16 displays two sample fiducials.

Figure 1.17: Sample IGD markers [34].

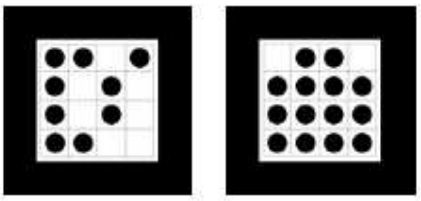


Figure 1.18: Sample SCR markers [34]

IGD The IGD marker system, as shown in Fig. 1.17, relies on a pattern in which the area is divided into 6×6 square tiles of equal size. The outer ring of tiles is always black to complete a square, just like the other marker systems. The inner area of 4×4 tiles is used for marker detection and orientation. The IGD system is only accessible to ARVIKA participants [28].

SCR The SCR system shown in Fig. 1.18 was also developed at Siemens Corporate Research (SCR) for localization and tracking in a variety of AR applications. This system also uses interior cells for marker detection, but they are filled with circles, instead of squares.

Zhang et al. [34] have conducted a quantitative comparison study to determine the processing times of each marker system, as is evidenced in Table 1.1.

The table shows that the ARToolKit performs the best with just one marker in the frame, but more time is needed with three markers, or even ten. However, in all cases the ARToolKit shows the best performance from a computational standpoint. Since the ARToolKit is the only software that is available under the GPL license, with source code provided, the remainder of this section will focus on previous experimental results evaluating its performance.

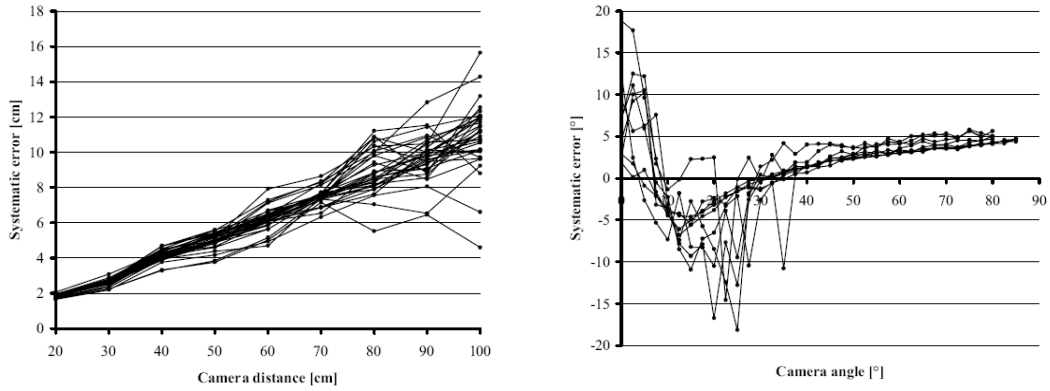
Table 1.1: Visual marker processing time in ms/frame [34]

Size	ROM/MPF	ATK	HOM	IGD	SCR
320×240	68x68/1	4.1	5.1	6.2	11.6
320×240	61x70/1	4.1	4.9	6.4	11.9
320×240	188x148/3	7.1	10.3	-	14.9
320×240	257x207/10	23.9	35.5	-	21.9

Abawi et al. [1] have studied the accuracy of the ARToolKit in detail. Since their paper was published in 2004, the current version of the ARToolKit may even show some slight improvements over these results. Their first experiment aimed to test the systematic error in calculating the distance from the marker. A camera was placed at a distance between 20 and 100 cm from the marker, in increments of 10 cm (9 variations). Furthermore, the angle was changed in steps of 2.5° (35 variations). This makes for a total of 315 different image capture locations. 250 frames were captured at each location, so that results in a total number of 78,750 individual data-sets. Based on these measurements, the systematic error for the distance is shown in Fig. 1.19(a). It is interesting to note that the systematic error starts at 2 cm, and almost only increases as the camera is moved farther away from the marker.

Next, the errors in the calculation of the angle are considered. In these experiments 0° is considered to be directly in front of the marker. Fig. 1.19(b) shows that there are considerable errors between zero and 25 degrees. This is attributed to the fact that in this position there is very little skew to the marker image, and it is difficult to correctly calculate the angle based on that small amount of skew. At greater angles the skew becomes larger, and therefore the angle is much easier to estimate.

Finally, a "map" is presented in Fig. 1.20 that plots the accuracy as a function of camera distance and relative camera angle. This shows that the best pose estimates are given close to the marker, and at angles between 30 and 70 degrees.



(a) Distance errors for different angles (b) Angle errors for different distances

Figure 1.19: Systematic measurement error when using ARToolKit [1]

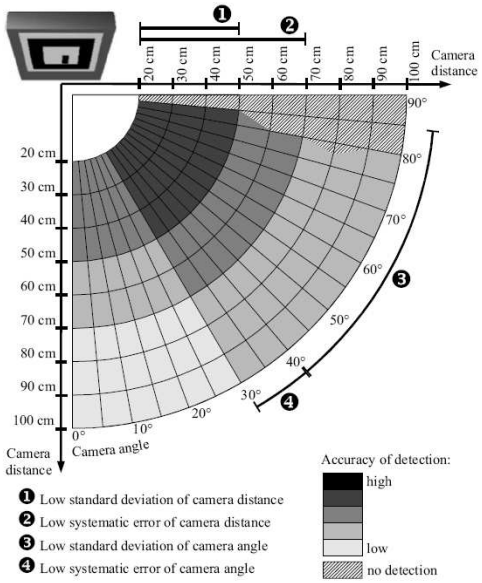


Figure 1.20: ARToolKit accuracy as a function of camera distance and relative camera angle [1]

1.5 Overview of Self-Deployment Techniques

There are many different ways to approach the problem of self-deployment in WSNs. However, there has not been quite as much research on this topic as on localization, and therefore numerous problems remain to be solved. The complexity of self-deployment methodologies depends largely on the assumptions that are made about the environment, and what kind of environment coverage is desired. Some scenarios require the placement of sensors along a particular route, perhaps a train track or a street, others require uniform placement in the region of interest (ROI). Unfortunately, many papers and research projects have evaluated deployment algorithms only using simulations, which may fail to consider some of the problems that may arise in real-world applications. These issues may include things such as:

- Energy constraints limiting the amount of movement or computation a node can perform before running out of battery power. This may not apply to solar powered MSPs.
- Many papers fail to consider the difficulty or cost of sensing the environment. It is usually not easy to determine what the environment looks like, or who the neighboring nodes may be. Further, even if such sensing capabilities are present, the equipment is likely to have significant power consumption needs.

Further, the type of self-deployment algorithm that is suitable in a particular situation depends heavily on the hardware that is being used on each MSP. One of the hardware challenges is to maximize the overall system lifetime with full coverage by balancing energy consumption of all nodes to be about equal [11]. This is particularly difficult in deployment schemes where some nodes may travel farther than others, so average moving distances should be minimized.

Heo et al. [11] propose a distributed deployment algorithm that introduces the concept of forces between sensor nodes. Nodes that are very close together would repel each other. Therefore, given a number of nodes placed into an ROI, nodes will repel each other until a uniform distribution is achieved. It is important to note here that there are no specific target locations, only the requirement that the sensors spread out evenly. The authors make the assumption that the locations of nodes can

easily be found using a GPS system or using iterative multilateration. Of course, as mentioned in Sec. 1.3.2, GPS only works outdoors. Howard et al. [13] propose a similar procedure, with the addition of also assigning forces, or so-called potentials, to obstacles. Therefore, nodes are repelled from each other and obstacles at the same time.

Parker et al. [22] propose deployment in heterogeneous sensor networks using "helper" robots. The challenge is that the most basic nodes do not have the capability to self-localize, or to avoid obstacles. Therefore, the helper robots, with more capable sensing and processing capabilities, visually detect the less powerful nodes, calculate their positions based on visible features in the environment, and then guide them to their target locations. This requires that line of sight detection of each node is always possible. There may be multiple "helper" robots to facilitate the efficient deployment of all sensor nodes. This method makes use of color coded cylinders mounted on top of sensor nodes, which allow unique identification, as well as distance and pose estimation. In [31], Parker et al. further demonstrate the viability of the proposed algorithms by providing results gathered from implementing them on physical robots.

Miao et al. [19] propose an interesting algorithm to self-deploy a cluster of heterogeneous sensor nodes based on biological principles. Heterogeneous sensors are to assume positions similar to the mosaic pattern in the retina, to maximize coverage, and minimize cost. The assumption is made that nodes can localize themselves using GPS, etc., and that there will be no collisions between individual nodes. Sensors are also aware of the boundary of the ROI. Each node carries a simple set of rules for random movement to converge to an overall mosaic pattern. Simulations show convergence to a state of almost uniform coverage is successful.

1.6 Development and Implementation Goals

The objective of this thesis is to realize the following goals:

1. Design and construct a testbed of five Mobile Sensor Platforms to be used for the following items.

2. Implementation of a localization technique by writing new software or by adapting one of the previously discussed solutions that is already available, such as the ARToolKit.

3. Develop algorithms for self-deployment.

For the testbed to be able to complete all these tasks, it is necessary that it meets or exceeds all of the following specifications:

- The cost of each node should be $500 or less.
- Each node must have the ability to take images or video, and carry any other sensing equipment that may aid in the localization of the MSP cluster.
- Each node should be equipped with motors and wheels sufficiently strong to move throughout an average sized room in a matter of seconds.
- Each node must have sufficient processing power to run some common image processing algorithms.
- Each node must have enough battery power to run for at least two hours with a moderate to heavy processing load.
- All nodes should operate in an untethered mode, but still be able to communicate with each other. This makes it necessary that some kind of wireless network be used.

A localization algorithm that works effectively with a small sensor network is to be implemented. The method that is chosen should be scalable so that it may easily be extended to a sensor network that is larger than having just five members.

Both centralized and distributed localization schemes should be implemented in software.

1.7 Performance Metrics

There are a few very straight-forward ways in which we can measure the performance of the localization algorithm:

- Is the initial location reported accurately, and what is the average error during localization. Do false marker detections occur?
- Is the system able to track the location accurately as the system moves around the area? Can this information be used for self-deployment?
- How fast are the self-deployment algorithms? What causes the differences, if there are any?
- How does centralized deployment compare to distributed deployment?
- Are MSPs able to properly execute the software's commands. In other words, does the hardware design meet its goals for functionality?

1.8 Thesis Contribution

This thesis discusses the design and implementation of the items set forth in Sec. 1.6. More specifically, the following contributions are made:

1. A cluster of five MSPs is designed and constructed. The choice of sensors and individual components is explained. This design makes a compromise between mobility and computing power, where previously existing designs typically fall into categories of high processing power and low mobility, or high mobility and low processing capability.
2. The implementation of a localization technique is realized, and validated by use with the MSPs.
3. Algorithms for centralized and distributed self-deployment are designed and implemented. While there are many existing papers on this topic, they usually simulate their findings, and rarely do they actually test their algorithms with actual MSPs. This thesis work implements the developed algorithms on the actual testbed that was designed.

1.9 Thesis Outline

The organization of this thesis is as follows:

This chapter gave an introduction to localization and self-deployment techniques. A literature review is provided to discuss the different methods available, and to evaluate the merits of each. Implementation and design goals are set forth and explained.

Chapter two discusses the design and construction of the MSP testbed. Every aspect of the design is discussed, ranging from the selection of parts, through part placement and wiring of the drive system. Software and drivers are also explained. Improvements are made in this design based on the previous version of the MSP.

In Chapter three the most important sections can be found. These are the development of centralized and distributed self-deployment algorithms. The algorithms are designed to work with the MSP testbed, and the specific software implementation is discussed. The various functional blocks that comprise the software package for centralized and distributed self-deployment are given.

The experimental setup and results are presented in Chapter four. The qualities of the MSP design are discussed, and experimental test results obtained by running the algorithms are presented. The two different deployment techniques are compared.

Finally, Chapter five completes the thesis with a conclusion and recommendations for future work. Improvements for the MSP design and software algorithms are provided.

Chapter 2

Mobile Sensor Platform Design

As mentioned in Sec. 1.6, one of the objectives of this thesis is to construct a mobile sensor platform testbed consisting of five nodes. These five nodes are then used to implement a localization algorithm and sensor node distribution in an indoor environment. The MSP should also be able to apply image processing algorithms on images taken by the platform itself. The wireless network in the indoor environment can be utilized to facilitate communication between the nodes and a controller workstation.

2.1 Assumptions

Before the platform design can be started, a few assumptions must be made about how the mobile platform will be used. These constraints are used to narrow down the design requirements, and to guide the purchase of the parts required to assemble the MSP. The Mobile Sensor Platform:

- will be designed for indoor use. Outdoor use may be possible as well, but it is not a primary design consideration;
- must be able to move around in its environment at an acceptable speed of about a foot per second. This means that motors will be required, and it is also assumed that the flooring will be smooth, made of materials such as tile, tightly woven carpet, vinyl, etc. Further, the MSPs will not have to deal with

other dangerous conditions, such as as uneven surfaces, stairs and drop-offs, or other moving objects (with the exception of other MSPs);

- will be required to carry out all actions autonomously. This means that the testbed must have processing capabilities that are sufficient to poll the sensors and make decisions individually;
- must be able to recognize and avoid obstacles in its path. Collisions should be avoided at all cost. The appropriate sensors should be used to enable obstacle avoidance;
- should cost approximately $500 per node, or less. It is desirable that future MSP platforms would be cheaper, and be more powerful. Technological advances and falling prices of hardware make this possible.

2.2 Initial Design Considerations

Before the design of the platform is started, so-called robotic "kit" solutions should also be looked at. While some of the systems available met all of the outlined requirements, they tend to be far more expensive than anything custom built. It was therefore decided to build the new MSP platforms from scratch using individual parts. The Advanced Imaging & Collaborative Information Processing (AICIP) [24] lab has already built several mobile sensor platforms for target classification purposes, so valuable lessons learned from those projects should be be considered in creating a new design. Before the actual design process is started, shortcomings of the old platform are discussed. The previous three designs are shown in Fig. 2.1. There were issues with various components of the system:

- The MSP was equipped with a Mini-ITX motherboard, powered by a small DC-DC power supply. This power supply was meant to snap directly onto the motherboard itself, for a space saving design. Unfortunately, the Power Supply Unit (PSU) did not fit, and it was therefore necessary to mount it separately. Thus, an ATX power-supply cable was run from the PSU to the motherboard.

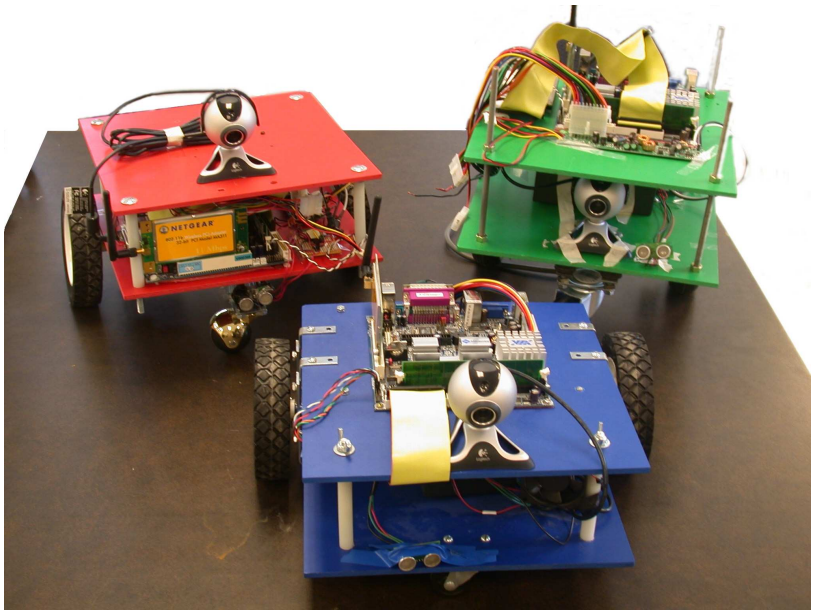
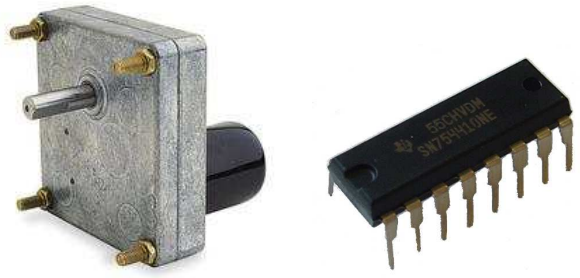


Figure 2.1: Mobile sensor platform version 1

The use of this cable consumed a large amount of space and was therefore not desirable for the more space-efficient new MSP.

- The second issue also has to do with the power supply. The motors, as well as the motor controller chip, were connected directly to and driven by the computer PSU. This heavy load, in addition to some accidental abuse caused several of the PSUs to burn out. Since these power supplies are costly to replace, this problem must also be rectified.

- All three of the previous platforms used a typical differential steering mechanism for moving through the environment. This method featured a primitive non-motorized caster as a third wheel. Depending on the prior orientation of this caster, it would sometimes resist rotating in a new direction, thereby throwing off the path of the platform. This was especially problematic because the robots were not equipped with any wheel-encoders, and therefore were unaware of this condition when it occurred. Three different types of casters exhibited the undesirable behavior described above; thus, a new third-wheel solution should be found.



(a) Dayton gearhead motor [10]

(b) H Bridge IC

Figure 2.2: Drive system components

- Finally, the batteries used as power sources for the first generation of MSPs were quite bulky; this hindered the mobility of the platforms as well as requiring more current to be provided to the motors to overcome the additional mass. It is desired to reduce the physical size and mass of the MSP power source to improve mobility and reduce motor load.

2.3 Hardware Design

Since this is the second version of the Mobile Sensor Platform for the AICIP lab, the first platform will guide the design of this project, taking into consideration the problems outlined in Sec. 2.2. This section will discuss each component individually and explain why it was selected.

2.3.1 Drive System

The previous drive system consisted of two Dayton gearhead motors, shown in Fig. 2.2(a), used in a differential steering configuration. This arrangement performed very well; thus, it was chosen as the basis of the new drive system, as well. The motors are 12VDC, but also work at lower voltages. The motors are driven by a dual H-Bridge chip, the SN754410 (Fig. 2.2(b)). The IC requires a 5V logic power supply, and is able to drive two motors at up to 36V with a continuous current of up to 1A, and

Figure 2.3: Omni-wheel

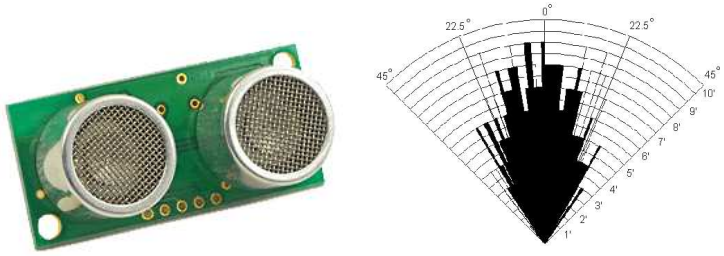
Figure 2.4: Logitech QuickCam 4000 Pro

a peak current of 2A for very short times. Pulse width modulation (PWM) can be used to run the motors at different speeds.

As stated in Sec. 2.2, the casters used as third wheels on previous designs prevented the platform from turning properly and consistently. The problem worsened when a larger caster was implemented. This malfunction is solved on the new sensor platform design by using an innovative omni-directional wheel, as shown in Fig. 2.3. This type of wheel is composed of eight smaller rollers. The unique construction allows the omni wheel to roll naturally by rotating about its main axis while also gliding smoothly in a lateral direction.

2.3.2 Sensing

The imaging sensor of choice remains the Logitech QuickCam 4000 Pro, shown in Fig. 2.4, as the newer 5000 series model is not yet supported by any Linux drivers. The driver for the QuickCam 4000 Pro is now maintained by a different individual than during the construction of MSP version 1, and kernel support has improved greatly. This should result in better performance and integration when writing software for the



(a) Devantech SRF04 (b) Beam pattern

Figure 2.5: Ultrasonic range sensor

camera. The camera is able to take pictures at a resolution of up to 1.3 megapixels, with a default setting of 640×480 pixels, and interfaces with any computer via USB.

There is also no change in the range sensor that is used. The Devantech SRF04 Ultrasonic Ranger (Fig. 2.5(a)) is easily controlled via the parallel port. The only problem with this device is the small size and difficulty in soldering the connections. The range sensor works by transmitting a pulse of sound outside the range of human hearing. The sound wave is reflected back to the ranger by any object in its path. The distance of the object is then obtained by measuring the time it takes for the sonic wave to bounce back to the sensor. The range sensor has an approximate beam pattern that is 60 in width (Fig. 2.5(b)), and less than 10 feet deep.

2.3.3 Computation and Storage

The motherboard used on the MSP version 1 was the VIA EPIA ME-6000 running at 600Mhz. The new motherboard is the VIA EPIA M10000 running at 1Ghz, shown in Fig. 2.6. The board is only 7×7 in and features one memory expansion slot, integrated audio, video, and ethernet, and one PCI expansion slot. This kind of motherboard is excellent for this type of application because it is powerful, yet very compact. Due to the faster processor and improved system architecture, significant improvements can be expected in execution time of computationally intensive algorithms. This may make it possible to run some image processing algorithms in realtime, where this was not possible using the slower motherboard. It is reasonable to expect that this new

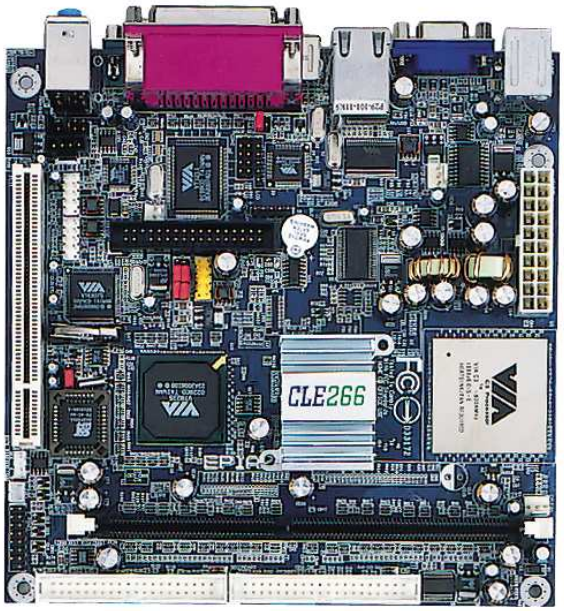


Figure 2.6: VIA EPIA-M10000

board will also reduce the runtime of the battery, since a faster processor tends to consume more power.

The motherboard uses DDR266 memory. The old platform was outfitted with 256MB; for the new platform it was possible to obtain 512MB for about the same price. Doubling memory should reduce use of the swap file, and therefore also significantly increase performance.

There is not a great need for storage on these Mobile Sensor Platforms. Therefore, a cheap 20GB hard drive is used just as before.

2.3.4 Communication

It is typical for wireless sensor networks to have slow communication links due to energy and cost constraints. For this reason, the MSP will keep using the older 802.11b wireless cards that were used previously, the Netgear MA311. This card is easy to use, and works well with the wireless network in the testing environment. This wireless device is natively supported in Linux by the PRISM driver. Other cards have

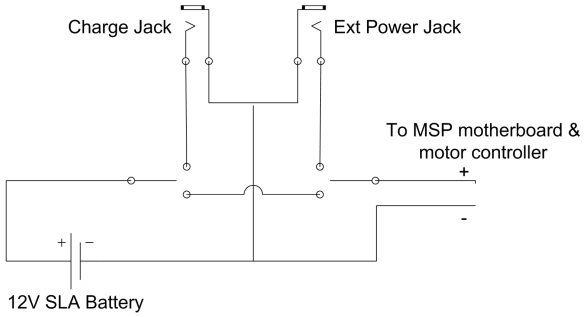


Figure 2.7: Power distribution circuit

been tested using the ndiswrapper utility for windows drivers, but this resulted in a significant performance decrease.

2.3.5 Power

The old system used a power supply that had a limited range of input voltages (11-14V). To make the new system more resistant against voltage fluctuations and accidental damage, a power supply with an 11-30V input range was selected. The new power supply also attaches directly onto the motherboard, which saves space, and is more aesthetically pleasing.

Both versions of the system use 12V sealed lead acid (SLA) batteries. The advantage of using these batteries is that they are much cheaper than other types of batteries. Unfortunately they are very heavy and make up about half of the weight of the platform. The new MSP design is going to have a few added features for power distribution. Two SPDT (Single Pole Dual Throw) switches are used to make it possible to switch between powering the MSP from the battery, or an external power source, and to either use the internal battery, or to charge it. A circuit schematic is shown in Fig. 2.7. It is also possible to charge the battery while the MSP is being powered from an external source. Having these switches also means that the battery can be disconnected without actually removing it, which is useful because the system consumes power even if it is not booted up. Further, the ability to now charge the battery without removing it from the MSP chassis is a major improvement; the installation and removal of the battery in the previous iteration of MSPs was the

primary source of accidental damage to the platforms. The potential for accidental damage to the platforms is further reduced by utilizing different sized connectors for the external power and charging jacks; this eliminates the possibility of an accidental polarity reversal.

2.3.6 Mounting Platform

The first version MSP was built using 12×12 inch PVC sheets which are easily machinable. Since this material is filled with tiny air bubbles it is also very light. Its small amount of flexibility allows easy assembly of all parts even if something does not fit perfectly. Because this material is easy to work with it is also used on the new MSP. Various angle brackets are needed to attach everything to the PVC sheets.

2.4 Hardware Installation

Since a compact design is highly desirable, various configurations were considered to achieve the smallest possible layout. Of the three previous designs, the blue robot was the most compact. It was the only one that had the motors attached in between the two PVC pieces, whereas the other two designs mounted these below the bottom platform, causing the MSP to be much taller. This compact design is used as a starting point for the new MSP configuration.

2.4.1 Chassis Assembly

In version 1 the hard drive was mounted essentially between the wheels in the back of the platform, with the battery installed from the front. This required two metal rods as support near the front of the platform. In the new layout, the hard drive is moved to the front of the MSP, and acts as the support for the upper platform in place of the rods, saving both weight and space. This means that the upper platform is held entirely by the motor mounts and the hard drive. Furthermore, the battery will now be located directly in between the motors, which results in better weight distribution overall.

Figure 2.8: Mounted omni-wheel

Next, two angle brackets are used to mount the omni-directional wheel in a slot at the front of the base platform, as shown in Fig. 2.8. Since the battery is placed in between the motors in the back, there will be less weight bearing down on this third wheel, and therefore it should rotate and glide with ease on almost any type of surface. In addition, because this setup is very low to the ground, it should make for a very stable platform base.

Previously the driving wheels were attached to the gearhead motor shaft by drilling a hole through the shaft, or by gluing the wheel directly to it. The wheels come with a small plastic sleeve inserted in them that holds a manufacturer's label in place. It turns out that this plastic sleeve is easily glued onto the motor shaft (Fig. 2.9), and the wheel fits quite snugly onto that sleeve.

2.4.2 Motherboard and Sensor Placement

Good placement of motherboard and sensor components is critical to building an effective and compact MSP. It is obvious that the sonar ranger and the camera should be placed near the front. The camera comes with a small stand that is easily glued to the PVC. To place the range sensor, a notch is made directly in front of the camera. The notch will hold the sensor securely in place, whereas on the first platform it was merely taped to the chassis. The first MSP design then placed the motherboard such that the ports were facing the back of the platform, causing the connecting wires to

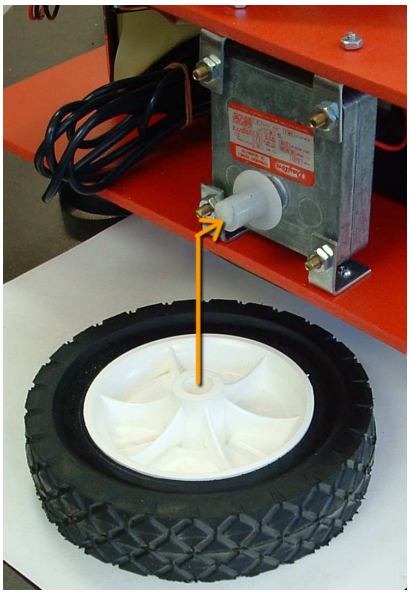


Figure 2.9: Wheel sleeve mounted on motor shaft

be routed over the edge of the PVC down to the lower deck. Not only did this look bad, but it made the MSP more fragile, and therefore handling of the MSP more difficult. To remedy this situation, the new design takes advantage of the fact that the PVC is very easy to cut. The mainboard is placed so that the parallel port and the IDE connector are facing the right and the left side of the chassis respectively. Wires coming from these ports are routed through holes in the PVC directly to the lower platform.

2.4.3 Motor Driver and Circuitry

As discussed in section Sec. 2.3.1, the MSP utilizes an H-Bridge to control the motors. This IC is able to drive two motors at up to 1A each. This 16pin DIP is mounted on a small PC board, so connections are made easily. Input signals to the IC come from a parallel port connection, and the output signals run directly to the motors. Because the IC has internal clamp diodes and current limiters, no discrete components are required for safe operation without causing short circuits or overheating the chip. The H-Bridge has two inputs for each motor, for a total of four data inputs. To cause a motor to move, one of the motor's inputs must be held high, the other low.

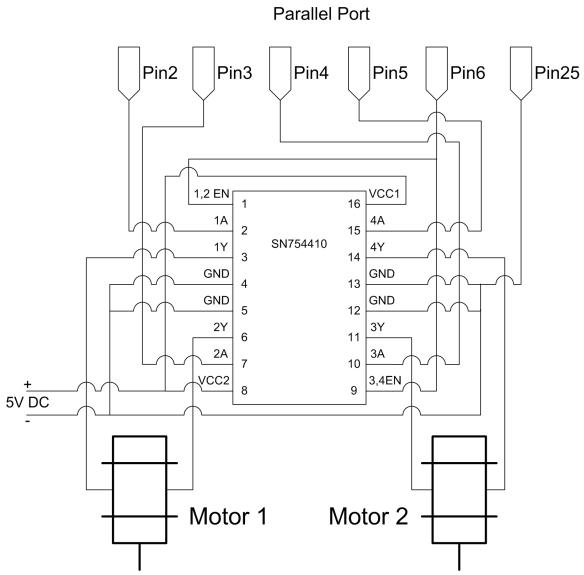
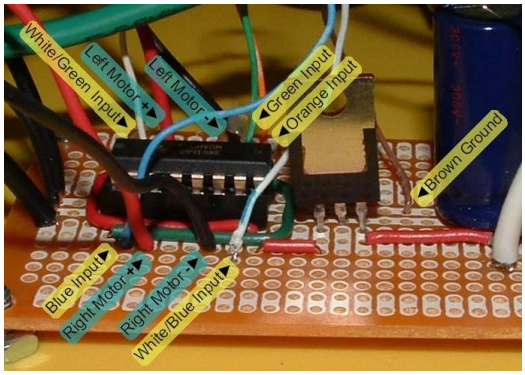


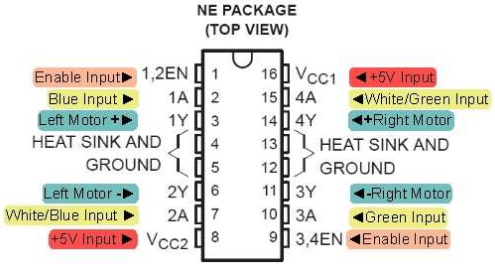
Figure 2.10: H-Bridge circuit

Reversing the polarity reverses the direction. The software must supply the proper high/low levels to move the motors in the desired direction. In addition, PWM is used to drive the motor at speeds less than 100%. Parallel port pins two through five are used to send these signals to the H-Bridge. The chip also has an enable/disable pin for each motor driver, which was permanently tied to 5V on the previous MSP design. The new setup adds an additional signal wire from the parallel port (pin 6) to these enable/disable pins, so that the H-Bridge IC can be turned on/off via the software. This can be used to conserve power when the MSP will not be moving for a while. The H-Bridge has four ground terminals that also serve as a heatsink. All four of these pins are soldered together to facilitate heat dissipation. The ground pins are also tied to a ground pin 25 on the parallel port. A basic schematic showing how the H-Bridge is connected to the parallel port is shown in Fig. 2.10, and a picture of the actual implementation is given in Fig. 2.11.

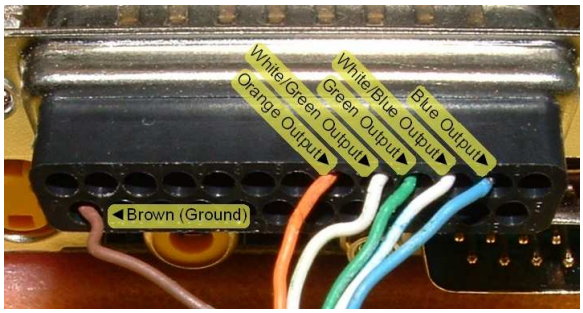
The IC has a recommended logic supply voltage of 5V, so a voltage regulator is used to convert the 12V supplied by the battery. As mentioned previously, hooking the IC directly to the motherboard power supply is undesirable, because it may draw too much current, which in turn may damage the power supply. Furthermore, there



(a) H-Bridge Mounted on a small circuit board, next to the voltage regulator and capacitor. Input signals(yellow) and motor outputs(cyan) are labeled



(b) H-Bridge pin assignments. The four ground pins also serve as heatsink



(c) Parallel Port pinout

Figure 2.11: H-Bridge signal and motor connections

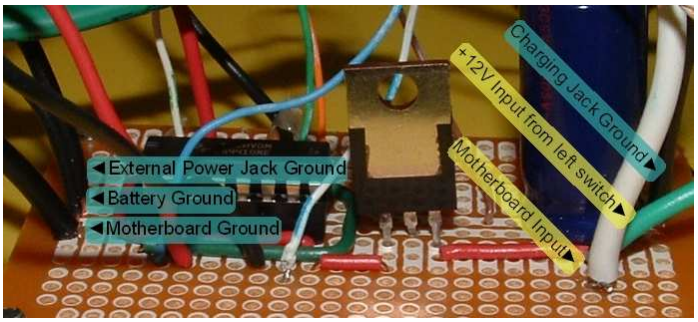


Figure 2.12: Power connections

is a separate pin for the motor power supply. However, since the motors run a bit too fast at 12V, they will also be connected to the 5V supply from the voltage regulator. Each motor can draw in excess of 500mA at full speed, so it is necessary to use at least two voltage regulators, each having a maximum current rating of up to 1A. Two voltage regulators hooked up in parallel are able to provide up to 2A of current to the H-Bridge and the motors, which should be adequate.

In Sec. 2.3.5 it was also explained that two switches are used so that the system can be powered either from the internal battery or from an external power source. To make this feature even more useful a 1000μF capacitor is added which allows switching between the two modes even while the system is running. The capacitor and the voltage regulators are both mounted on the PC board with the H-Bridge IC, as shown in Fig. 2.12. The PC board is also where all ground and 12V wires are connected. The trace running along the center of the PC board is ground, so all components have convenient access to it. The ground wires for the external power jack, the battery, and the motherboard are all attached on the left, and the ground wire for the charging jack is soldered to the right side.

This completes the hardware design discussion for the MSP. A picture of one of the completed sensor platforms can be found in Fig. 2.13, and a complete circuit schematic is shown in Fig. 2.14.

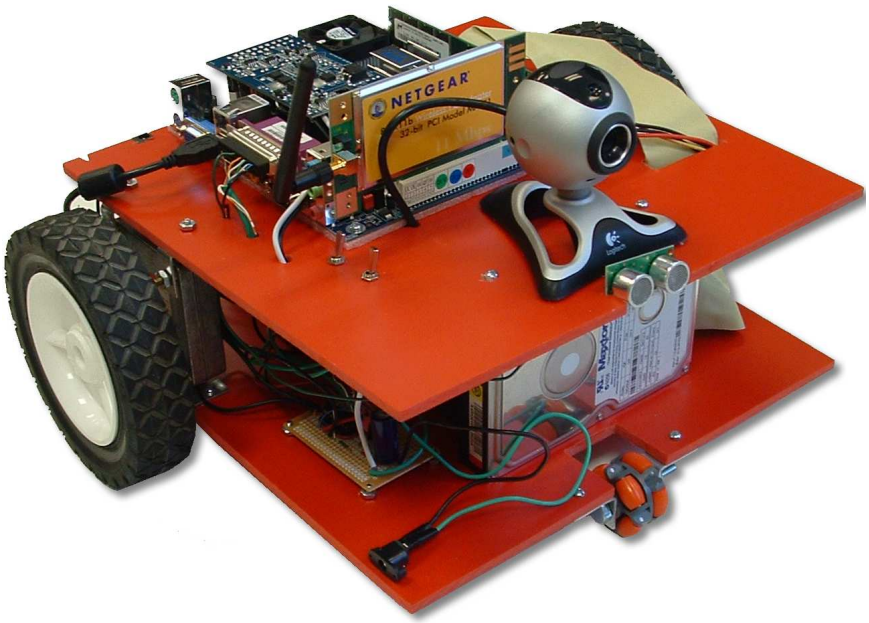


Figure 2.13: Fully assembled MSP

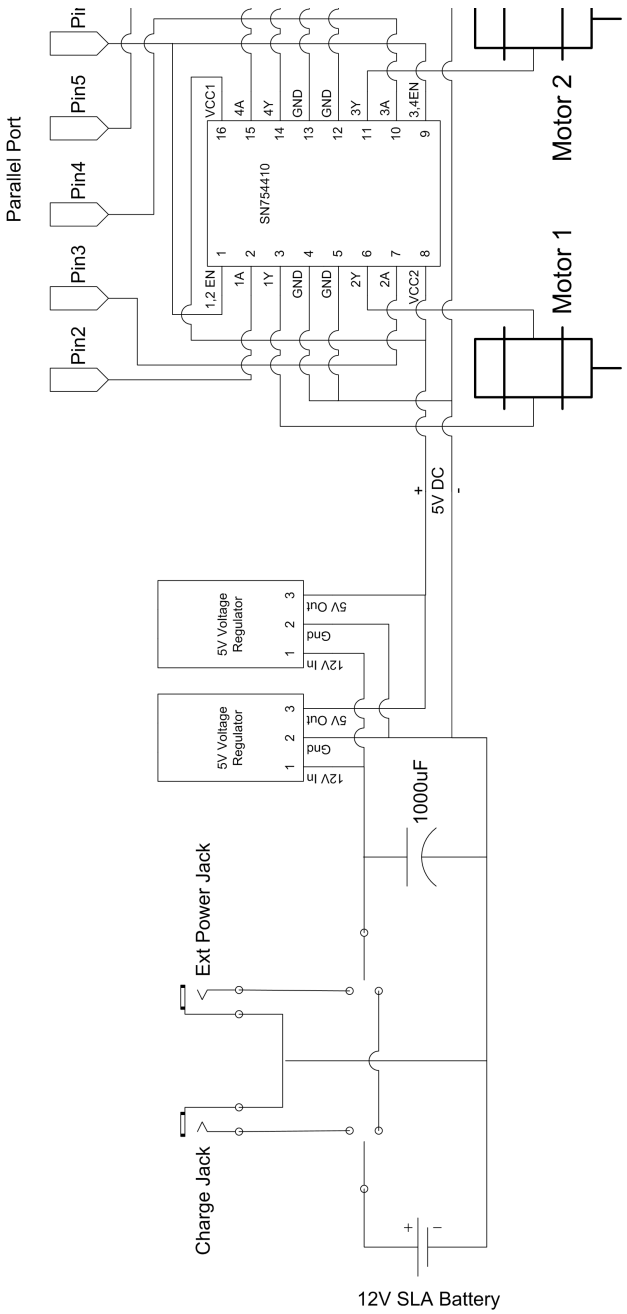


Figure 2.14: Complete circuit schematic

2.5 Software Setup

This section describes a few of the details of the software configuration. Several of the software packages have been updated significantly since the last version of the MSP was configured, so there are some interesting new features that can be made use of.

2.5.1 Operating System

Each MSP is configured with the most current version of the Fedora Linux operating system that was available at the time of installing the system. Fedora Core was chosen because many people are familiar with it. Also, Linux is available for free under the GNU software license; thus, software will not contribute to the cost of the MSP. Further, an earlier version of Fedora Core worked well on the first generation.

2.5.2 Camera Driver

As mentioned previously, there have been some changes in the maintenance of the camera driver. In fact, the new kernel that comes with the latest version of Fedora no longer supports the older driver. The Phillips Webcam (PWC) driver is now being developed by a different individual; kernel support has improved greatly. Video4Linux (V4L) is the software package that provides streaming video support to the Linux family of operating systems. The PWC driver is designed to be used through V4L, which means that any software that uses V4L is ready to use the camera.

2.5.3 Motor and Sonar Drivers

The parallel port is used to interface with the H-Bridge and the sonar ranger. Writing commands to the parallel port is fairly simple. On Linux the port must be opened using the ioperm() function. Individual bytes are then written to the port using the outb() function. There are two points that must be noted when writing software to use the parallel port: Compiler optimizations must be enabled in order to pull in the correct libraries on all platforms. Also, access to the parallel port requires root privileges. A work-around using the *sudoers* mechanism would be preferred, but so far that does not work properly. The motor driver that was written for the first

generation MSP is used, with some minor modifications. A few lines are added to enable support for enabling the H-Bridge through pin six of the parallel port. The sonar ranger driver is used exactly as it was on the first MSP. A signal is sent to the sensor via the parallel port to obtain a reading. The parallel port driver then measures the time for this signal to return on a different pin. A trivial calculation is then performed to obtain the actual distance that was measured.

Chapter 3

Localization and Distributed Self-Deployment

The most popular localization techniques were described in Sec. 1.4. In this chapter the selection and implementation of a localization method is discussed. Then an algorithm and software implementation is presented that uses the method that was selected to localize the MSPs within the testing area, which will be described in more detail later. Results obtained by running this software on the MSP tesbed are given in the following chapter.

3.1 Selecting a Localization Method

The most desirable case would be to implement localization using the sensors that are already present on the MSP, without having to buy additional hardware. The proposed MSP is equipped with two types of sensors: The camera and the ultrasonic ranger. Given the available hardware, it may be possible to implement localization using dead-reckoning, hop counting, wireless triangulation using TOA or RSS, and possibly even wireless angle of arrival when you consider that the platform is able to rotate.

As mentioned previously, dead-reckoning is one of the easiest localization methods to implement, but it requires knowledge of the starting point of the MSP. For this thesis work it is assumed that the wireless sensor nodes are randomly deployed, so

dead-reckoning is not helpful. Furthermore, dead-reckoning works best with wheel encoders, which are currently not present on the testbed. However, these could be added at a very modest cost.

The presence of the 802.11b wireless network card would allow for the implementation of a number of localization methods. However, all of these have some problems in the current testbed setup. First, wireless hop counting is most useful for determining routing and proximity information, but it does not give accurate data about the location to within several meters. Also, with wireless hop counting it is assumed that each node is within reach of only a few neighbors, thereby making it necessary to use a mesh network for data relaying anyway. However, it is the case that the localization algorithm to be implemented should be able to localize nodes even if they are all close to each other.

Wireless triangulation is one of the most popular methods for localization in wireless sensor networks. Unfortunately it is necessary that at least three wireless access points or beacons be visible to the MSP so that consistent and unambiguous localization can be achieved using TOA and RSS. The testing environment is currently only within reach of one such access point, which means that additional access points would be required. Setting up new access points is problematic because it would add significant cost to the localization implementation. There are also some problems with each particular kind of wireless triangulation. Time of arrival requires very precise timing. Since wireless signals travel about 1 foot in 1 ns, sub-nanosecond timing would be desirable. This would obviously be difficult to achieve on the standard wireless cards without purchasing additional hardware. Triangulation using received signal strength would also be very difficult because there are many obstacles between the MSPs and the present wireless access point, or potentially additional wireless access point locations. This would lead to unknown amounts of attenuation in various locations, and thus RSS is not a good choice for the indoor testing environment.

As mentioned previously, angle of arrival localization typically requires antenna arrays. Using just one antenna such an array could be simulated by having the MSP move around and take TOA or RSS readings at different locations. However, this would require highly precise movements on behalf of the MSP, and therefore is not feasible.

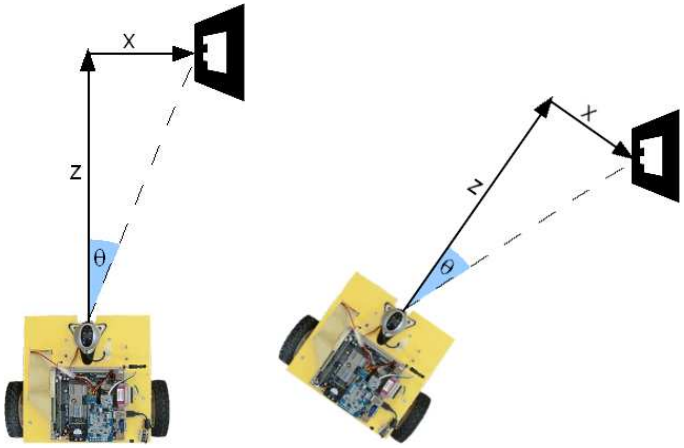
Ruling out these previous methods leaves visual marker detection as a highly attractive option. Because a camera is available on each MSP anyway, this method adds no cost to the design of the MSP testbed. In Sec. 1.4.4 several different possibilities for visual marker detection were explored. There are several widely used structured marker systems, but a number of them are not freely available.

The ARToolKit software package is not only free and open-source, but it comes with excellent documentation and community support. Also, as described in Sec. 1.4.4, the ARToolKit is among the best performers as far as detection accuracy and computation speed are concerned. This package offers C/C++ libraries that allow the detection of pre-defined markers, as well as their estimated pose in real-time. The ARToolKit software is primarily used for Augmented Reality applications, but it should be possible to use the libraries only for localization, essentially ignoring the Augmented Reality functions.

3.2 ARToolKit Details

The behavior of the ARToolKit is now explained. The software works by loading a training data set for each marker that is used, as chosen by the user. It is possible to load several hundred markers, but the more markers there are, the greater the likelihood of receiving an erroneous marker classification. When the ARToolKit is initialized it loads the training set into a database along with an ID that uniquely identifies each marker. When a marker is detected, its ID number is reported, along with a matrix that contains the pose estimate. The ARToolKit is capable of detecting multiple markers in a single video frame. When this happens, an array of marker IDs and an array of pose matrices is returned.

By default, the library returns the matrix corresponding to the coordinate system (CS) that gives the position of the marker in the camera's coordinate system. The matrix gives three values: x, the lateral translation, y, the vertical translation, and z, the distance to the x-y plane the marker is located on. Note that x and y can be negative, but z cannot. It is important to also note that in this coordinate system the x-y plane does not usually coincide with the plane of the marker, unless the camera is



(a) The marker is located at distance z and translation x in camera CS

(b) Distances x and z are unchanged, but the position of the MSP with respect to the marker is different

Figure 3.1: Camera coordinate system

located directly in front of the marker image. The coordinate system is corresponding to this matrix is shown in Fig. 3.1.

The height is omitted since it is not used to localize the MSP. It is easy to see that it is possible for x and z to remain unchanged, even though the position of the MSP relative to the marker is different. This means that given the same x and z coordinates, the MSP could be at an infinite number of locations along a semicircle. However, there are two useful pieces of information that can be obtained from this coordinate system:

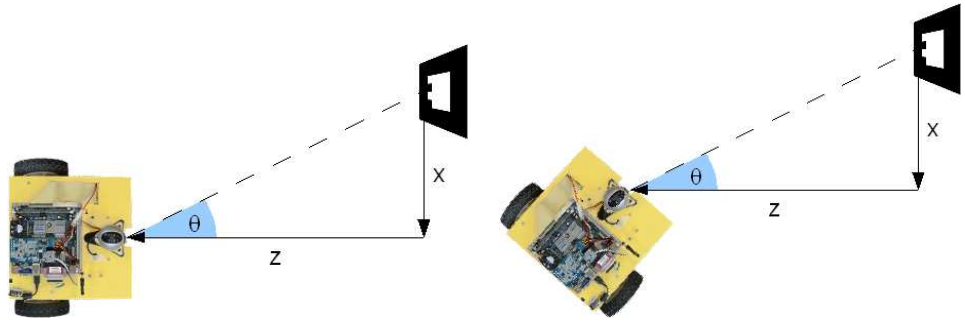
- The angle θ_{marker} at which the marker is found with respect to the center of the image is given by

$$\theta_{marker} = \arctan \frac{x}{z} \tag{3.1}$$

- The total distance between the camera and the marker is given by

$$d = \sqrt{x^2 + y^2 + z^2} \tag{3.2}$$

The camera coordinate system does not provide unambiguous information about the location of the MSP with respect to the marker. Thus, after the geometry of the



(a) The camera is located at distance z and translation x in the marker CS

(b) As the MSP turns x and z are unchanged

Figure 3.2: Marker coordinate system

marker is detected by the ARToolKit it is necessary to call a transformation function that will return the marker coordinate system. This coordinate system is shown in Fig. 3.2. The marker coordinate system provides the information that is lacking in the camera coordinate system to completely localize the camera:

- The angle θ_{msp} at which the camera is found with respect to the marker's plane is given by

$$\theta_{msp} = -\arctan\frac{x}{z} \tag{3.3}$$

  The negative of the angle is taken to make it consistent with the coordinate system of the Graphical User Interface, which will be described later in this chapter.

- The total distance between the camera and the marker is again given by

$$d = \sqrt{x^2 + y^2 + z^2} \tag{3.4}$$

Next, it is necessary to define the algorithms that should be implemented in the software.

3.3 Algorithm Development

As mentioned in Sec. 1.6, it is desired to implement centralized and distributed deployment algorithms, and to evaluate how each of them performs on the actual testbed. This means that depending on the mode of operation, the majority of the processing will be done either by the MSP itself, or by a central processing node, which could also be one of the MSPs, or an external computer. Regardless of running centralized or distributed localization, the marker detection itself must always be run directly on each MSP. Transmitting images to another location for remote processing is too costly from an energy consumption standpoint. Furthermore, transmitting images for remote processing would introduce undesirable delay.

In both operating modes each MSP will use the ARToolKit library to detect the fiducial markers. Markers are placed around the perimeter of the testing area. Since the MSPs are randomly deployed they should roam around until they are able to locate one of the markers for pose determination.

Since the starting location of each MSP is completely random, there is no way of knowing where a marker might be found at first. Therefore, after being initially deployed each MSP will follow the steps given by Algorithm 1 until a marker is detected in a video frame; this will be called the *lost* routine, since it causes the MSP to roam around until it is no longer lost.

As soon as a marker is detected in the field of view the lost routine is aborted, regardless of which step the MSP was currently at. If the MSP is very far from the marker at the time of detection it is possible that the pose estimate is not accurate. For this reason the MSP should move closer to the detected marker until it can be said with reasonable certainty that the pose estimate is good. Since the field of vision narrows as the platform is approaching a marker, it is necessary to track the location of the marker in the image frame and adjust the heading of the MSP to move directly towards it. Otherwise, the marker may move out of the field of vision and the lost routine will have to start over. This tracking is accomplished by measuring the angle between the center of the image and the actual location of the marker. The angle is obtained from the camera coordinate system using Eq. 3.1. The angle is then used to make adjustments to the motor speeds to correct the heading of the MSP. For

Algorithm 1: Localization Algorithm for Centralized and Distributed Deployment.

```
Data: Images taken at a 15-frame per second (fps) rate while on the move, marker
      coordinates
Result: Distance to marker (d), angle to marker (θ), position of MSP in the map.
while true do
    Resolve d and θ for every frame while making the 360 degree turn;
    if Marker resolved during the turn then
        Break from the while loop;
    end
    Move 1 meter forward;
    if obstacle detected by sonar then
        turn right by 90 degrees;
    end
end
if MSP is farther than 1.5 meters from the marker then
    move towards marker until within 1.5 meters;
end
Calculate the MSP position in the map using d and θ;
```

example, if the marker is in the left half of the image, the MSP will veer to the left until the marker is centered in the image. Once the marker is detected to be within a range of 1.5 meters, the sensor node will stop and accept the pose estimate calculated at that point. If the marker is already within 1.5 meters the first time it is detected, the MSP will not move any closer. What happens next depends on the mode of operation.

3.3.1 Centralized Deployment

In the centralized localization mode the coordinates of the MSP with respect to the marker coordinate system along with the detected marker ID are sent to the central processing node. This node has knowledge of the location of each marker in the map of the testing area, and can calculate the position of each MSP based on the pose estimate that is received. This calculation is performed immediately, and the position of the MSP is then drawn to the screen by the GUI. The GUI is essential to quickly evaluating the performance of the localization. The central processing node is also tasked with optimally distributing the nodes to pre-defined target locations. To

Algorithm 2: Centralized Deployment Algorithm.

Data: Marker locations, images captured at 15fps.
Result: Deployed MSPs.
Each MSP connects to the central processing server;
while *Not all MSPs localized* **do**
    MSPs localize themselves according to Algorithm 1;
    The pose estimate is sent to the server;
    Server uses pose estimate to localize the MSP;
end
Mark all targets and MSPs as "unassigned";
while *Not all MSPs and targets "assigned"* **do**
    Calculate center of MSP cluster;
    Calculate distance to each unassigned target;
    Select unassigned target with the greatest distance;
    Determine which unassigned MSP is closest to this target by calculating the distance to each;
    This MSP is marked as assigned to this target;
    Path planning is performed and the MSP is instructed to approach the target;
end

minimize the distance that each node must travel, the software waits to receive pose estimates from every platform before starting the distribution routine. This is also done to avoid sending a sensor node to a target location, only to find out seconds later that another node is already much closer to that target. The distribution algorithm is according to the steps shown in Algorithm 2.

3.3.2 Distributed Deployment

In the distributed localization mode there is no central processing node to make calculations and decisions for the localization process. Instead, all calculations and decision making are carried out on each MSP directly. However, the external node is still needed for the purpose of sending start and stop commands to the MSP cluster. In this mode each node has complete knowledge of all marker locations in the testing area. The algorithm is guided by the following steps shown in Algorithm 3.

Algorithm 3: Distributed Self-Deployment Algorithm.

Data: Marker locations, images captured at 15fps, and the environmental map.
Result: MSPs deployed.
Each MSP connects to the central processing server;
Each MSP attempts to connect to all other MSPs;
MSPs search the environment to estimate its pose as described in Algorithm 1;
Determine the location of the MSP within the map using the pose estimate and the map;
Mark all targets as "unassigned";
while *MSP not assigned* **do**
    Calculate distance from MSP to each unassigned target;
    Select unassigned target with the shortest distance;
    Send a message to all MSPs that are connected to ask for permission to approach target;
    if *permission denied message received from other MSPs within a period of waiting time* **then**
        Label marker as "assigned";
    end
    else
        Label marker as assigned to self;
    end
end
Plan path to target;
Approach target;

3.4 Software Implementation

The most challenging part of implementing this system is writing the software. To make the software easier to manage it is segmented into several functional blocks. These objects can then be called by the main program as needed. Further, two separate executables are created for the central processing node and the MSPs. The two executables will also be referred to as server and client since by default the central processing node waits for incoming connections, and the MSP will make outgoing connections at startup. However, both of them do not exclusively play the roles of client and server. The two programs have some of the functional blocks in common, and also some objects which are unique to each. The parts that are used by both programs are written such that they function differently depending on which executable they are compiled into. The discussion that follows considers each of these objects at a time, and begins with the parts that belong to the client software alone. Each part is written as a class in C++, and object oriented programming principles are followed.

3.4.1 ARToolKit Interface

The ARToolKit is written in C and is made so that it compiles on Windows and Linux. On each operating system it uses the appropriate drivers to connect directly to a USB or firewire camera. Fortunately, on Linux the ARToolKit is configured to use the Video4Linux architecture. This is ideal because the camera driver for the Logitech 4000 Pro is made specifically for use with V4L. Thus, setting the ARToolKit up to use the cameras present on the MSP testbed is almost effortless. However, there are some issues with the ARToolKit libraries and gcc4 that must be fixed. This concerns several path names to OpenGL libraries, and also some problems arising from the fact that gcc4 is much stricter than gcc3, which apparently is still the version of the compiler used by most people at the time of this writing.

The next major hurdle that presents itself is the fact that the ARToolKit is not written as a collection of classes, but as a collection of static C libraries. The solution consists of creating a small C++ wrapper that exposes a few public methods for interfacing. Hence, the class that wraps the ARToolKit is called arInterface. An

alternative version of the ARToolKit that is written in C++, called ARToolKit Plus was considered for use, as well, but it does not have integrated camera support. The arInterface creates a new thread specifically for use by the ARToolKit so that it is capable of detecting markers even when the main program is doing other things.

Since the ARToolKit software package is typically used for Augmented Reality applications, starting it causes a window to open up that shows the live video stream. This is undesirable when the software is running on the MSP, because there is no need for graphics display on each sensor node. It would be great if this window could be suppressed in the future.

The arInterface class can be instructed to initiate marker recognition, to send video frames to the central processing node without marker recognition, or both. When video is requested, each video frame is passed directly to the socket class to be sent to the central processing node for display in the GUI. Further, when markers are detected in the field of vision, the angle of the MSP with respect to the marker is calculated based on the camera coordinate system, and this angle is then passed to the localization class along with the coordinates of the MSP in the marker coordinate system.

3.4.2 Motor and Sonar Control

As described in Sec. 2.5.3, motor and sonar drivers were already developed for the first version of the MSP, and only minor modifications were made to the motor class to add an enable/disable pin. To make moving the MSP even easier for this version of the MSP, another class, named Control, is created. The Control class creates an additional level of abstraction for issuing commands to the lower level motor and sonar classes. Other objects can run member functions such as turn or emergency stop (estop), and the control class will call the appropriate Motor control member functions to make the MSP turn, or to tell all motors to stop immediately.

When a sequence of moves is to be executed this is performed inside of a newly created thread. This is done so that a sequence can be aborted at any time by killing that thread, without having to wait for a particular move to complete. This kind

Table 3.1: MSPXY class used to store data about MSP locations and final target locations. The class used for markers is the same with the exception that it lacks the target member.

Data Type	Name	Explanation
int	x	the x coordinate
int	y	the y coordinate
int	angle	the angle in the map coordinate system
int	target	stores the id number of the assigned object

of behavior is necessary, for example, for aborting the lost function, or aborting any other move because an emergency stop instruction is issued by the user.

3.4.3 Self-Deployment

The localization class, which is common to both the client and server programs, is at the heart of the entire system. This class does all the calculations to localize all the nodes. There is an integer variable, called *mode*, that is used to set whether the class should operate in centralized mode or distributed mode, or in a special central processing node mode. The centralized and distributed modes are discussed individually:

- **Centralized mode** In the client software, upon receiving pose data and a marker ID from the arInterface class, the data are not processed and instead simply forwarded to the sockets class to be sent to the central processing node.

  The Sockets class running on the central processing node (the server) receives this data as described in Sec. 3.4.4, and then passes it to this localization class. At this point a new record for this MSP is created within the localization class if this is the first time a message is received from it. In this case this is actually a class, of which a new instance is created when a new record is needed, and it is then stored into a Standard Template Library (STL) Map. The class is shown in Table 3.1.

  The actual location of the MSP is calculated according to the marker locations stored in the map. Depending on the orientation of a marker in the map coordinate system, the x and z distances of the MSP pose estimate are either added

Table 3.2: Signs used to calculate the localization of MSPs. Markers may be affixed to any wall, so they may be facing in one of four directions

Marker angle	dx & dy
0 facing down	dx = x dy = z
90 facing left	dx = -z dy = x
180 facing up	dx = -x dy = -z
270 facing right	dx = z dy = -x

or subtracted to/from the x and y coordinates of the marker itself. dx and dy are used to represent the coordinates that are added/subtracted. Table 3.2 shows the values of dx and dy, and the equations to calculate estimated x and y coordinates for each MSP are then given by:

$$\begin{aligned} estx &= dx + x_{marker} \\ exty &= dy + y_{marker} \end{aligned} \tag{3.5}$$

Note that the map coordinate system is based on the orientation of the GUI, which follows the standard coordinate system used by all graphical display systems; that is, y facing downward, and angles are measured positive in the clockwise direction.

Next, the angle of the MSP with respect to the marker is calculated as shown in Eq. 3.3, and the angle at which the MSP is oriented in the map coordinate system overall is then given by

$$\theta = \text{marker angle} - \theta_{marker} + \theta_{msp} \tag{3.6}$$

where marker angle is the angle of the fiducial in the map coordinate system. Again, note that positive angles are measured clockwise.

Once all MSPs are localized, it is time to distribute them according to the steps outlined in Sec. 3.3.1. At this point there are two STL maps that hold data about MSPs and final target locations; these are used to determine how the MSPs should be assigned. Once an assignment is made a path planning method is called, which calculates movement instructions for each MSP and these are then sent to the nodes for execution.

- **Distributed mode** In distributed localization mode the class does not send pose data to the central processing node, but the localization object processes the data directly on each MSP. For this purpose, the class loads the marker map and final destination data, and once it receives a pose estimate it localizes itself based on the knowledge it has that is given in the map. These calculations are performed exactly as described by the equations shown in the previous paragraphs on centralized localization. However, after a node determines its own location, it determines the target location that is closest and requests permission from other MSPs to approach that location. The algorithm is described in detail in Sec. 3.3.2.

3.4.4 Network Communication

The socket library is one of the parts at the core of each program because it enables the MSPs to communicate with each other and the central processing node. This class is also one of the first to be instantiated by the server and client programs. When the instance is created, the class immediately creates a new thread to listen for incoming connections and sets up a data structure to hold information about the machines that connect. The information that is kept in the data structure is shown in Table 3.3. Some of this data is also maintained for use by the GUI and the localization class. Whenever a new connection is made, be it incoming or outgoing, the information is added to the data structure, and when a connection is closed, the data is removed.

The networking class also provides a method to connect to any other machine given a hostname. However, before a new connection is made the above mentioned data structure is checked to determine if a connection to that particular machine already exists. Duplicate connections are not allowed, since they would serve no

Table 3.3: Connected client data

Data Type	Name	Explanation
int	fd	socket number associated with client
char	ip	IP address of client
char	name	hostname of client
int	video	status flag indicating whether video is on or off
int	artoolkit	status flag indicating whether artoolkit is on or off
int	mspId	number of MSP uniquely identifying it

purpose. The most important task of the networking class is obviously sending data back and forth between the different nodes. There are basically two types of data that can be exchanged: The pose estimates associated with MSP IDs and raw image data to be sent to the main processing node for display in the GUI. MSPs are not able (and should not be able to) send images to each other. The data structure used to send pose data across the wireless network is shown in Table 3.4.

This data structure is also used for sending other types of commands. These include instructions such as telling an MSP to send video, or telling it to begin marker detection; other instructions may tell an MSP to begin localization in a certain mode, or even to perform an emergency stop. To send these special types of instructions the marker number, which usually starts at one, is set to a negative number. When data is received by a node, the networking class checks the value of the marknum member. A negative value means that the data structure is a special command, and the actual value may indicate what this specific command is. For example, receiving a packet with a marknum field that is equal to -200 means that the remaining members of that transmitted data structure contain an angle and a distance that tell the MSP where to move to. If a data packet arrives and it does not have a negative marknum field, it means that it is an actual pose estimate, and instead of issuing a special command, it is passed into the localization class for further processing.

3.4.5 Completing the MSP Client Software

The MSP client software is tied together by a main program that creates instances of each of the classes explained above. A flowchart depicting the data flow between

Table 3.4: Pose data structure

Data Type	Name	Explanation
int	mspId	the number assigned to that particular MSP
int	marknum	the marker number that is detected
int	lateral	the lateral translation of the MSP
int	distance	the distance of the MSP from the plane of the marker
int	vertical	the vertical translation of the MSP
int	angle	the angle of the marker with respect to the center of the image

all of these objects is shown in Fig. 3.3. The arrows indicate data flow, and flow of instructions.

3.4.6 Completing the Server Software with a Graphical User Interface

The Graphical User Interface (GUI) is used on the central processing node to easily issue instructions to the MSP cluster, and to visualize the localization and distribution result. The interface is written entirely using the Qt toolkit C++ class libraries, and therefore can be compiled on Linux, Windows, and even MacOS. This is something that can not be said of the client software, as it makes heavy use of hardware drivers that are unique to Linux. A block diagram showing the top-level objects being used for the GUI application is shown in Fig. 3.4. A screen-shot of the GUI right after startup is shown in Fig. 3.5.

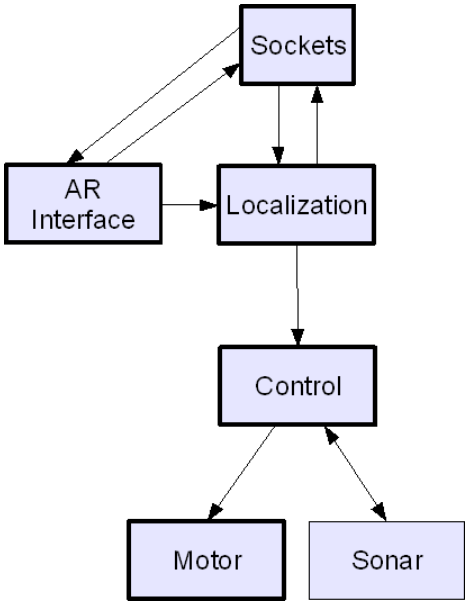


Figure 3.3: MSP client block diagram. The sonar object is the only one that does not have at least one thread

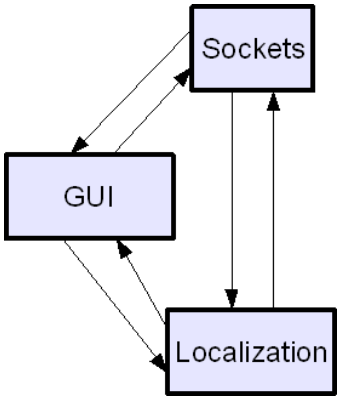


Figure 3.4: Central processing node block diagram

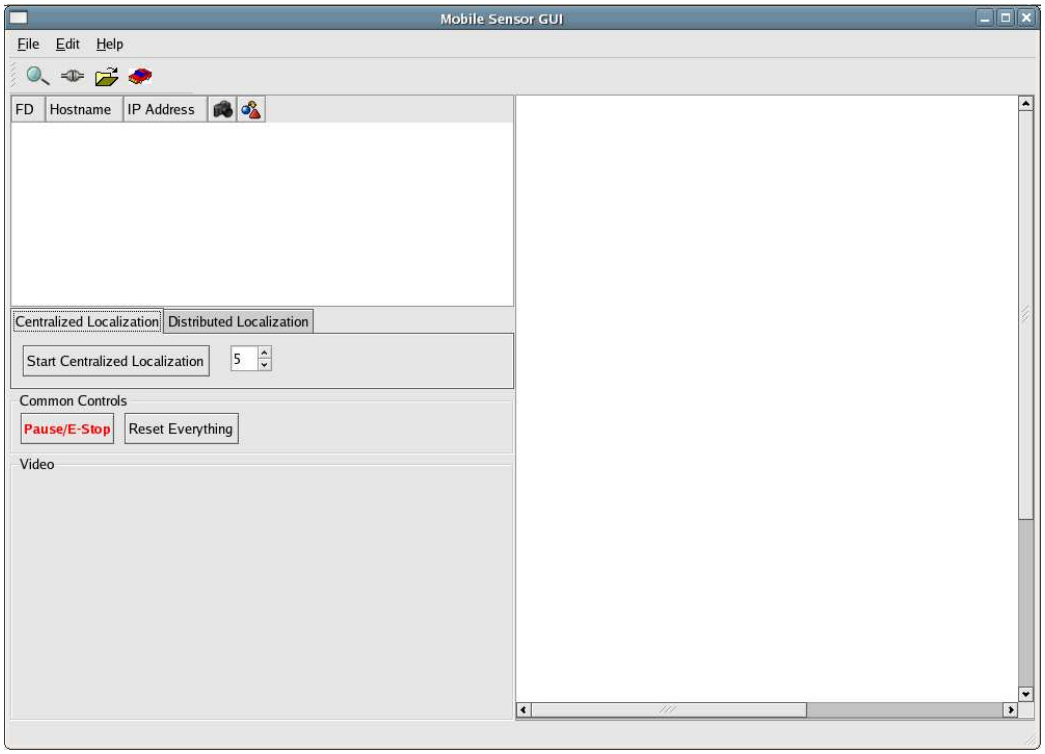


Figure 3.5: Graphical user interface right after starting up. It has an area for displaying the connected clients, several buttons to control the cluster, an area to display live video streams, and a large box on the right to display the map as MSPs localize themselves

Chapter 4

Experimental Results and Discussions

The preceding chapters discussed the details of localization and deployment algorithms, and software was written to implement centralized and distributed deployment of the MSPs. In this chapter the experimental setup and results will be discussed. In the process of conducting these experiments some ideas have been gathered for future improvements, and these are addressed in the following chapter.

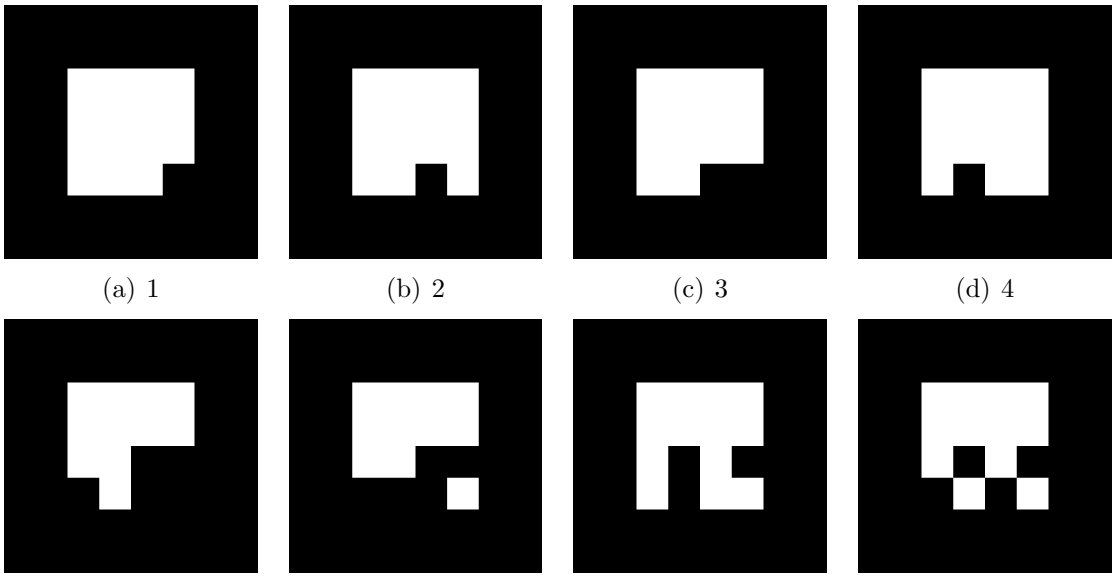
4.1 Experimental Setup

The software described in the previous chapter was tested every step of the way as it was being written, using a single marker and a camera. However, the real efficacy of the system is determined by the successful deployment of multiple MSPs in the testing area. The testing environment consists of an enclosed space of about 4×4 meters. The perimeter is formed by white foamcore board of two different heights, with just one side open to allow for easy access. The flooring is made of cloudy gray tile, and is very smooth and flat, which should be an ideal surface for the MSPs to move around. Artificial lighting is used, with the blinds shut to keep sunlight from creating patterns of very high contrast. Since the marker detection algorithm uses thresholding to create a binary image, it is beneficial to keep the lighting as constant as possible, as a way to ensure that the best possible detection is achieved. The area

Figure 4.1: Testing area surrounded by white foamcore board on three sides. The inside of the area is kept clear of obstacles

is cleared of all dirt and any obstacles. A picture of the testing environment is shown in Fig. 4.1.

To allow for each of the MSPs to be localized, eight markers are affixed to the foamcore board around the perimeter of the area. It is easy to create custom markers, but for this experimental setup a standard set produced by the Geometric Design and Computation (GDC) research group is used [14]. The markers are designed by dividing the internal area into a 4×4 grid, and using each grid point as a digit for counting in binary. So, a marker of value one has the lower right hand square filled in, a marker of value two has the second square from the bottom right hand side filled in, and so on. The entire interior can be filled in this way, but care must be taken not create two patterns that are the same under rotation. Such patterns must be skipped. Eight markers that were used for the experiment are shown in Fig. 4.2. The values encoded into the marker are 1, 2, 3, 4, 56, 59, 77, and 83. The last four markers were chosen for maximum dissimilarity, in order to test the detection result as compared to the more similar markers 1-4.

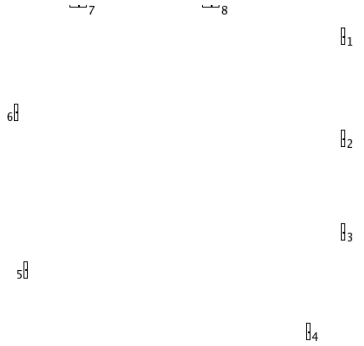


(a) 1 (b) 2 (c) 3 (d) 4

(e) 56 (f) 59 (g) 77 (h) 83

Figure 4.2: Markers with different binary values encoded into the center white square. The markers have values as shown in the individual captions

(a) Fiducials mounted on the foamcore wall



(b) Numbered marker map. The numbers shown are consecutive IDs assigned to markers

Figure 4.3: Fiducial placement in the testing area. All markers are facing inward

The fiducials are 10×10 cm in size, printed on bright white paper using a laser printer to get the highest possible contrast. Again, this is to make certain that there is sufficient contrast for the thresholding process to properly detect each marker that is in the field of view. The markers are spaced roughly one meter apart, but it is not necessary for them to be spaced perfectly uniformly, as long as their exact locations are known. Each marker is attached to the perimeter so that its center is about 20cm off the ground, and right side up; orientation is critical, since pose in relation to the marker is deducted from the internal features and orientation of said features. The height, however, is not that critical, as long as the camera on each MSP has a clear and unobstructed view of the fiducial. It is also very important that the marker be flat, otherwise the geometry of the marker would be distorted in the eccentricity detection process. Fig. 4.3(a) shows several markers mounted on the perimeter foamcore wall.

A map showing the location of each marker is given in Fig. 4.3(b). This map is actually generated by the GUI. To load the marker locations, their marker ID, x and y coordinates, as well as their angle must be specified in a text file that is read in by the software. The values in the configuration file are shown in Table 4.1. As mentioned earlier, the coordinate system is that used by many graphical display systems, which means that the positive x axis is pointing to the right, the positive y axis is pointing downward, and angles are measured positive in the clockwise direction, starting from the x axis. Therefore, the markers that are facing downward at the top have an angle

Table 4.1: Eight fiducials used in the testing area. Binary number encoded into each pattern is given by the column pattern number

Marker ID	Pattern Number	X	Y	Angle
1	1	396	36	90
2	2	396	154	90
3	3	396	262	90
4	4	356	377	90
5	56	30	305	270
6	59	19	123	270
7	83	91	0	0
8	77	244	0	0

of zero degrees, the markers on the right hand side have an angle of 90 degrees, and so on.

In addition to having knowledge of the marker locations, the software must also know the final target locations. These are also specified manually, and loaded from another configuration text file. It is acceptable to define more targets than there are MSPs. Five targets are defined for this experimental setup, since this conveniently places a target in each corner, and one in the center. The coordinates that are specified as targets at this time are given in Table 4.2.

4.2 Localization Result

The methodology for localizing MSPs based on the markers found in the testing area is described in Sec. 3.2. Software was written to implement those algorithms, and the results obtained by using that software are now discussed. Since the ARToolKit is mature software, and the algorithm it uses is very robust, localization using the markers is highly successful. Marker detection is successful even at distances greater than a couple of meters. However, the quality of the pose estimate declines sharply at distances greater than 1.5 meters. For this reason, pose estimates are only trusted at a close distance. Example locations of MSPs when they are localized are shown in Fig. 4.4. Considering the points mentioned above, the localization software exhibits the following behavior:

Table 4.2: Final target locations of the MSPs

Target ID	X	Y
1	91	61
2	305	61
3	198	182
4	91	305
5	305	305

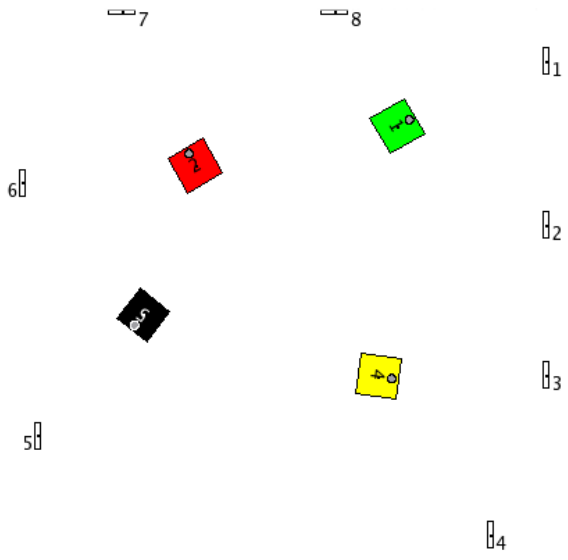


Figure 4.4: Four localized MSPs.

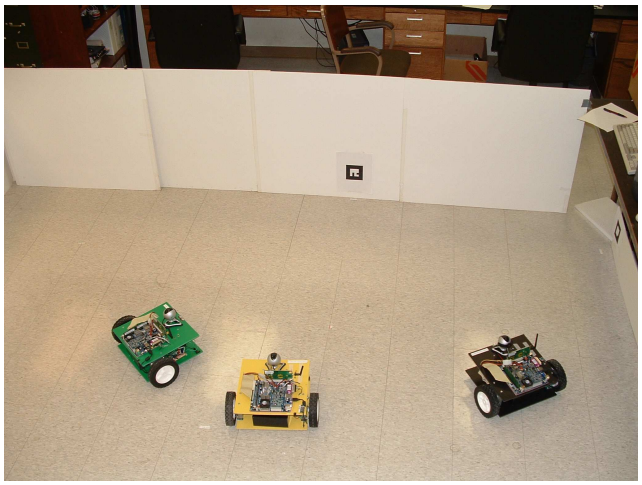
- As shown in Fig. 1.20, it is true that pose estimates are the most accurate when the MSP is viewing the marker from an angle. At positions that are very close to being exactly centered, the eccentricity of the marker is very minute, and a false estimate is sometimes returned. For example, the platform may be located at 5° from the center axis, but the ARToolKit actually reports −5°. When this occurs at very small angles, it often does not introduce appreciable error, since the effective difference in position is relatively small, on the order of 10-20cm. At larger angles (15° or more) this pose estimate introduces serious errors.

- The rate of false marker classifications, that is, identifying a marker as another marker, is exceedingly low. This is in part due to the fact that there are at most only eight markers in the training set. This event occurs only once every couple of minutes, and at a frame rate of approximately 10-15 frames/sec. This means that the frequency of false classifications is far less than 1%.

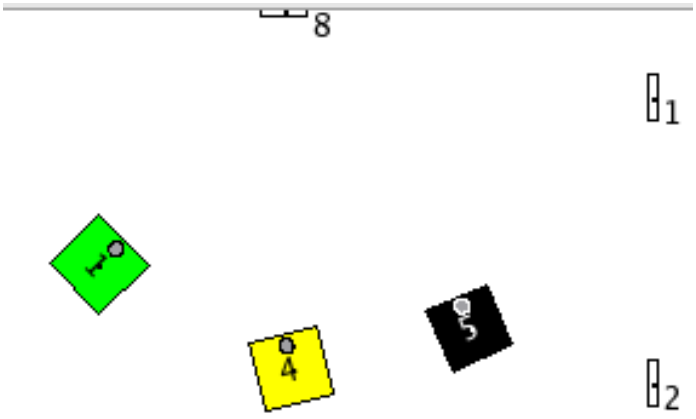
- False positive marker detections occur when the ARToolKit library segments a square in a video frame, but it is not actually a marker. This never happens when the white foamcore is in the background, but only when the MSP is facing an unenclosed side. Objects that will cause such false positive marker detections are the legs and armrests of chairs, the feet of some tables, etc. Since the platform is assumed to operate in an area with uniform background, these objects should be eliminated. Therefore, the rate of false positive marker detections is also far less than 1%.

- While it is theoretically possible that the ARToolKit library would not detect a marker at all for whatever reasons, this event is not observed during testing, due in part to the fact that the artificial lighting creates very ideal testing conditions, and markers are kept clear of obstructions.

Fig. 4.5(a) shows several MSPs that are stationary after moving close to a marker to localize themselves. These same MSPs are shown in Fig. 4.5(b) from the GUI's perspective. The GUI draws exactly the position reported by each MSP. It can be seen that the yellow MSP is slightly too far to the right. Since localization error is



(a) Picture showing the position of MSPs after localization is complete



(b) GUI representation of those same positions

Figure 4.5: Localization of three MSPs

Table 4.3: Measurements from the localization of the yellow MSP

reported(cm)		actual (cm)		difference (cm)		Euclidean
x	y	x	y	x	y	distance
339	122	337.82	127	-1.18	5	5.14
292	145	304.8	137.16	12.8	-7.84	15.01
345	96	353.06	93.98	8.06	-2.02	8.31
287	113	281.94	111.76	-5.06	-1.24	5.21
341	141	337.82	148.59	-3.18	7.59	8.23
214	86	219.71	87.63	5.71	1.63	5.94
136	89	142.24	95.25	6.24	6.25	8.83
					average	8.10
					STD	3.42

the greatest contributing factor to problems in the overall deployment of the MSPs, measurements were taken to quantify the localization accuracy.

Seven trials are conducted with each MSP. The reported location and the actual measured location are recorded, and the difference is calculated. Then, the euclidian distance is used to calculate the overall reported error from the actual location. The mean and standard deviation of the Euclidean error are calculated for each trial, shown together with the data in Tables. 4.3, 4.4, 4.5, and 4.6.

Since the area has a size of 4×4 meters, and the overall average error is calculated to be $14.7cm$, we can find the overall approximate accuracy of the localization to be

$$accuracy = \left(1 - \frac{14.7\text{cm}}{400\text{cm}}\right) \times 100\% = 96.3\% \tag{4.1}$$

Of course, if you consider that in actuality it is acceptable for an MSP to be localized a couple centimeters from its true location, the effective accuracy could be considered to be even higher.

The overall mean and standard deviation for all trials of each MSP are shown in Fig. 4.6. This clearly shows that there is a wide range of accuracies between different MSPs.

Table 4.4: Measurements from the localization of the black MSP

reported(cm)		actual (cm)		difference (cm)		Euclidean
x	y	x	y	x	y	distance
158	118	152.4	132.08	-5.6	14.08	15.15
192	142	213.36	162.56	21.36	20.56	29.65
221	155	187.96	144.78	-33.04	-10.22	34.58
250	43	240.03	55.88	-9.97	12.88	16.29
268	97	256.54	96.52	-11.46	-0.48	11.47
99	160	92.71	165.1	-6.29	5.1	8.10
23	123	52.07	152.4	29.07	29.4	41.35
					average	22.37
					STD	12.74

Table 4.5: Measurements from the localization of the green MSP

reported(cm)		actual (cm)		difference (cm)		Euclidean
x	y	x	y	x	y	distance
200	93	198.12	105.41	-1.88	12.41	12.55
57	147	62.23	123.19	5.23	-23.81	24.38
71	52	62.23	53.34	-8.77	1.34	8.87
161	72	160.02	76.2	-0.98	4.2	4.31
290	99	302.26	100.33	12.26	1.33	12.33
283	135	281.94	135.89	-1.06	0.89	1.38
249	150	237.49	186.69	-11.51	36.69	38.45
					average	14.61
					STD	12.83

Table 4.6: Measurements from the localization of the red MSP

reported(cm)		actual (cm)		difference (cm)		Euclidean
x	y	x	y	x	y	distance
157	123	130.81	143.51	-26.19	20.51	33.27
183	102	184.15	100.33	1.15	-1.67	2.03
209	88	203.2	116.84	-5.8	28.84	29.42
249	78	251.46	93.98	2.46	15.98	16.17
291	202	288.29	214.63	-2.71	12.63	12.92
134	127	132.08	127	-1.92	0	1.92
166	62	166.37	62.23	0.37	0.23	0.44
					average	13.74
					STD	13.46

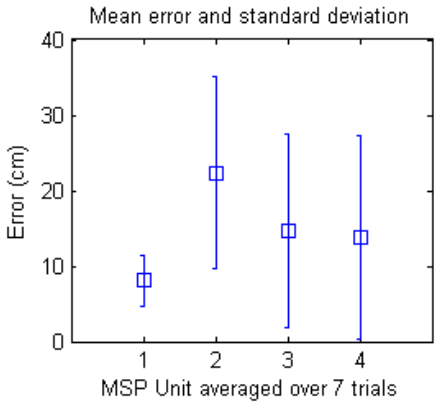


Figure 4.6: Average localization error and standard deviation (STD)

4.3 Self-Deployment Result

The quality of the deployment result depends heavily on the accuracy of prior localization, and the errors outlined above can have a great impact on deployment results. Even very slight errors in the estimated angle can lead to serious problems, propagating and increasing the error the farther the MSP has to travel to reach the intended target. As the MSP moves towards a target it is usually the case that there are no markers close enough to guide the MSP, and this is why they are ignored. So, in essence the MSP is approaching the target blindly, directed only by the prior knowledge of the distance that needs to be traveled. The MSP design is lacking wheel-encoders for dead-reckoning, and therefore the MSP must rely on known motor speed to travel the desired distance. Some of the MSPs do not travel in consistently straight lines when instructed to do so, and this causes further error. This behavior may be caused by any of the following:

- There may be slight differences in the manufacturing of each motor, causing them to rotate at slightly different speeds, provided the same voltage. Since the motors have a gearhead unit attached, the varying resistance of this mechanism may also prevent a motor from achieving its full rotational speed;
- The H-Bridge may have "uneven" circuitry, meaning that one of the motors may be driven at a slightly higher voltage than the other, which would have

a significant impact on speed. Furthermore, as the H-Bridge heats up this problem may be aggravated, if it does not heat evenly;

- The voltage regulators do not all appear to be consistent. While all of them supply voltages very close to 5V, it appears that several of them have a lower maximum current tolerance than others, and therefore self-protect at different current levels by cutting off the power supply.

- There may be some wheel slippage on the floor. Another problem is that over time the wheel may become loose on the motor shaft, and begins slipping. This problem is easy to fix, once it is observed. Also, in very rare instances the omni-wheel appears to get stuck for a brief moment on small gaps between the tiles. Generally, though, once the omni-wheel is moving at a decent speed there are no problems observed.

- There is some built-in error due to the fact that some of the wheels are not exactly the same size. The wheels used for the construction of these MSPs are lawnmower wheels, and those are obviously not manufactured to high precision. More accurate dead-reckoning may be accomplished by carefully selecting wheels that have more consistent sizes.

To characterize the error that is introduced by the movement of the MSP in a straight line, an experiment is conducted. Each MSP is instructed to travel in a straight line for 1 meter, and the lateral deviation from the path is measured. Table. 4.7 shows ten trials of this experiment, with the average deviation for each MSP shown on the last line. The average deviations are also compared in Fig. 4.7, which makes it clear that the last MSP, the red platform, has the worst deviation of all. Much time was spent calibrating this MSP, and the behavior of the motors is so erratic that it was impossible to compensate for the error. The other MSPs were also calibrated to correct for curvature in the path, but with more success than the red platform. The overall average error for all trials of all four MSPs is 7.5cm.

Table 4.7: Measurements of deviation from the path of each MSP over 10 trials. The last line shows the average deviation from the path. All measurements are in cm, where positive and negative values indicate deviation to the right and the left, respectively

yellow	black	green	red
-1.91	0.00	-0.25	19.05
-6.99	-3.18	-2.54	21.59
-7.62	-5.72	1.91	29.21
-3.18	-2.54	4.45	24.13
-7.62	-3.81	-3.18	13.34
-9.53	-4.57	-8.89	15.24
-7.62	-5.72	0.00	11.43
-9.53	-6.35	1.27	7.62
-8.26	-4.45	2.54	8.89
-5.08	-2.03	6.35	10.80
6.73	3.84	3.14	16.13

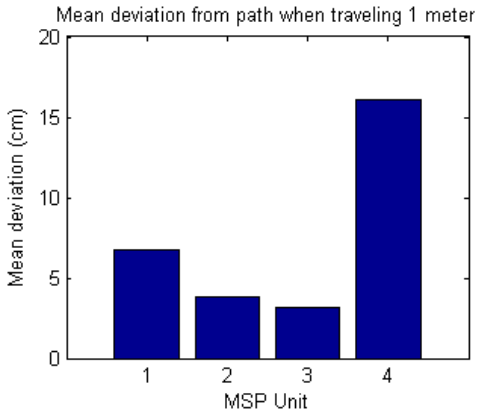


Figure 4.7: Average deviation in cm from the path for each MSP as it travels 1 meter

4.3.1 Centralized Deployment

The centralized deployment algorithm should theoretically achieve the most optimal deployment result because it considers all targets and assigns MSPs such that the distances traveled will be the smallest possible. The time it takes for this algorithm to complete, however, may be greater because the central processing node waits for all nodes to be localized before deployment decisions are made. The deployment algorithm itself performs as expected. MSPs are properly assigned to the targets that are farthest away first, in order to minimize the distance traveled. Path planning also performs as desired, so that MSPs turn towards the correct target and begin their approach.

However, as mentioned previously, since the motors of the MSPs do not perform exactly uniformly every time, error is introduced in the actual target approach. Sometimes an MSP turns slightly, when it really should be going in a straight line.

4.3.2 Distributed Deployment

The distributed self-deployment algorithm also performs as desired. This algorithm should theoretically take less time, since the nodes must not wait for the cluster head to make any decisions, but the target assignments would be sub-optimal. Again, error in the final target approach is introduced by the motor behavior that was described above.

4.3.3 Method Comparison

Both methods successfully deploy MSPs to the correct locations, given that the localization result is reliable. One of the biggest differences is the speed at which the two algorithms achieve full deployment. Since the centralized algorithm waits for all nodes to be localized before initiating the deployment stage, it could lead to long delays if a node has difficulty localizing itself. An experiment is conducted to measure the time it takes each method to complete. Four MSPs are placed randomly in the testing area, and then the software is started. Complete time measurements are shown in Table. 4.8. When running six trials of each deployment method with

Table 4.8: Runtime comparison between centralized and distributed deployment methods. The last row shows average times for each method

centralized	distributed
50	37
40	31
25	34
37	30
28	16
26	24
34.3	28.6

these four MSPs, the centralized and distributed self-deployment methods achieve an average completion time of 34.3 and 28.6 seconds, respectively. This means that during these trial runs, distributed self-deployment was 16.5% faster than centralized deployment, which is consistent with previous expectations.

It turns out to be true that centralized localization takes longer because the software does not begin deploying MSPs until all have been localized, and sometimes this time can be significant. Distributed deployment is prone to other problems, however. Since the deployment is not intelligently coordinated for all MSPs at once, it is possible that the paths of two MSPs will cross as they move in nearly opposite directions, which can lead to delays as one or both of them stop moving to avoid a collision. In addition to that, there is much larger wireless connection overhead, as all MSPs are connected to each other. Furthermore, when all connections are established as the distributed algorithm starts, there can be some connection problems if all MSPs try to do this at once. To work around this problem a couple hundred milliseconds of delay were added between each connection attempt. As each MSP makes several connections, this introduces delay to the total time it takes for the distributed algorithm to execute.

Chapter 5

Conclusions and Future Work

Localization and self-deployment in visual sensor networks are fields of research that currently have a great amount of activity, but there are still many problems that remain unsolved. Many of these problems have been researched in great detail, and most often simulations are offered to validate the proposed algorithms. Unfortunately, real-world implementations pose quite a few technical challenges that make many of these simulations infeasible, or very difficult to realize. In this thesis algorithms for centralized and distributed self-deployment were implemented and then tested on an actual custom-built testbed. The testbed is equipped with the devices necessary to perform a large variety of different tasks. The centralized self-deployment algorithm requires that communication be established to a cluster head, which can be an external node. The distributed self-deployment algorithm requires that each MSP connects to every other MSP to coordinate some very basic decisions. Both the software and the testbed performed satisfactorily, but there are still a few design issues that should be resolved for improved performance. Concerning the hardware specifically, components used by the MSP will become smaller and cheaper in the future, and this will allow for an even better, more compact MSP design. Some of the improvements that can be made to both the hardware and the software are listed in the following section.

5.1 Future Work

This thesis successfully designed a new MSP and implemented a localization and deployment algorithm in software, but there a number of problems that must still be addressed both in hardware and software design. Improvements that can be made to the hardware depend heavily on the availability and the cost of hardware, sensor and processing equipment.

5.1.1 Hardware Improvements

The functionality of a hardware design of Mobile Sensor Platforms is not only constrained by technology itself, but also by its cost. There are many types of sensors that would make great additions to any Mobile Sensor Platform, but they may not be feasible at this point from a cost perspective. A few of the improvements that would be most useful are:

- The MSP would benefit greatly from a more compact and lighter battery. The SLA batteries used currently are very bulky and heavy, but unfortunately they are the only cost-effective option available at the time. NimH and Li-Ion batteries, like the ones used in portable electronics have much higher charge density, but can cost up to ten times as much as SLA batteries. A lighter battery would have a number of highly desirable side effects: The supporting platform could be made of lighter material as it would have much less to support. It also means that the platform would be easier to move, and the motors would therefore consume less current. This could even allow for the usage of a smaller motor all-together.

- Another issue that could greatly reduce the demands on the battery would be a more efficient processing unit. The motherboard with processor and hard drive can consume upward of 2A in an idle state. This is way too much for a truly mobile sensor platform. It would be advantageous to develop a system that powers on instantly, so that it could be sent to a low-power mode to be woken up only when needed. The hard drive itself also consumes a fair amount of energy since it is not designed for low power applications. Smaller laptop hard

drives address the issue of power consumption, but at a greatly increased cost. In recent years a plethora of solid state memory cards have been brought to the market, and prices are falling rapidly. These storage devices provide a large amount of storage in a very small space and with very low power consumption requirements. As prices continue to fall it is very likely that small memory cards will ultimately replace hard drives on the MSP design.

- Since the motors consume up to 1A of current when used concurrently at full speed, it would be desirable to find a more efficient replacement for these, as well. Weaker motors consume less power by design. However, the required motor strength (torque) is constrained by the overall weight of the platform, which largely depends on the weight of the battery as discussed above.

- The distance ranging capability of the MSP is limited by the field of view of the range sensor. The ability to measure distance in more directions is highly desirable, as this would aid in localization and path planning around obstacles. It would also allow the platform to travel in reverse over large distances without bumping into obstacles, whereas the current design must turn around to face forward when traveling to a destination. There several ways in which this problem can be addressed.

  Firstly, range sensors are relatively cheap, so it is possible to place more of them around the perimeter of the MSP, which would allow the processing unit to obtain range measurements in several directions simultaneously. The problem with this solution is how to control multiple range sensors at once, and the minimal increase in overall power consumption contributed by each additional unit. Most commercially available MSPs utilize multiple sonar rangers facing in various directions.

  Secondly, it is possible to mount the range sensor on a servo motor to allow it to rotate from -90° to 90°. This would allow for range sensing in all directions using just two range sensors and two servos, or optionally mounting two range sensors on one servo on top of the MSP. The disadvantage to this setup would be the added complexity of controlling a servo motor to point in a certain direction to take a range measurement.

Another way to improve range sensing data would be to use a laser range scanner. While these devices offer much higher accuracy and resolution, they do so at a much higher cost than traditional sonar rangers.

- The camera used for both versions of the MSP has adequate resolution, but it has a limited field of view, which is essentially the same limitation that applies to the range sensor. A few of the options to increase the area that can be captured at once include using a wide angle lense, mounting the camera on a servo as was suggested for the range sensor above, or installing multiple cameras onto the MSP.

- In this version of the MSP, to prevent the computer PSU from being overloaded the motors and H-Bridge were powered by two 5V voltage regulators. Unfortunately these simple regulators are very inefficient, and it would be desirable to investigate more energy efficient circuits for the next MSP version.

- The overall design of the MSP still needs to be made more compact. Much space could be saved by implementing some of the suggestions outlined above, such as replacing the hard drive with a small memory card attached directly to the motherboard. Further, space usage could be optimized by mounting things closer together, which would have the downside of making assembly and maintenance more complicated.

5.1.2 Software and Algorithm Improvements

A large number of additional features can be added to the MSP cluster by implementing new algorithms in the software. This thesis successfully implemented centralized and distributed self-deployment, but there remain some improvements that can be made to those two algorithms. There are also a number of other features that would help to enhance the overall performance of either type of deployment algorithm. Some of the suggested improvements are listed below:

- The software is currently lacking mapping capability. The addition of a map building component would greatly enhance the autonomy of the system, as it

would allow for self-deployment without prior knowledge of the environment. The deployment strategy would have two separate steps. The first would be for each MSP to map its surroundings and collaboratively build a map given the result from each. In the second step the MSPs would then self-deploy, as they do currently. Map building can be quite difficult using only marker detection and range measurements. Image processing with edge/wall detection could prove very beneficial.

- In its current configuration it is necessary for the operator of the MSP cluster to decide on the mode of operation. It would be ideal if the MSPs could decide for themselves whether to execute a centralized or distributed self-deployment algorithm. Allowing for a hybrid mode between centralized and distributed operation modes would also be a major improvement. This could be structured so that MSPs start in a centralized mode by default, but fall back onto making their own decisions if they decide that they have waited too long for instructions from the MSP cluster head.

- To facilitate the easy design of the above items, it would be practical to make the code even more modular than it already is. The current class structure would make it challenging to switch between operating modes while one algorithm is already executing.

- For completely autonomous operation in any type of environment algorithms for natural landmark detection should be developed. The current setup requires that the structured markers are placed around the perimeter of the testing area, which means that the software cannot perform without a human previously placing such markers.

- The second generation MSP has added some features to allow for energy conservation when the MSP is not moving around. The software currently does not take full advantage of the mechanisms to put all devices possible into a low-power mode when it would be helpful. For example, an output signal has been added to the parallel port that allows for the H-Bridge to be disabled that is currently not used. The operating system manages power consumption of the

processor itself, but further steps could be taken to make this more effective. Also, the hard drive consumes a considerable amount of current even when its not being used. Sending the hard drive to sleep mode could increase battery life, but it should be investigated how much current is required to make it spin back up to full speed, in order to determine when it is worth to actually power it down.

Bibliography

Bibliography

[1] D. F. Abawi, J. Bienwald, and R. Dörner. Accuracy in optical tracking with fiducial markers: An accuracy function for artoolkit. In *Proceedings of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality*, page 8pp, September 2002.

[2] ARToolKit Team. *ARToolKit Website*, 2005. http://www.hitl.washington.edu/artoolkit.

[3] Scott Boskovich. *The Ninth Tomorrow Robots*, 2005. http://users.adelphia.net/ skbosko/.

[4] United States Coast Guard Navigation Center. *GPS General Information*, 2005. http://www.navcen.uscg.gov/gps/.

[5] Igor Cesko. *PC Infrared Receiver*, 2006. http://www.alldiy.info/IR-home.html.

[6] N. X. Dao, B-J. You, and S-R. Oh. Visual navigation for indoor mobile robots using a single camera. In *Proceedings of the Intelligent Robots and Systems*, pages 1992–1997, August 2005.

[7] DARPA. *Project Website*, 2005. http://www.darpa.com.

[8] Spark Fun Electronics. *Spark Fun Electronics*, 2006. http://www.sparkfun.coms.

[9] M. Fiala. Comparing artag and artoolkit plus fiducial marker systems. In *IEEE International Workshop on Haptic Audio Visual Environments and their Applications*, pages 148–153, October 2005.

[10] Inc. Grainger. *Grainger Product Catalog*, 2006. http://www.grainger.com.

[11] N. Heo and P.K. Varshney. An intelligent deployment and clustering algorithm for a distributed mobile sensor network. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, volume 5, pages 4576–4581, October 2003.

[12] J. Hightower and G. Borriello. Location systems for ubiquitous computing. 34(8):57–66, 2001.

[13] A. Howard, M. J. Mataric, and G. S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. In *Autonomous Robots, Special Issue on Intelligent Embedded Systems*, volume 13, 2002.

[14] D. Johnson, C. Berthiaume, and B. Witkowski. *Augmented Reality - ARToolkit Patternmaker*, 2002. http://www.cs.utah.edu/gdc/projects/augmentedreality/.

[15] A. M. Ladd, K. E. Bekris, A. P. Rudys, D. S. Wallach, and L. E. Kavraki. On the feasibility of using wireless ethernet for indoor localization. *IEEE Transactions on Robotics and Automation*, 20(3):555–559, June 2004.

[16] J. G. Lim and S. V. Rao. A grid-based location estimation scheme using hop counts for multi-hop wireless sensor networks. In *International Workshop on Wireless Ad-Hoc Networks*, pages 330–334, May 2004.

[17] Active Robots Ltd. *Active Robots - Wheel Encoders*, 2006. http://www.active-robots.com/products/motorsandwheels/wheel-encoders.shtml.

[18] G.W. Lucas. *A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators*, 2001. http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html.

[19] L. Miao, H. Qi, and F. Wang. Biologically-inspired self-deployable heterogeneous mobile sensor networks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Edmonton, Alberta, Canada, August 2005.

[20] U.S. Centennial of Flight Commission. *Centennial of Flight*, 2003. http://www.centennialofflight.gov/essay/Dictionary/Gyroscope/DI105.htm.

[21] C.B. Owen, F. Xiao, and P. Middlin. What is the best fiducial? In *First IEEE Internation Workshop, Augmented Reality Toolkit*, page 8pp, September 2002.

[22] L.E. Parker, B. Kannan, X. Fu, and Y. Tang. Heterogeneous mobile sensor net deployment using robot herding and line-of-sight formations. In *Proceedings of the Intelligent Robots and Systems*, pages 2488–2493, October 2003.

[23] N. Patwari, J. N. Ash, S. Kyperountas, A. O. Hero, R. L. Moses, and N. S. Correal. Locating the nodes: Cooperative localization in wireless sensor networks. *IEEE Signal Processing Magazine*, pages 54–68, 2005.

[24] H. Qi. *Advanced Imaging and Collaborative Information Processing Laboratory*, 2006. http://aicip.ece.utk.edu.

[25] J. Ryde and H. Hu. Fast circular landmark detection for cooperative localisation and mapping. In *Proceedings of the International Conference on Robotics and Automation*, pages 2745–2750, April 2005.

[26] E. Sahin and P. Gaudiano. Development of a visual object localization module for mobile robots. In *European Workshop on Advanced Mobile Robots*, pages 65–72, September 1999.

[27] SICK AG. *SICK Product Catalog*, 2005. http://ecatalog.sick.com.

[28] Siemens. *ARVIKA*, 1999. http://www.arvika.de.

[29] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. *IEEE Network*, 18(4):45–50, 2004.

[30] H. J. Sohn and B. K. Kim. A robust localization algorithm for mobile robots with laser range finders. In *Proceedings of the International Conference on Robotics and Automation*, pages 3545–3550, April 2005.

[31] Y. Tang, B. Birch, and L.E. Parker. Planning mobile sensor net deployment for navigationally-challenged sensor nodes. In *Proceedings of the International Conference on Robotics and Automation*, volume 1, pages 172–179, April 2004.

[32] H-J. von der Hardt, D. Wolf, and R. Husson. The dead reckoning localization system of the wheeled mobile robot romane. In *Proceedings of the British Machine Vision Conference*, pages 603–610, 1996.

[33] Wikipedia. *Photoresistor - Wikipedia*, 2006. http://en.wikipedia.org/wiki/Photoresistor.

[34] X. Zhang, S. Fronz, and N. Navab. Visual marker detection and decoding in ar systems: A comparative study. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, pages 97–106, September 2002.

Vita

Christopher Allan Beall was born in Frankfurt, Germany, on June 15th 1982. He went to school at the Gymnasium Gernsheim in Germany, until age 16. He then moved to the Knoxville, Tennessee area with his family. He graduated from Lenoir City High School after attending for two years. In the fall of 2000 he began his undergraduate study at the University of Tennessee in Knoxville, with a major in Computer Engineering. Beginning with his sophomore year he had an Undergraduate Assistantship with the Engage Freshmen Engineering Program. He earned his Bachelor of Science degree in 2004, and in the fall of the same year started pursuing his Master of Science degree in Computer Engineering. During his graduate studies he held a Teaching Assistantship with the Engage Freshmen Engineering program, where he taught introductory physics and computer courses for two years. During the summer of 2005 he then joined the Advanced Imaging and Collaborative Information Processing lab lead by Dr. Hairong Qi. In the lab he began work on implementing localization and self-deployment of visual sensor networks, using a real testbed. He will be graduating with a Master of Science Degree in Computer Engineering in August, 2006.