6-2006

Robot Localization Using Visual Image Mapping

Carrie D. Crews

ROBOT LOCALIZATION USING VISUAL IMAGE MAPPING

THESIS

Carrie D. Crews, First Lieutenant, USAF

AFIT/GCS/ENG/06-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCS/ENG/06-03

ROBOT LOCALIZATION USING VISUAL IMAGE MAPPING

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Carrie D. Crews, BS

First Lieutenant, USAF

March 2006

AFIT/GCS/ENG/06-03

ROBOT LOCALIZATION USING VISUAL IMAGE MAPPING

Carrie D. Crews, BS

First Lieutenant, USAF

Approved:

_____	_____
Dr. Gilbert Peterson (Chairman)	Date
_____	_____
Dr. John Raquet (Member)	Date
_____	_____
Andrew Learn, Maj, USAF (Member)	Date

Acknowledgments

I would like to give thanks to God for blessing me with a group of people integral to my successful completion of this work. First, my family is the cornerstone of my strength and sanity and I am ever so grateful to have them to go home to each day.

Secondly, I extend my profound gratitude to Major Theresa Jamison for being my friend, mentor, and sister in spirit. I am a better person having known her.

Lastly, I can't express the impact my advisor, professors, and fellow students have had on my experience here. I fully appreciate every second I spent in their company.

Carrie D. Crews

AFIT/GCS/ENG/06-03

Abstract

One critical step in providing the Air Force the capability to explore unknown environments is for an autonomous agent to determine its location. The calculation of the robot's pose is an optimization problem making use of the robot's internal navigation sensors and data fusion of range sensor readings in calculating the most likely pose. This data fusion process requires the simultaneous generation of a map which the autonomous vehicle can then use for obstacle avoidance, communication with other agents in the same environment, and target location. Our solution entails mounting a Class 1 laser to an ERS-7 AIBO. The laser projects a horizontal line on obstacles in the AIBO camera's field of view. Range readings are determined by capturing and processing multiple image frames, resolving the laser line to the horizon, and extracting distance information to each obstacle. This range data is then used in conjunction with mapping and localization software to accurately navigate the AIBO.

Table of Contents

Page

Abstract iv
Acknowledgments v
Table of Contents vi
List of Figures viii
ROBOT LOCALIZATION USING VISUAL IMAGE MAPPING 1
I. Introduction 1
1.1 Rationale 1
1.2 Problem Statement 2
1.3 Approach 3
1.4 Thesis Outline 5
,II. Literature Review 7
2.1 Localization 8
2.2 Localization Algorithms 10
2.3 Striping Laser and Camera Implementations 25
2.4 Simultaneous Localization and Mapping (SLAM) 29
2.5 Summary 33
III. Methodology 35
3.1 Overview 35
3.2 The AIBO 36
3.2 Capturing Images 38
3.4 Extracting Laser Line 42
3.5 Estimating Horizon Line 44

3.5 Determining Distance 50

3.6 Determining Pose 58

3.8 Localization 62

3.9 Summary 62

IV. Results and Analysis 63

4.1 Sensor Model 63

4.2 Pose Model 67

4.3 Mapping 69

4.5 Conclusion 75

V. Future Work and Conclusions 76

5.1 Estimation and Assumption Alternatives 76

5.2 Future Extensions 79

5.3 Conclusions 80

Appendix A 82

Appendix B 89

Appendix C 99

List of Figures

Page

Figure 1: Pixel Projection 24

Figure 2: Camera/Laser Triangulation 26

Figure 3: Camera/Laser Geometric Model 28

Figure 4: Theoretical Concept of Determining Distance Using Horizon and Striping Laser 35

Figure 5: Tekkotsu Data Flow 37

Figure 6: Camera Configuration 38

Figure 7: YUV to RGB Conversion 40

Figure 8: Color Vision Train 41

Figure 9: Laser Line Colors 42

Figure 10: Horizon using Vanishing Lines 44

Figure 11: Horizon Using Intersecting Planes 45

Figure 12: AIBO Diagram 1 46

Figure 13: AIBO Frame Translations 47

Figure 14: AIBO Frame Translation 48

Figure 15: Triangulation 50

Figure 16: Sensor Model Error 53

Figure 17: Sensor Model 54

Figure 18: Geometric Sensor Model 55

Figure 19: AIBO/Pioneer Coordinate Systems 56

Figure 20: AIBO Motion Tracking 57

Figure 21: (left)Straight Walk: 1 meter (Right) Angular Walk: 1 meter 58

Figure 22: Sensor Model Data Statistics(left) and Curve Fit Equation Surface Plot..... 60

Figure 23: Sensor Readings Without Walk 64

Figure 24: Sensor Readings While Walking..... 64

Figure 25: Sensor Readings of a Wall without Head Motion 65

Figure 26: Sensor Readings with Pan Motion 65

Figure 27: Laser Lines in Different Lighting..... 66

Figure 28: Hallway Maze, (left) Top View, (right) robot's view 67

Figure 29: Continuity of Control: Left Turn Pose Tracking. (Left) Actual plotted pose data, (Right) Mapped pose data 68

Figure 30: Continuity of Control: Right Turn Pose Tracking. (Left) Actual plotted data, (Right) Mapped pose data 68

Figure 31: Accumulated Pose (x,y) Error 69

Figure 32: AIBO Mazes..... 70

Figure 33: Remote Controlled Navigation Through Hallway Maze, Trial 1..... 71

Figure 34: Remote Controlled Navigation Through Hallway Maze, Trial 2..... 71

Figure 35: Programmed Navigation Through Simulated Hallway Maze, Trial 1 72

Figure 36: Programmed Navigation Through Simulated Hallway Maze, Trial 2 72

Figure 37: Simulated Maze with Small Left Turn 73

Figure 38: Simulated Maze with Small Right Turn..... 73

Figure 39: Continuous Turn Pose Estimations: (Left) Left Turn, (Right) Right Turn 74

Figure 40: Simulated Round Maze: Off The Ground 74

Figure 41: Single Run Through Maze 75

Figure 42: Three Runs Through Maze..... 75

ROBOT LOCALIZATION USING VISUAL IMAGE MAPPING

I. Introduction

The ability for a robot to localize itself is a critical step in creating a fully autonomous robot. Essential to localization is the relationship between a map of the robot's environment and its sensor, from which the robot is able to localize (determine its location in the environment). Localization and mapping requires two accurate pieces of information: sensor information which is interpreted as the locations of objects in the robot's environment and a mathematical representation of the motion of the robot in question. Although both sensor and motion models are important, the motion model must be as accurate as possible, since it provides correct robot estimation of its current location in the world.

This research focuses on studying the localization of a quadruped robot while creating and correcting a physical map derived solely from information provided by the robot's vision system.

1.1 Rationale

The evolution of robotics in commercial service provides a new facility for exploring environments without risking the loss of human life. A well-known and publicized application is the remote controlled vehicle used by bomb squads [23]. This application easily extends to use in military missions [23]. As the terrorist community grows more fearless and ever stealthier, such vehicles provide the military with the means

of exploring environments too hazardous for human entry. The enemies of this country will go to any necessary means to hide themselves and their weapons. This includes "booby trapping" buildings including schools, hospitals, and office buildings. The robot's ability to navigate these types of environments autonomously significantly reduces the loss of life often suffered during human exploration. The legged robots have two advantages over wheeled or tread robots—1.) the ability to explore environments with rougher terrains 2.) the ability to fit into smaller enclosures.

This research envisions sending a robot with a striping laser into an unknown environment to collect data. The striping laser provides a more accurate sensor reading than traditional sonar sensors that have a 30° cone of possible locations for each sensor reading. The striping laser reduces the noise the 3D cone to a 2D range of possibilities. The laser is also small and lightweight as not to add excessive proportion and weight to the robot. As the robot navigates the rooms or buildings, it sends its images to a source that processes the information and extracts physical features from the environment.

1.2 Problem Statement

Localization and mapping solutions are successful under constraints of specialized environments using various types of object detection algorithms. The problem becomes more complex when applying these solutions to dynamic environments. The concept of using vision as a sensor for object detection centers on being able to detect patterns in the image corresponding with known features in the environment. The goal of this research is to overcome the hurdle of dynamic environments and stray away from the "known" by using components common to all

images (introducing one such component using a striping laser) to detect features of the unknown environment. Utilizing this information with a Monte Carlo localization technique allows the robot to build a physical map of "where" it has been and accurately estimate "where" it is currently.

1.3 Approach

We address this problem by representing one image collected from a host's camera as a single "scan" from a generic sensor. Two representations of each image are processed--segmented and raw. The segmented image is used to derive the location of the laser line in the image. The horizon line of the image is estimated and projected onto the raw image. Once the locations of these two lines are located, they are moved to mirror their true positions in the real world. This process requires the horizon's rotation angle to be determined, then rotating both lines by the negative of that angle. On order to compensate for the nodding of the head, the laser line is translated to mirror the distance between the horizon and the center of the image. The distance information provided by comparing the laser line and horizon line in each image provides us with information from which we build a local map. The map consists of two vital pieces of information--the robot's position in the environment and the distances to obstacles reported by the sensors. The distance measurements are calculated relative to the base of the robot, compensating for pan motion of the head as well as the geometric relationship between the laser line pixels and the center of the robot. The mapping software utilizes this information to determine the location of the obstacle in its grid-based world. The robot's position is derived from the distance the robot has traveled since its last image was

captured and processed and the angular velocity the robot is traveling, from which rectangular coordinates are derived. The location on the mapping software's grid is based on the accumulated rectangular coordinates and the direction the robot was last facing. The local maps are calculated and maximized separately, first forward during mapping, then backward over previously constructed local map. The global map is corrected as a result of the local maps being reconstructed based on the maximized pose.

The theory behind the vision component of this application is the traditional landmark detection using segmented vision. In previous research, the landmarks are distinguished by color and/or pattern and the images are segmented, extracting these colored features[12, 20, 22, 30, 35]. The robot's pose is estimated by identifying the landmarks captured in the image. As an alternative, we replace the color-coded landmarks with a laser line, projected into the image to define the shape and orientation of the objects, versus capturing and processing the entire detected object. Since the laser line segments are not natural to the image, but introduced by the laser-mounted robot, the dependency on a specific environment is reduced in this research. Additionally, the estimated horizon of the image is also based on the robot stance instead of the traditional technique of using sets of vanishing lines extracted from the image. As a result, our local maps can be built in various environments (light-dependent) since we don't depend on the natural image itself to detect obstacles, but instead use the image as a plane upon which these two lines in are projected in order to measure the distance between them. This distance measure is intended to provide the same information as any other distance feedback sensor.

1.4 Thesis Outline

Chapter II presents a history of different localization solutions implemented with varying level of success. This includes three implemented algorithms: Kalman Filter, grid-based Markov, and Monte Carlo localizations, the latter being the most frequently applied to localizing the AIBO mobile robot. Additionally, the *simultaneous localization and mapping (SLAM)* problem is presented with a brief discussion of the solution used in this research. Although these solutions are widely used and modified, the implementation of this research included two known variables: one, it is being applied to the AIBO robot, and two, it is using the AIBO's vision system as the primary sensor. Hence, Chapter II also reviews several systems which localize the AIBO using modified versions of the Monte Carlo localization algorithm and the combination of Markov localization with Kalman filtering. These extended versions of the original solutions utilized vision for the sensor model. Each of the cited solutions is accompanied by the challenges associated the algorithms as well as proposed modifications to improve performance.

Due to the unique nature of the laser and vision distance extraction technique, three demonstrations of collaboratively using vision with a Class 1 striping laser are reviewed.

Chapter III documents the theory behind the steps necessary to process images, estimate the horizon, extract the laser line, and develop a *sensor* and *motion model* for the AIBO robot. The actual methods used in implementing the theoretical concepts are also covered in the chapter. For each successive step, we describe our prevailing assumptions

and compromises made to accommodate constraints of reduced complexity, computation synchronization, and eventual real-time implementation.

Chapter IV describes the analysis of the SLAM solution using the information and the techniques described in Chapter III. Testing maps without localization which determine the accuracy of both the object detection using the horizon/laser line distance correspondence and pose estimation are developed through a series of physical tests and calibrations. The specific nature of these calibrations is described in Chapter III. Once familiar with the impact of the *sensor model* and pose estimation on the accuracy of the mapping computations and resigned to a threshold of inaccuracy, the results from the mapping algorithm are collected and analyzed. For the rest of Chapter IV, we analyze the localization calculations and determine each parameters' influence on the outcome and make adjustments which improve accuracy.

Finally, Chapter V reviews the estimates made throughout the implementation discussed in Chapter III. For each estimate, reasoning is provided as well as possible alternative processes for making the estimation that may result in improved accuracy. The final section of the chapter provides a brief overview of the conclusion that were drawn from the research, extensions for future development, and recommendations to improve this specific research topic.

,II. Literature Review

This chapter presents related research addressing the problem of localization with mobile robots. The specific interest of this research concerns implementing a localization solution, using images generated from the onboard camera on an AIBO robot mounted with a split-beam infrared laser.

Many techniques focus on allowing mobile robots to move about their environment autonomously. Autonomous navigation relies on the interpretation of information from the robot's sensors which, is filtered and produces data used in determining the current position of the robot in its environment. There exist several approaches in proposing solutions to the localization problem. Given that sensor data is far from absolute and not necessarily reliable, a need remains for estimation when using this data to calculate the robot's pose explaining why the most successful localization solutions are probabilistic in nature. A brief review of the localization problem and three popular solutions--Monte Carlo localization, grid-based Markov localization, and localization using the Kalman filter--are included in the following sections. Many of these techniques depend on having a good map of the physical environment. In dynamic environments, where a map isn't available, it is possible to build the map using *simultaneous localization and mapping (SLAM).* This chapter includes a section devoted to research on this subject.

This research intends to utilize an AIBO as its agent which moves about and gathers images, providing information about the environment from which *SLAM* software builds a map. There are many research projects which involve localizing an AIBO using

vision. Subsequent sections introduce several such localization implementations, using the AIBO's camera, or its camera coupled with its laser range finder, to provide sensor data.

The final set of concepts this chapter presents are those involving the use of a striping laser. In these publications, lasers are used for determining distance by triangulating between the laser and a camera, aligning images, and for performing camera calibrations.

2.1 Localization

Localization is a fundamental capability requirement to make significant headway in the development of a pure autonomous robot. A map of its environment, a history of its sensory perceptions, and its recently executed actions are the three categories of information robots required to deduce their current position in the environment (pose). This deduction is broken down into two key problem areas [27]: global position estimation (GPE) and local position tracking (LPT). The first is the most complex, since the robot's position must be determined without any *a priori* pose information. In contrast, the LPT problem begins as soon as the robot has localized itself within its map, keeping track of the robot's position as it moves over time. The common thread among the solutions is that the state of the robot is a vector consisting of the *(x,y)* position and orientation, θ, at a any given time *T*.

$$\mathbf{x} = [x, y, \theta]^T \tag{1}$$

The estimation of this state is "an instance of Bayesian filtering problem where we are interested in constructing the posterior density." [27]

$$p(\mathbf{x}_k \mid Z^k) \tag{2}$$

Where Z^k, the set of all measurements, is collected by the sensor up to the k^{th} sample $\{z_k, i = 1..k\}$ and x_k represents one of all possible states. The decision of how to represent the distribution (1) is the primary factor making each localization solution unique. As much as these solutions are diverse, they share a common recursive formula consisting of a *Prediction Phase* and an *Update Phase* used in computing the posterior density at each time step. Each phase of the computation uses two mathematical models in deriving an approximated representation of the robot's state.

The Prediction Phase uses a *motion model* in predicting the current position of the robot, represented as this predictive probability distribution function:

$$p(\mathbf{x}_k \mid Z^{k-1}) \tag{3}$$

This model makes use of the Markov assumption, in that the state of the robot is only dependent on its previous state ($\mathbf{x}_{k-1}$) and some known control input (**u**),

$$p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_{k-1}) \tag{4}$$

This equation denotes the probability of our current position, given our previous position and last control input, allowing the computation of the corresponding predictive probability using integration:

$$p(\mathbf{x}_k \mid Z^{k-1}) = \int p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_{k-1}) p(\mathbf{x}_{k-1} \mid Z^{k-1}) d\mathbf{x}_{k-1} \tag{5}$$

This phase ultimately computes the probability that the estimated pose is accurate by applying the *motion model* to the estimate.

The Update Phase uses the information from the sensors in a *measurement model*

$$p(\mathbf{z}_k \mid \mathbf{x}_k) \tag{6}$$

representing the likelihood that $\mathbf{z}_k$ is observed given its current pose $\mathbf{x}_k$. Using this model, the posterior density is calculated using Bayes theorem:

$$p(\mathbf{x}_k \mid Z^k) = \frac{p(\mathbf{z}_k \mid \mathbf{x}_k) p(\mathbf{x}_k \mid Z^{k-1})}{p(\mathbf{z}_k \mid Z^{k-1})} \tag{7}$$

This two-phase process is recursively performed over previous states until it reaches the initial state, which is handled differently for the GPE and LPT problems. An important fact to remember is that there is no *model* that is perfect, especially when modeling a system using information from sensors. Once again, sensors do not necessarily provide accurate or complete data from which to build these system representations. The following section presents a few solutions to the localization problem.

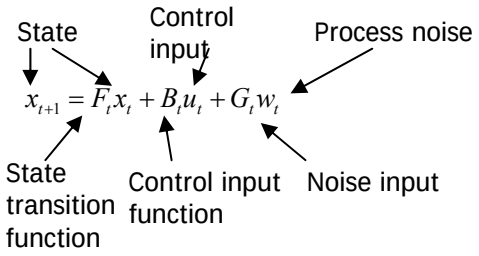
2.2 Localization Algorithms

There are three popular solutions to the localization problem having various levels of implementation success: Kalman-filter based, Markov grid-based, and Monte Carlo localization. Each solution is developed in hopes of mitigating *sensor model* and *motion model* inaccuracies.

2.2.1 Kalman Filter

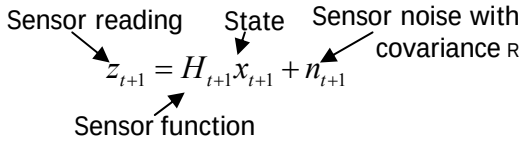
The first and most straightforward approach at solving the localization problem is using pure Kalman filters. The Kalman filter is a recursive data processing algorithm [16] which processes all measurements provided to it, producing an estimate of the value of the variable of interest. The filter uses three pieces of information in calculating this estimate: knowledge of the system and measurement device dynamics, statistical

description of the system noises, measurement errors, and uncertainty in the dynamics models, and any available information about the initial conditions of the variable of interest [16]. In the case of localization, both the *motion* and *measurement* (sensor) *models* are Gaussian distributions. The Kalman filter is composed of two components, the motion model [22]:



(9)

which is used in the *Prediction Phase* to estimate the current position based on its previous position, and the measurement model:



(10)

The *sensor model* is used during the pose update phase for refining the current pose using sensor data. When these components are used with the calculations to compute the Minimum Mean Square Error (MMSE) estimate of the state and covariance, the Kalman filter's resulting representation of the localization *motion model* [22] is:

$\hat{x}_{t+1/t} = F_t \hat{x}_{t/t} + B_t u_t$	- State estimate is updated from system dynamics
$P_{t+1/t} = F_t P_{t/t} F_t^T + G_t Q_t G_t^T$	- Uncertainty estimate *GROWS*

and the sensor model is:

$\hat{z}_{t+1} = H_{t+1} \hat{x}_{t+1/t}$	- Compute expected value of sensor reading
$r_{t+1} = z_{t+1} - \hat{z}_{t+1}$	- Compute the difference between expected and "true"
$S_{t+1} = H_{t+1} P_{t+1/t} H_{t+1}^T + R_{t+1}$	-Compute covariance of sensor reading
$K_{t+1} = P_{t+1/t} H_{t+1}^T S_{t+1}^{-1}$	- Compute the Kalman Gain (how much to correct est.)
$\hat{x}_{t+1/t+1} = \hat{x}_{t+1/t} + K_{t+1} r_{t+1}$	- Multiply residual times gain to correct state estimate
$P_{t+1/t+1} = P_{t+1/t} - P_{t+1/t} H_{t+1}^T S_{t+1}^{-1} H_{t+1} P_{t+1/t}$	- Uncertainty estimate SHRINKS

Table 1: Kalman Filter Equations

It is noteworthy to highlight the linear nature of these equations. When applied to mobile robots, the linearity of the filter introduces a limitation in that mobile robot dynamics are not linear. Hence, the nonlinear system of a robot must be modeled with a linear process model by making some small-angle assumptions. The linearization of the system results in an increase in state error residual since it is not the best estimate. The weakness of using pure Kalman filters in localization is that only one hypothesis can be represented if the filter's optimality is to be maintained. Additionally, the filter is not capable of handling 1) non-Gaussian *motion* and *sensor models*, 2) multi-modal densities of global localization, and 3) is unable to recover from local tracking failures. Most of these

weaknesses can be corrected with some extensions of the Kalman filter, but they cause the solution to be less optimal [22,19].

Although there is loss optimality in applying an extension of the Kalman Filter [EKF] to localization, it has been successfully used to build a map of the robot's environment while localizing. In [20], a feature-based concurrent mapping and localization, also known as SLAM, algorithm is introduced. As localization solutions need to be applicable to dynamic environments, the proposed solution performs SLAM without the *a priori* knowledge of a global map or known robot location. The technique presented in [20] initializes a local map relative to the current vehicle location upon initialization of a *motion*. At each step of the *motion*, the EKF prediction and updating algorithm is used to estimate the vehicle's current location and location of environmental features using the sensor and motion models described in the previous text. Other than avoiding optimistic estimations by using the EKF, the only additional component to consider in using Kalman Filters for mapping is developing a hypothesis associating the sensor return with its corresponding feature in the map. The application of a Hough transform along the vehicle locations of a local map gives a hypothesis $H_t = [j_1, j_2, ..., j_s]$ associating each sonar return *i,* at instant *t* with *i*=1,..*s,* with its corresponding feature F_{j_i} [20]. The theoretic distance from *i* to F_{j_i} is a function of the vehicle and feature location, $\mathbf{h}_{ij_i}$, found in the map state vector. This resulting distance measure is given as

$$\mathbf{z}_t = \mathbf{h}_t(\mathbf{x}_t) + \mathbf{w}_t$$

where $\mathbf{w}_t$ is measurement noise. Linearization around the current map estimate yields:

$$\mathbf{z}_t \cong \mathbf{h}_t(\hat{\mathbf{x}}_{t|t-1}) + \mathbf{H}_t(\mathbf{x}_t - \hat{\mathbf{x}}_{t|t-1})$$
$$\mathbf{H}_t = \frac{\partial \mathbf{h}_t}{\partial \mathbf{x}_t}|_{(\hat{\mathbf{x}}_{t|t-1})}$$

which is used to obtain a new estimation of the state using the standard EKF update equations listed in Table 1. Characteristically, the resulting local maps are independent of any prior knowledge and only depend on the odometry readings and sensory data collected during the steps and the data association hypothesis [20]. As they become available, each local map is added to the stochastic global map. Compatible features found in both the local and global maps are gathered and used to update the global map.

2.2.2 Markov Model

The second approach to solving the localization problem is the Markov method. This approach is directed at the global localization problem by maintaining a probability density over the space of all locations of a robot in its environment in order to globally estimate the position of the robot in its environment. One variant of the Markov method [7] uses a fine-grained and metric discretization of the state space, providing more accurate position estimates and the ability to incorporate raw, unfiltered, sensory input. It also addresses the general assumption that the robot's environment is static, making it vulnerable to failure in dynamic environments. This vulnerability is overcome by only updating the probability density with measurements produced by objects that are very likely to be contained in the robot's map. The Markov model addresses the limitation of only maintaining a single hypothesis as found with the Kalman filter approach by

maintaining a probability distribution over the space of all such hypotheses. In this approach, the sensor data z_t can be either camera or odometry readings $(d_0,...,d_{T-1})$, and the constant

$$\alpha_T = \frac{1}{P(z_t \mid d_0,...d_{T-1})}$$

is independent of the random variable corresponding to the true location at time $t(L_T)$. The actual location of the robot is not known, but can be represented as a probability, computed as

$$p(x_k) = \alpha_T P(z_T \mid \mathbf{x}_k) p(\mathbf{x}_k^{T-1}), \tag{11}$$

when the most recent data received is a sensor measurement, or

$$p(x_k^T) = \int P(\mathbf{x}_k^T \mid a_T, x) p(\mathbf{x}_k^{T-1}) dl', \tag{12}$$

where the data is an odometry measurement. This belief equation computes the probability distribution $p(x_k^T)$ of the possibility that its location at time T is x_k. The equation denotes the sensor measurement as z and odometry reading by a. In this representation, the *motion model* is denoted by the $P(\mathbf{x}^t \mid a, \mathbf{x}^{t-1})$ while the *perception model* is denoted as $P(z \mid \mathbf{x})$. Since this belief is most often approximated via a fine-grained grid, it is able to represent multi-modal distributions, unlike its Kalman filter counterpart. It also eliminates the need to use landmarks to estimate the position of the robot, thus allowing the raw sensor data to be incorporated into the belief update. Unlike most other Markov-based algorithms, the set of distances used to compute $P(z \mid x)$ only

includes the distance to the closest obstacle in the direction of that sensor, reducing computational complexity for real-time implementation.

2.2.3 Monte Carlo Localization

The final solution presented is the Monte Carlo Localization method. As noted in the previous discussion, each approach differs in the chosen representation of the probability densities. In this method [27], the density function $p(\mathbf{x}_k \mid Z^k)$ is not directly described, but represented using a set of random samples ($S_k = \{s_k^i; i = 1..N\}$) taken from $p(\mathbf{x}_k \mid Z^k)$. Such methodology is taken from earlier work done on Bayesian filtering with particle-based density representation and applied in this approach by using Monte Carlo methods to update the probability density. The samples approximately reconstruct the probability density, and then the sample set S_k is recursively computed at each time step *k*. A general particle filter such as the bootstrap or Monte Carlo filter can be used to perform this recursive computation. This algorithm also prescribes to the Prediction and Update Phase process. In the Prediction Phase, the motion model is applied to each sample s_{k-1}^i in the set of previously computed samples S_{k-1} by sampling $p(\mathbf{x}_k \mid a_{k-1,} s_{k-1}^i)$ resulting in a new sample $s_k'^i$ as a member of the new set S_k' that approximates a random sample from $p(\mathbf{x}_k \mid Z^{k-1})$. At this point, there has been no sensor data, $\mathbf{z}_k$ incorporated into this approximation, which leads us to the Update Phase. Here, the measurement $\mathbf{z}_k$ and the (weight) likelihood of the each sample in S_k' given the sensor

measurement $m_k^i = p(\mathbf{z}_k \mid s_k'^i)$ are taken into consideration in the obtaining a new S_k by getting one sample s_k^j from the weighted set $\{s_k'^i, m_k^i\}$. By resampling, $s_k'^i$ has a high likelihood associated with it, so S_k approximates a sample from $p(\mathbf{x}_k \mid Z^k)$. These phases are performed recursively until time *k=0* is reached. Like the Markov approach, the Monte Carlo method is successful in overcoming the single-modal weakness of the Kalman filter by representing multi-modal distributions which is key to global localization in the robot's world. Meanwhile, it uses far less memory than the Markov grid-based approach and is more accurate in its approximation.

The solutions presented above are algorithms addressing localization problems for mobile robots, implemented on different robots with varying levels of success. The next section of this paper addresses localization solutions implemented on the particular robot of interest in this research, the AIBO.

2.2.2 AIBO Specific Localization Implementations

The predominant work in the area of localization with the AIBO robot consists of directly targeting requirements for the RoboCup Quadruped League competition. Most competing teams implement the Monte Carlo Localization algorithm on the AIBO, utilizing the camera, infrared laser, or a combination of both for collecting sensor data. The tournament takes place in a specialized environment, a 280 cm x 180 cm playing field. The images are used to locate specific landmarks (goals, markers, and flags) and use their dimensions and known locations to estimate distance to them [31]. In other applications, the same color-dependent concepts are used with the exception of extracting

edges of the landmarks instead of processing the entire object [21]. The techniques for processing the images for distance data all have one common thread--they are dependent on the static environment and color-coded features with known locations upon which they derive pose. In addition to being constrained to the field, AIBO-specific localization methods are limited by the AIBO's processing power and on-board camera. Although the camera, a 350k pixel Charge-Coupled Device (CCD) capable of 16.8 million colors output and producing up to 30 frames per second of real video, the robot is unable to process the images to this detail so the standard output is a 176x144 pixel image [3]. The cumulative constraints of the research presented in the following sections are taken into consideration in the implementation described in the next chapter.

2.2.2.1 Landmark-Based Localization

Due to the regulated environment of the Robocup competition, the most widely used technique for determining the pose of the robot on the field is by landmark detection. Several algorithms utilize this methodology in developing localization solutions for the AIBO. An experiment conducted in [9] uses variants and combinations of the Kalman filter, Markov, and Monte Carlo localization algorithms and implements them on the AIBO robot. One of these applications models the data derived from images, accompanied by range data from the AIBO's infrared laser, in the sensor model. Results indicate that in comparing these three approaches [9] the more robust and accurate solution is achieved by combining Markov localization with Kalman filtering (ML-EKF). Each of the variants is briefly presented in the next section.

The variant described in [9] is a unique combination of the Markov localization (ML) with the Kalman filter (EKF), which uses a two-dimensional Markov localization grid containing only possible robot positions, not its orientation. Landmark observations are integrated into this grid. If this observation has a high probability of being true, it is integrated into the EKF also. If this occurs, the distributions of the ML grid and the EKF are compared using a χ^2 test [9]. Although this approach performs quick computations and efficiently outputs the EKF state, it is limited when using dense sensor matching instead of landmarks for navigation.

The second solution is a variant of the Monte Carlo Localization (MCL). Similar to other versions of MCL, the concept consists of a random weighted sampling to represent the probability distribution. Different versions of MCL are developed by modifying the method used for adding samples to the sample set. The first method (SRL) is sensor resetting localization, where the samples are drawn according to the likelihood of the accuracy of the current observation. Samples, or fractions of samples, are added when the average likelihood of the observation $\tilde{p}$ exceeds the threshold p_t.

$$\tilde{p} = \sum_i p(\mathbf{z}_n \mid \mathbf{x}_i) / n$$

This equation denotes $\mathbf{z}_n$ as the sensor measurement at iteration n and $\mathbf{x}_i$ as robot position. The second method (Mix-MCL), adds a fixed number of samples to the distribution, adds the current probability density to the weight of the sample. This method was developed for extremely accurate sensor information [9]. The final method (A-MCL) uses the combination of two smoothed estimates of the observation likelihoods, one being

a long-term average and the other a short-term average of the observation likelihood. This method only adds samples if the short-term estimate is less than the reciprocal of the long-term average.

To compare the ML-EKF, SRL, A-MCL, and Mix-MCL solutions, an AIBO was programmed to observe colored landmarks. The best results were found in the ML-EKF, SRL1, and A-MCL. The EKL didn't deal with noise well, SRL2's parameter settings increase the uncertainty, and Mix-MCL adds to the weight of the above-mentioned noisy samples, further increasing the uncertainty. In the kidnapped robot problem or global localization, ML-EKF, SRL2, Mix-MCL, and A-MCL prevailed for recovery time, but the Mix-MCL required the most processing time. The fastest, but least successful in the experiment was EKF.

Several other competitors use the MCL approach in conjunction with landmark detection. In [24], a case study describes the enhancements of MCL algorithm to increase accuracy and performance in a mobile legged robot reliant on only its vision system. The localization solutions discussed in previous sections tried to improve performance of their algorithms by making modifications to the sampling techniques. In the approach taken in [24], adjustments are not constrained to sampling methods. To gain the desired level of accuracy, the basically competent variant of MCL is enhanced with three additional components:

1. Maintain a history of landmarks
2. Update estimates using *empirically-computed landmark distance model* in addition to heading

3. Tuning and extending the motion model for improved odometry calculation

The prevailing research used as a foundation for this particular solution also came from the arena of RoboCup legged soccer. In such studies, the *sensor model,* described earlier in this document, updates are based on sensed locations of landmarks that are known to be in the environment. Knowing its current location, the robot then determines the *expected* bearing angle of each of the landmarks seen in the current frame, $\alpha_{\exp}^{(l)}, l \in L$. The posterior probability of a single observation is then estimated based on how well the measured bearing $\alpha_{meas}^{(l)}$ matches the expected bearing $\alpha_{\exp}^{(l)}$ of the sensed location *s*.

$$s(\alpha_{meas}^{(l)}, \alpha_{\exp}^{(l)}) = \begin{cases} e^{-50\omega_l} if \omega_l < 1 \\ e^{-50(2-\omega_l)} otherwise \end{cases} \quad [24]$$

In this equation, angular velocity, $\omega_l = \frac{\left|\alpha_{meas}^{(l)} - \alpha_{\exp}^{(l)}\right|}{\pi}$ and the resulting probability of $p = \prod_{l \in L} s(\alpha_{meas}^{(l)}, \alpha_{\exp}^{(l)})$. Finally, the particle's probability is updated with the filter function[22]:

$$p_{new} = \begin{cases} p_{old} + .01 & if \quad p > p_{old} + 1 \\ p_{old} - 0.05 & if \quad p < p_{old} - 0.05 \\ p & otherwise \end{cases} \quad [36]$$

What makes this sensor model unique is that the distances to the landmarks are ignored, since their estimates are quite noisy when using vision and their calculations have a non-linear bias that degrade localization. This *sensor model* is capable of handling the kidnapped robot problem using a unique version of *reseeding.* Unlike traditional MCL

reseeding methods, the version in [24] does not require two landmarks to be seen concurrently because it keeps a history of landmarks, whose distance and angle are adjusted each frame based on the known pose. When a landmark is seen in successive frames, the distance and angle measurements are averaged, weighted by their confidence, and then used as input for reseeding. These archived landmarks are deleted from the history if they exceed an age threshold, or if the robot has been moved a significant distance. To incorporate the noisy distances into the update phase of the calculation, a corrective function (based on X and Y coordinates) is used to improve the distance estimate within a 5% error:

$$y_i \mid_{y_i \in Y} = a_0 + a_1 x_i + a_2 x_i^2 + a_3 x_i^3 \mid x_i \in X$$

where a_i are estimated coefficients derived when provided measured values x and actual values y.

The motion model of the basic MCL used in this experiment $p(x_n^T \mid a^{T-1}, \mathbf{x}_k^{T-1})$ where $\mathbf{x}_k^{T-1}$ is the old pose estimate, a^{T-1} is the last action command, and x_k^T is the new pose estimate [24]. This model was extended to eliminate the oscillation around the target location by allowing the robot to move at full speed until it comes within a threshold distance from the target location. Once inside 300*mm* of the target, its speed is reduced to $\frac{1}{10}$ its normal speed. This reduction in speed significantly reduced oscillation and improved localization accuracy.

When all three enhancements were implemented, tests of the localization accuracy were conducted using an ERS-7 robot achieving a 50% reduction in position

error and >50% improvement in orientation without a significant increase in the time taken to reach the target location. In addition to its ability to move to a target location, another test of the ability to stay localized to the target, *stability,* was performed. The extended motion model produced over 50% increase in distance accuracy and over 35% orientation accuracy with less than one second of additional processing time. When presented with collisions or kidnapping problems, the enhanced MCL reduced the increase in error to only 56%. The algorithms in [9] and [24] share the same process of identifying landmarks, they segment the image for the colors they *know* identify the landmarks, then process the landmarks for their size and dimensions. To reduce the amount of image processing necessary, other solutions rely on using edges instead of distance to objects.

In [36], the algorithm is weaned from processing the entire image, extracting only those features needed for localization, making it less impacted by lighting. This localization solution, based on MCL and using landmark detection, was developed by a team of students in Germany in preparation for the RoboCup soccer tournament. Traditionally, preprocessed (segmented) images are used in detecting the features needed for localization. This segmentation labels the pixels in a manner that ignores the influence of surrounding pixels. Since this solution is targeting known features (flags, goals,), a basic pattern recognition algorithm is used to extract the features; scanning vertical lines and marking pixels that show a significant change in U or V channels. When detecting lines, an edge detection algorithm can be used similarly using changes in the Y channel values. These image processing techniques provide the edges of the features, which are to

be used during self-localization. The traditional MCL uses distances and directions to the landmarks to perform localization. By using only the edges, there are more points of reference per image [36].

2.2.2.2 Edge Based Localization

The *bearings* on the edges are calculated once the hypothetical camera pose for a particle is known. The robot's position can sometimes be calculated using these bearings on the landmarks, but since these calculations aren't always accurate, they are only treated as hypotheses. The possible positions replace samples in the distribution with a probability of $1 - p_i'$. If not enough positions are calculated to replace the samples in the set, random samples are used. A significant improvement was noted in the implementation of these MCL variants. They are still constrained by being a landmark-based algorithm, but does not suffice in a landmark-free environment. Since the overriding goal is to compete these robots against a human team on a real field, it was suggested that a line-based localization be developed. Such an approach was initiated by the same two authors in [21]. The key logic in the new approach is to achieve speed by not processing all pixels of an image, rather concentrate on detection

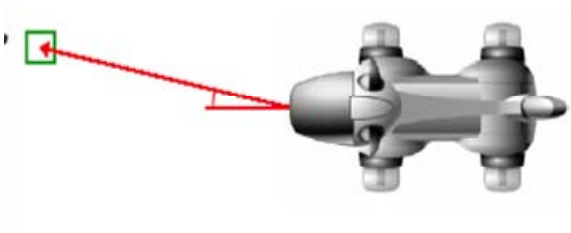


Figure 1: Pixel Projection

of lines and edges using color classification. The distance to the point on the edge can be calculated by projecting the pixel onto the ground plane as shown in Figure 1.

In using this line-based approach to self-localization, MCL is again used in moving the particles according to the *motion model*:

$$pose_{new} = pose_{old} + \Delta_{odometry} + \Delta_{error}$$

The *observation model* describes the probability for taking certain measurements at certain locations [21]. In this representation, the processing the camera images results in points on edges, which in turn are projected onto the field yielding an offset relative to the center of the robot body. The horizontal and vertical angles to the point are then calculated and compared with the measured angles in determining the most probable robot position.

With such diverse implementations of localization solutions in hand, the process to incorporate the use of a laser line to determine robot pose is less complicated. In the following discussion, several pieces of research are presented, involving using a laser to derive range values. Such applications provide support for the theoretical concept this research--deriving distance by projecting a laser line into an image.

2.3 Striping Laser and Camera Implementations

The subsequent sections introduce applications of collaborative use of a camera and a laser for providing distance information, as a relocation tool, and for camera calibration. Each application introduces different mathematical and algorithmic relationships between a camera and the striping laser, providing a theoretical foundation for this research.

The first use of a striping laser in combination with a camera is to determine the distance from the robot to the object reflecting the laser line [17]. When using a striping laser for distance calculations, a simple triangulation protocol is used. The triangle is formed by the camera and laser line as depicted in Figure 2. Here, *s* represents the distance between the camera and the laser producing hardware. The angle α represents the angular distance between a straight line drawn from the camera to the laser line and a

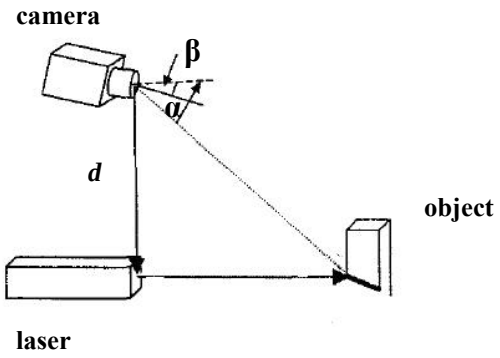


Figure 2: Camera/Laser Triangulation

horizontal line from the camera to the horizon (vanishing view point). With this information, the distance between the laser and the obstacle is calculated as

$$d = s \bullet \cot(\alpha) . \tag{8}$$

To obtain the geometric relationship described in the figure, the laser line must first be extracted from the image. A variety of techniques can perform this extraction, e.g. pattern recognition, edge detection, etc. Next, the geometric relationship between the camera and the laser plane are measured [17]. Finally, the distance is calculated with

equation (8). Now that the distance in the image is calculated, its relative real world distance is computed:

$$\Delta d_t \sim d^2 / s \,. \tag{9}$$

In essence, the disparity in the distance measures is equivalent to the distance between two image pixels measured as real world distance. This concept is implemented in the Chapter III for deriving distance for map-building and self-localization.

Another application collaboratively using a camera and a striping laser determines the location of a robot by locating its camera. Here, combining geometric information from the striping laser and information from the image is used to more accurately estimate the sensor location [18]. Using multiple sensors generating a single observation helps classify *nodes* on a topological map. These nodes are then used in relocation. First, the pixels on the laser line are used to detect vertical planes in the scene, which then have the images texture mapped to them, resulting in a non-scaled image. The relationship between image pixels and the selected plane can be described as:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

where (x,y) are the coordinates in the image, X,Y,Z is the reference system defining a selected plane, K being the inner calibration of the camera, t is its location, λ is the scaling factor, and $\begin{bmatrix} r_1 & r_2 \end{bmatrix}$ are the first two rows of the rotation matrix. Once the non-scaled images are produced, their similarities can be used to align two observations of the same scene taken with different perspectives. In order to extend this concept into a

localization solution (relative to previous observations), there must exist a topological map with some pre-calculated node locations [18]. Once the measured observations are ascertained, the same alignment technique is used to align the observation with the pre-calculated nodes on the topological map. In experiments performed, fallacies were found in solutions due to insufficient overlap of views, poorly textured areas, occlusions of the area, failed segmentation of walls, and specular reflections and lighting changes [18]. Although these issues did cause false positives/negatives during testing, the overall success was measured by the number of times the vehicle was found in the map and correctly located, excluding the true negatives, earning a success rate of over 83%.

Finally, a camera and striping laser have been used together to perform calibrations on a laser range finder [35]. The unique technique in [35] provides the model with precise initial estimations by applying an evolutionary algorithm to tune the initial parameters. The upcoming work takes note of how this geometric relationship is not only mathematically modeled, but also how the movement of the robot, upon which the laser and camera are mounted, affects this model's parameters. In this system model, an undistorted system model is derived from a three-dimensional point $\mathbf{P}(x, y, z)$ seen in Figure 3.

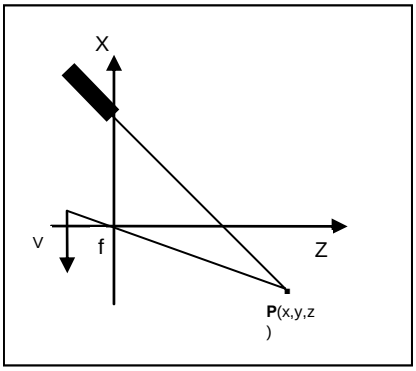


Figure 3: Camera/Laser Geometric Model

Here, the point **P** is transformed, first into an undistorted, two-dimensional sensor coordinate, then into a distorted coordinate using a radial lens distortion. The resulting image coordinate is transformed into a pixel location using magnification coefficients and the center of the image. This pixel is used to compute the 3D coordinates **P**(x,y,z) of the *illuminated scene point* [35]. Unlike the other applications using a laser and camera, this particular research introduces a new factor of using geometric models in such computations; kinematics. The calibration performed in [35] is directly affected by the kinematics of the robotic arm where the camera an laser are mounted.

The research presented in the previous sections demonstrates that it is possible to use geometric relationships between the camera, laser, and obstacle to determine the distance to that obstacle. The research provided thus far encompasses localization methods, sensors used to provide the data for localization, and a few unique applications involving the collaborative efforts of a camera and laser. There exists another solution that extends the use of sensors beyond localization. The data collected, in the manners described above, is also integral to map-building. A map is a critical piece of information in the localization process. If a map isn't known ahead of time, one must be built as the robot navigates its way through its environment, a process known as *Simultaneous Localization and Mapping.*

2.4 Simultaneous Localization and Mapping (SLAM)

Localization is not the only problem faced in the development of autonomous robots. Another concept that is fervently studied is *simultaneous localization and*

mapping (SLAM), where not only is the pose of the robot in its environment estimated, but induced from a map and sensor readings, where the map is built as the robot explores the environment. Of course, since this matches the real world and includes the robot noisy motor controllers and sensors inhibited by noise, the algorithms developed as solutions to this problem are probabilistic in nature. The most popular solutions used in estimating the map and robot location are Kalman filters[16, 19], Dempster's expectation maximization algorithm, and those algorithms that identify objects in the environment [26, 25]. Each of these methodologies are characterized by the types of information they produce such as identifying objects or significant features in the environment or verifying the accuracy of a sensor measurement over time. Regardless of the approach taken, there are two significant sources of information that must be dealt with when using *SLAM*.

The first source of information is identical to that used with localization algorithms; sensors. The biggest problem with sensors is dealing with noise. This noise isn't necessarily caused by the usual inaccuracies of the sensors themselves, but by the second source of data; the motion commands (controls) issued during environment exploration [26]. The *sensor* and *motion models* are independent of each other, but have a dependency through the map. So the job of the *SLAM* algorithm is to compensate for such errors, as well as complications including the high dimensionality of entities being mapped, the correspondence (data association) problem, and the dynamic nature of the environment being mapped [26]. Each family of algorithms is able to handle a limited number of these complications. Two such approaches, the Kalman filter and Expectation Maximization, are widely used.

Kalman filters, as discussed previously, are Bayes filters whic represent posterior distributions with Gaussians. When working with mapping, the Gaussian model is the full state vector [26]

$$s_t = (\mathbf{x}_t, m)^T \tag{13}$$

made up of the robot's pose $\boldsymbol{x}$ and the map m. Using Kalman filters for mapping requires three primary assumptions be made: the motion model must be linear with added Gaussian noise, the same goes for the sensor model, and initial uncertainty must be Gaussian [26]. With this in mind, the pose and sensor functions must be *linearized*, since they are not traditionally linear functions. Once linearization has been performed, the standard Kalman filter equations (Table 1) can be used. When estimating a map, not everything in the environment is going to be known ahead of time, so as each new feature is stumbled upon, a separate Kalman filter used. If that feature is repeatedly seen, it is added to the feature list for the map. In general, Kalman filters are most well known for their ability to "estimate the full posterior" making it possible to maintain most likely map and pose locations and a full uncertainty map coupled with the ability to converge to a true map and robot location. As with any probabilistic solution, there is a limitation to using Kalman filters for mapping; the Gaussian noise assumption. Generally this becomes significant when dealing with the *correspondence problem*, being able to associate individual sensor measurements with features in the map [26]. The maps produced contain location of landmark-type features, but little geometric information about the environment.

In contrast, the *Expectation Maximization (EM)* algorithm solves the *correspondence problem* by repeatedly relocalizing the robot relative to the present map instead of the pose posteriors. In EM, the posteriors are calculated for a given map, *expectation step,* and then the most likely map given these pose estimations is calculated, *maximization step.* These steps, performed iteratively starting with an empty map, produce a more accurate map. The *expectation step* is so named because it builds on the expectation that the path of the robot is known and calculates :

$$p(\mathbf{x}_\tau \mid m^{[i]}, d^t) \tag{14}$$

Which is the posterior for the pose $\mathbf{x}_\tau$ conditioned on all data leading up to time *t,* d^t and the *i*-th map $m^{[i]}$. In contrast to standard localization, data over the entire interval *[1…t]* is used to estimate the posterior pose at time τ, even if $\tau < t$. The *maximization step* then finds a new map m that maximizes the log likelihood of the sensor measurements $\log p(z_\tau \mid x_\tau, m)$, for all τ and all poses x^t and under the expectation calculated in the *expectation step* [26]. *EM* produces maps that are topologically correct, given correspondence problems presented by such things as large loops. The only pitfall of the *EM* algorithm is that it is an offline algorithm and subject to local maxima.

There are currently no successful implementations of *SLAM* on an AIBO due to the requirement for an accurate motion model, as the accelerometer sensor are unreliable for inferring pose information.

2.5 Summary

The aforementioned research describing localization implementations, both generic and AIBO specific, and applications combining a camera and striping laser describe the techniques required to develop the solution presented in this publication. The core problem to be solved is one of localization, specifically using the AIBO ERS-7. Like the AIBO-specific localization techniques discussed earlier, this research focuses on Monte Carlo-style localization. In contrast to existing solutions, this research moves away from relying on colored landmarks and lines for determining the robot's pose. Instead, localization of an AIBO robot in a non-soccer environment and have it simultaneously localize and map its environment.

In order to use the vision sensor to provide the sensor data for localization, one must extract geometric relationships to determine the range between the robot and obstacles captured by its camera. In this research, a striping laser is attached to the robot and the projected laser beam is then captured in the camera images. Since the laser is mounted in line with the camera, its orientation in the image never changes. Hence, the laser line seen in the image provides a horizontal reference for the skew of the image itself. The research implementing active triangulation with a laser line in an image provides the basic tools for determining distance between an obstacle and the robot by using the geometric relationship between the laser, camera, and obstacle. In our case, the configuration is different, so additional research is performed to transform the system model relative to our specific configuration. Additionally, the kinematic chains specific to the AIBO are far more complicated than those for the 5 DOF arm [35]. This chapter

briefly covered various mathematical applications and solutions that are compiled and modified to provide a solution to a more dynamic localization process using the ERS-7 AIBO.

III. Methodology

This chapter presents the research and implementation aimed to support the theory that distance between an AIBO mobile robot and an obstacle can be determined by the distance between a static feature native to all images a pattern projected onto the image. This distance is used within the sensor model and is coupled with the pose of the robot and its motion model and fed into a simultaneous localization and mapping (SLAM) application.

3.1 Overview

In this research, the AIBO ERS-7 serves as an autonomous platform from which images are gathered for SLAM. Sensor readings are provided by a disparity between the horizon and a laser line projected into the AIBO's camera frame as shown in Figure 4.

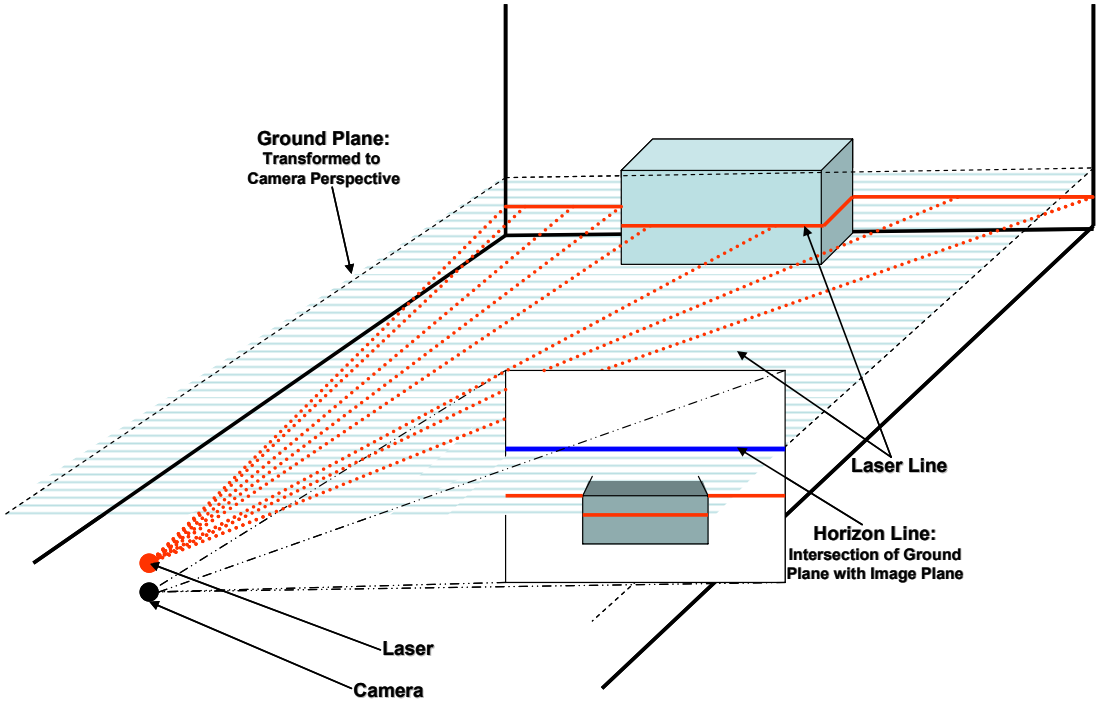


Figure 4: Theoretical Concept of Determining Distance Using Horizon and Striping Laser

This is based on a relationship between the distance and position of objects (the laser line in this research): those that are far away are close to the horizon. The disparity between the pixels of the two lines have relative real-world distance [1]. Therefore, each pixel on the laser line is assigned a relative real-world distance, representing our sensor reading for that location in front of the robot.

3.2 The AIBO

The utilization of the many assets of the AIBO requires a thorough understanding of the hardware native to the robot. The physical characteristics of the robot include: 576MHz processor, 64 MB RAM, 802.11b wireless ethernet (standard), MemoryStick reader/writer, 18 PID joints, each with force sensing, 26 independent LEDs, 350k pixel video camera with 16.8 million colors at 30 frames per second, 3 IR distance sensors, 3 accelerometers, 10 pressure sensitive buttons (two on head, three on back, four feet, and one under belly), and 1 button under the mouth. The components critical to this research are the camera, the wireless Ethernet, and the joints of the legs and head [5]. Due to limited RAM and processing power of the robot, the majority of the processing for this research is done offline through wireless communication.

As for software selected, the open source Tekkotsu API developed at Carnegie Mellon University is built directly on top of the Open-R architecture, is used [2,8]. Open-R was initially designed to create a standard architecture for “entertainment” robots. It provides an interface for sensors and actuators, methods of obtaining information from functions of these components, and has a layered architecture based on

Asperos [8]. Open-R supports Tekkotsu's object-oriented and event-passing architecture, depicted in Figure 5 [2], making full use of the template and inheritance features of C++. It was originally written for the Sony AIBO, but can also be compiled for Linux, Mac OS X, and any other BSD-based OS. The framework is designed to handle routine tasks for the user, allowing them to concentrate on higher level tasks. Some of the services Tekkotsu provides include basic visual processing, forward and inverse kinematics solvers, remote monitoring and teleoperation tools, and wireless networking support. Tekkotsu builds on several third party libraries, such as ROBOOP (general kinematics), and NEWMAT (matrix operations) [2]. Tekkotsu's internal data flow is shown in Figure 5.

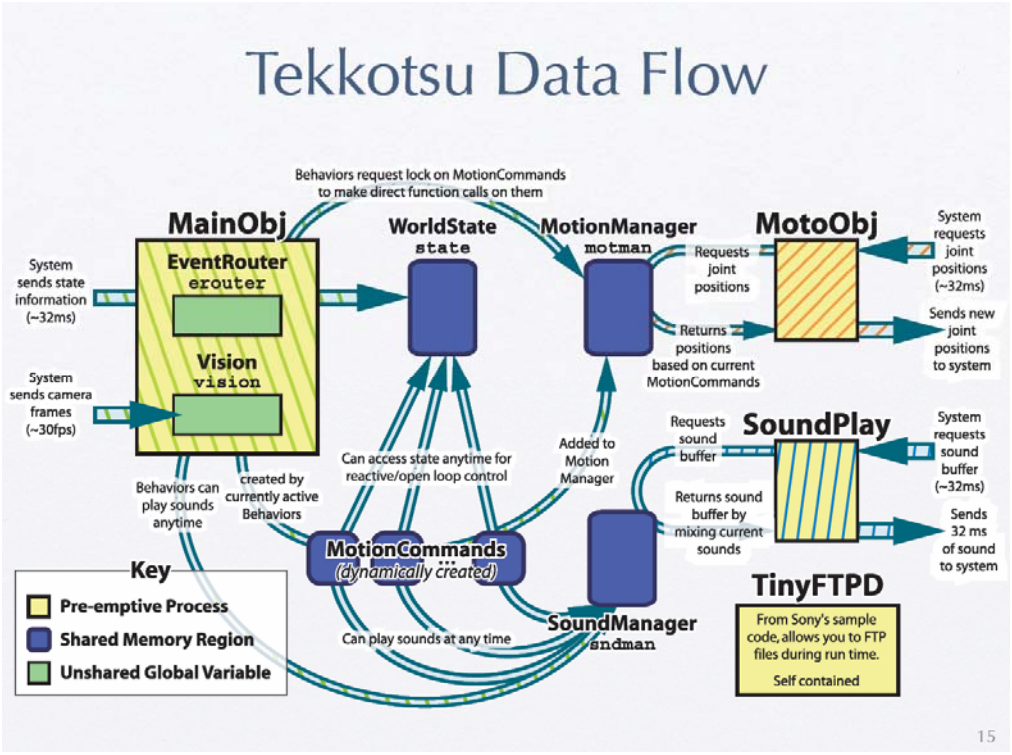


Figure 5: Tekkotsu Data Flow

In order to use Tekkotsu, one must establish a separate port and a server for each type of data to be transmitted.

3.2 Capturing Images

With the AIBO's onboard camera as the primary sensor upon which to build the *sensor model* for use in computing pose, it is critical to become familiar with the images the camera records, how Tekkotsu manipulates them, and how to transmit those images. On the client side, we develop the software to receive these images and restore them to their original format.

The AIBO's camera is a Charge Coupled Device (CCD) camera able to capture and store images with a resolution up to 416x320 pixels. The Tekkotsu framework allows users to configure the vision system. A simple configuration file contains modifiable settings such as white balance, compression, type of image, etc. (see Appendix A for entire configuration file).

```
white balance = indoor
gain = mid
shutter speed = fast
resolution = full
rawcam_encoding = color
rawcam_compression = none
rawcam_compress_quality = 85
rawcam_y_skip = 2
rawcam_uv_skip = 3
raw_transport = udp
rle_transport = udp
```

Figure 6: Camera Configuration

To reduce transmission time, reduce loss of information in transmission, and conserve memory, the camera configuration was set to the values as shown in Figure 6. The gain and shutter speed settings control the amount of noise and motion blur. Higher gain and slower shutter speed brighten the image, but increase noise and increase motion blur. In the aforementioned configuration (Figure 6), the *rawcam* settings affect the traditional image while the *rlecam* settings are concerned with the

segmented image engine, also native to this CCD camera. There are various additional settings in the configuration of the camera that allow for customizing the segmentation of the image that were not altered for this research. The configuration file located in Appendix A provides a brief description of each of the configuration settings mentioned above. The image segmentation process is discussed in section 3.4. Once the camera is configured and the transmission type determined (UDP/TCP), the simple activation of the camera servers automatically transmits images over ports 10011(*raw* images) and 10012(segmented images) to their registered clients.

3.2.1 Raw Image

The *raw* image mirrors the real world without manipulation. Unlike traditional images, the AIBO's CCD camera records images in YUV standard. The YUV model defines a color space in terms of one luminance and two chrominance components. YUV is used in the PAL and NTSC systems of television broadcasting, which is the standard in much of the world. YUV models human perception of color more closely than the standard RGB model used in computer graphics hardware, but not as closely as the HSV color space. Y stands for the luminance component (the brightness) and U and V are the chrominance (color) components. The YCbCr or YPbPr color space, used in component video, is derived from it (Cb/Pb and Cr/Pr are simply scaled versions of U and V), and are sometimes inaccurately termed "YUV" [34]. In the image buffer described above, the YUV information is broken down into its components and compressed. The compression settings call for skipping $\log_2 2$ of Y channel pixels and $\log_2 3$ U and V channel pixels. As a result, a 208 x 160 (Y,U,V) image, is encoded as a 104 x 80 (Y), 52 x 40 (U & V)

image. As mentioned before, this compression reduces the transmission time, which is critical to processing the images in real-time. Once the buffered image data is parsed and stored in the structure, the client software uncompresses the image and converts it to RGB for viewing. This conversion is possible since YUV signals are created from an original RGB (red, green and blue) source. The weighted values of R, G and B are added together to produce a single Y signal, representing the overall brightness, or luminance, of that spot. The U signal is then created by subtracting the Y from the blue signal of the original RGB, and then scaling; and V by subtracting the Y from the red, and then scaling by a different factor. The RGB values from the YUV values are derived with the algorithm in Figure 7.

```
– buffersize =( sizeof raw_images.data )/4;
– for (q = 0; q< buffersize; q+=3)
– {
    • int C = raw_images.data[q] - 16;
    • int D = raw_images.data[q+2]-128;
    • int E = raw_images.data[q+1]-128;

    • raw_images.data[q] = clip((298*C+409*E+128)>>8);
    • raw_images.data[q+1] = clip((298*C-100*D-208*E+128)>>8);
    • raw_images.data[q+2] = clip((298*C+516*D+128)>>8);
– }
```

- with *clip()* defined as follows:

```
• void clip(int x)
• {
•         if(x<0) return 0;
•         if(x>255)return 255;
•         else return x;
• }
```

Figure 7: YUV to RGB Conversion

The purpose of this function is to keep the converted values within the RGB range of $0 \le x \le 255$. At this point, the raw image is restored to its original form. Since the *raw* image is fully reconstructed, the system then reads and processes the *segmented* image.

3.2.2 Segmented Image

Segmentation of an image is simply the removal of all the unwanted colors from the image. Provided an image from which to extract a feature that can be uniquely identified by its color, segmentation eliminates the complexity of extracting the laser line. The segmentation process is done in Tekkotsu by taking a series of sample images that represent those good objects and feeding them through a calibration tool that builds a threshhold and color file(s) included in the segmentation

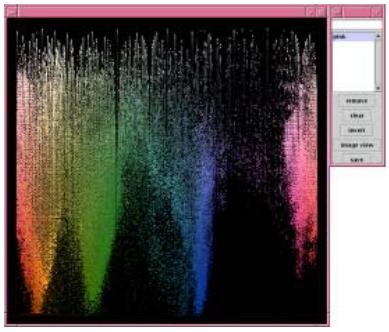


Figure 8: Color Vision Train

configuration. These files are automatically loaded at boot-up of the robot, telling the Tekkotsu behaviors which colors are searched for in the image. There are two java classes that handle creating the segmentation setup. The first is VisionTrain, which allows us to send it a series of images (taken from the AIBO). VisionTrain creates a color palette based on the colors in those images, and select the colors to retain. The second tool is called VisionSegment, which lets us check our test segmentation, by feeding it the configuration created in VisionTrain and the same set of sample images it outputs the effect that our segmentation had on the set of sample images.[29] In this case, we calibrated the segmentation engine to keep only the red associated with the laser.

Unfortunately, the laser line may not be the only red object in the camera's view. Our segmentation is also guilty of identifying such things as orange cones, pink cups, and red variations on clothing and skin pigmentation. It was decided not to recalibrate to reduce these random objects, since the color of the laser line changes as the color of the object that reflects the laser changes. Black backgrounds produce red lines, while lighter backgrounds cause the line to be more pink in hue.

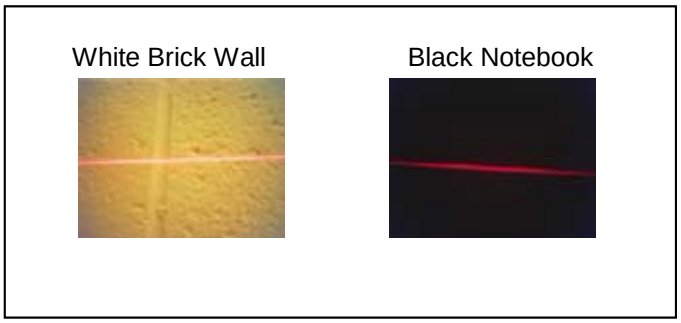


Figure 9: Laser Line Colors

Tekkotsu's SegmentedCamBehavior encodes the segmented images, preparing them for wireless transmission. The client software uses the same methodology to capture and parse the segmented image buffer as for the raw buffer. The only difference is in the reconstruction of the image. The process began with converting the segmented image to a white background with the preserved color being black. With the successful reconstruction of the segmented image, it is now processed to extract the laser line and store the relative information.

3.4 Extracting Laser Line

To process this segmented image, the Open Source Vision Library (OpenCV) [10] developed by Intel is used. The first step is to convert the array of integers representing the image to an OpenCV-friendly format (IplImage). The new IplImage now references

the segmented image array, able to be passed around between OpenCV methods for processing. The next critical process is determining where the laser lines appear in the image. In order to use the laser line pixels in the estimation of distance, we must know their location in reference to the horizon line.

The tools found in the OpenCV library provide a means to extract the line segments and store their endpoints using Hough Transforms. The underlying principle of the Hough transform is that there are an infinite number of potential lines that pass through any point, each at a different orientation. The purpose of the transform is to determine which of these theoretical lines pass through most *features* in an image - that is, which lines fit most closely to the data in the image. In the standard Hough transform, each line is represented by two parameters, commonly called r and θ, which represent the length and angle from the origin of a normal to the line in question. In other words, a line is described as being at an angle 90° from θ, and being r units away from the origin at its closest point. This representation of the two parameters is sometimes referred to as *Hough space*. A set of points which form a straight line produces Hough transforms which cross at the parameters for that line [33]. In this particular case, the probabilistic Hough transform is used since it is more efficient in pictures containing a few long linear segments. It returns line segments rather than the whole lines. Every segment is represented by starting and ending points. These line segments (their endpoints) are stored in an OpenCV object (cvLine).

3.5 Estimating Horizon Line

Since the horizon is used as the fixed reference in the image, the first most critical estimation to be made in this research is the location of this horizon. The information known about the motion of the robot and camera tells us a lot about the horizon. The same concept is used in [13] to calibrate a camera for motion for robotic applications. There are several ways that the horizon has been calculated. The most popular tactic is using vanishing lines in the image [4, 13]. Once the vanishing lines are identified, estimates of their vanishing points are determined. By connecting these two points, found at an infinite distance from the camera, it is possible to determine the location of the horizon in the image shown in Figure 10 [13].

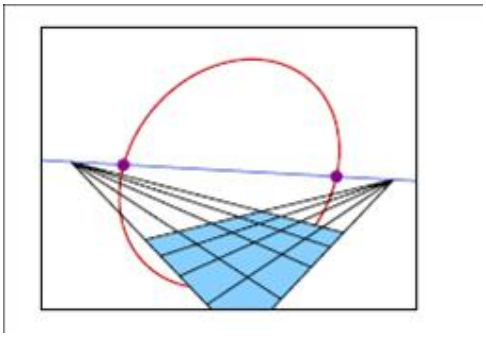


Figure 10: Horizon using Vanishing Lines

Extracting the vanishing points shown in Figure 10 is accomplished using projective reconstruction and stereo imaging to calculate the horizon points. Unfortunately, the resolution of the AIBO's camera being visible, coupled with only image features within 67.056 cm of the camera, inhibits the extraction of these vanishing lines. Due to the poor resolution, another method was used in estimating the location of the horizon in the image.

The horizon is defined as the intersection of the ground plane, transformed to the camera's perspective, and the image frame as shown in Figure 11 [11]. For the AIBO, there are many parameters to consider in deriving the ground plane equation of a robot with 15 degrees of freedom (3 degrees for each leg, 3 degrees for the camera). The robot

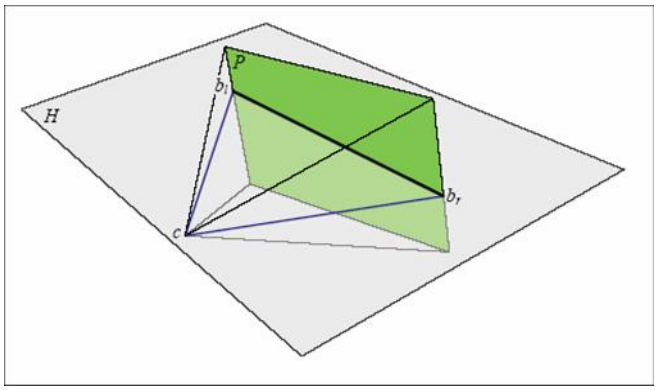


Figure 11: Horizon Using Intersecting Planes

itself is capable of providing us with information about its stance as well as the camera's rotation. As features of the robot are discussed, refer to Figures 12 and 13. The purpose of the ground plane equation is to capture the tilt of the robot's body, that is, the angle between the robot and ground along the x axis [30] which directly impacts the horizon's position and *z* axis which impacts the rotation in the image. The pan about a vertical pan axis , Figure 12 axis y_3 ,[13] also changes the position of the horizon, as discussed later in this section. Thankfully, the Tekkotsu software developed a behavior to calculate the ground plane equation (GroundPlaneBehavior) using the location of each foot in reference to the *base frame*. The original behavior ("GroundPlaneBehavior") used the (x,y,z) location of the three feet on the ground, along with the accelerator values, to

capture the tilt of the robot. The accelerators returns a real valued estimate of the robot's acceleration along the *x,y,z* axes [30]. For our purposes, the behavior is modified to exclude the accelerators, since the horizon's rotation isn't affected by those values (left/right, forward/reverse, up/down) and the sensors are so noisy they actually decay the solution.

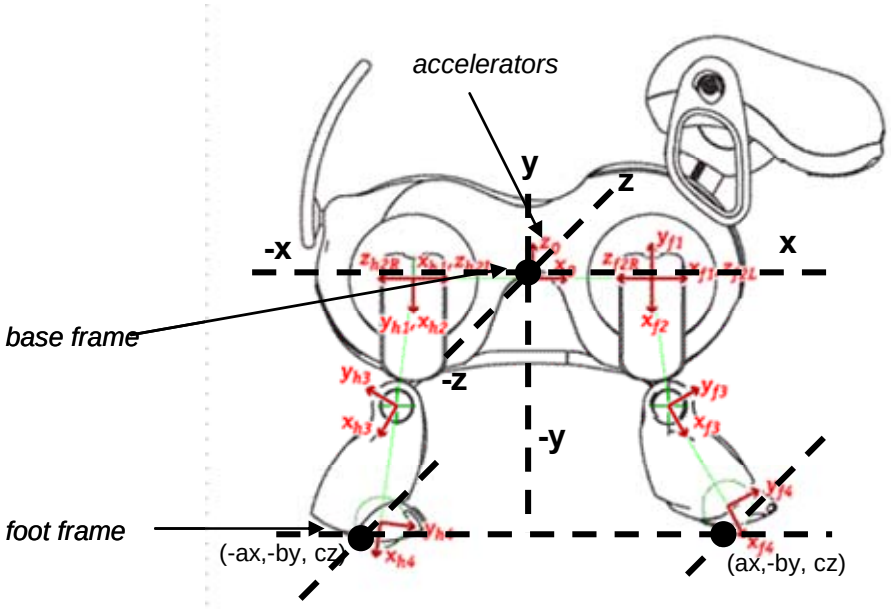


Figure 12: AIBO Diagram 1

The initial assumption made is that the ground was flat (0,0,0). The ground plane equation is derived by fitting a plane to the three "down" legs; represented by three translation vectors in reference to the base frame. Before beginning the transformation of the ground plane vector to the camera frame, it must be noted that the camera rotation and not the camera height affects the horizon's position and rotation [1]. The transformation translation is shown in Figure 12 and 13. Each transformation introduces

error into the calculations, since each robot and each motor are not identical.

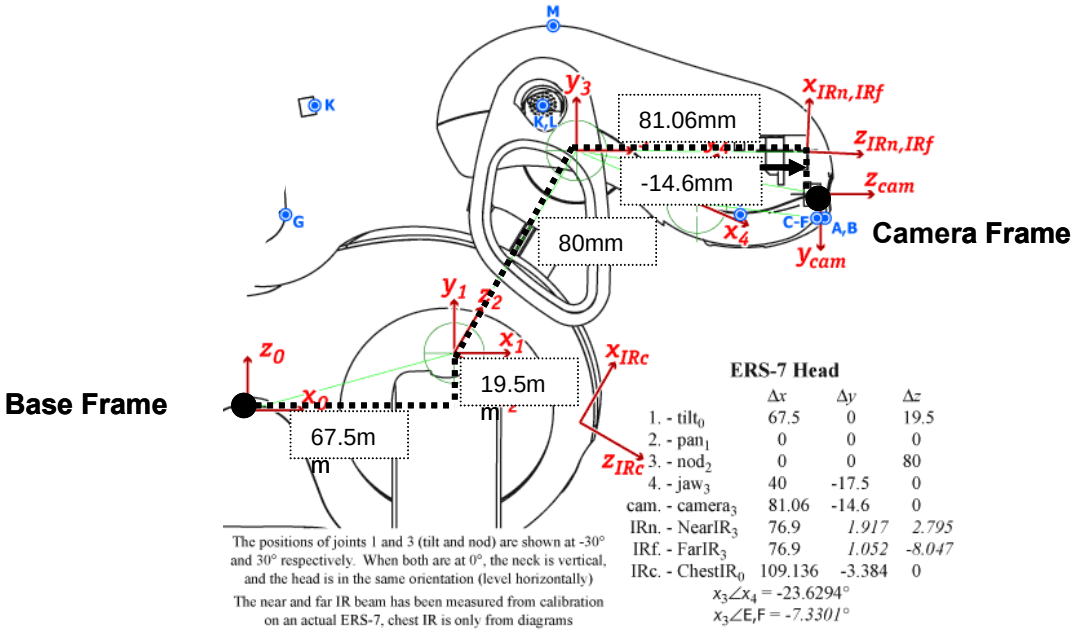
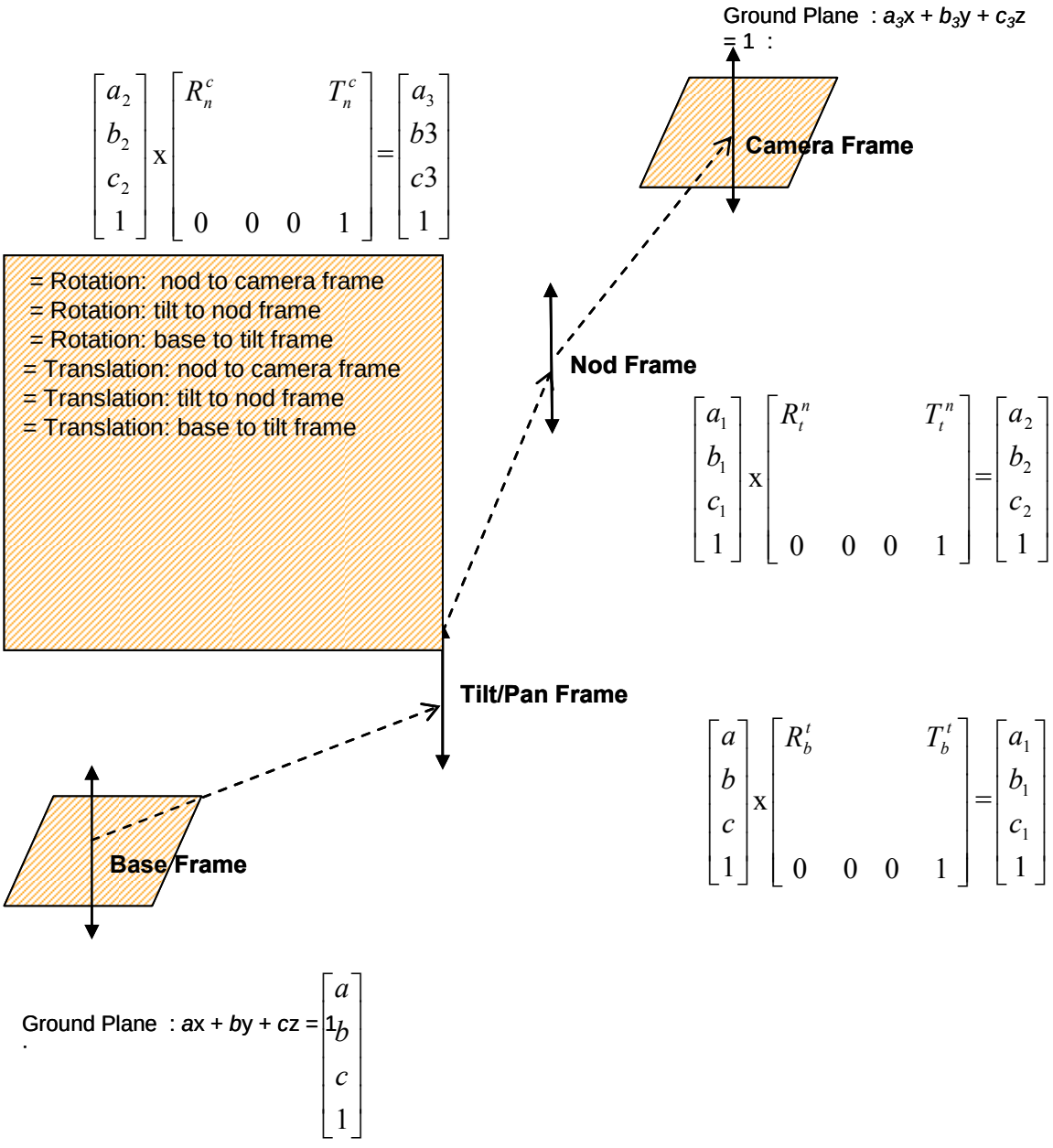


Figure 13: AIBO Frame Translations

Once the estimation of the ground plane is generated and transformed to the camera's perspective, the ground plane vector is projected to a distance in the *z* (relative to the *base frame*) direction as to intersect the image plane. Since the camera's resolution prevents the image from picking up the laser line at a distance greater than approximately 67.056 cm , we use a distance of 91.44 cm to simulate the infinite distance of the horizon. The horizon is drawn on the image by selecting a point in space using our simulated infinite distance forward. The point is also rotated about the up/down axis *(y)* to compensate for

Figure 14: AIBO Frame Translation



the left/right pan of the head. These *(x,y)* values are substituted into the ground plane equation, $z = \frac{1 - a_3 x - b_3 y}{c_3}$, to compute the *z* coordinate. By adding the vector of coefficients to this *x,y,z* point, we have two points which represent the normal of the horizon. To find the corresponding location of the horizon in the image, the *(x,y,z)* coordinates are converted to pixel coordinates x_p, y_p:

$$x_p = \frac{w-1}{2} + \left(\frac{w}{\tan(f_h)}\right)\left(\frac{x_r}{z_r}\right)$$
$$y_p = \frac{h-1}{2} + \left(\frac{h}{\tan(f_v)}\right)\left(\frac{y_r}{z_r}\right)$$

where x_r, y_r, z_r are real world distances and f_h and f_v are the horizontal and vertical field of vision (FOV) respectively, and *w and h* are image width and height (in pixels). The OpenCV Image Processing Library (which uses a Bresenham algorithm) is then used to draw the normal of the horizon line in the image to test the accuracy of the estimation. This vertical line provides us information about the robot's stance as well as the horizon's rotation in the image. At this point, the horizon and lines extracted from the segmented image provide the information necessary to approximate the distance between the robot and the obstacle reflecting the striping laser beam.

3.5 Determining Distance

There are several methods to determine distance between a camera and an object. The most popular method is triangulation, as discussed in Section 2.2.3. The camera, laser, and horizon have a geometric relationship and the distance between the laser and obstacle is determined by plugging the angle and distance between the angle and camera into the Pythagorean Theorem, solving for the "adjacent side" of the right triangle. The configuration of the camera and laser on the AIBO doesn't provide us a right triangle relationship as in laser/camera configurations described in Chapter 2 (see Figure 6).

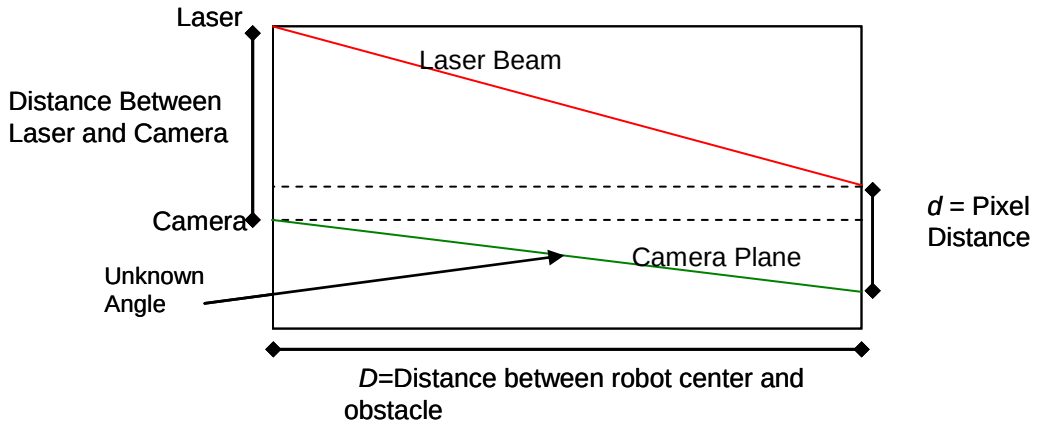


Figure 15: Triangulation

Therefore, a different approach is taken in determining the distance *D*. The approach taken in this research assigns a real-world distance *D* for each pixel in the image. To begin this process, the simplest calculation is the pixel distance *d*. First, the horizon is oriented in its natural position in the image; the center. This adjustment must be mirrored by every pixel in the image. Since determining the pixel distance is only dependent on the relationship between the laser line and horizon line, corresponding adjustments are

only made to those pixels containing the laser line. The actual adjustments to both lines requires the computing the horizon's angle of rotation relative to the horizontal. This is easily derived from the perpendicular's endpoints x_1, x_2, y_1, y_2 [4]:

$$slope = \frac{(x_2 - x_1)}{(y_1 - y_2)}$$

$$\alpha = \arctan(slope)$$

The horizon is rotated by $-\alpha$ so it is parallel to the image frame horizontal. The laser line segments must maintain the angular relationship to the horizon line, so each pixel of the laser line must also be rotated by $-\alpha$.

$$x' = x\cos(-\alpha) - y\sin(-\alpha)$$
$$y' = x\sin(-\alpha) + y\cos(-\alpha)$$

In addition to the rotation, there is the need to compensate for the nod of the head. Since the horizon's position in the image changes relative to the nod angle, the laser line segments must reflect the same motion to maintain a true relationship with the horizon. The distance between the horizon, y_h and its natural position in the image (center), y_c is used as the translation value $d = y_h - y_c$ applied to the laser line segments, $y'' = y' + d$. Since only the endpoints of each of these line segments are retained, an algorithm is needed to locate each pixel of the line segment, as if we are drawing the line pixel by pixel. Tracing through pixels of a line in an image is not as simple as iterating through the *x* coordinates, adding the *slope* of the line to the *y* coordinates due to the native

structure of a pixel being an integer. Thus, a version of the Bresenham's Line Algorithm is used to decide which pixel values are classified as being laser line pixels. Bresenham's Line Algorithm determines which points on a 2-dimensional raster should be plotted in order to form a close approximation to a straight line between two given points [32]. Each pixel classified as a laser line pixel is rotated by $-\alpha$ and translated by d. The original x,y values and the rotated/translated x,y values are in the appropriate index of an array of structures, created to maintain the metrics of each pixel in the image. The number of pixels between the line segment pixel and corresponding horizon line pixel is initially stored in this structure as *distance*. After the entire image is processed and the pixel information stored, the client program is modified as to run calibration tests to develop the *sensor model*.

3.5.1 Sensor Model

To develop the *sensor model,* the robot is placed at discrete distances (measured in mm) from a box placed in its field of view as to reflect the laser. The client program is executed ten times at each one-centimeter intervals, ranging one foot to 91.44 cm from the center of body. The decision establishing 91.44 cm as the maximum distance threshold is based on the camera's inability to capture the laser line at a distance at greater than 67.06 cm from the robots center of body. For each of the one centimeter intervals, the corresponding pixel distances are recorded (see Table 1). These pixel distances subsequently provide indexes into a lookup table used during program execution to return an estimated distance. Figure 16 shows the error in distance induced within one standard deviation from the mean. The array is able to return the relative real-

world distance for the indexed pixel distance.

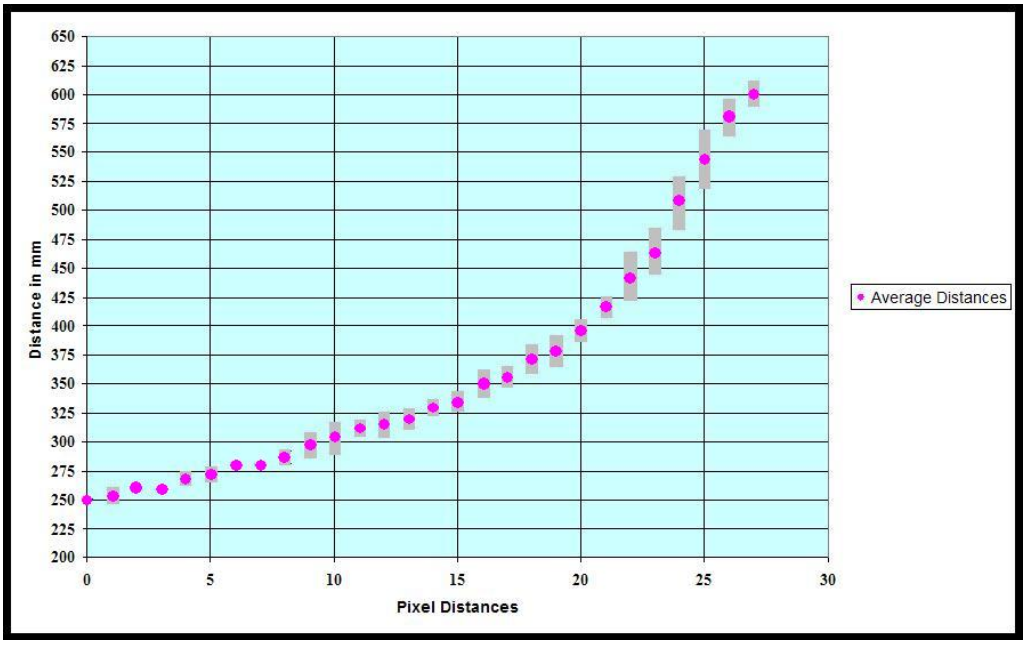


Figure 16: Sensor Model Error

The approximations are critical components of the *sensor model* the SLAM software needs to build the map of the environment as well as correcting the estimated pose. The SLAM software using sensor data produced by this client software is developed for a Pioneer wheeled robot. Hence, the *sensor model* developed in this research must conform to characteristics of the sonar sensor model of the Pioneer. To make this adjustment, a single scan of the sensor is represented by 104 pixels (one row) of the image. In essence, the distance information stored in each column of the image cts as though it were a distance reading from one of 104 sensor readings (Figure 17).

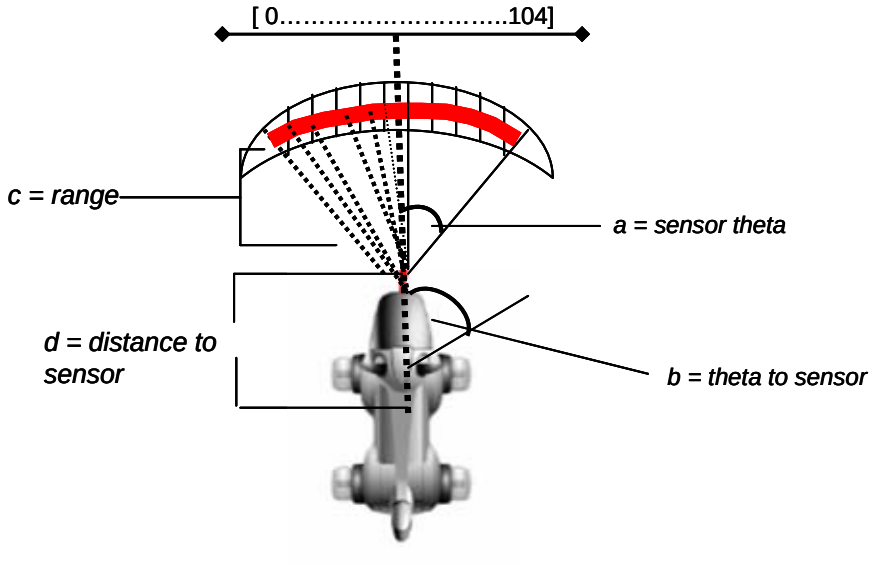


Figure 17: Sensor Model

The sensor model provides the *distance to sensor (d), range(c), sensor_theta (a), theta to sensor(b)* for mapping. These values are computed using geometric relationships depicted in Figure 18

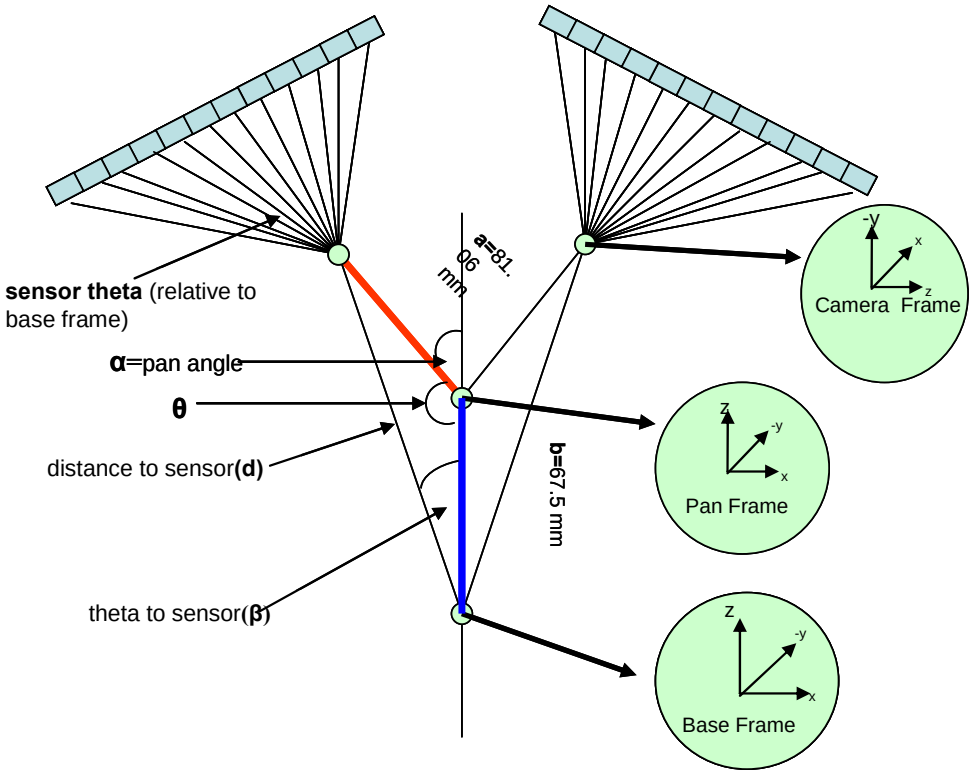


Figure 18: Geometric Sensor Model

First, the *sensor theta* describes the angle measured from the camera to each pixel:

$$sensor\_theta = \frac{HorizFOV}{104} * pixel\_number - 51$$

where *HorizFOV* is the Horizontal field of view, and *pixel_number* is the pixel (0-104). Next, the *distance to theta* and *theta to sensor* values are computed in the following manner (reference Figure 18):

$$\theta = \pi - \alpha$$

$$d = \sqrt{a^2 + b^2 - 2ab\cos(\theta)}$$

$$\beta = \arcsin(a * \sin(\theta) / d)$$

Once these four values are computed, they are converted from the AIBO coordinate system to the Pioneer coordinate system (see Figure 19), then written to a file for processing by the SLAM software.

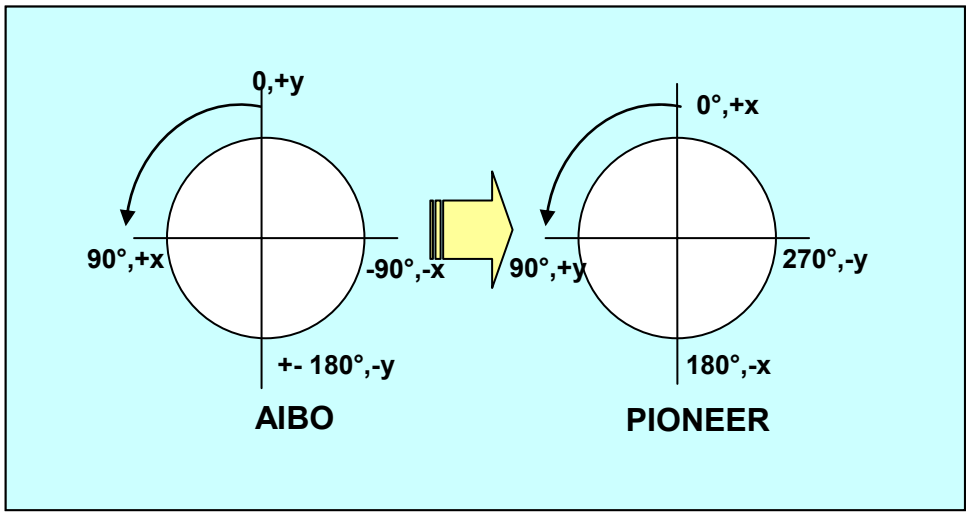


Figure 19: AIBO/Pioneer Coordinate Systems

The last component of SLAM developed is a *motion model* of the AIBO's walk. The *motion model* needs to accurately determine the location of the robot having traveled some distance in a certain direction. To model the AIBO's walk, a timing sequence is used to break the walk down into measurable distances. There are two reasons a timing sequence was used to discretize the walk. First, identifying a single step using joint cycles is inaccurate due to the dynamics of the quadrant trot. There are a total of 54 parameters which describe the walk. The step is found to consist of twelve joint adjustments, but pinpointing the first value of the cycle is extremely tricky. Since the walk cycle never repeats a joint angle, to calculate distance traveled in a single "step" is

not consistent. Second, the complexity of using joint cycles is increased when synchronizing each step sequence with the captured images. The images are uniquely identified by a timestamp that begins recording at boot-up of the robot. To maintain consistency between the milliseconds elapsed between image captures and distance traveled, the walk was modeled using time slices. First, timed trials are performed to reveal that it took the AIBO approximately 7940 ms to travel about1 meter. Next, one meter was measured and the AIBO was programmed to walk that distance at 100 mm/sec for over 100 runs. Notice our first assumption of a constant forward velocity. Each of these runs was recorded (see Figure 20) using a marker attached to the AIBO.

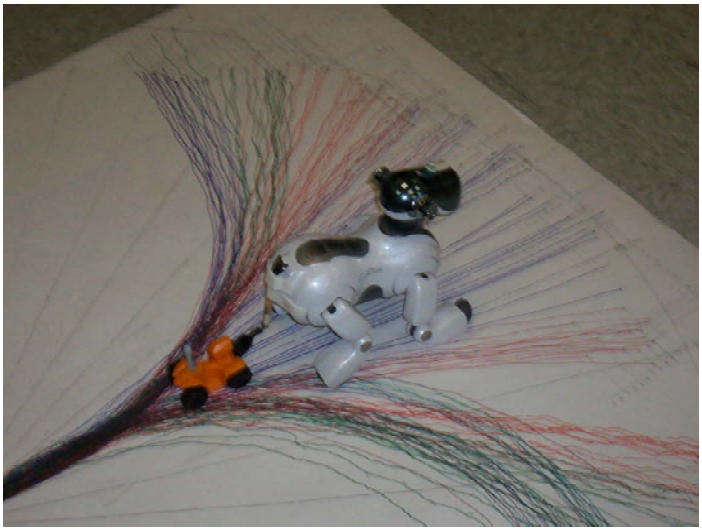


Figure 20: AIBO Motion Tracking

The assumptions made for future use of this motion model is that the AIBO would maintain a 100 mm/sec pace forward and turn only by adjusting angular velocity (radians/sec) control. Figure 21 show the results of 10 runs of walking straight for a meter and Figure 21 shows the results of recording turns ranging from -50° to 50°

degrees within that meter. Keeping with the time-slicing modeling approach, the turns were accomplished within the 1 meter by programming the robot to walk for 1000-5000 milliseconds with an angular velocity of 10 degrees + offset (0.040 radians) for 15 trials. For the remaining seconds, the robot walks straight forward. The marker trails provide measurable *(x,y)* locations at various distances. The trials were performed on a 1/2 inch grid posterboard. Points are collected at five distances, 1, 0.5, 0.25, 0.175, and 0.0875 meters, and the resulting (x, y, θ) are measured with θ as the robot's heading. The data points are analyzed and a curve-fit is performed, providing us a polynomial equation that establishes a relationship between the *(x,y)* and θ. Given a set of *x,y* coordinates, the resulting polynomial reproduces the curves found in the two figures below. Deriving this relationship is critical for determining pose as discussed in the next section.

Figure 21: (left)Straight Walk: 1 meter (Right) Angular Walk: 1 meter

3.6 Determining Pose

Critical to the mapping portion of SLAM is the robot pose and the distance to the obstacles detected by the sensor. The pose for the AIBO is calculated using two pieces of

information extracted from the robot in real-time. First is how far the AIBO has traveled since its last camera snapshot. The concern here is that the distance traveled is synchronized with the image that has been processed. To do this, the timestamp of the image is used to determine how long the robot has walked since the previous image. This time (milliseconds), $(t_2 - t_1)$ is divided by the established time for 1 meter of forward motion (7820 ms), providing us with distance traveled, *r* (measured in mm).

$$r = \frac{(t_2 - t_1)}{7820} * 1000$$

Next, the angular velocity, ω (radians/sec), captured directly from the remote control software, provides us with direction of the walk, θ (measured in radians):

$$\theta = \frac{\omega * (t_2 - t_1)}{1000}$$

The corresponding rectangular coordinates is calculated from these polar coordinates using the simple calculations:

$$x = r\cos\theta$$
$$y = r\sin\theta$$

The heading of the robot, *z* is computed from the parametric equation produced by the curve-fitting software [37]:

$$z = a + bx^0y^1 + cx^0y^2 + dx^1y^0 + ex^1y^1 + fx^1y^2 + gx^2y^0 + hx^2y^1 + ix^2y^2$$

which identifies the following coefficients:

Coefficients

a = 1.2053355738433053E-02
b = 2.1166856360492894E-02
c = -3.0603347809746204E-06
d = 1.6425508250918786E-04
e = -4.6808579487236736E-05
f = 1.3109629918730776E-08
g = -1.4198703166187395E-07
h = 2.7997429433335293E-08
i = -1.1698188063611851E-11

Absolute Error

Minimum:	-2.574403E-01
Maximum:	2.301725E-01
Mean:	-4.850480E-08
Median:	-1.769848E-02
Sample Variance:	1.115304E-02
Sample Std Dev:	1.056079E-01
Pop. Variance:	1.148107E-02
Pop. Std Dev:	1.071497E-01
Variation:	-2.177268E+06
Skew:	-2.417118E-01
Kurtosis:	1.870974E-01

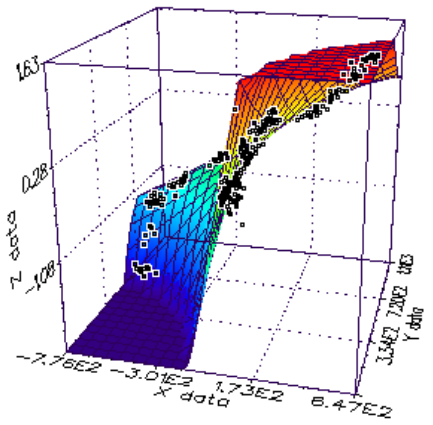


Figure 22: Sensor Model Data Statistics(left) and Curve Fit Equation Surface Plot

This equation is *fit* (see Figure 22) to the average x,y,θ values calculated for data collected at 10° increments and the previously described distances. Their distribution is depicted in Figure 22. The (x,y,z) represent the robot's estimation of its current pose. Initially, the robot's location is (0,0,0) at $t_0 = 0$. Subsequent (x,y) values are accumulated at each time step t (every 3 image frames), rotated by the heading of the previous time slice, $t-1$,

$$x_t += x\cos(\theta_{t-1}) - y\sin(\theta_{t-1})$$
$$y_t += x\sin(\theta_{t-1}) + y\cos(\theta_{t-1})$$

and stored with the corresponding sensor information for that time slice. Finally, the heading, *z,* holds the heading. The SLAM software uses the compilation of information collected over the exploration of a maze to build the map of the AIBO's environment. This is done by plotting the pose and sensor distance readings on a grid having a granularity of 5 cm, meaning each grid cell represents 5 cm in the real world. Each sensor reading describes what is seen by the robot relative to its center of mass. To generalize, the *distance to sensor* describes how far the specified pixel is from the center of the robot, the *theta to sensor* represents the angle between the center of the robot and its camera (accounting for the pan motion), the *range* is the distance between the camera and the pixel, and finally the *sensor theta* is an angular relationship between the camera and each pixel represented in the *scan.* Each sensor reading is plotted using the geometric relationship of these four values as the probability of that grid location accounting for an actual detected object increases. In other words, if the same pixel is identified as containing the reflected laser beam over numerous *scans*, the belief of its true existence grows iteratively stronger. Once the map has been built, the mapping software reviews the map, cleaning up those plotted points with probabilities below the established threshold, leaving us with what we believe is an accurate map of the environment. The next phase of the SLAM implementation involves using the sensor and motion models to determine the accuracy of the pose/sensor estimates currently present in the map.

3.8 Localization

For each set of distanced traveled (the local map), the software compares the estimated pose of the robot with the probability distribution of locations based on the standard deviations derived during testing. It then maximizes the distribution, choosing the particle with the highest probability as the robot's updated pose. The subsequent local map is adjusted to compensate for the newly update pose. In this manner, the robot traces through the robots initial estimations, correcting the map previously constructed.

3.9 Summary

The processes described in this chapter are focused on providing existing SLAM software with the information necessary to perform localization and mapping. The images are used as sensor inputs for what the robot sees as it walks around. The pose is an estimation of its current position relative to its starting location. Our peers in research and academics all over the world successfully use landmark and line-based localization with the AIBO. The primary challenge in this research is developing accurate *sensor* and *motion models* for SLAM. Other research and development teams aren't successfully implementing SLAM with the AIBO because of the complexity developing an accurate motion model for a quadruped robot with 20 degrees of freedom [24]. We have made some assumptions and performed some manual calculations to reduce some that "gray area". In the following chapters present the analysis of both the mapping and localization results using the estimations made in this research software.

IV. Results and Analysis

This chapter presents the factors that impact the sensor data and pose determination. Results of testing and analysis of the motion and sensor models described in the previous chapter are also discussed.

4.1 Sensor Model

The *sensor model*, regardless of the development technique used, is prone to error. In this particular application, error is introduced into the model by several factors, the most detrimental being motion and the environment.

The two central motions contributing to error in the *sensor model* are 1) Body Motion 2) Head Motion and 3) Image Granularity. The jolting walk of the AIBO causes the images captured by the camera to contain *noise*. This *noise* reflects the inability of the camera to capture accurate positions of image participants due constant, rigorous movement. By the time the image is captured, things have moved on in the motion cycle. The calculations relying on extracting accurate locations of features in the image cannot compensate for these random motions. Such *noise* has a negative impact on the accuracy of our distance calculations. In the figures below, the AIBO is traversing a simulated hallway, scanning walls to its left and right. The distances derived from the relative disparity between the horizon and laser line are depicted as the increasingly darkening colors. It is noted that introducing walking motion to this behavior results in a more random sets of distance distributions than if scanning the walls without motion.

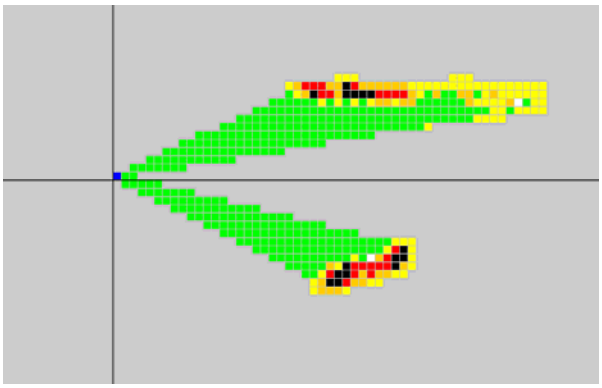


Figure 23: Sensor Readings Without Walk

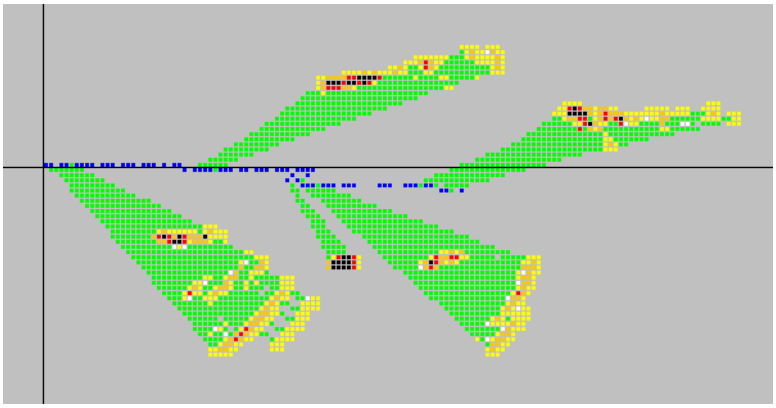


Figure 24: Sensor Readings While Walking

Another factor affecting the accuracy of sensor readings is head motion. The motors of the robot are constantly updating their *state* (every 32 ms), hence the joints are never fully motionless. This affects accuracy of manipulating image data using elements of the AIBO's *world state.* For example, commanding the AIBO to pan it's head from left to right, but telling it not to nod/tilt its head, will not result in nod and tilt elements of the *world state* remaining in their neutral angles, 0° and 30° respectively. While monitoring the updates of the *world state,* slight fluctuations (± ~0.213°) of these

settings are noticed. This preempts any assumption that the control input is truly duplicated by the robot, one obstacle to performing accurate active triangulation. This technique relies on geometric relationships whose consistency cannot be guaranteed, eliminating it as a viable option for determining distance. Figures 25 and 26 show the distance distributions when detecting a wall directly in front of the AIBO with and without panning head motion.

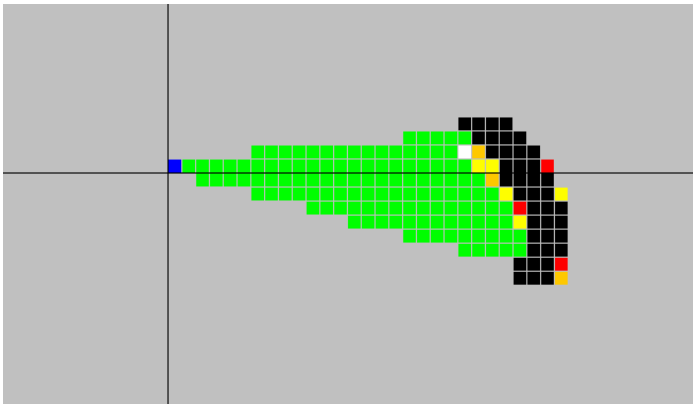


Figure 25: Sensor Readings of a Wall without Head Motion

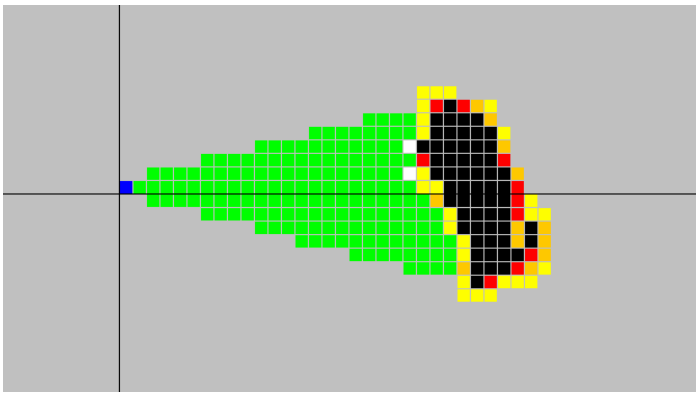


Figure 26: Sensor Readings with Pan Motion

Finally, the environment setting plays a crucial role in the ability to extract the information necessary to accurately describe the locations of features in its image. For

most applications, the overriding concern is lighting. Proper lighting is crucial to "seeing" those objects in the robot's environment to extract particular features or dimensions. In this application, incorporating the Class 1 laser introduces a new facet to the lighting issue. To detect this laser, the darker the environment reflecting the laser, the more pronounced the laser appears in the image. Especially since the detection relies of segmenting the color of the beam from the image. In Figure 27, the left column is the original image and the right column shows the extracted line. This is an example of the

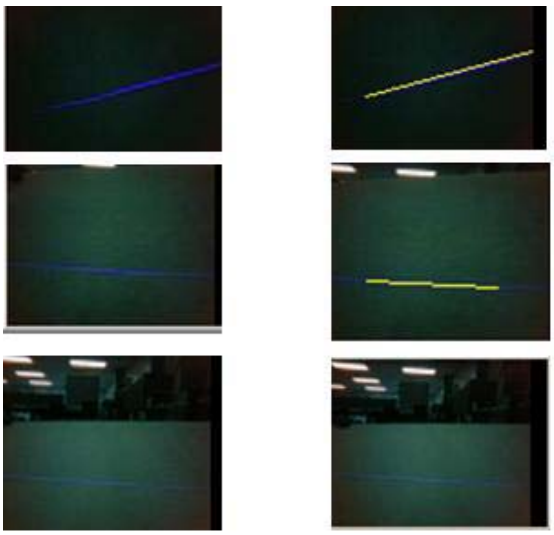


Figure 27: Laser Lines in Different Lighting

increasing brightness of an image reducing the accuracy of the extracted line. When testing the impact of lighting on laser line extraction, over 75% of the images fell into the to darker ranges when navigating the mazes (see Figure 28). This is primarily due to the proximity of the maze walls/obstacles to the camera when the laser line becomes visible, because as the robot moves closer to the maze walls/obstacles, more light filters out of the images.

Figure 28: Hallway Maze, (left) Top View, (right) robot's view

4.2 Pose Model

The accuracy of the motion model doesn't necessarily suffer from the same *noisy* factors that the *sensor model* does. The error of the pose is due in part to the assumptions made during development of the model and in part, the method of data collection upon which the model is based. For example, the assumption that the robot has a forward velocity of 100 mm/sec for all pose calculations may be erroneous. As the angular velocity is varied, the robot is no longer traveling "forward" at that set speed. The model developed in this research performs well when programmed with a constant control input, which makes sense since a constant motion control was used during data collection for the model. Performance deteriorates when the control input is not as smooth, most of them occurring when the robot was driven by remote control. Generally, this means that if the robot is told to perform a continuous 2° turn to the right, and no other control variations are introduced, the robot's pose tracks more accurately. In contrast, if the robot is reactively driven remotely, the resulting pose shows much more *noise* in its tracking. The figures below provide a sample of pose derivations in both situations.

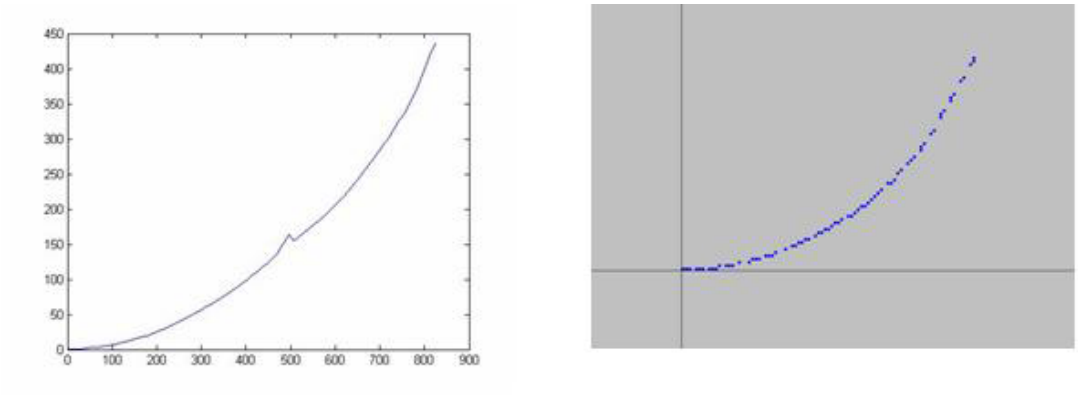


Figure 29: Continuity of Control: Left Turn Pose Tracking. (Left) Actual plotted pose data, (Right) Mapped pose data

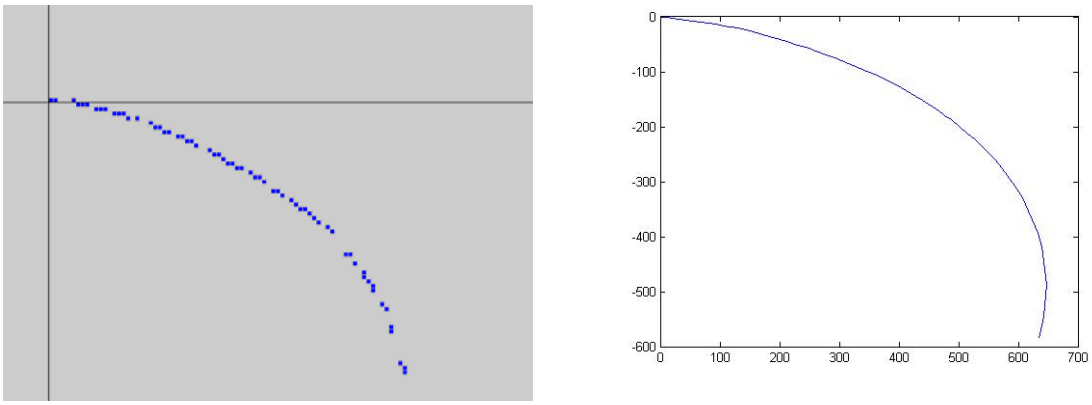


Figure 30: Continuity of Control: Right Turn Pose Tracking. (Left) Actual plotted data, (Right) Mapped pose data

The error of each small distance traveled is accumulated as the *(x,y)* coordinates are accumulated when mapping the pose, shown in Figure 31.

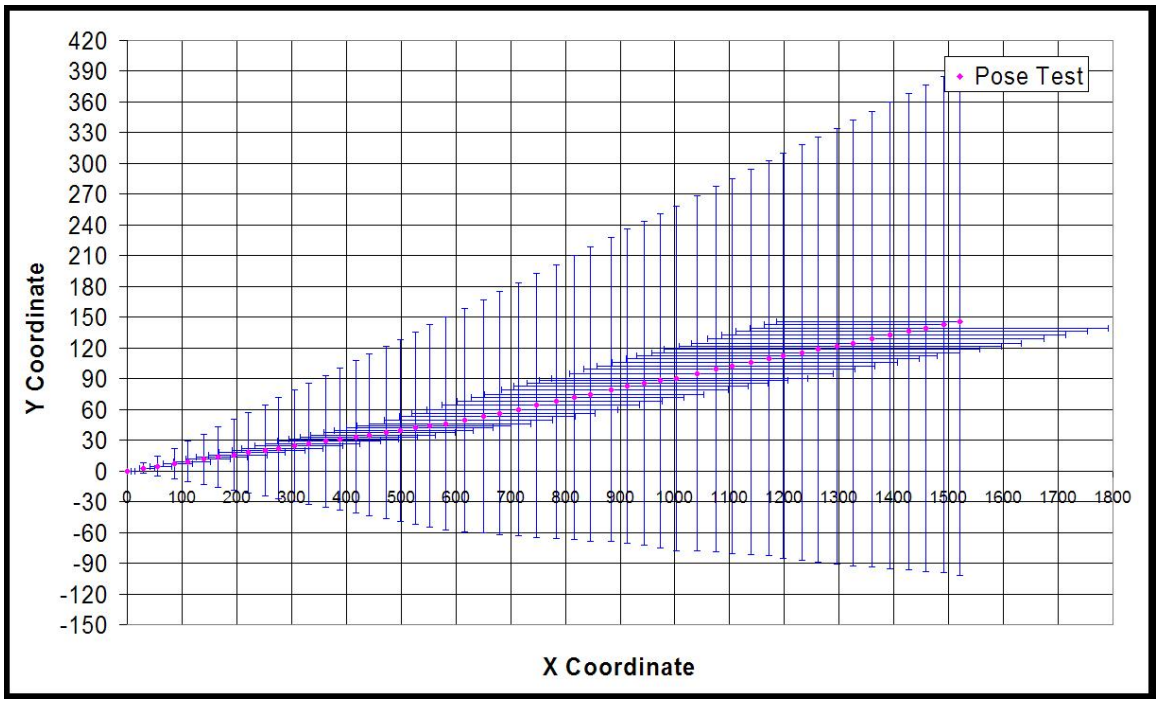


Figure 31: Accumulated Pose (x,y) Error

4.3 Mapping

The process of mapping involves the collaboration of the pose and sensor data to formally create a map of the environment as the robot navigates its environment. The plausible error of this map is an extension of the *noise* found in each of the models described in previous sections. To test the accuracy of the *sensor* and *motion models*, 2 environments were physically designed (see Figure 32), while 2 were simulated. The two mazes through which the robot was navigated were characteristic of the data collection methods described in Chapter 3, only requiring the controller to make small adjustments to the angular velocity setting.

Figure 32: AIBO Mazes

The noise described for the pose model propagates to the map as drift. In the tests run in the straight hallway, the drift is always toward the robot's right. This drift first appeared in data collection, illustrated in Figure 33 and exists when the robot controls the movement. When manually driven, the drift can be compensated out of the map, but this introduces pose error as discussed in section 4.2 . The following maps are test runs through the straight hallway seen above in Figure 32. The large versions of the map are at a 1cm granularity, meaning each grid contains 1 cm of real world space. The smaller versions are at 5cm grid size. Each map suffers from gaps in sensor readings. The two contributing factors for the gaps are head pan speed and navigation technique. In all cases, the head doesn't pan fast enough to capture images of wall segments directly opposite each other, the images are staggered from left to right. For the remote controlled navigation, there are larger gaps in the sensor readings because the robot is swerving from left to right as it is controlled through the maze. The maps for the simulated mazes look cleaner because the walls maintain a constant distance from the

robot, so the only contributing factor is the panning speed of the head. The shape of the sensor scans is slanted because the laser reflects a diagonal line on the walls when not looking at the wall head-on, as in Figures 26 and 27.

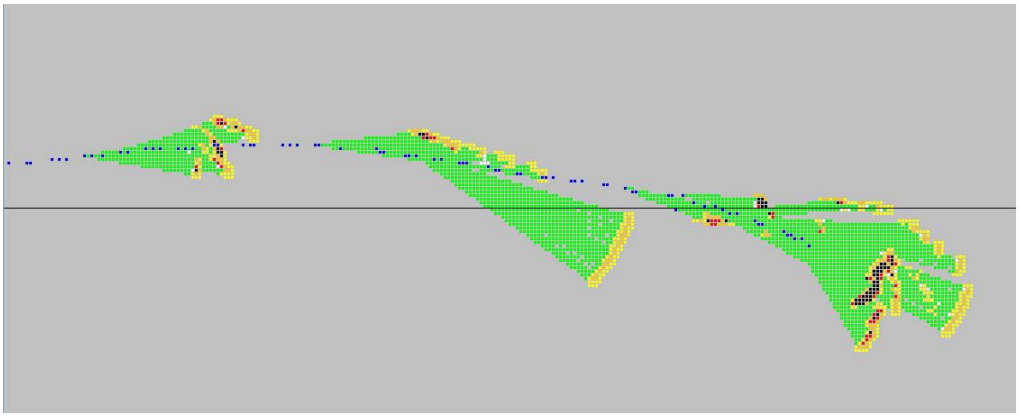


Figure 33: Remote Controlled Navigation Through Hallway Maze, Trial 1

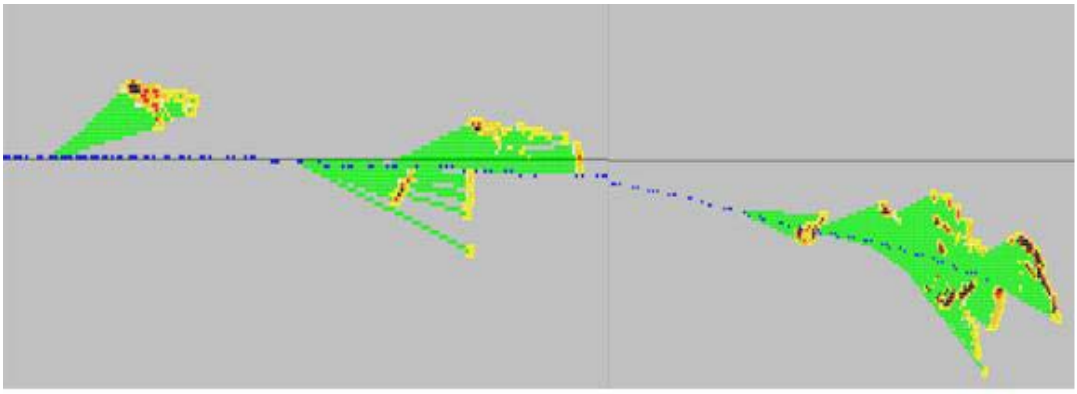


Figure 34: Remote Controlled Navigation Through Hallway Maze, Trial 2

The maps shown in Figures 33 and 34 are the result of manually navigating the AIBO through the physical hallway shown in Figure 32. To illustrate the contrast between the sensor data gathered without the noise-induced walk and sensor data shown in the figures above, the same tests are executed in a simulated hallway without walking (but tests are shown in Figure 35 and 36.

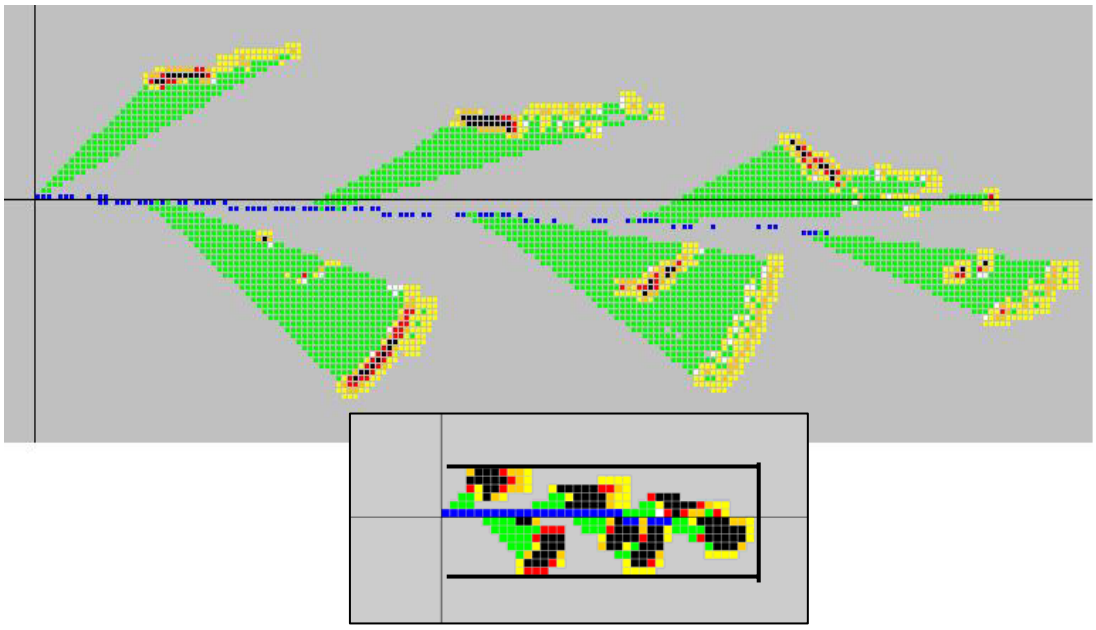


Figure 35: Programmed Navigation Through Simulated Hallway Maze, Trial 1

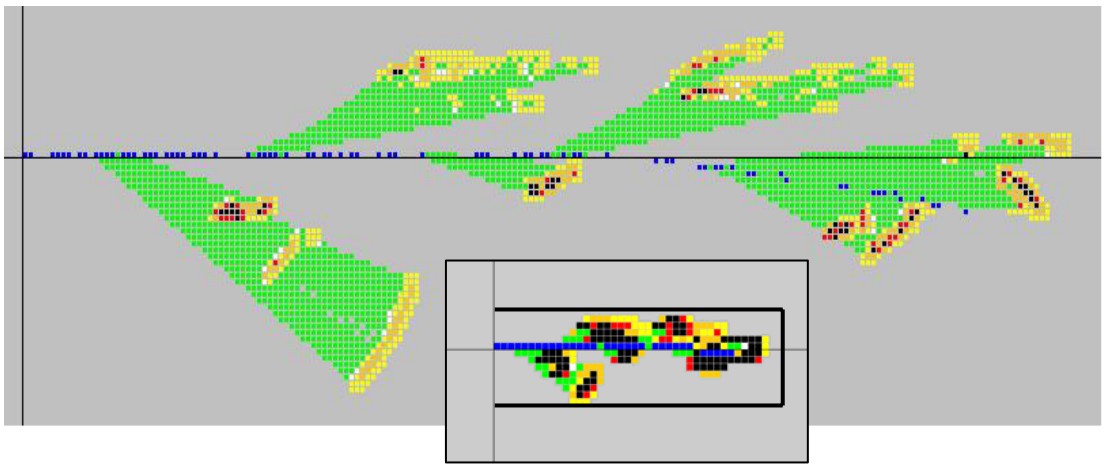


Figure 36: Programmed Navigation Through Simulated Hallway Maze, Trial 2

The simulated mazes were hallways with slow gradient curves to the right and to the left. These tests revealed a shortfall of the sensor data collection. If the robot's head was turned toward the wall it was turning towards, the opposites side of the hall is sometimes missed entirely. It is also dependent on the skill of the controller driving the robot. The more smoothly the robot maintains heading, the more distinct the sensor

scans. The next set of tests was performed in a simulated maze requiring the robot to make small turns to the left or to the right.

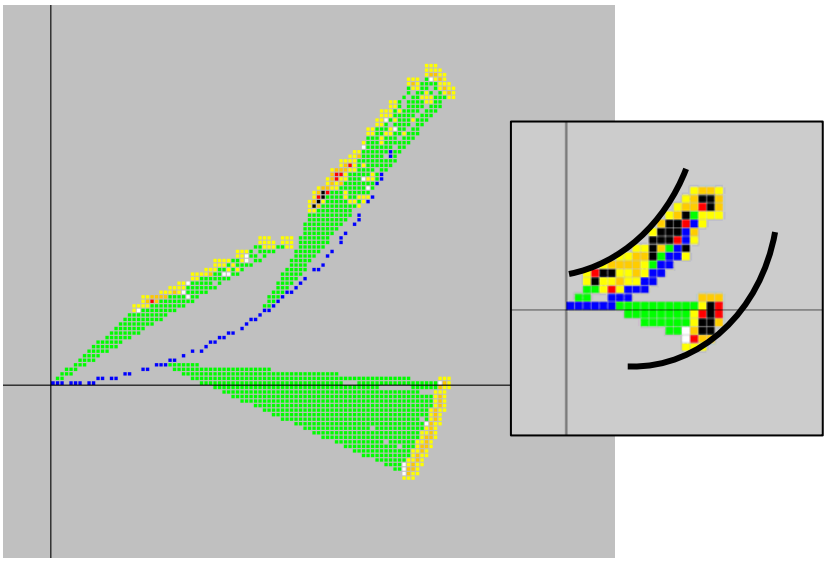


Figure 37: Simulated Maze with Small Left Turn

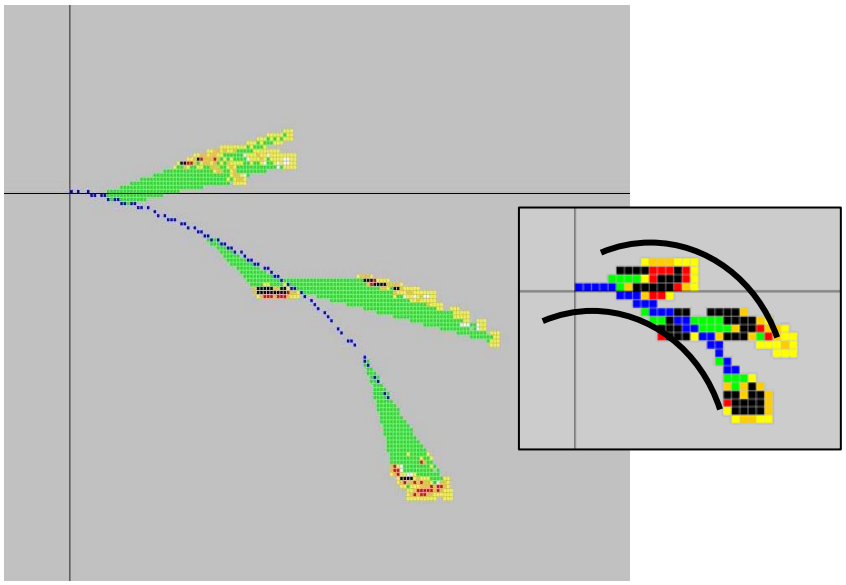


Figure 38: Simulated Maze with Small Right Turn

The final aspect of the pose calculation and sensor data representation tested with the Mapping system is a circular maze. Since each test of the pose demonstrated the ability of the client software to accurately derive the pose, despite small discrepancies in the sensor data, it is presumed that the maze traversal contains accurate pose information, but the sensor readings will overlap and be quite noisy. It is proposed that the some of the inaccuracy in the pose stems from inconsistencies in the panning of the head. It was observed that the head didn't always pan completely to the right, as well as sometimes containing a noticeable jerk when the head reached the maximum pan angle. In such cases, the robot was rebooted and the tests were duplicated. The following figures include three test sets. The first map set is the pose estimations (without sensor data) for a continuous right and left turn.

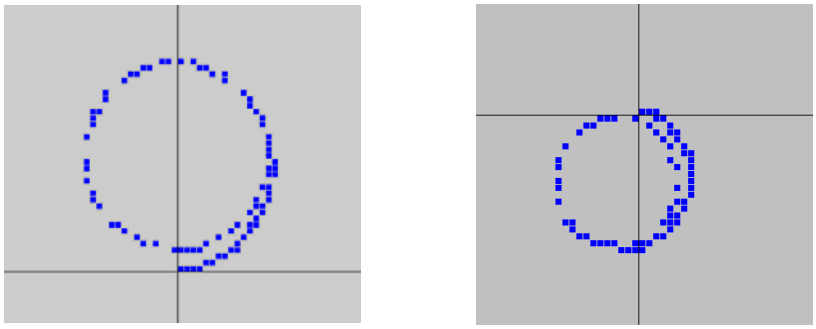


Figure 39: Continuous Turn Pose Estimations: (Left) Left Turn, (Right) Right Turn

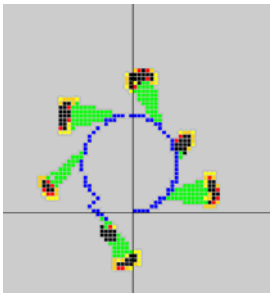


Figure 40: Simulated Round Maze: Off The Ground

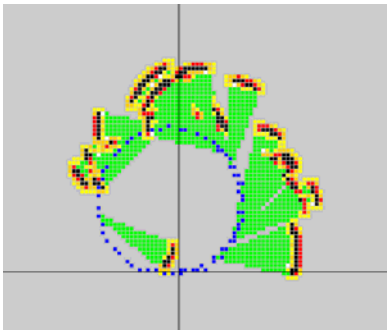


Figure 41: Single Run Through Maze

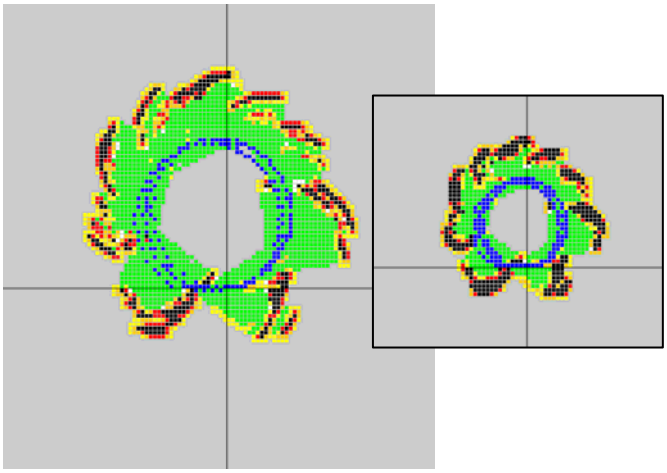


Figure 42: Three Runs Through Maze

4.5 Conclusion

The tests performed in this chapter support the plausibility that distance can be determined by projecting a laser into the image frame. Progress is impeded by hardware and robot complexity, but there exists promise in fine tuning the methods described in this research for vastly improving the accuracy of both models. Suggested alternative methods to improve the development of the *motion* and *sensor models* are discussed in Chapter 5.

V. Future Work and Conclusions

This chapter discusses a variety of modifications to this research that would improve the results of the localization computations and also provides conclusions based on this research.

5.1 Estimation and Assumption Alternatives

The accuracy of the *sensor* and *motion models* is dependent on establishing configurations of the vision and motion systems that reduce the number of estimations and assumptions made to reduce complexity of calculations.

5.1.1 Horizon Estimate

One critical estimate made in developing the *sensor model* concerns deriving the position of the horizon in the image. Using the location of 91.44 cm in space in reference to the *base frame* to determine the location of the horizon is a sound method, but not effective with the current walk. Unfortunately, the robot's walk doesn't involve all four feet, resembling more of a crawl by supporting itself on its rear feet and slightly below the *knee joint* on the front legs. Therefore, since the locations of the points used to create the ground plane are based on the translation vectors of the feet in reference to the *base frame*, the resulting plane tends to be inaccurate. Modifying these vectors involves measuring the distance between the *knee joint* and the point of the leg that contacts the ground and adjusting the translation vectors of the front legs. This in itself isn't a perfect solution, since the contact point of the front leg changes as the leg rotates. An alternative approach is to design a unique walk, exhibiting characteristics that are easier to measure. In doing so, you have a deeper understanding of the body rotations impacting the position

of the horizon. Future implications of designing a unique walk are reducing of the number of assumptions and estimations necessary to develop an accurate *motion model.*

Assumptions are also made when determining the point of intersection of the ground plane (represented by a vector) and the image plane. This requires the calculation of the real world points of intersection and their location in the image. A projection results in a 3D point being by identified by a 2D coordinate. Unfortunately, there isn't an exact science in determining the number of mm each pixel covers; hence this estimation introduces a percentage of error into the calculations. We are confined to using the fields of view and resolution to calculate the projection. In future extension, it would behoove us to perform a precise extrinsic and intrinsic calibration of the camera to provide more accurate measurements of image features.

5.1.2 Sensor Model

Of the two models developed in this research, the *sensor model* has the fewest parameters and is the most flexibility in describing how to determine the distance to obstacles in the robot's environment. The distance is derived from the pixel distance between the horizon line and laser line in the segmented image. Although calculating the pixel distance between these two lines is a precise integer operation, the translation of pixels into real-world measures (mm) isn't as deterministic. Without the previously mentioned camera calibration, this projection calculation requires some educated guess work. In addition to calibrating the camera to increase distance accuracy, using active triangulation to derive the distance to the robot is a more robust method than that used in this research. In our method it is noted that for distances exceeding 40 cm between the robot and obstacle, 4-5 consecutive world distances share the same recorded pixel

distance. Additionally, the average pixel distance used as a reference for each distance, since each distance has inconsistent relative pixel distances. The resulting disparities are recorded in the table found in Appendix C. Since the distance accuracy is dependent on precision in positioning of the horizon, an alternative is to investigate all factors that affect the position of the horizon in the image and establish a camera/laser configuration more conducive to active triangulation. As mentioned before, horizon position factors include stabilizing joint positions of the legs and head and extracting precise kinematic states of the walk, then compensating for these deviations in determining where the horizon appears in the image. Increasing the known information about the kinematics of the robot will propagate throughout the *sensor model.* Angle accuracy and stability are directly reflected in determining the geometric relationship between the robot's center, the sensor, and the obstacle, further improving the robot's knowledge of where an obstacle is relative to its pose.

Estimations in this research are not restricted to developing the *sensor model*. The most complex model developed in this research is the *motion model*. The relative error increased when

5.1.3 Motion Model

Presently, there isn't an established technique for developing an accurate *motion model* for the AIBO. Due to the 54 parameters involved in analyzing the AIBO's walk, assumptions are made to reduce the complexity of the *motion model* and the unknowns outside the scope of this project. The manners in which the characteristics of the walk are gathered have implications for future work. In generating the motion model, we assume a static forward velocity (100 mm/sec). With this set, the controller input only

provides changes in angular velocity to maneuver the robot through the maze. This assumption constrains the motions of the robot in implementation as well as the type of environment navigated. For example, since the test was restricted to smaller turn radiuses, the *motion model* does not accurately determine its pose in an environment comprised of 90° turns. In future extensions, expanding the tested motion sets representing a broader spectrum of navigated environments should better represent the robots navigational capabilities. Additionally, since the robot isn't capable of reporting its estimated pose, this is determined it empirically by gathering information about the walk. This process involves testing the results of the controller inputs which drive the robot straight with/without turning. Compiling the resulting location (x, y, θ) information provides us an estimate of pose given a control input. The distribution $(\Delta x, \Delta y, \Delta\theta)$ of these locations is used as the *motion model* for localization. Although the test set relies on a set forward velocity, the impact of changes in angular velocity on the actual forward velocity is not addressed, another parameter for future investigation.

5.2 Future Extensions

Suggestions for extension of this research are two-fold. First, the camera's poor-resolution restricts the environment to navigate. Without the presence of "good" lighting and distinct patterns in the environment, extracting lines and other features is extremely difficult and inconsistent. For this project, poor resolution restricts the local map to within 67.056 square cm of the robot and requires an increasing number of local maps to build a global map of the robot's physical environment.

The possibility of augmenting the robot with a better camera is nonexistent since the ability to access hardware by the robot is limited to what is already on board. The AIBO doesn't support external sensors being incorporated into its system configuration.

The final extension of this research is to transition to real-time mapping, eliminating the bottleneck created by writing to a file. The theoretical concept of research such as this is to be able to use the data processed from the images while navigating, hence reading and writing files is not conducive to the "big picture" implementation of such concepts.

5.3 Conclusions

This goal of this research was development of the sensor and motion models necessary for SLAM to build a map and self localize, using vision as the primary sensor. Although the resulting models weren't as successful as hoped, it provided insight into previously unidentified factors that must be considered when selecting a robot platform for mapping and localization, specifically head and body motion, image granularity, camera resolution, and accelerometer accuracy.

The theory behind using the relationship between the horizon of the image and the laser line to determine the distance between the robot and an object in its path is supported as plausible by this research. The resulting distances did not achieve the expected accuracy, not due to faulty theory, but due to the nature of the robot used, the poor resolution of the camera installed in the robot, and the scope of the project. Implementing some of the alternatives described above may lead to more precise models for use in SLAM software. Additionally, the lessons learned in this research provide

insight into the impact of kinematic motion on images, key to future success in implementing the same methodology on the new Wheg robot platform.

Appendix A

```
######################################################################
#####################   Tekkotsu config   ############################
######################################################################
######################## $Revision: 1.5 $ ############################
################### $Date: 2005/06/07 00:57:38 $ #####################
######################################################################
######################################################################

######################################################################
######################################################################
[Wireless]
######################################################################
######################################################################
# unique id for Aibo (not used by Tekkotsu, but you might want it...)
id=1

######################################################################
######################################################################
[Vision]
######################################################################
######################################################################

# white_balance  indoor | flourescent | outdoor
<ERS-2*>
white_balance=flourescent
</ERS-2*>
<ERS-7>
white_balance=indoor
</ERS-7>

# gain          low | mid | high
# higher gain will brighten the image, but increases noise
gain=high

# shutter_speed  slow | mid | fast
# slower shutter will brighten image, but increases motion blur
<ERS-2*>
shutter_speed=mid
</ERS-2*>
<ERS-7>
shutter_speed=slow
</ERS-7>

# resolution     quarter | half | full
# this is the resolution vision's object recognition system will run at
resolution=full


### Color Segmentation Threshold files ###
# Threshold (.tm) files define the mapping from full color to indexed color
# You can uncomment more than one of these - they will be loaded into
# separate channels of the segmenter.  The only cost of loading more
# threshold files is memory - the CPU cost of actual segmenting is
# only done when the channel is accessed.

# Included options for color threshold file:
<ERS-2*>
# phb.tm - pink, skin (hand), and blue
```

```
#   note: "skin" is just of people who work in our lab - not a general sampling... :(
# general.tm - general colors, previously 'default'
# ball.tm - standard Sony pink ball definition
# pb.tm - pink and blue
#thresh=config/phb.tm
#thresh=config/general.tm
#thresh=config/ball.tm
#thresh=config/pb.tm
thresh=config/ttt.tm
</ERS-2*>
<ERS-7>
# 7red.tm - just your usual pink/red/purple color detection, nothing too fancy
# ball.tm - standard Sony pink ball definition
thresh=config/7red.tm
thresh=config/ball.tm
</ERS-7>

# the .col file gives names and a "typical" color for display
# the indexes numbers it contains correspond to indexes in the .tm file
#colors=config/default.col
colors=config/ttt.col


### Image Streaming Format ###
# These parameters control the video stream over wireless ethernet
# transport can be either 'udp' or 'tcp'
rawcam_port=10011
rawcam_transport=udp
rle_port=10012
rle_transport=udp

# pause between raw image grabs: 0 for fast-as-possible, 100 for 10 FPS, etc
# in milliseconds
rle_interval=0

# rawcam_encoding   color | y_only | uv_only | u_only | v_only | y_dx_only | y_dy_only | y_dxdy_only
rawcam_encoding=color

# compression       none | jpeg
rawcam_compression=jpeg

# quality of jpeg compression 0-100
rawcam_compress_quality=85

# pause between raw image grabs: 0 for fast-as-possible, 100 for 10 FPS
# in milliseconds
rawcam_interval=0

# apparently someone at sony thinks it's a good idea to replace some
# pixels in each camera image with information like the frame number
# and CDT count.  if non-zero, will replace those pixels with the
# actual image pixel value in RawCamGenerator
restore_image=1

# jpeg algorithm: 'islow' (integer, slow, but quality), 'ifast' (integer, fast, but rough), 'float' (floating point)
jpeg_dct_method=ifast

# log_2 of number of pixels to skip, 0 sends reconstructed double
#   resolution (mainly useful for Y channel, others are just resampled)
#   our eyes are more sensitive to intensity (y channel) so you might
```

```
#   want to send the UV channels at a lower resolution (higher skip) as
#   a form of compression
# rawcam_y_skip is used when in sending single channel, regardless of
#   which channel
# valid values are 0-5
rawcam_y_skip=2
rawcam_uv_skip=3

# you can send the original segmented image
# or an RLE compressed version (which includes some noise removal)
#rlecam_compression   none | rle
rlecam_compression=rle

# this is the channel of the seg cam which should be sent.
# corresponds to the index of the .tm file you want in thresh
rlecam_channel=0

# this is the log_2 of pixels to skip when sending RLE encoded
# segmented camera images, same idea as rawcam_*_skip
rlecam_skip=1


### Camera Calibration ###
# see Config::vision_config::{computeRay,computePixel} to convert
# between world coordinates and pixel coordinates using these values

# focal length (in pixels)
focal_len_x = 198.807
focal_len_y = 200.333

# center of optical projection (in pixels)
principle_point_x = 102.689
principle_point_y = 85.0399

# skew of CCD
skew = 0

# Radial distortion terms
kc1_r2 = -0.147005
kc2_r4 = 0.38485
kc5_r6 = 0

# Tangential distortion terms
kc3_tan1 = -0.00347777
kc4_tan2 = 0.00012873

# resolution at which calibration images were taken
calibration_res_x = 208
calibration_res_y = 160


##############################################################################
##############################################################################
[Main]
##############################################################################
##############################################################################
console_port=10001
stderr_port=10002
error_level=0
debug_level=0
```

```
verbose_level=0
wsjoints_port=10031
wspids_port=10032
walkControl_port=10050
aibo3d_port=10051
headControl_port=10052
estopControl_port=10053
stewart_port=10055
wmmonitor_port=10061
use_VT100=true
# pause between writes: 0 for fast-as-possible, 100 for 10 FPS, etc.
# in milliseconds
worldState_interval=0


##########################################################################
##########################################################################
[Behaviors]
##########################################################################
##########################################################################

### FlashIPAddrBehavior ###

# You probably already know the first 3 bytes for your network
# so you might only want the last byte for brevity
# (valid values are 1 through 4)
flash_bytes=4

# Do you want to automatically trigger this on boot?
# Will use a priority of kEmergencyPriority+1 in order to override
# the emergency stop's status animation
flash_on_start=0

# your-stuff-here?

##########################################################################
##########################################################################
[Controller]
##########################################################################
##########################################################################
gui_port=10020
select_snd=whiip.wav
next_snd=toc.wav
prev_snd=tick.wav
read_snd=ping.wav
cancel_snd=whoop.wav
error_snd=fart.wav

##########################################################################
##########################################################################
[Motion]
##########################################################################
##########################################################################

# Any motion related paths which are not absolute (i.e. do not
# start with '/') will be assumed to be relative to this directory
root=data/motion

# This is the default set of walk parameters
walk=walk.prm
```

```
# The file specified by "kinematics" should define the kinematic
# chains which form your robot.
# "kinematic_chains" lists the names of the chains which should be
# loaded from that file
<ERS-2*>
<ERS-210>
kinematics=/config/ers210.kin
kinematic_chains=Body
kinematic_chains=Mouth
</ERS-210>
<ERS-220>
kinematics=/config/ers220.kin
kinematic_chains=Body
</ERS-220>
kinematic_chains=IR
</ERS-2*>
<ERS-7>
kinematics=/config/ers7.kin
kinematic_chains=Body
kinematic_chains=Mouth
kinematic_chains=NearIR
kinematic_chains=FarIR
kinematic_chains=ChestIR
</ERS-7>
kinematic_chains=LFr
kinematic_chains=RFr
kinematic_chains=LBk
kinematic_chains=RBk
kinematic_chains=Camera

# These calibration parameters should specify the value to multiply a
# desired position by in order to cause the joint to actually reach
# that position.  This is then used both to calibrate joint values
# which are sent to the system, and also sensor values which are
# received back.
# An unspecified joint is by default '1' which will then pass values
# through unmodified.  Only PID joints are calibrated (i.e. LEDs and
# ears are not)
<ERS-7>
#Only the knees and rotors have been calibrated
#This is just kind of a rough calibration since
#I don't know how well it will generalize across
#individual robots anyway.
calibrate:LFr:rotor=0.972
calibrate:LFr:knee~=0.944
calibrate:RFr:rotor=0.972
calibrate:RFr:knee~=0.944
calibrate:LBk:rotor=0.972
calibrate:LBk:knee~=0.944
calibrate:RBk:rotor=0.972
calibrate:RBk:knee~=0.944
</ERS-7>
<ERS-2*>
#ERS-2xx seems to be fairly well calibrated by system, but
#you can always try to do better...
</ERS-2*>

# Sounds to play when turning estop on and off
estop_on_snd=skid.wav
```

```
estop_off_snd=yap.wav

# These values are used by some behaviors to limit the
# speed of the head to reduce wear on the joints
# Units: radians per second
<ERS-2*>
max_head_tilt_speed=2.1
max_head_pan_speed=3.0
max_head_roll_speed=3.0
</ERS-2*>
<ERS-7>
#the pan speed is revised down from Sony's maximum a bit
max_head_tilt_speed=3.18522588
max_head_pan_speed=5.78140315
max_head_roll_speed=5.78140315
</ERS-7>

# If non-zero, robot should attempt to change directions instantaniously
# If zero, robot should change directions more fluidly (following some internal acceleration calibration)
inf_walk_accel=0

console_port=10003
stderr_port=10004

######################################################################
######################################################################
[Sound]
######################################################################
######################################################################
root=data/sound
# volume = mute | level_1 | level_2 | level_3 | <direct dB setting: 0x8000 - 0xFFFF>
# if you directly set the decibel level, be warned sony recommends against going above 0xF600
# However, I believe the commercial software on the ERS-7 runs at 0xFF00
# going above 0xF800 on a ERS-210 causes distortion (clipping) - full volume on a ERS-7 sounds fine
though.
volume=level_3

# Sound playback currently requires all sounds to be the same bit
# rate.  Aperios further requires only either 8bit/8KHz or 16bit/16KHz
# formats
sample_rate=16000
sample_bits=16

# Preload is a list of sounds to cache at boot
# can be either root relative or full path
preload=skid.wav
preload=yap.wav

# Audio streaming settings
# Audio from the AIBO's microphones
streaming.mic_port=10070
streaming.mic_sample_rate=16000
streaming.mic_bits=16
streaming.mic_stereo=true

# Audio to the AIBO's speakers
streaming.speaker_port=10071
# Length of the speaker streaming buffer (ms)
# Streamed samples are sent to the sound manager in packets of this length
streaming.speaker_frame_length=64
```

```
# Maximum delay (ms) during playback of received samples
# If the playback queue gets longer it is emptied.
streaming.speaker_max_delay=1000
```

Appendix B

AIBO Motion Model								
y	x	theta						
0.000	0.000	0.000						
1 Meter								
-112.713	966.788	-0.174						
-50.800	949.325	-0.017						
-146.050	958.850	-0.017						
-79.375	946.150	-0.122						
-112.713	966.788	-0.174						
-114.300	952.500	-0.140						
-120.650	928.688	-0.157						
-138.113	1003.300	-0.157						
-146.050	958.850	-0.017						
-153.988	906.780	-0.140						
-157.163	966.788	-0.174						
-168.275	962.025	-0.192						
-153.988	906.780	-0.140	AVERAGE			STDEV		
-120.650	928.688	-0.157	-126.773	950.164	-0.127	32.396	25.800	0.062
.5 Meter								
-31.750	423.863	-0.140						
-38.100	409.575	-0.140						
-34.925	384.175	-0.157						
-39.688	381.000	-0.192	AVERAGE			STDEV		
-45.720	412.750	-0.192	-38.037	402.273	-0.164	5.267	18.770	0.026
.25 Meter								
-6.350	255.588	-0.157						
-12.700	265.113	-0.209						
-14.288	258.128	-0.105						
-15.875	270.828	-0.140						
-16.828	269.875	-0.174	AVERAGE			STDEV		

-19.050	263.525	-0.157	-14.182	263.843	-0.157	4.407	6.127	0.035
.175 Meter								
-6.350	107.950	-0.035						
7.938	103.188	0.017						
6.350	98.425	0.052						
17.463	96.838	0.070	AVERAGE			STDEV		
9.525	90.488	0.105	6.509	97.102	0.052	7.774	8.121	0.054
4.128	85.725	0.105						
.0875 Meter								
-1.588	52.388	0.000						
1.588	50.800	0.052						
9.525	47.625	0.209						
6.350	46.355	0.174	AVERAGE			STDEV		
-1.588	44.450	0.000	2.858	48.324	0.087	4.944	3.241	0.099
1 Meter								
-263.525	850.900	-0.523						
-287.338	896.963	-0.506						
-300.038	857.250	-0.541						
-314.325	874.078	-0.593						
-320.675	876.300	-0.576						
-320.675	909.638	-0.576						
-323.850	903.288	-0.541						
-323.850	882.650	-0.576						
-327.025	890.588	-0.506						
-334.963	884.238	-0.628						
-336.550	847.725	-0.558						
-347.663	863.600	-0.593						
-350.838	887.413	-0.628						
-350.838	879.475	-0.611	AVERAGE			STDEV		
-374.650	862.013	-0.680	-325.120	877.741	-0.576	27.419	18.654	0.049
.5 Meter								
-44.450	385.763	-0.366						
-53.975	419.100	-0.366						
-57.150	412.750	-0.279						
-50.800	390.525	-0.331						
-44.450	400.050	-0.384	AVERAGE			STDEV		
-41.275	382.588	-0.297	-48.683	398.463	-0.337	6.243	14.892	0.042

.25 Meter								
-11.113	283.528	-0.401						
-15.875	283.528	-0.401						
-16.828	287.338	-0.349						
-16.828	277.813	-0.454						
-23.813	276.225	-0.314	AVERAGE			STDEV		
-36.513	276.225	-0.419	-20.161	280.776	-0.390	8.980	4.656	0.050
1 Meter								
-501.650	809.625	-0.837						
-504.825	823.913	-0.802						
-506.413	809.625	-0.837						
-509.588	804.863	-0.907						
-511.175	800.100	-0.872						
-512.763	790.575	-0.890						
-519.113	809.625	-0.959						
-525.463	811.213	-1.012						
-533.400	820.738	-0.907						
-544.513	795.338	-0.942						
-547.688	766.763	-1.134						
-557.213	781.050	-0.942						
-558.800	768.350	-0.977						
-565.150	766.763	-0.977	AVERAGE			STDEV		
-574.675	755.650	-0.942	-531.495	794.279	-0.929	24.646	21.689	0.082
.5 Meter								
-101.600	400.050	-0.558						
-85.725	385.128	-0.593						
-93.028	397.828	-0.576						
-122.873	385.128	-0.558						
-53.975	372.428	-0.628	AVERAGE			STDEV		
-101.600	371.513	-0.645	-93.133	385.346	-0.593	22.874	12.086	0.037
1 Meter								
-584.200	666.750	-1.169						
-596.900	677.863	-1.012						
-596.900	720.725	-1.151						

-609.600	703.263	-1.134						
-614.363	706.438	-1.116						
-615.950	666.750	-1.116						
-622.300	698.500	-1.204						
-627.063	649.288	-1.116						
-636.588	649.288	-1.116						
-641.350	700.088	-1.099						
-644.525	703.263	-1.221						
-644.525	717.550	-1.186						
-647.700	655.638	-1.204						
-649.288	706.438	-1.029						
-654.050	688.975	-1.064	AVERAGE			STDEV		
-663.575	639.763	-1.099	-628.055	684.411	-1.127	23.294	26.481	0.061
.5Meter								
-92.075	365.125	-0.733						
-61.913	396.875	-0.663						
-85.725	366.713	-0.680						
-66.675	365.125	-0.558	AVERAGE			STDEV		
-79.375	368.300	-0.768	-77.153	372.428	-0.680	12.680	13.730	0.080
1 Meter								
-601.663	596.900	-2.250						
-609.600	603.250	-1.587						
-615.950	596.900	-1.465						
-622.300	593.725	-1.430						
-639.763	573.088	-1.570						
-644.525	600.075	-2.181						
-644.525	546.100	-1.692						
-647.700	588.963	-2.146						
-651.510	550.863	-2.111						
-657.225	555.625	-2.250						
-666.750	549.275	-2.163						
-671.513	547.688	-2.146						
-673.100	542.925	-2.111						
-	558.800	-	AVERAGE			STDEV		

709.613		2.058						
-711.200	584.200	-2.181	-651.129	572.558	-1.956	32.378	23.026	0.307
.5 Meter								
-38.100	434.975	-0.994						
-57.150	446.088	-0.611						
-74.549	409.575	0.733						
-49.213	409.575	-0.558	AVERAGE			STDEV		
-29.528	425.450	-0.436	-49.708	425.133	-0.373	17.426	15.970	0.653
1 Meter								
-15.875	1028.700	-0.017						
-20.638	1028.700	-0.021						
-22.860	975.360	-0.021						
-49.022	1054.100	-0.052						
-57.150	1041.400	-0.066						
-65.088	1054.100	-0.070						
-77.788	1016.000	-0.085						
-92.075	1028.700	-0.096						
-98.425	1000.125	-0.105	AVERAGE			STDEV		
-101.600	1008.063	-0.105	-60.052	1023.525	-0.064	32.630	24.569	0.035
.5 Meter								
-41.275	481.013	-0.122						
-31.750	485.775	-0.052						
-39.688	492.125	-0.105						
-38.100	493.103	-0.140						
-47.625	496.888	-0.070						
-23.813	496.888	-0.035						
-31.750	501.650	-0.122						
-22.225	508.000	-0.017						
-31.750	514.350	-0.035						
-22.225	492.125	-0.017						
-12.700	496.888	-0.017	AVERAGE			STDEV		
-14.288	498.475	-0.035	-29.766	496.440	-0.064	10.918	8.977	0.046
.25 Meter								
-20.638	238.125	-0.087						
-15.875	238.125	-0.070						

-14.288	242.888	-0.105						
-20.638	244.475	-0.140						
-16.828	245.428	-0.035						
-9.525	244.475	0.017						
-12.700	238.125	-0.087						
-79.375	239.078	-0.105						
-11.113	242.888	-0.017	AVERAGE			STDEV		
-12.700	250.825	0.000	-21.368	242.443	-0.063	20.716	4.151	0.051
.175 Meter								
-6.350	123.825	-0.070						
-6.350	117.475	-0.052						
-1.588	115.888	-0.052						
-10.478	130.175	-0.087						
-4.128	127.000	-0.035						
-9.525	123.825	-0.052						
-12.700	117.475	-0.105						
-9.525	115.888	-0.122						
-6.350	115.888	-0.035						
-7.938	112.078	-0.070						
-3.175	112.078	-0.052	AVERAGE			STDEV		
-9.525	111.125	-0.105	-7.303	118.560	-0.070	3.260	6.214	0.029
.0875 Meter								
-4.128	44.450	-0.052						
-1.588	49.213	-0.122						
-4.128	50.800	-0.087						
-1.588	53.975	-0.070						
-3.175	57.150	-0.052						
-6.350	52.388	-0.017						
-3.175	53.975	-0.035						
-4.128	59.373	-0.070						
-3.175	60.325	-0.052						
-7.938	66.675	-0.122						
-6.350	69.850	-0.070						
-3.175	71.438	-0.017	AVERAGE			STDEV		

-1.588	73.025	-0.017	-3.883	58.664	-0.060	1.971	9.133	0.035
.04375 Meter								
0.000	15.875	0.000						
-2.540	19.050	-0.017						
-1.588	20.638	-0.035						
0.000	25.400	-0.017						
-1.588	28.575	-0.105						
-1.588	33.338	-0.087						
0.000	34.925	-0.052						
-3.175	28.575	-0.140	AVERAGE			STDEV		
-3.175	31.750	-0.122	-1.517	26.458	-0.064	1.297	6.688	0.051
1 Meter								
152.400	987.425	0.192						
152.400	987.425	0.192						
203.200	957.263	0.314						
111.125	946.150	0.087						
127.000	981.075	0.105						
138.113	982.663	0.140						
152.400	987.425	0.192						
169.863	933.450	0.244						
203.200	957.263	0.314						
169.863	933.450	0.244	AVERAGE			STDEV		
127.000	981.075	0.105	155.142	966.788	0.193	29.810	21.836	0.080
.5 Meter								
20.638	461.328	0.070						
0.000	431.800	0.035						
3.175	403.225	0.122						
25.400	442.913	0.087	AVERAGE			STDEV		
7.938	406.400	0.087	11.430	429.133	0.080	11.078	24.604	0.032
.25 Meter								
29.528	283.528	0.384						
28.575	285.750	0.314						
20.638	273.050	0.262						
15.875	269.875	0.244						
20.638	283.528	0.297	AVERAGE			STDEV		
7.938	282.575	0.227	20.532	279.718	0.288	8.074	6.556	0.057
.175 Meter								
-16.828	111.125	-0.070						
-19.050	101.600	-0.157						
-6.350	96.838	-0.035						
-4.128	92.075	0.000						
-3.175	88.900	-0.052	AVERAGE			STDEV		

-9.525	80.963	-0.070	-9.843	95.250	-0.064	6.678	10.482	0.053
.0875 Meter								
-6.350	52.388	-0.087						
-12.700	50.800	-0.140						
-6.350	49.213	-0.087						
-9.525	47.625	-0.122	AVERAGE			STDEV		
-3.175	46.038	-0.035	-7.620	49.213	-0.094	3.620	2.510	0.040
1 Meter								
227.013	917.575	0.209						
284.163	907.256	0.331						
212.725	919.163	0.279						
220.663	939.800	0.331						
187.325	904.875	0.227						
195.263	901.700	0.279						
196.850	903.288	0.209						
212.725	919.163	0.279						
219.075	939.800	0.279						
220.663	939.800	0.331						
212.725	919.163	0.279						
227.013	917.575	0.209						
241.300	915.988	0.279						
254.000	931.863	0.349						
227.013	917.575	0.209	AVERAGE			STDEV		
284.163	907.256	0.331	226.417	918.865	0.276	27.938	12.910	0.050
.5 Meter								
34.925	419.100	0.192						
39.688	384.175	0.349						
69.850	414.338	0.262						
54.928	396.875	0.262	AVERAGE			STDEV		
25.400	396.875	0.227	44.958	402.273	0.258	17.535	14.270	0.058
.25 Meter								
7.938	257.175	0.314						
7.938	274.638	0.384						
9.525	276.225	0.401						
15.875	277.813	0.436						
6.350	279.400	0.454	AVERAGE			STDEV		
3.175	269.875	0.366	8.467	272.521	0.393	4.220	8.198	0.050
1 Meter								
292.100	927.100	0.454						
298.450	915.988	0.523						
298.450	903.288	0.523						
301.625	901.700	0.471						
307.975	906.463	0.523						
307.975	895.350	0.576						

319.088	952.500	0.576						
333.375	946.150	0.611						
341.313	895.350	0.558						
360.680	908.050	0.628						
301.625	901.700	0.471						
390.525	887.413	0.663						
292.100	927.100	0.611						
319.088	956.628	0.576						
301.625	941.388	0.471						
360.680	908.050	0.628	AVERAGE			STDEV		
319.088	965.200	0.576	320.339	919.966	0.555	28.179	24.231	0.064
.5 Meter								
61.913	450.850	0.488						
65.088	474.028	0.558						
54.928	461.328	0.454						
54.928	435.928	0.454	AVERAGE			STDEV		
57.150	469.900	0.541	58.801	458.407	0.499	4.526	15.387	0.049
1 Meter								
384.175	812.800	0.977						
412.750	817.563	0.890						
417.513	817.563	1.151						
428.625	819.150	0.977						
430.213	790.575	0.994						
434.975	821.055	1.064						
434.975	800.100	0.977						
439.738	774.700	0.977						
439.738	809.625	1.029						
444.500	790.575	1.012						
447.675	803.275	0.925						
469.900	777.875	0.994						
428.625	819.150	0.977						
434.975	800.100	0.977	AVERAGE			STDEV		
500.063	741.363	1.116	436.563	799.698	1.002	25.633	22.047	0.067
.5 Meter								
80.328	515.938	0.523						
82.550	492.125	0.611						
95.250	476.250	0.541						
90.488	503.238	0.488	AVERAGE			STDEV		
71.438	485.775	0.558	84.011	494.665	0.544	9.247	15.416	0.045
1 Meter								
488.950	708.025	1.221						
520.700	681.038	1.343						
501.650	677.863	1.326						
508.000	665.163	1.326						
519.113	668.338	1.308						
520.700	681.038	1.343						
525.463	690.563	1.343						

539.750	660.400	1.396						
549.275	644.525	1.413						
550.863	661.988	1.361						
561.975	650.875	1.378						
563.563	668.338	1.291						
563.563	631.825	1.378						
568.325	650.875	1.396						
577.850	639.763	1.413	AVERAGE			STDEV		
582.613	628.650	1.448	540.147	663.079	1.355	28.684	21.687	0.055
.5 Meter								
101.600	495.300	0.698						
105.728	479.425	0.663						
114.300	469.900	0.680						
100.013	465.138	0.698	AVERAGE			STDEV		
84.138	469.900	0.663	101.156	475.933	0.680	11.009	12.006	0.017

Appendix C

Sensor Model										
Mm	pixels	pixels	pixels	pixels	pixels	pixels	pixels	pixels	AVERAGE	STD DEV
250.000	1.000	1.000	0.000	1.000	0.000	0.000	1.000	0.000	0.500	0.5345
260.000	1.000	4.000	2.000	3.000	2.000	1.000	3.000	4.000	2.500	1.1952
270.000	4.000	4.000	5.000	4.000	5.000	4.000	4.000	4.000	4.250	0.4629
280.000	7.000	6.000	6.000	5.000	8.000	7.000	5.000	6.000	6.250	1.0351
290.000	9.000	10.000	8.000	9.000	8.000	9.000	9.000	10.000	9.000	0.7559
304.800	9.000	9.000	10.000	12.000	11.000	9.000	12.000	9.000	10.125	1.3562
314.800	12.000	10.000	11.000	11.000	13.000	12.000	11.000	10.000	11.250	1.0351
324.800	14.000	15.000	14.000	12.000	13.000	14.000	12.000	15.000	13.625	1.1877
334.800	15.000	14.000	14.000	16.000	14.000	15.000	16.000	14.000	14.750	0.8864
344.800	16.000	15.000	16.000	16.000	17.000	16.000	16.000	15.000	15.875	0.6409
354.800	17.000	16.000	18.000	17.000	19.000	17.000	17.000	16.000	17.125	0.9910
364.800	16.000	17.000	18.000	17.000	18.000	16.000	17.000	17.000	17.000	0.7559
374.800	18.000	18.000	19.000	19.000	18.000	18.000	19.000	18.000	18.375	0.5175
384.800	20.000	18.000	21.000	19.000	19.000	20.000	19.000	18.000	19.250	1.0351
394.800	20.000	19.000	20.000	20.000	20.000	20.000	20.000	19.000	19.750	0.4629
404.800	20.000	21.000	20.000	21.000	20.000	20.000	21.000	21.000	20.500	0.5345
414.800	21.000	22.000	20.000	22.000	21.000	21.000	22.000	22.000	21.375	0.7440
424.800	22.000	22.000	22.000	22.000	21.000	22.000	22.000	22.000	21.875	0.3536
434.800	22.000	22.000	22.000	22.000	21.000	22.000	22.000	22.000	21.875	0.3536
444.800	23.000	22.000	22.000	23.000	22.000	23.000	23.000	22.000	22.500	0.5345
454.800	23.000	23.000	22.000	23.000	21.000	23.000	23.000	23.000	22.625	0.7440
464.800	23.000	22.000	22.000	21.000	22.000	23.000	21.000	22.000	22.000	0.7559
474.800	24.000	23.000	22.000	23.000	22.000	24.000	23.000	23.000	23.000	0.7559
484.800	24.000	23.000	24.000	23.000	23.000	24.000	23.000	23.000	23.375	0.5175
494.800	24.000	24.000	24.000	24.000	24.000	24.000	24.000	24.000	24.000	0.0000
504.800	24.000	25.000	24.000	23.000	23.000	24.000	23.000	25.000	23.875	0.8345
514.800	24.000	25.000	25.000	25.000	24.000	24.000	25.000	25.000	24.625	0.5175
524.800	25.000	25.000	24.000	25.000	24.000	25.000	25.000	25.000	24.750	0.4629
534.800	25.000	24.000	24.000	25.000	25.000	25.000	25.000	24.000	24.625	0.5175

544.800	25.000	25.000	25.000	25.000	23.000	25.000	25.000	25.000	24.750	0.7071
554.800	25.000	25.000	25.000	26.000	25.000	25.000	26.000	25.000	25.250	0.4629
564.800	26.000	24.000	26.000	25.000	26.000	26.000	25.000	24.000	25.250	0.8864
574.800	26.000	25.000	26.000	25.000	26.000	26.000	25.000	25.000	25.500	0.5345
584.800	26.000	26.000	27.000	26.000	26.000	26.000	26.000	26.000	26.125	0.3536
594.800	27.000	26.000	27.000	26.000	25.000	27.000	26.000	26.000	26.250	0.7071
609.600	27.000	27.000	25.000	26.000	26.000	27.000	26.000	27.000	26.375	0.7440

Bibliography

1. Bach, J. and M. J'ungel. "Using pattern matching on a flexible, horizon-aligned grid for robotic vision". *Concurrency, Specification and Programming* - CSP'2002, 1:11–19, 2002.

2. Carnegie Mellon University Tekkostu Project. "About AIBO Programming". URL http://www.cs.cmu.edu/~tekkotsu/AiboInfo.html.

3. Chen, H., E. Glassman, C. Liao, Y. Martin, L. Shank and J. Stahlman. "AIBO Motion and Vision Algorithms", 2005. URL http://www-2.cs.cmu.edu/~tekkotsu/media/pgss_2003_paper.doc.

4. Cornall, T. "A Low Computation Method to Determine Horizon Angle from Video". *Department of Electrical and Computer Systems Engineering Technical Report,* MECSE-4-2004.

5. David C., K. Yuen and B. MacDonald. "An evaluation of sequential monte carlo technique for simultaneous localization and map-building".I*EEE International Conference on Robotics and Automation 2003*, Taipei, 2003.

6. Dubrawski, A. and B. Siemiatkowska. "A Method for Tracking Pose of a Mobile Robot Equipped with a Scanner Laser Range Finder". *Proceedings of the 1998 IEEE International Conference on Robotics & Automation,* Leuven, Belgium, May 1998.

7. Fox D., W. Burgard, and S. Thrun. "Markov localization for mobile robots in dynamic environments". *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

8. Fujita, M. and K. Kageyama. "An open architecture for robot entertainment". *Proceedings of the First international Conference on Autonomous Agents* (Marina del Rey, California, United States, February 05 - 08, 1997). AGENTS '97. ACM Press, New York, NY, 435-442.

9. Guttmann, J.-S. and D. Fox. "An experimental comparison of localization methods continued". *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'02)*, 2002.

10. Intel. "Open Source Computer Vision Library". URL http://www.intel.com/technology/computing/opencv/index.htm.

11. J¨ungel, M., J. Hoffmann, and M. L¨otzsch, "A real-time auto-adjusting vision system for robotic soccer," in 7th International Workshop on RoboCup 2003 (Robot

World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence, (Padova, Italy), Springer, 2004.

12. J'ungel, M., Hoffmann and J. Lotzsch. A Vision Based System for Goal-Directed Obstacle Avoidance used in RC'03 Obstacle Avoidance Challenge". *Robocup WorldCup VIII, Lecture Notes in Artificial Intelligence,* Springer 2005.

13. Knight,J., A. Zisserman and I. Reid. "Linear Auto-Calibration for Ground Plane Motion". *Proc IEEE Conf on Computer Vision and Pattern Recognition*, Madison, Wisconsin, June 16 - 22 2003.

14. Martin,F., R. Gonzalez-Careaga, P. Barrera, J. Canas and V. Matellan. "Vision Based Localization for a Legged Robot". URL http://gsyc.escet.urjc.es/~fmartin/archivos/locali.pdf

15. MacMahan, W. and J. Bunting. "Puppy Vision Enhancement: Proposed Enhancements to NUbot 2002 Vision System". URL http://murray.newcastle.edu.au/users/students/2002/c3012299/enhance.html.

16. Maybeck, P. *Stochastic models, estimation, and control Volume 1*. New York: Academic Press, 1979.

17. Mertz, C.; J. Kozar, J.R. Miller and C. Thorpe. "Eye-safe laser line striper for outside use", *Intelligent Vehicle Symposium*, 2002. IEEE , Volume 2 , 17-21 June 2002. pp. 507 - 512 vol.2.

18. Negenborn, R.. "Kalman Localization and Kalman Filters on Finding Your Position in a Noisy World", chapter 7, pp. 91-107, 2003.

19. Neira, J., D. Ortin and J.M.M. Montiel. "Relocation using laser and vision". *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004.

20. Neira, J., J. Tardos, P. Newman and J. Leonard. "Robust Mapping and Localization in Indoor Environments using Sonar Data". *The International Journal of Robotics Research,* 2002.

21. Rofer, T. and M. Jungel, "Fast and robust edge-based localization in the Sony four-legged robot league," *7th International Workshop on RoboCup 2003*, *Lecture Notes in Artificial Intelligence*, (Padova, Italy), Springer, 2004.

22. Rybski, R.. "Mobile Robot Localization and Mapping using the Kalman Filter". URL http://www.csu.cmu.edu/~robosoccer/cmrobobits/lectures/kalman.ppt.

23. Shachtman, N. “The Baghdad Bomb Squad”. *Wired Magazine*, Issue 13.11, November 2005.

24. Sridharan, M., G. Kuhlmann, and P. Stone. “Practical vision-based Monte carlo localization on a legged robot”.*The International Conference on Robotics and Automation*, 2005.

25. Thrun, S. (2002). “Particle filters in robotics”. *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*.

26. Thrun, S. “Robotic Mapping: A Survey,”*Exploring Artificial Intelligence in the NewMillennium*, G. Lakemeyer and B. Nevel, eds., Morgan Kaufmann, 2002.

27. Thrun, S., Dellaert, F., Burgard, W. and Fox, D. “Monte Carlo Localization form Mobile Robots”. *Artificial Intelligence Journal,* 2001.

28. Thrun, S, D. Fox, W. Burgard and F. Dellaert, “Robust Monte Carlo localization for mobile robots”. *Artif. Intell. J.* **128** (2001) (1–2), pp. 99–141.

29. Turner, S. “Implementing Segmented Vision Using Tekkotsu”. *Robotics Seminar,* 2003. URL http://www.ils.albany.edu/robotics/SegmentedVisionInTekkotsu.doc.

30. Vail, D. and M. Veloso. “Learning from accelerometer data on a legged robot”. *IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, 2004.

31. Veloso, M., E. Winner, S. Lenser, J. Bruce and T. Balch. “Vision-servoed Localization and behavior-based planning for an autonomous quadruped legged robot”. *In Proceeding of AIPS-00.* 2000.

32. Wikipedia Foundation. “Bresenham’s line algorithm”. Sept 2004 URL http://en.wikipedia.org/wiki/Bresenham’s_line_algorithm.

33. Wikipedia Foundation. “Hough Transfrom”. September 2004. URL http://en.wikipedia.org/wiki/Hough_Transform.

34. Wikipedia Foundation. “YUV”. September 2004. URL http://wikipedia.org/wiki/YUV.

35. Yoon,J., J. Lee and E. Kang. “ Simultaneous Intrinsic and Extrinsic Calibration of Hand Mounted Laser Range Finder“. *Proceedings of the Ninth IASTED International Conference,* September 12-14, 2005, Benidorm, Spain.

36. Yuen, D. and B. MacDonald. “Vision-Based Localization Algorithm Based on Landmark Matching, Triangulation, Reconstruction, and Comparision”, *IEEE Transactions on Robotics,* vol. 21, no. 2, April 2005.

37. ZunZun. “Interactive 2-Dimensional and 3-Dimensional Data Modeling”. URL http://zunzun.com.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
23-06-2006	Master's Thesis	August 2004 – March 2006

4. TITLE AND SUBTITLE

ROBOT LOCALIZATION USING VISUAL IMAGE MAPPING

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Crews, Carrie, First Lieutenant, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/06-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL/SNRP
Attn: Dr. Michael Miller
2241 Avionics Circle
WPAFB OH 45433 785-6127 ext.4274

10. SPONSOR/MONITOR'S ACRONYM(S)
SNRP

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

One critical step in providing the Air Force the capability to explore unknown environments is for an autonomous agent to be able to determine its location. The calculation of the robot's pose is an optimization problem making use of the robot's internal navigation sensors and data fusion of range sensor readings to find the most likely pose. This data fusion process requires the simultaneous generation of a map which the autonomous vehicle can then use to avoid obstacles, communicate with other agents in the same environment, and locate targets. Our solution entails mounting a Class 1 laser to an ERS-7 AIBO. The laser projects a horizontal line on obstacles in the AIBO camera's field of view. Range readings are determined by capturing and processing multiple image frames, resolving the laser line to the horizon, and extract distance information to each obstacle. This range data is then used in conjunction with mapping an localization software to accurately navigate the AIBO.

15. SUBJECT TERMS

Localization, mapping, AIBO, robot, image processing, distance, laser.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert Peterson, AFIT/ENG
U	U	U	UU	117	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636, ext 4281 (gilbert.peterson@afit.edu)