

**A High-Level Modeling Framework for Total System
Performance of Automated Target Recognition in
Distributed Surveillance Systems**

A Thesis
Presented to
The Faculty of the School of Engineering and Applied Science
University of Virginia

Shankha Basu
May 2008

APPROVAL SHEET

The thesis is submitted in partial fulfillment
of the requirements for the degree of
Master of Science (Systems and Information Engineering)

Shankha Basu

This thesis has been read and approved by the examining Committee:

Professor Peter Beling

Thesis advisor

Professor Barry Horowitz

Committee chairperson

Professor Michael DeVore

Committee member

Accepted for the School of Engineering and Applied Science :

Dean, School of Engineering and Applied Science

Date -----

Abstract

This thesis presents a unified modeling approach for the performance analysis of automatic target recognition systems built within a framework of distributed surveillance systems. An idea of characterizing an ATR system with the help of independent sets of control vectors defining the modeling, recognition and hardware characteristics is presented and developed. Additionally, the concept of a system performance vector is introduced to characterize the performance space of an ATR system in terms of the variable components of the control vectors. The performance space that this model characterizes not only encompasses recognition accuracy but also involves other measures of resource consumption such as system throughput and energy usage, presenting a unified view of ATR system performance.

These ideas are used to build and model an actual system implementation on a distributed platform of mobile computing devices. The generic models of system performance are evolved and tailored to demonstrate the process of characterizing the system performance for a specific recognition problem and system topology. The results of this work also demonstrate relationships between recognition accuracy and system resource usage for a real ATR system implementation. Actual tradeoffs between resource usage and recognition accuracy are analyzed and inferences developed. The effect of system control parameters on system performance is quantified and the entire performance space for the set of available controls is characterized and presented.

To Ma and Baba

Contents

List of Tables	iii
List of Figures	iv
Acknowledgements	vi
1 Introduction	1
2 Literature Review	7
2.1 Hardware models	7
2.2 Model-Based Recognition Techniques	10
2.3 Recognition System Performance	11
2.4 Proposed enhancements to ATR System Modeling	12
3 ATR System Characterization	13
4 Performance Characterization	18
5 Recognition Methodology and Algorithm	21
6 Models of Resource Consumption	25
6.1 Hardware Performance Model	26
6.2 Hardware Performance Measures	30
7 Models of Recognition Accuracy	34
7.1 Noise Induction in 3D vehicle models	35
7.2 Discretized Statistical Model Building	36
7.3 Recognition System Implementation and Model Refinement	39
7.4 Recognition Template Masks	40

7.5 Accuracy Estimation	42
8 Resource Usage Results and Model Validation	48
8.1 ATR Testbed Configuration	49
8.2 Resource Usage Model Simplification	50
8.3 Resource Consumption Results	54
8.4 Resource Consumption Model Evaluations	61
8.5 Resource Consumption and Recognition Accuracy Tradeoffs	66
9 Conclusion	69
Bibliography	72
APPENDIX A - Image Rendering and Noise Simulation	76
APPENDIX B - System Implementation	99

List of Tables

7.1	Confusion matrix for recognition performance for complexity level 2	43
7.2	Confusion matrix for recognition performance for complexity level 3	44
7.3	Confusion matrix for recognition performance for complexity level 4	44
7.4	Confusion matrix for recognition performance for complexity level 5	44
7.5	Confusion matrix for recognition performance for complexity level 6	44
7.6	Confusion matrix for recognition performance for complexity level 7	44
8.1	List of available system states along with their associated CPU, Bus and Memory frequencies	55
8.2	Average Execution time values in Seconds for different recognition complexities and system states	57
8.3	Average Power consumption values in Watts for different recognition complexities and system states	57
8.4	Average Energy consumption values in Joules for different recognition complexities and system states	57
8.5	Coefficient values of execution time model variables for different levels of database complexity	63
8.6	Coefficient values of energy consumption model variables for different levels of database complexity	63
8.7	Residual errors for execution time model	65
8.8	Residual errors for energy consumption model	65

List of Figures

6.1	A generic model for a wireless ATR system	25
7-1	3-D Model (Wire Frame) of a Vehicle	35
7-2	Points on Car Body	35
7-3	Displacement of Points after noise addition	36
7-4	Rendered Grayscale images. (a) original model; (b) noisy model	36
7-5	Surface Discretization. (a) $k=1$, first order; (b) $k=2$, second order	37
7-6	Images from two regions of model complexity, $k=7$. (a) and (c) Normalized Mean; (b) and (d) Variance	39
7-7	Target Class and System average recognition accuracy for various levels of database complexity	46
7-8	System average recognition accuracy for various levels of database complexity and differential increase in accuracy for each complexity level	47
8.1	Plot showing variations in CPU and Bus frequencies for different system states	56
8.2	Line and Bar plots for Execution times in seconds for different complexity (k) levels and system states	58
8.3	Power consumption (Watts) plot for various levels of database complexity (k) and system states	58
8.4	Line and Bar plots for Energy Consumption (Joules) for different complexity (k) levels and system states	59
8.5	Comparison of experimental and modeled values for execution time and	64

energy consumption

8.6	Plot of Residual errors for execution time and energy consumption	65
8.7	Plot of Recognition accuracy and average execution time for various complexity levels and hardware system states	67
8.8	Plot of Recognition accuracy and average energy consumption for various complexity levels and hardware system states	68

Acknowledgements

I offer my sincerest thanks and gratitude to several people in the Dept. of Systems Engineering at the University of Virginia. I would especially like to thank Prof. Bass, Prof. Beling, Prof. DeVore and Prof. Horowitz, all of whom have encouraged me to progress with this work. A special note of thanks to Prof. Beling for taking great interest in this research. I really appreciate his efforts to take this work to completion and for his great consideration in accommodating my schedules. I also thank Prof. Horowitz for the resources and advice he has always provided for implementing the ideas developed as part of this thesis. I thank him for allowing me unfettered access to the facilities of the WICAT center without which these ideas would only have remained on paper.

Most of this work was conceptualized and conducted under the guidance of Prof. Michael DeVore and I will be ever thankful to him for the support and guidance that I have received from him. He has always been steadfast in giving me the right directions and I thank him for the endless hours that he has spent with me, giving me great lessons in approaching complex engineering problems in a very elegant way, combining theory with practical aspects of design.

I would like to thank my family members who have always been there for me in spite of the physical distances that separate us. A special note of thanks to my parents and my brother, who have provided me with selfless affection, and have encouraged and motivated me throughout my life, and without their help and best wishes I would not have been here. I also cannot thank Manali enough for all her love and support over the

last few years. She has been a pillar of strength and an eternal source of encouragement for me and I thank her for being there.

I would also like to mention Jianping, who has helped me throughout the course of this work, Deepak, who has taken a keen interest in my research and has always accommodated my demands for time away from work to progress with this thesis, and Xin and Kirtan for their great friendship, advice and insights.

This work has been supported by the Office of Naval Research through grants N00014-03-0110 and N00014-06-1-0061.

Shankha Basu

May 2008

Chapter 1

Introduction

Modern surveillance systems provide advanced capabilities to monitor relevant activities within a region of interest in a designated environment. A surveillance system typically consists of sensor platforms to gather data from its surroundings and in many cases provide mechanisms to store the data gathered and possess a mechanism to detect particular phenomenon after analysis of the data.

Advancements in wireless network technology and vast improvements in computational power available on mobile sensor platforms provide us the capability to implement Automated Target Recognition (ATR) systems contained within the infrastructural framework of such surveillance systems. Such a system can consist of a multiple number of surveillance-recognition nodes stringed together over a wireless network with each node possessing the ability to gather data from its surroundings, the ability to process the data and also have local storage to store and retrieve data to solve a recognition problem on its own. All such nodes may have the ability to co-operate with each other in such an environment and collectively process sensor data to reach a classification result after each sensor collection. Such a classification process is ideal for a distributed surveillance system since it provides the ability to harness the combined computational power and data storage of the whole network, thereby increasing the likelihood of obtaining a classification much faster, and also with more accuracy after a particular collection.

As part of this work, our general goal is to demonstrate and formally characterize the architecture and develop models for analyzing the performance characteristics of such an ATR system. In many such computational systems there are tradeoffs between consumed computational resources, system throughput, and the quality of results output from the system. Common examples exist in gaming and image-rendering applications where the realism of the presented scenes and the speed of rendering vary with the type of computational platform employed. It is a common practice for these applications, when implemented on a distributed platform, to adapt their behavior dynamically to make the best use of available resources. As an example, in the case of distributed gaming applications, the frame rate and information content for frames is lowered for slower communication links and increased when faster links are available. In this thesis we explore and model the relationships between quality, throughput and hardware resources for such computationally intensive applications and apply these ideas to present a high level performance model for target recognition systems built around a data model with variable complexity.

An automated target recognition system classifies targets by gathering data from sensor systems and then evaluating them against stored statistical models of various target classes. We present a unified systems modeling framework to characterize the performance characteristics of ATR systems deploying flexible hardware architectures and databases of varying complexity and size. A typical target recognition system consists of a combination of hardware and software components, which co-operate to recognize inputs to the system and generate responses in the form of classification results.

We present a generic performance model for an ATR system configured as a collaborative network of single processor systems, each of which has a local database of recognition templates. The primary focus of this work is to present a methodology for the high-level performance modeling of a broad category of such data-driven ATR system architectures so as to enable a system designer to choose a combination of software and hardware resources to achieve a set of desired values for various performance parameters. The model also lends itself to a quick exploration of the performance space so as to dynamically and optimally allocate the available resources in case of scenarios where there is a possibility of a change in resource availability and quality. The model hides the complexities of the individual hardware, networking and software performance parameters to present a light-weight integrated performance model which can be utilized efficiently to reconfigure and redesign an ATR system on the fly as and when need arises. The idea behind developing such a model is to provide the designer with an efficient abstraction to let him analyze the overall performance without having to individually investigate the performance of the various components of the ATR system in minute detail.

A fundamental performance measure of a recognition system is the ability to classify targets accurately given a set of target models. However, this ability to characterize targets accurately is also tied to the processing power of the hardware system and the accuracy of the target models present in the recognition system. While highly complex databases may provide better accuracy, they might add to the computational time and energy consumed by the underlying hardware system. In practical operational scenarios there might be hard limits on the processing power and electrical energy available to the

ATR system. There might also be cases where the size and accuracy of the target model databases will be a constraining factor. As a practical example, consider an ATR system operating on a handheld computing device running on batteries and having limited storage for target model databases. In such a scenario, the system operator might like to achieve an optimal performance scenario where he would like to trade off a certain amount of accuracy for the sake of conserving electrical energy by considering models of lesser complexity which gives him the ability to classify more images than would be possible with databases of the highest complexity. The operator may also opt for a lesser clock speed on the processor which will conserve electrical energy but will add to the computational time. On the other hand, given a system with no constraints on energy or storage size, the performance will be constrained only by the performance of the recognition algorithm and the size and accuracy of the target model databases. In such a scenario, it might be possible to run the system at the highest processor speed to achieve the shortest computational time.

The model that we propose will enable a designer or an operator to select a desired level of performance by varying various controllable operational attributes of the system. The performance of the system can thus be characterized as a function of the various controls available, such as the system power, processor speed, database complexity etc. The performance function that results from such a characterization is multidimensional in nature encompassing both hardware system performance and performance of the recognition algorithm employed and carves out a performance space as a function of the various control parameters. The model provides a performance vector which consists of various parameters and lends itself to exploration along the various dimensions of control

available for the system. This enables the designer or operator to choose a specific point of operation in the performance space by providing him with values for each control parameter. On the other hand it also provides him with the ability to predict the performance of the system for a given set of control parameter values. It also provides the ability to consider the various trade-offs in terms of performance by considering various operating points by varying the various control parameter values.

We demonstrate the use of this model by applying it to an ATR system implemented on an embedded platform and employing databases of simulated optical imagery to derive a set of performance characteristics for various processor speeds and separate levels of recognition model complexities. We implement a representative ATR system on an embedded hardware platform employing the Intel®PXA 255 processor based on the Intel® XScale® architecture. The ATR system employs a single processor and stores recognition models in a local file system.

The rest of this thesis is organized as follows: Chapter 2 presents a literature review of various hardware performance modeling techniques and model based recognition systems. Chapter 3 describes a methodology for ATR system characterization with the help of system control vectors for modeling, computational and hardware subsystems. Chapter 4 lays down the performance characterization vector for an ATR system. Chapter 5 describes our target modeling methodology and the recognition methodology applied in our ATR system implementation. Chapter 6 illustrates the hardware model development and the framing of the hardware performance measures for our system. Chapters 7 and 8 describe an actual system implementation and evaluation of performance characteristics

of the system on an experimental platform. Chapter 9 outlines the contribution of this research.

Chapter 2

Literature Review

2.1 Hardware models

The performance of an ATR system in terms of the time required to completely identify a target depends on the underlying hardware architecture. A lot of work has been done in computer system modeling and performance characterization and various measures have been developed to model generic application performance.

Two broad classes of performance evaluation tools have evolved for modeling system performance of computer architectures: Simulation tools and Analytical methods.

Simulation methods enable modeling a system in different configurations without implementing it, potentially allowing the monitoring of an unlimited number of factors that affect performance. An analytical model is a mathematical description of a system, and often can easily show how the performance of a system varies with changes in parameters such as the variation in execution time with the number of processors, letting researchers analyze regions of the parameter space that affect performance in a positive or adverse manner.

The three areas in hardware performance and modeling that are of primary interest to us are:

- 1) Performance characterization of computer systems using simulation tools.
- 2) Performance analysis of Parallel Processing Systems
- 3) Performance modeling of embedded systems, which are gaining increasing importance in today's environments.

Boise and Warn [1] present a very simple model for predicting computer system performance. They describe a cyclic queuing model that is generic in nature and may apply to a variety of computer systems. Austin et al. [41] present the modeling tool SimpleScalar, which uses a simple model that focuses on a design's primary components leaving out minute details. The SimpleScalar methodology uses an execution-driven simulation technique that requires reproduction of the execution of instructions on the machine. Simulation-based power consumption modeling of generic and embedded systems [36 – 37] also often use simulation cores based on the SimpleScalar model. Prete et al. [3] describe a trace-driven simulator that simulates a shared memory, common-bus multiprocessor system with private caches. Larson et al. [5] propose an improvement over trace driven techniques in their simulator, MASE, which incorporates a micro-functional performance model that not only does timing analyses of instructions and data, but also executes instructions similar to microprocessors. They also propose a memory interface that is more realistic by including non-deterministic memory latency modeling.

Adve and Vernon [6] put forward the case for an analytical model for parallel program architectures. They use a conceptually simple technique called a task graph analysis that provides detailed performance prediction for shared memory programs with arbitrary task graphs, a wide variety of task scheduling policies and significant communication and resource contention. Peterson and Chamberlain [7] have developed an analytic model for synchronous iterative algorithms running on distributed-memory MIMD machines, and refined it for discrete-event simulation.

The use of queuing networks to model the performance of parallel processing systems is also widely prevalent in literature [8 - 12]. The Message Passing Interface (MPI) [2]

has evolved as an industrial standard for implementing parallel programs and possible MPI implementations of ATR systems may provide efficient ways to tackle computation intensive ATR problems on a parallel network of workstations. The performance modeling of parallel computer systems have primarily focused on dedicated systems. Recently, non-dedicated network clusters have also assumed significance for high performance computing. Gong et al [12] present a mathematical model to predict performance for such a non-dedicated environment with heterogeneous machine utilizations and service distributions.

The emergence of stable computing platforms on embedded systems, both standalone and distributed, provides an opportunity to assemble highly portable ATR systems on such devices. Various performance modeling techniques have been researched for generic embedded programs and the results may be extended to ATR systems built on such devices. Yen and Wolf [13] present an algorithm for upper and lower bounds for the performance analysis of a set of tasks executing on a heterogeneous distributed system for a general model problem. Wolf [14], [15] presents a survey on the design and development of embedded systems to describe the relationship between hardware and software architecture for embedded system design. Li and Malik [16] provide a framework for performance analysis of embedded programs using implicit path enumeration. They present a method to tightly bind the execution time performance of a given program on a given processor.

Asaad and Bapty [17] present a case study on modeling an ATR system to show the effectiveness of their performance modeling environment for adaptive parallel embedded systems. An adaptive computing system can operate in multiple modes of operation at

any instant. Each mode of operation is regarded as a static computing system. The reconfiguration of the system from time to time gives rise to the adaptive behavior.

Power consumption is an important criterion for the design of computing systems and has drawn significant amount of attention from the research community. Benini and Micheli [18] present a survey of design methods for energy efficient design. Low power processing systems may incorporate dynamically variable voltage supply. Ishihara and Yashura [19] present a model for variable voltage processor and establish theorems for power-delay optimization.

Pouwelse et al [20] examine the performance of a low-power StrongARM embedded Linux platform (LART) in two modes: idle and cpu-intensive. They determine the cases of system performance, memory performance and application performance experimentally.

Tan et al. [21] present a high-level software energy estimation method using the concept of characterization-based macro-modeling where software components like functions or sub-routines are characterized using an accurate lower-level energy model of the target processor.

2.2 Model-Based Recognition Techniques

An innovative model-based approach for ATR based on SAR imagery is presented in [27] employing an optimal search strategy for selecting a candidate among many possible hypotheses [28]. The paper discusses a method to examine a complex hypothesis space for ATR using hypothesis refinement and generation.

The emergence of wireless networks and embedded devices has provided an opportunity to implement distributed strategies for target classification over inexpensive networks of wireless sensor networks. Kotecha et al [38] present a classifier for multitarget recognition in such an environment.

2.3 Recognition System Performance

The performance of recognition systems is often analyzed in terms of the accuracy or probability of error [29, 30, 31] but there have been earlier efforts at combining application and hardware level performance in the recognition systems domain including studies of speech recognition algorithms in terms of accuracy, memory requirements and speed [32] and analysis of recognition error rates with respect to various coding/decoding schemes [33].

DeVore and O’Sullivan have previously developed analytic models models for ATR from SAR imagery relating recognition accuracy to the classification time through the number and speed of processors [34] and to the power dissipated by the system [35].

Much of the above work in performance modeling of hardware systems have often focused on low level simulation of processor architectures and communication networks. Such models are often too detailed for the high level modeling of complex ATR systems which are often designed as software applications running on top of an underlying hardware platform.

Hardware and recognition performance have traditionally been investigated independently of each other. The current work is unique in providing a combined high-level model of hardware and recognition performance for a generic ATR system. This

paper presents a unique unified performance model for an ATR system from a holistic systems perspective. Such a unified model provides a high level of abstraction to the system designer and enables rapid exploration of design space along with dynamic reconfiguration of hardware and algorithmic properties. The framework suggested allows the joint manipulation of hardware and software properties of an ATR system through various control parameters to achieve certain hard performance requirements such as processing time, electrical power consumption or recognition accuracy etc.

2.4 Proposed enhancements to ATR System Modeling

Hardware and recognition performance have traditionally been investigated independently of each other. The current work is unique in providing a combined high-level model of hardware and recognition performance for a generic ATR system. This thesis presents a unique unified performance model for an ATR system from a holistic systems perspective. Such a unified model provides a high level of abstraction to the system designer and enables rapid exploration of design space along with dynamic reconfiguration of hardware and algorithmic properties. The framework suggested allows the joint manipulation of hardware and software properties of an ATR system through various control parameters to achieve certain hard performance requirements such as processing time, electrical power consumption or recognition accuracy etc.

Chapter 3

ATR System Characterization

A generic ATR system can be decomposed into the following three major subsystems for the purposes of system definition and characterization.

1. The modeling and training system i.e. the database component of the system that contains the models of the classes of objects we intend to recognize.
2. The computational algorithm and other statistical techniques we employ to solve the recognition problem.
3. The hardware architecture of the system that provides us the platform to run our computational techniques to achieve our objectives of target classification.

This approach of breaking a generic ATR system into these three primary subcomponents gives us a great amount of flexibility to model a vast category of ATR systems.

3.1 System control vector

We model the functional dependence of the performance space of an ATR system as a function of a system control vector that captures the characteristics of the various subsystems i.e. the modeling, recognition and hardware subcomponents. This approach provides us with parameters that represent the various operational scenarios for the system and encompasses both hardware and software characteristics of the system.

Distinct categorization of the controls we have at our disposal helps us to easily examine various operational scenarios and to select a desired point of operation in terms of system performance. As an example, we might consider using simpler database models if we have a hard limit on computational time available with a fixed number of processors. On the other hand we might increase the complexity of the databases if we have additional processors at our disposal with the same limits on computational time.

We partition the system control vector into 3 sets of components representing the control parameters of generic target recognition systems. We explain the properties of each control vector component in the following sections.

3.1.1 The Approximation Control Vector (α)

The approximation control vector represents the accuracy of the target databases on which the recognition system is based. The constituents of α may vary according to various system configurations. In our system implementation, α characterizes the properties of the optical imagery used to build the recognition models as well as the approximation parameters for recognition models built with the optical imagery. Components of α include various properties of the imagery used in building a target model and also index the number of terms in a piecewise constant approximation of the mean and variance of pixel values in the imagery as a function of the location of the camera with respect to the vehicles in a three dimensional space. The location of the camera is defined in terms of the distance of the camera and the azimuth and elevation angles of the camera with respect to the geometric center of the vehicle.

The various components of α in our ATR system implementation are as follows:

1. Image Pixel Width, width of the sample images in pixels.

2. Image Pixel Height, height of the sample images in pixels.
3. Image bits per pixel, the bit depth of the images.
4. Noise Intensity, the variance of the zero mean Gaussian noise present in the model.
5. Discretization distance, the distance from which the imagery has been collected.
6. Model Complexity, which specifies the total number of perspectives or the number of regions of discretization around the target.

Increases in model complexity, size of imagery and bit depths of images may enable us to achieve higher recognition accuracy but will come with a higher cost in terms of computational time and electrical energy spent. On the other hand, recognition models employing less noisy and detailed imagery might improve the recognition performance but might not affect the usage of electrical energy and computational time when compared to a recognition model database built with noisier imagery photographed from a farther distance.

3.1.2 The Scope Control Vector (σ)

The scope control vector σ , denotes the scope of the recognition problem to be solved. The scope control vector includes components such as the number of classes present in the model database, the number of observations of a target that is available to the system to correctly identify a target class, sensor properties, and operating conditions etc. The vector might also include the kind of the recognition algorithm employed to solve the classification problem.

In our system implementation the scope control vector consists of the following components:

1. Assumed statistical distribution of the observations on which the recognition model databases are built.
2. Dependency model for pixel values.
3. Number of target classes to be differentiated.
4. Algorithm employed
5. Number of Input Observations to the ATR system.

All of the above parameters can be considered as controls which take on various values to influence the performance of the system. Different distribution assumptions for the pixel values result in different likelihood functions which in turn increase or decrease computational time required in classifying a target but might give rise to differences in recognition accuracy. Similarly, if the individual distributions of pixel values are considered to be independent of each other, the computation of likelihood values of each model is much faster and energy conserving but comes with a trade-off in terms of recognition accuracy. Multiple input observations may again increase the recognition accuracy but will also increase the computational cost associated with the recognition algorithm.

3.1.3 Hardware Control Vector (η)

The hardware control vector (η) represents the hardware configuration of the system. A target recognition system may be designed on various configurations, a standalone single processor system or on a distributed computing environment of homogeneous or heterogeneous nature. An embedded system may also provide an opportunity to

implement a recognition system. Representative constituents of the hardware control vector may include the number of processors in the system, system memory, the clock frequency of the processors, the communication bandwidth available to the processors, the available memory size on the processing nodes etc. The elements of the hardware control vector for our system are outlined in Chapter 6.

The justification for using the hardware properties for a system as a control vector arises from the fact that a given operational system might have hard upper limits on the hardware resources available to the system. A system might have limitations in terms of power, memory, number of processors and communication bandwidth etc. For a given operational scenario, we might not have full resource availability at our disposal and might have to consider trading off some of the resources and augment other parameters to achieve a certain level of performance. There might also be cases where a single component of the hardware control vector imposes an absolute limit on the system performance. As an example, increasing the processor speed or the number of processors might not help if a system is limited by communication bandwidth, being unable to obtain more throughputs from target model databases located at remote locations.

Chapter 4

Performance Characterization

The probability of error is a fundamental measure of performance for most target recognition systems and is indicative of the quality of results that an ATR system can achieve for various operational scenarios. The conditional entropy of the target image given the classification result provides us a basis for classifying the system performance in terms of the mutual information between the target database models and the target images the system classifies and also provides us the opportunity to transmit the recognition results in a more efficient way utilizing a lesser number of bits to encode the target image given a recognition template. A highly accurate database of target models resulting in close matches will result in very low entropy values.

In terms of hardware performance, the computational time taken to solve a problem is of essential importance in modern computational systems, which in turn is directly linked to the number of processor cycles consumed by the hardware platform. The number of data bits communicated gives us an estimate of the recognition problem as well is indicative of time taken for communication purposes in distributed architectures. Electrical energy consumption is also a measure of primary importance for most embedded and mobile systems these days and assumes significance in our performance model as it is a limiting factor on the recognition performance we can achieve if we are faced with hard limits on electrical energy we can consume.

4.1 Performance Vector

A vector $\delta(\alpha, \sigma, \eta)$ denotes the measures of performance for a recognition system. The parameters that constitute the performance vector are dependent on the control vector triple (α, σ, η) . The components of δ represent the recognition and hardware performance measures of an ATR system as a function of the system control vectors.

The problem that we look into primarily consists of characterizing a performance space $\delta(A; \Sigma, H) = \{ \delta(\alpha; \sigma, \eta) : \alpha \in A, \sigma \in \Sigma, \eta \in H \}$ for a target recognition system. The dimensionality of the performance space is of a maximum order of $\dim(\alpha) + \dim(\sigma) + \dim(\eta)$ where $\dim(\alpha)$, $\dim(\sigma)$, $\dim(\eta)$ are the free parameters in the control vector sets α , σ and η .

The components of the system performance vector, δ , for our system implementation are outlined below and can be grouped into three major groups:

Measures of accuracy

- 1) $P_{\text{Err}}(\alpha, \sigma)$, the probability that a target is wrongly classified.

Measures of throughput

- 3) $T(\alpha; \sigma, \eta)$, the time performance parameter, i.e. the time required by the system to classify a target.
- 4) N_{Bits} , the number of bits communicated to process a single target model.

Measures of resource consumption

- 5) $E(\alpha; \sigma, \eta)$, the total electrical energy utilized by a system to classify a target.

6) $P(\alpha;\sigma,\eta)$, the average electrical power consumption, given by

$$P(\alpha;\sigma,\eta) = E(\alpha;\sigma,\eta)/T(\alpha;\sigma,\eta).$$

There are several other potential candidates for performance characterization that we do not specifically aim to model as part of this work. Latency measurements and probability of false alarm are two such examples.

Chapter 5

Recognition Methodology and Algorithm

In the ATR system that we develop and characterize, we take inputs in the form of two dimensional digital grayscale images of vehicles in various environmental settings. The target recognition algorithm then works by conducting a search over a database of target descriptions or models, which in our case is a set of statistical models that characterize the distribution of pixel intensities of the target surface as functions of target class and pose relative to the camera. The target models that we employ assume independent Gaussian distributions for pixel values in the vehicle images, conditioned on the pose of the vehicle with respect to the camera. Our target model databases consist of mean and variance values of pixel intensities of the target surface which are estimated from a repository of training images. The recognition methodology then consists of maximizing the likelihood for a given input image over various target models available in the database.

The recognition algorithm we derive in this Chapter is based on the modeling methodology that we have developed which consists of building a statistical mean and variance database of the pixel values contained in the images of the target photographed from various perspectives. The examination of the distribution characteristics of the pixel values R_i of a target image where $i = (x,y)$, the pixel index, when photographed from a mean position (r, θ, ϕ) in space, (where (r, θ, ϕ) represents the location of a point in three-dimensional space in polar co-ordinates), follows a Gaussian distribution with mean M_i

and variance S_i , conditioned upon a target class a , pixel location i , and mean camera location (r, θ, ϕ) , represented in polar co-ordinates.

Under the assumption that the individual pixel values are all independent in a given image, the probability density for the entire image of the target is a product of the individual pixel densities and is given as:

$$P(R|a, r, \theta, \phi) = \prod_{i=1}^{N(a,r,\theta,\phi)} \frac{1}{\sqrt{(2\pi S_i(a, r, \theta, \phi))}} \exp - \frac{(R_i(a, r, \theta, \phi) - M_i(a, r, \theta, \phi))^2}{(2 S_i(a, r, \theta, \phi))} \quad (5-1)$$

where N is the total number of relevant pixels on the target model. It is also to be noted that the distribution is conditional upon the location of the sensor platform i.e. the camera, with the image plane always normal to center of the target.

The number of pixels of interest, $N(a, r, \theta, \phi)$, depend on the location of the car with respect to the camera and also on the number of relevant pixels that are available to us in the mean-variance model.

For a given input image, our recognition algorithm is designed to identify the most likely **target class a and the most likely target pose (r, θ, ϕ)** from all available models using a generalized likelihood ratio test [39]. We also have to account for the fact that for a given input image the region of interest or the location of the center of the target may not always coincide with the center of the image. This requires us to extract and evaluate l possible regions within the input image itself and then compute the likelihoods for each such case within the input image itself. The target and the region of discretization are then selected for the case l that maximizes the probability density as in

$$\begin{bmatrix} a \\ r \\ \theta \\ \phi \\ l \end{bmatrix} = \arg \max_{[a,r,\theta,\phi,l]^T} \prod_{i=1}^{(N(a,r,\theta,\phi))} \frac{1}{\sqrt{(2\pi S_i(a,r,\theta,\phi))}} \exp \left[-\frac{(R_i(a,r,\theta,\phi) - M_i(a,r,\theta,\phi))^2}{(2 S_i(a,r,\theta,\phi))} \right] \quad (5-2)$$

In our experimental setup the distance for the models from the camera is constant for all models so the above equation can be simplified to exclude r . The solution to the above equation yields the target class $\mathbf{a}$, and the most likely pose which is characterized by the angles θ and ϕ and also the most likely location l of the target within the input image.

The above equation holds true for the image of the object against a null background. However in a real life scenario, the image of a scene containing a target object will also have a background along with the object so the equation has to be modified to compensate for the presence of a background. The compensated equation takes the following form for 16 bit grayscale images

$$\begin{aligned} \begin{bmatrix} a, r, \theta, \phi, l \end{bmatrix}^T &= \arg \max_{[a,r,\theta,\phi,l]^T} \left\{ -0.5 \sum_{i=1}^{(N(a,r,\theta,\phi))} \ln(2\pi S_i(a,r,\theta,\phi)) \right. \\ &\quad \left. - \sum_{i=1}^{(N(a,r,\theta,\phi))} \frac{(R_i(a,r,\theta,\phi) - M_i(a,r,\theta,\phi))^2}{(2 S_i(a,r,\theta,\phi))} + (N_T - N(a,r,\theta,\phi)) \ln \left(\frac{1}{65536} \right) \right\} \end{aligned} \quad (5-3)$$

In our scenario, we assume a gaussian distribution for the pixel values which lie on the target surface, and a uniform distribution for the rest of the $\mathbf{N_T - N(a,r,\theta,\phi)}$ pixels, where $\mathbf{N_T}$ is the total number of pixels in the target image and $\mathbf{N(a,r,\theta,\phi)}$ is the number of pixels that correspond to the target surface in the target model. The term $(\mathbf{N_T - N(a,r,\theta,\phi)}) \ln(1/65536)$ accounts for the assumption that background pixel intensities

follow a uniform distribution in grayscale values from 0 to 2^{16} . N_T is the total number of pixels in the scene and is constant for the input image for all the models we test. Therefore, N_T may be neglected in the above expression for the likelihood test so the above expression simplifies to

$$\begin{aligned} \left[a, r, \theta, \phi, l \right]^T = \arg \max_{[a, r, \theta, \phi, l]^T} & \left\{ -0.5 \sum_{i=1}^{(N(a, r, \theta, \phi))} \ln (2 \pi S_i(a, r, \theta, \phi)) \right. \\ & \left. - \sum_{i=1}^{(N(a, r, \theta, \phi))} \frac{(R_i(a, r, \theta, \phi) - M_i(a, r, \theta, \phi))^2}{(2 S_i(a, r, \theta, \phi))} - N(a, k, \theta, \phi) \ln \left(\frac{1}{65536} \right) \right\} \end{aligned} \quad (5-4)$$

Chapter 6

Models of Resource Consumption

In this Chapter we lay down the models of resource consumption for a generic ATR system implementation. The generic ATR hardware system we propose consists of a wireless network of processing nodes with each node possessing a local database in the form of a collection of files resident on a CF flash memory card containing models for various vehicles and sufficient local memory to store all pixels of a digital image of a vehicle. During the operation of the ATR system, one of the nodes behaves as a manager node and acts as the source of images for all other computing nodes.

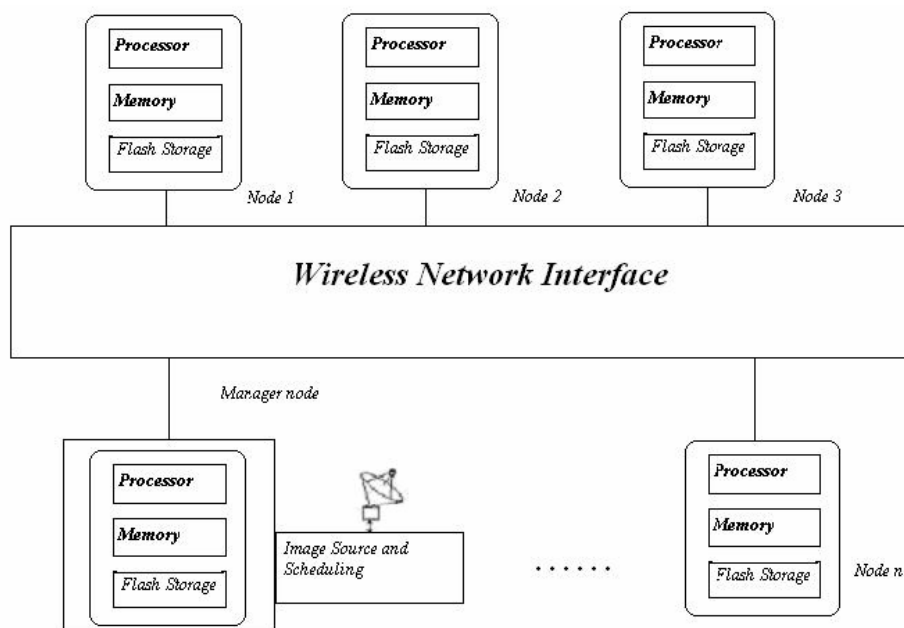


Fig. 6.1. A generic model for a wireless ATR system

The recognition algorithm in itself employs a simple distributed architecture. Each node receives an image from a manager node and computes the likelihood for each image transmitted to them and sends the results back to the manager node which fuses the results from each node to make a final decision.

The manager node implements a scheduling policy that allocates jobs on a first come first serve basis to any node that finishes a task and has computing resources available. We assume a scenario where the manager node may not have broadcast capabilities and communicates with each node individually.

6.1 Hardware Performance Model

In Section 3.1.3 we stated a basic definition of the hardware control vector η , a constituent of the system control vector triple (α, σ, η) . The constituent elements of η for our system implementation are derived in this section. The hardware control vector elements are then used to develop the hardware performance models for the ATR system implementation in later sections.

The hardware parameters for our system implementation are as follows:

1. The number of processors available in the system, N_p
2. The processor clock frequency, F_p
3. System Memory Frequency, F_m .
4. System Bus Frequency, F_b .
5. Clock cycles per instruction, CPI.
6. Processor capacitive constant, C_p .
7. Switching factor, α_s .

8. Slope of the processor's voltage-frequency curve, K_v .
9. Y-intercept of the processor's voltage-frequency curve, V_0 .
10. Memory access time per word, T_{MEM} .
11. Processor Word Size, W .
12. N_s , the number of input images to the ATR system.
13. Size of node local memory, M_{SIZE} .
14. Number of statistical models stored at each node, N_c .
15. Memory idle power constant, P_M .
16. Flash Memory power constant, P_{CF} .
17. Energy expended on memory loss, E_M .
18. Recognition algorithm computational instruction count per pixel, RIC .
19. Database access throughput, BW .
20. $f(k)$, the number of database models processed image expressed as a function of recognition model complexity, k .

We use these parameters to specify the basic models for processor, network and memory components of the system and develop the generic hardware performance model based on them in later sections.

6.1.1 Processor Model

The classic equation for dynamic power consumption in VLSI circuits is $P = \alpha_s C_P F_P V_P^2$. Combining this relationship with the linear relationship for dynamic voltage scaling circuits [23], [24], [26], $V_P = K_v F_P + V_0$, we obtain the energy consumed for executing n instructions as,

$$E = \alpha_s C_P F_P (K_v F_P + V_0)^2 \cdot CPI \cdot \frac{n}{F_P} = \alpha_s C_P (K_v F_P + V_0)^2 \cdot CPI \cdot n \quad (6-1a)$$

where the term $CPI \cdot n/F_P$ is the time required to execute n instructions. α_s and CPI are values that are determined to predict long term energy consumption so as to evaluate average power and energy performance of the system. In scenarios where frequency scaling is applied without voltage scaling or where system architectures limit voltage scaling options, the above equation can be simplified to model the energy consumption for n instructions as

$$E = \alpha_s C_P V_P^2 \cdot CPI \cdot n \quad (6-1b)$$

6.1.2 Memory Model

We model the combination of node local memory and processor data cache as imposing an average processor delay of T_{MEM} seconds for each word read from or written to the processor. The memory and processor may or may not be clocked together. We model this relationship as $T_{MEM} = \beta/f_P$, where β accounts for the difference in system bus and processor clock speeds and taking into account some average cache miss rate. In many architectures, where memory and system buses operate asynchronously, performance can vary unexpectedly with different bus to memory frequency ratios.

DRAM memory can be modeled as having two components to its energy consumption [25]. The first is linear in the number of bits stored and represents the idle power required to refresh the dynamic storage cells. The second is linear in the number of bits accessed and represents the power required for driving the sense amplifiers and

delivering the signals on-chip to the processor. The average energy consumed during an interval of τ seconds during which n bits are moved between the processor and a memory of M_{SIZE} bits is $\tau P_M M_{SIZE} + n E_M$.

CompactFlash memory used as a local storage for database models also contributes to processor delay and energy consumption during the course of algorithm execution. The CompactFlash[40] specification states the maximum average RMS current for CompactFlash cards as 75 mA at 3.3V and 100 mA at 5V. The average power consumed by the CF card can thus be approximated by a average power consumption P_{CF} , the product of steady state voltage and average input current during program execution involving a steady state data access rate for the CF card. The delay in processing due to CF card access is modeled as a time delay due to bandwidth constraints and is incorporated into the model as database access time (Eqn 6-8).

6.1.3 Network Model

We model the design of the ATR system as a group of mobile wireless hosts, with one of the nodes acting as a fixed manager node and implementing a control policy. The energy consumption of a an IEEE 802.11 wireless network interface [41] in an ad-hoc mode can be approximated by a linear equation

$$E = m \times size + b$$

where a fixed part is associated with the channel acquisition and an incremental part which is proportional to the size of the packet. The communication energy for transmitting n bits can then be represented by

$$\frac{n}{size} (m \cdot size + b) = (m + b') n = E_{NET} \cdot n$$

where the size of the packet is assumed to be constant.

Additionally, protocol processing by the processor and data copies (moving the data between the network interface and local memory) also contributes to the energy costs associated with networking at each node.

The communication energy consumed at a node thus involves a signaling term of E_{NET} Joules/bit, a processing term accounting for CIC instructions per bit, and a term representing memory access. The average energy, E_{COM} consumed by a node while communicating n bits can then be represented as the sum of the above two terms

$$E_{COM} = \left(E_{NET} + C_P (K_v F_p + V_0)^2 \cdot CPI \cdot CIC + E_m \right) n \quad (6-2)$$

6.2 Hardware Performance Measures

In this section we define measures of performance in terms of the steady state behavior of a processing node from the time image distribution begins until all available target models have been processed. The total number of bits involved while processing an image consists of the bit size of the input images retrieved from the manager node over the network and the total bit size of the locally resident statistical models processed by the recognition algorithm. The images are communicated over a network to all processing nodes for recognition processing. The data size of these transmissions is

$$N_{BITS}(k) = (N_P N_T) \cdot B P P \quad (6-3)$$

The argmax function of (Eq 5-4) comprises the bulk of the computational overhead in the recognition algorithm. Given a single image and single mean-variance sub-model, we compute the likelihood for each pixel value, assuming independence between various pixel values. Associating an average of RIC instructions for each pixel to compute the

expression in the argmax function, and assuming that we precompute the first term of the expression and store it in a database, we require $RIC \cdot N(a, \theta, \phi)$ processor instructions to calculate the likelihood value for a single input image and a single mean-variance sub-model. This will be accompanied by $3N(a, \theta, \phi) \cdot BPP / W$ read operations to retrieve R_i , M_i and S_i from local memory (assuming one clock cycle each with pre-fetch). These arithmetic and memory operations will be performed for each of the N_S target images, $f(k)$ sub-models, and N_{TC} target classes for a total of

$$N_{CYC}(k) = \left(RIC \cdot CPI + 3 \frac{(BPP)}{W} \right) \cdot N_S \sum_{j=1}^{N_{TC}} \sum_{i=1}^{f(k)} N(a_j, (\theta, \phi)_i) \quad (6-4)$$

clock cycles spent on computation for each model.

The total number of processor cycles spent is a sum of the number of cycles required for likelihood computation and the cycles required to fetch the statistical models and input image and store them in memory. We also add a constant overhead N_{COM} for all the nodes involved in processing to account for the time spent in communication, i.e. receiving image(s) for classification from the manager node and transmitting the results back to the manager node.

$$N_{CYC_{TOT}} = \left(RIC \cdot CPI + 3 \frac{(BPP)}{W} \right) N_S \sum_{j=1}^{N_{TC}} N(a, (\theta, \phi)_i) + \frac{N_{BITS}(k)}{W} + N_{COM} \quad (6-5)$$

These are divided evenly over N_P processors clocked at F_P cycles per second, for an elapsed computation time of approximately

$$T_{\text{COMP}}(k) = \frac{N_{\text{CYCTOT}}}{(N_P F_P)} \quad (6-6)$$

The number of bits that must be communicated from the model database is given by

$$\left(\sum_{(a, \theta, \phi)} 2 N(a, \theta, \phi) \right) \cdot B P P \quad (6-7)$$

and represents the total number of combinations of statistical models stored over all nodes.

We have modeled the database as a collection of files residing on a local file system mounted on a compact flash card and accessible via an interface capable of BW bits per second, so the amount of time spent in database communication is approximately

$$T_{\text{DB}}(k) = \left(\sum_{(a, \theta, \phi)} 2 N(a, \theta, \phi) \right) \cdot \frac{BPP}{BW} \quad (6-8)$$

Assuming that a part of the models might be cached in local memory, or may be communicated over a network, the database access time can be multiplied by a factor ρ . Thus the average elapsed time to classify an image chip is the sum of time consumed for computation and database access and is given by

$$T_{\text{TOT}}(k) = T_{\text{COMP}}(k) + \rho T_{\text{DB}}(k) \quad (6-9)$$

where ρ may vary between 0 (completely cached) to 1 (no caching) for locally resident databases, or may take values more than one to account for network communication overheads in case of network databases. Assuming a constant cache miss rate, where a

part of the database is cached in memory and the rest resides on external storage, the cumulative database access time can roughly be assumed to be inversely proportional to system bus frequency, F_b for a constant bandwidth BW.

The energy consumed consists of electrical energy corresponding to N_{CYTOT} processing cycles, energy consumed in memory reads while processing each recognition template, E_{MR} , energy consumed in network communication, E_{COM} and additionally, idle memory also consumes power during the whole of the computational time $T_{TOT}(k)$ and is proportional to the size of the memory. So the total energy utilized while processing N_{TC} models of order k for N_S input images is

$$E_{TOT} = \alpha_s C_p (K_V F_p + V_0)^2 N_{CYTOT}(k) + P_M M_{SIZE} T_{TOT}(k) + E_{MR}(k) + E_{COM}$$

(6-10)

These models present a very generic approach to model the resource consumption values for time and energy for any ATR system. However, there is further scope for simplification once the actual implementation of the ATR system is finalized. There are several constants in these models which may or may not be known to the designer, while there are variables such as F_p and k which might be readily known and might be the only controls that are available to characterize the performance curve. Additionally, systems may not support frequency and voltage scaling, and there might be other system frequencies in the system which may introduce new variables. However, these equations can serve as a very good starting point for further refinement and tailoring to meet the needs of a final implementation. In Chapter 8, we demonstrate this idea by deriving a real life model for a testbed ATR application by further refining and evolving the models developed in this Chapter.

Chapter 7

Models of Recognition Accuracy

In this Chapter we begin by describing the steps involved in building a database of vehicle models for recognition in our system implementation. The process of recognition model building consists of generating three dimensional noisy models of various vehicles and then generating images of the noisy surfaces with the help of simulated optical imagery. The addition of noise to the vehicle models help to simulate variations in surface properties of the various targets as well as to account for noise in the image capturing process associated with the use of optical cameras.

In later sections we examine the performance of the ATR algorithm developed in Chapter 5 when applied to the recognition database models developed in this Chapter. The database that we implement consists of recognition templates for four different vehicles for various levels of recognition database complexity. We show that the recognition performance of the ATR algorithm varies non-linearly for increasing levels of database complexity and characterize a accuracy-complexity curve for the algorithm.

The results in this Chapter are derived from tests involving around 4,000 test images divided across four different vehicle types. Each image is tested at six different levels of database complexity to yield around 24,000 test results which are then processed to obtain a measure of accuracy for each complexity level. Each test image incorporates a noisy background in addition to the picture of a vehicle located at the center of the image. The vehicles are photographed at random camera positions resulting in randomized poses

for the vehicles in the images. No test images are included from the training set of images that are utilized to build the recognition database templates.

7.1 Noise Induction in 3D vehicle models

We model image variations within a specific target class for a given pose by simulating minor surface deformations over the whole vehicle body. The procedure for obtaining such image variations after incorporating noise in the 3-D models of vehicles is described in this section. The process consists of extracting the set of unique points in space that make up the body of the vehicle and then adding a certain amount of Gaussian noise with a known mean and variance to each point on the vehicle surface to generate a unique noisy set of points. The original set of points in the 3-D model is then replaced with the new set of noisy points and an images are generated from the noisy model using simulation.

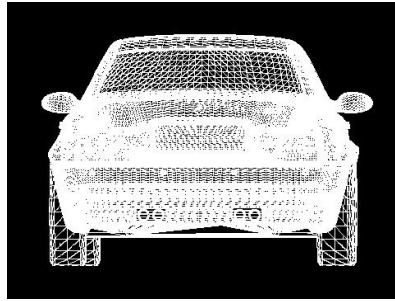


Fig. 7-1: 3-D Model (Wire Frame) of a Vehicle

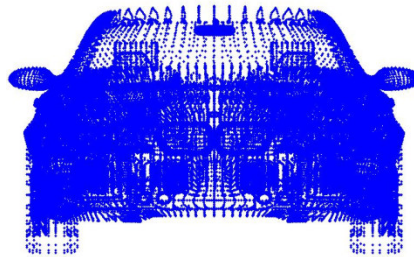


Fig. 7-2: Points on Car Body

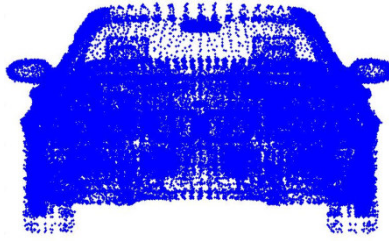
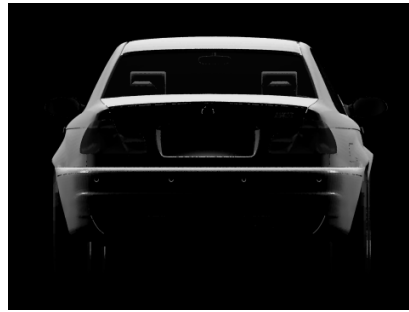
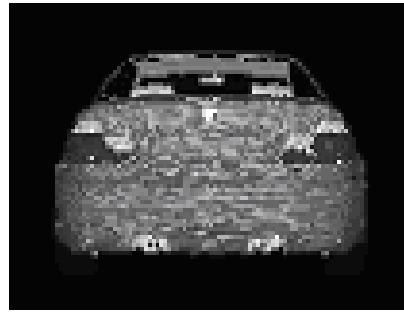


Fig.7-3: Displacement of Points after noise addition



(a)



(b)

Fig.7-4: Rendered Grayscale images. (a) original model; (b) noisy model

7.2 Discretized Statistical Model Building

In an ideal modeling case, to build a complete statistical description of the target vehicle, we need to have a very large amount of images for any target pose relative to the camera. However, the effort to build a model in such a manner would be impossible and practically infeasible. The solution to this problem lies in building an approximate model by discretizing the three-dimensional space around the target and then taking images around the discrete points.

For the purposes of image generation and modeling, the three dimensional space around the object can be thought of as a spheroid. For any radial distance between the target and the camera, we can then represent the relative pose as a point on the sphere

surface and by placing the camera at various points on the surface of the sphere we generate images of the target from different perspectives.

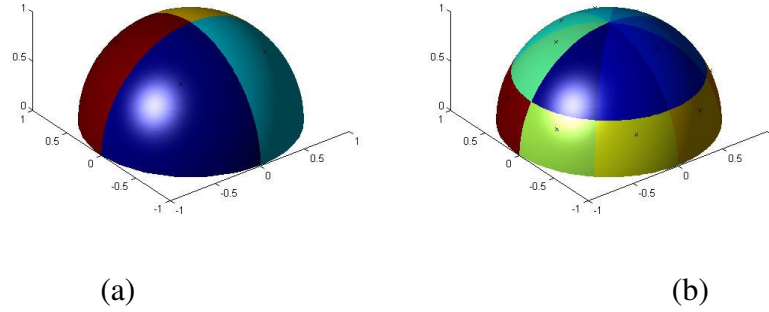


Fig. 7-5: Surface Discretization. (a) $k=1$, first order; (b) $k=2$, second order

We make an assumption that for recognition purposes we are interested only in the top view of the target vehicle. We thus discretize the surface of the top part of the sphere into n divisions of equal area, such that $n=2^{2k}$, where k is a non-zero, non-negative integer. The value k is also a subcomponent of the approximation control vector, α , which indexes the number of terms in a piecewise constant approximation of the mean and variance of pixel values in the imagery as a function of the location of the camera. We call the number k , which is constrained to be a non-negative integer, the order of discretization or the index of model complexity.

For a particular order of discretization, k , the regions $R_{s,t}$ on the sphere are called the regions of discretization and can be defined in spherical co-ordinates :

$$R_{s,t}(r, \theta, \Phi) = \left\{ (r, \theta, \Phi) : r = \rho, \theta \in \left[(s-1) \pi / 2^k, s \pi / 2^k \right], \right. \\ \left. \phi \in \left[\left(\cos^{-1} (t-1) / 2^{k-1} \right), \cos^{-1} (t / 2^{k-1}) \right] \right\}$$

(7-1)

The center of each region of discretization corresponds to a discrete point in space (r, Θ, Φ) , where r is the radius of the sphere, Θ is the angle that the center of the region $R_{s,t}$ makes with the positive x axis in the x - y plane, Φ is the angle that surface normal passing through the center of $R_{s,t}$ makes with z axis.

The whole surface of the sphere is a union of all the regions

$$\cup R_{s,t} \text{ for } s = 1, 2, \dots, 2^{k+1}; t = 1, 2, \dots, 2^{k-1}.$$

The next step in our modeling process consists of building a mean and variance perspective of the car by placing the camera within a discretization region $R_{s,t}$ and taking various training images. The statistical model building is achieved by taking a set of N_p images by placing the camera N_p times on the surface defined by the region $R_{s,t}$ such that whole region is uniformly covered. For each image, the focal plane of the camera forms a tangent to the sphere and points at the origin, which is centered at the geometric center of the sphere. The geometric center of the target is also aligned with the center of the sphere.

After generating the images, we calculate the mean and the variance values for all the pixels for the N_p images generated from a particular region $R_{s,t}$.

For N_p images of size $(X \times Y)$ pixels, the estimated mean intensity $I_{\text{avg}}(x, y)$ of a pixel $P(x, y)$ is given by

$$I_{\text{avg}} = (1 / N_p) \sum_1^{N_p} I_n(x, y) \quad (7-2)$$

The estimated variance for each pixel $P(x, y)$ for the N_p observations is given by

$$S(x, y) = (1 / N_p) \sum_1^{N_p} \{I_n(x, y) - I_{\text{avg}}(x, y)\}^2 \quad (7-3)$$

The complete statistical model thus consists of $X \times Y$ mean and variance values for each region $R_{s,t}$. The whole target model consists of all the models of the individual regions for a total of $X \times Y \times 2^{2k}$ mean and variance values for a particular order of discretization k and distance r .

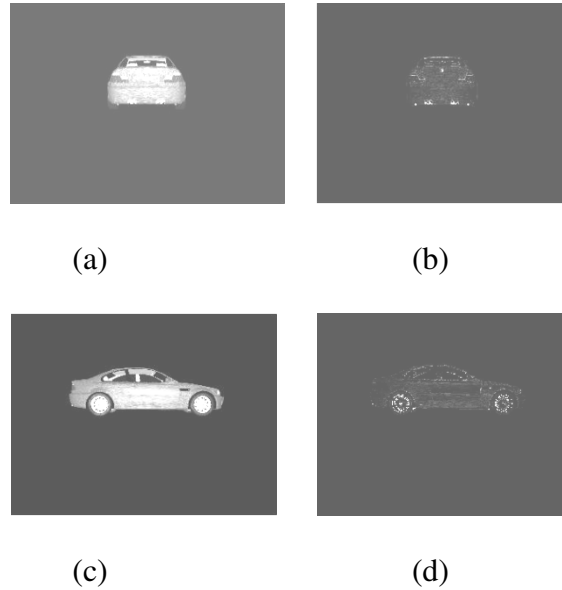


Fig. 7-6: Images from two regions of model complexity, $k=7$. (a) and (c) Normalized Mean; (b) and (d) Variance

7.3 Recognition System Implementation and Model Refinement

In the hardware implementation of an ATR system, various model enhancements or refinements may have to be incorporated to enable the recognition system to function on the target hardware platform. Specific examples of constraints may include systems with processors which do not have any native support for floating point arithmetic or constraints on system memory size and available storage. In our experimental implementation on a platform built around the Intel PXA255 processor we typically run

across many such limitations. In this section we address those issues and lay out the modifications employed to enable an ATR system to run successfully on such a platform.

In the final implementation of our experimental system we deal with mean pixel intensities of the order of 2^{16} and variances of the order of 2^{32} , which are typical for 16 bit grayscale imagery. The likelihood computation in eqn. 5-4 requires the extensive use of logarithmic evaluations of a constant value and a division of the square of the difference of input image pixel and template pixel intensities by the pixel variance for all pixels in every recognition template. Logarithm calculations introduce a substantial overhead without the support for floating point operations on the processor. Additionally, multiplications and divisions also require additional processor time when done in a floating point mode with the help of a floating point emulator implemented in software.

To address these limitations, we limit our algorithms strictly to integer mode calculations that provide us with significant benefits in computational time and energy usage. We also introduce the concept of a recognition template mask, which is incorporated into our algorithm to calculate likelihood values only for pixels that possess non-zero mean values in recognition templates. We also introduce constant variances for the recognition template pixel values by discarding pre-calculated variances in favor of variance values, which are proportional to the number of valid pixels in the template mask. The formation of a recognition template mask is elaborated in the following section.

7.4 Recognition Template Masks

In eqns. 7-3 and 7-4 we calculate the mean and variance for pixel intensities for a specific recognition template model. As an addition parameter in our specific implementation we

introduce the concept of a template mask which takes on binary values for each pixel depending upon their location in the images from which recognition templates have been formed and also depending on whether the mean pixel intensity for a pixel is a non-zero positive integer.

For every pixel $P(x,y)$, the template mask takes on a value of one if and only if the mean pixel intensity is estimated to be non-zero positive integer and the pixel possesses non-zero intensity values in all constituent images from which the mean intensity is estimated. In all other cases the mask value for the pixel takes on a value of zero and is excluded from likelihood computations. The mask value $M(x,y)$ can be expressed as,

$$M(x, y) = \begin{cases} 1, & \forall I_{avg}(x, y) > 0 \wedge I_n(x, y) > 0 \text{ for } n = 1, 2, \dots, N \\ 0, & \forall I_{avg}(x, y) = 0 \vee I_n(x, y) = 0 \text{ for } n = 1, 2, \dots, N \end{cases} \quad (7-4)$$

where N is the total number of images processed in estimating average pixel intensities for a recognition template. Variance values are then revised for each pixel in the recognition template, with the substituted variance estimate being given by,

$$S(x, y) = \sum^{X \times Y} M(x, y) \quad (7-5)$$

where $X \times Y$ is the total number of pixels in a recognition template. Further elaboration on the technical aspects of recognition template generation from a repository of vehicle images is included in Appendix A.

7.5 Accuracy Estimation

In this section we present the results of applying the ATR algorithm developed in Chapter 6 involving the recognition database models presented in the previous sections of this chapter. We test recognition accuracy for all levels of complexity from levels 2 to 7, excluding level 1. The modeling exercise elaborated in the previous section fails to yield adequate estimates for mean pixel intensities at level 1 with a number of recognition templates yielding no pixels of interest after applying the recognition template mask. The rest of the levels yield sufficient information for a recognition exercise. The rest of this discussion restricts our analysis of system accuracy and performance to levels 2 through 7 and demonstrate the relationship between recognition performance and recognition database complexity.

The measure of accuracy for recognition performance, $P_{\text{Err}}(\alpha, \sigma)$, is determined by conducting multiple runs of the recognition algorithm by varying a singular component of the system control vector α for various poses of a known vehicle for each available target class in the model database. We keep components of the system control vector σ , which include the properties of the recognition algorithm and the number of target classes as a constant. The recognition database model complexity, k , a component of the control vector α is varied to draw out the relationship between recognition accuracy and recognition database complexity.

The target database of vehicles employed included four different vehicle types, identified by the symbols: *audi*, *bmw*, *bmw3* and *mlance*. The properties of each vehicle and their corresponding models are presented in Appendix A. The test images for determining the accuracy of recognition system are generated under similar conditions as

used in the generation of training images for database generation but with additional background clutter and random orientations. We subject these test images to the recognition algorithms developed previously in Chapter 5 involving the recognition database templates detailed in the earlier sections of this Chapter. The input images are of 320 by 240 pixels in resolution with the target lying in the center of the picture. Camera positions are randomly varied within a range of zero to 2π radians in azimuth and zero to $\frac{\pi}{2}$ radians in elevation.

The target recognition performance is computed in a series of tests for each level of database complexity, k , with each level yielding different accuracy numbers. We represent the results of the recognition tests in which each image is classified as one of the four target classes in the database for which models have been trained, with the help of confusion matrices. In each confusion matrix, the row heading gives the true class for each of the test images represented in a row. Each cell contains the number of test images which were classified as one of the target classes, with each target class represented in the column heading. The column at the far right shows the overall accuracy for each set of test images for every target.

Table 7-1: Confusion matrix for recognition performance for complexity level 2 ($k=2$)

	audi	bmw	bmw3	mlance	
audi	56	276	692	0	5.47%
bmw	26	380	593	0	38.04%
bmw3	52	248	724	0	70.70%
mlance	81	353	590	0	0.00%

Table 7-2: Confusion matrix for recognition performance for complexity level 3 (k=3)

	audi	bmw	bm3	mlance	
audi	938	31	55	0	91.60%
bmw	310	582	107	0	58.26%
bm3	369	26	629	0	61.43%
mlance	749	134	141	0	0.00%

Table 7-3: Confusion matrix for recognition performance for complexity level 4 (k=4)

	audi	bmw	bm3	mlance	
audi	1011	10	3	0	98.73%
bmw	87	910	1	1	91.09%
bm3	227	9	788	0	76.95%
mlance	707	136	81	100	9.77%

Table 7-4: Confusion matrix for recognition performance for complexity level 5 (k=5)

	audi	bmw	bm3	mlance	
audi	1017	7	0	0	99.32%
bmw	25	974	0	0	97.50%
bm3	16	8	1000	0	97.66%
mlance	148	85	30	761	74.32%

Table 7-5: Confusion matrix for recognition performance for complexity level 6 (k=6)

	audi	bmw	bm3	mlance	
audi	1017	7	0	0	99.32%
bmw	6	993	0	0	99.40%
bm3	6	8	1010	0	98.63%
mlance	7	12	8	997	97.36%

Table 7-6: Confusion matrix for recognition performance for complexity level 7 (k=7)

	audi	bmw	bm3	mlance	
audi	1017	6	1	0	99.32%
bmw	0	999	0	0	100.00%
bm3	4	7	1013	0	98.93%
mlance	4	7	1	1012	98.83%

Tables 7-1 through 7-6 depict the recognition performance for each class for different levels of database complexity with other algorithmic parameters remaining the same. We see marked increases in recognition accuracy with increases in database complexity at lower complexity levels with accuracy gains gradually fading off as we approach the highest level of complexity. We also observe that some classes fare very poorly at lower levels but gradually catch up in terms of recognition performance as higher levels of database complexity are approached.

In Fig 7-7, we depict the recognition accuracy of each target class along with the system average accuracy for each level of recognition database complexity. In Fig 7-8, we chart the system average percentage accuracy and the differential increase in accuracy with increasing levels of database complexity. We are primarily interested in the system average accuracy percentage figures as it forms a component of our performance vector . The figure also illustrates that the average accuracy of the system increases linearly with a steep gradient for complexity levels two to five but registers very marginal increases for levels five to seven. These results show that performance is highly dependent on the complexity and number of templates involved in the recognition process. Since database complexity is directly related to the data processing costs for the recognition algorithm and also related to the amount of data storage required for the storage of recognition database models, the recognition performance-complexity curve can be utilized to characterize other measures of resource consumption in an ATR system. In later chapters, we explore the effect of database complexity on various such measures of resource consumption.

Recognition accuracy estimation can usually be performed independently of any particular hardware implementation. In our test setup we undertake this exercise by simulating conditions similar to our implementation platform on a cluster of Linux servers. Approximations are made in the estimation of mean intensities of pixels in recognition templates by rounding the estimated floating point values to their nearest

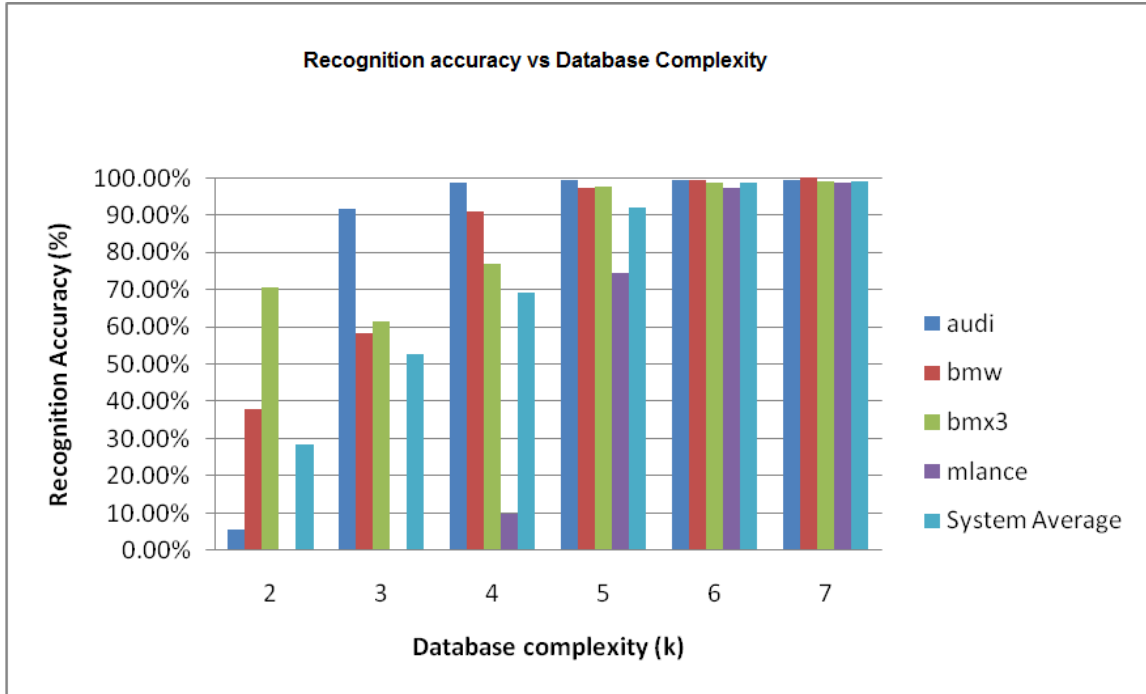


Fig 7-7: Target Class and System average recognition accuracy for various levels of database complexity

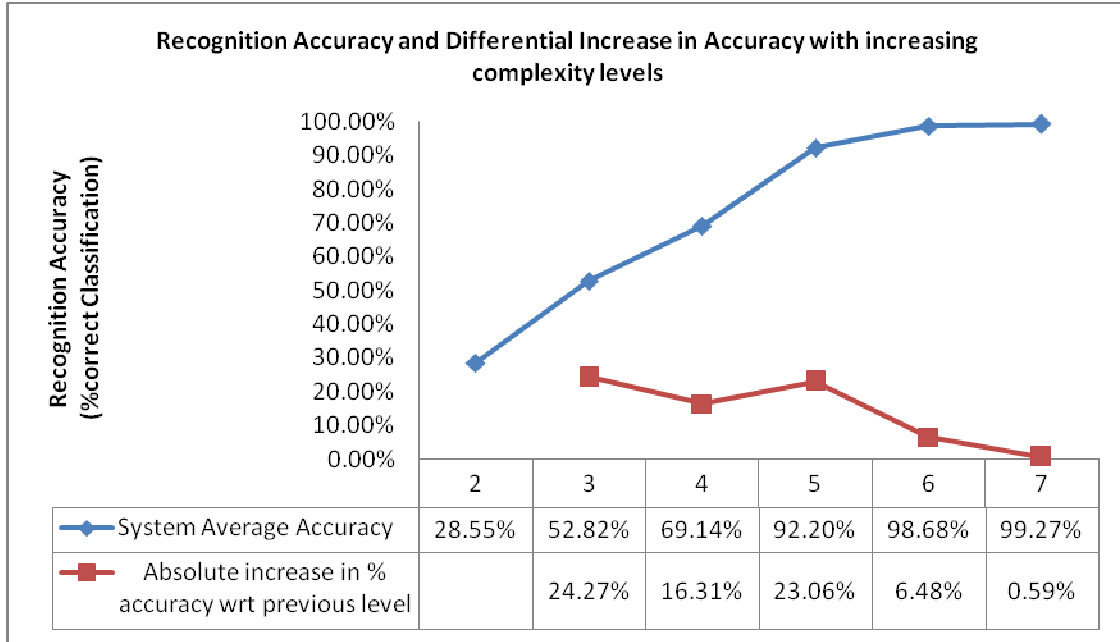


Fig 7-8: System average recognition accuracy for various levels of database complexity and differential increase in accuracy for each complexity level

integer values and storing such values in the database files as 16-bit integers. The number of relevant pixels in each template is pre-calculated and stored in each database file along with the template mask as discussed in earlier sections. Likelihood computations are also done strictly in integer arithmetic with logarithmic values used in the computations being stored in the database models along with the pixel intensity and mask information for recognition templates. In following such a modeling process we save a considerable amount of processing time and resources while executing our recognition algorithm. A discussion on the practical aspects of the actual hardware and software implementation of the experimental ATR system can be found in Appendix B.

Chapter 8

Resource Usage and Model Validation

In this Chapter we advance the concept of the generic models of resource consumption developed in Chapter 6 for actual evaluation on a real ATR testbed system. We simplify the models to eliminate factors that are beyond the control of the system designer and evolve the models to present a form suitable for a specific system implementation. The methodology we outline can serve as a basis to undertake performance characterization on any generic ATR system on a specific hardware platform. At the very outset, we present simplified closed form models for system throughput and energy consumption for the recognition algorithm implementation involving recognition databases presented in Chapter 7. Later on we present actual resource consumption data collected from our testbed implementation on a network of mobile computing platforms. We examine these results for seven different system states consisting of different combinations of processor and system bus frequencies on a mobile CPU. We then subject our resource consumption models to the data that we obtain on the testbed platform to evaluate the validity of our models. We finally demonstrate the robustness of our models by presenting the results from fitting our resource consumption models to the experimental data using a simple least squares regression method. The residual errors that we obtain from this exercise lie in the range of zero to seven percent for various operating points for the ATR system.

Lastly, we correlate the recognition accuracy of the system from Chapter 7 with resource usage results obtained in this Chapter and present the results of such a characterization. We illustrate the tradeoffs between accuracy and resource consumption

to show that the resource demands from an ATR system increase manifolds for incremental increases in system accuracy at higher levels of recognition performance. Additionally, we demonstrate that the most efficient operating point for the system lies in the highest processor and bus frequencies for almost all levels of recognition accuracy.

8.1 ATR Testbed Configuration

To demonstrate the concepts of resource modeling for an ATR system, we implement an ATR testbed system using a distributed configuration of mobile processing nodes. The basic topology of the ATR system consists of a central manager node and several server nodes. The manager node acts as the overall coordinator of the ATR system by distributing recognition tasks to the available server nodes and aggregating results from various server nodes to arrive at a final recognition classification for every test image. A detailed discussion on the actual implementation of the hardware and software design of the testbed is given in Appendix B.

The target system that we choose for building the ATR system consists of several nodes employing an embedded computing platform based on the Intel® XScale™7 (PXA255, 100 - 400MHZ) voltage and frequency scalable processor which allows us to vary the processor and system bus frequency to obtain seven distinct combinations. The single processor device has a node memory of 64 MB(SDRAM)and a flash memory of 64MB out of which approximately 25MB is available to user applications. We store the recognition database in the form of binary files on a CF card attached to every node. The manager and the server nodes communicate for the transmission of test images and recognition results. Upon receipt of a test image, the manager node distributes the image

to all available server nodes and keeps waiting for a recognition result from each node. Every server node processes the test image and sends back the best likelihood values to the manager node after executing the recognition algorithm against a database of recognition templates available locally. The manager node then selects the best value from all the results received to arrive at a global classification result.

In this framework, the total energy consumed and the total times taken are proportional to the absolute size of the recognition problem i.e. the size and the number of target models, assuming the tasks are distributed equally across all server nodes. In Chapter 5, we discussed that single target database comprises of 4^k sets of mean and mask values, corresponding to every level of database complexity, k , which are processed during the entire recognition process. This information can be utilized to describe a simpler hardware model in terms of k and other hardware parameters such as processor, memory and system bus frequencies. We simplify the resource consumption models derived in Chapter 6 by grouping together all non variable system parameters as constants and expressing the hardware measures of interest in terms of model complexity and other variable hardware control values.

8.2 Resource Usage Model Simplification

Determination of the values for each component of the hardware control vector forms the first step towards fitting the model for this implementation. While some values for a system may be available readily, we have to experimentally derive the rest of the values for our specific system.

The experimental platform being a distributed system of multiple processing nodes, we simplify the model further by looking at system resource consumption per node by averaging resource usage values obtained across all nodes. With every node being a single processor system, N_p is assigned a value of one. The processor word size, the number of models stored at each node, the bandwidth available and the size of the local node memory are also available but we do not intend to utilize those values at this stage as they fall outside the set of controllable parameters for the system.

While Sections 6.1 and 6.2 provide the generic framework for developing a hardware model for an ATR system, we further simplify the performance model by introducing constant variables as components of the hardware control vector(η) and expressing the measures of interest as functions of

1. k , the order of complexity which is also a component of α .
2. F_p , the processor clock frequency, a component of η .
3. F_b , system bus frequency, a component of η .

The resources consumed by the hardware platform are roughly proportional to the number of target models processed which is equal to 4^k for a particular level of target model complexity, k . Eqs 6-3 and 6-4 can then be simplified and expressed in the following forms

$$N_{BITS}(F_p, k) \approx A \cdot 4^k + B \quad (8-1)$$

$$N_{CYCTOT}(F_p, k) \approx C \cdot 4^k + D \quad (8-2)$$

where 4^k corresponds to the number of recognition templates to be processed for a recognition complexity level, k . Similarly, the computational times can be expressed as simplified forms of eqs. 6-6, 6-8. and 6-9

$$T_{COMP}(F_p, k) \approx E \cdot (F4^k + G)/F_p \quad (8-3)$$

$$T_{DB}(F_p, k) \approx H4^k \quad (8-4)$$

Additionally, we also consume some time when we encounter cache misses by fetching missed data from external memory. That amount of time can be assumed to be proportional to $1/F_b$ and 4^k , assuming an average cache miss rate for 4^k recognition models or templates processed for a given level of recognition complexity, k . The total time spent in computation can be then expressed as the sum of processor time given in eqn. 6-9 and the average delay in fetching data from memory as

$$T_{TOT}(F_p, F_b, k) \approx \left(\frac{I}{F_p} + \frac{J}{F_b} + H \right) \cdot 4^k \quad (8-5)$$

The total energy consumption for the ATR application follows from eq. 6-10 which is given below for reference

$$E_{TOT} = \alpha_s C_p (K_V F_p + V_0)^2 N_{CYCTOT}(k) + P_M M_{SIZE} T_{TOT}(k) + E_{MR}(k) + E_{COM} \quad (6-10)$$

The first term in the right hand side of this expression is the energy consumed by the processor and has two variable terms in F_p , the processor frequency and $N_{CYCTOT}(k)$, the total number of cycles spent in computation on the processor, which can be assumed to be proportional to 4^k . The idle memory energy consumption is proportional to the total time of the execution and the energy expended in memory reads can again be assumed to be proportional to 4^k . Additionally, the hardware platform will consume some leakage power and some amount of power to run peripheral devices and other operating system tasks for the duration of program execution given by Eqn. 8-5. The final expression for energy consumption that we arrive at includes all these components and models the energy for the system as a whole, for the duration of execution of the ATR algorithm. We express this energy in a simplified form as

$$E_{TOT}(F_p, F_b, k) \approx (LF_p^2 + MF_p + \frac{N}{F_p} + \frac{Q}{F_b} + R) \cdot 4^k$$

(8-6)

The coefficients I, J, H, L, M, N, Q and R form the components of a modified hardware control vector η' and can be established by experimentation.

The values for the coefficients in the simplified hardware models described above are determined by making multiple runs of the recognition algorithm on our chosen hardware platform. Our primary measures of interest are the total time of execution and total energy consumption. The power consumption is determined by sampling the steady state current with the ATR application kept running for an extended period of time. The product of the voltage and the current gives the steady state power consumption and

accounts for the power consumed by memory, peripherals and other components during the recognition process.

Eqs. 8-1 through 8-6 model the hardware performance measures as functions of k , F_b and F_p , which are the primary controls we employ as part of our investigation. The coefficients in Eqns. 8-5 and 8-6 can be determined by fitting the models to the experimental data collected. The resulting set of equations provides us with the performance model for our system implementation and also provides us with the opportunity to evaluate the validity of the model and the goodness of the fit.

8.3 Resource Consumption Results

In this section we present the results of the experiments conducted on our testbed ATR platform to measure the resource usage of the ATR system application. We primarily record two measures of resource consumption, the time taken by the system to arrive at a recognition result and the average power consumption of the system nodes while executing the recognition task. We estimate the energy consumption for the recognition time by multiplying the average time of program execution by the average power consumption during the period of execution. The test setup and measurement methodology is explained in Appendix B for reference.

The application server nodes are built on the Stargate mobile computing platform manufactured by Crossbow Inc, which incorporate an Intel® PXA255 processor implementing the Intel® XScale architecture. The PXA255 processor allows us to vary processor speed by accessing the clock configuration registers [22] for obtaining several different combinations of processor and bus frequencies. Table 8.1 lists the various

different combinations of system frequencies that we use to characterize the performance curves for our application. We classify the various different combinations of CPU and bus frequencies into seven different system states that we will use for future reference as part of this discussion. System changes are undertaken by writing different values to a memory mapped register called CCCR (core clock configuration register) accessed by implementing a loadable kernel module that is plugged into the operating system kernel at runtime. The kernel module provides an interface for accepting input values for executing a frequency change sequence through a file in the */proc* file system of the Linux operating system.

Table 8.1: List of available system states along with their associated CPU, Bus and Memory frequencies

System State	CPU frequency (MHz) F_p	Bus frequency (MHz) F_b	Memory frequency (MHz) F_m
1	99.5	50	99.5
2	199.1	50	99.5
3	199.1	99.5	99.5
4	298.6	50	99.5
5	298.6	99.5	99.5
6	398.1	99.5	99.5
7	398.1	196	99.5

The seven different states provide different combinations of CPU and bus speeds with CPU speeds varying from approximately 100MHz to 400MHz in increments of 100MHz and three different bus speeds of 50MHz, 100MHz and 200MHz. Not all combinations of CPU and bus frequency are however valid, with the available combinations being shown below in Fig 8.1.

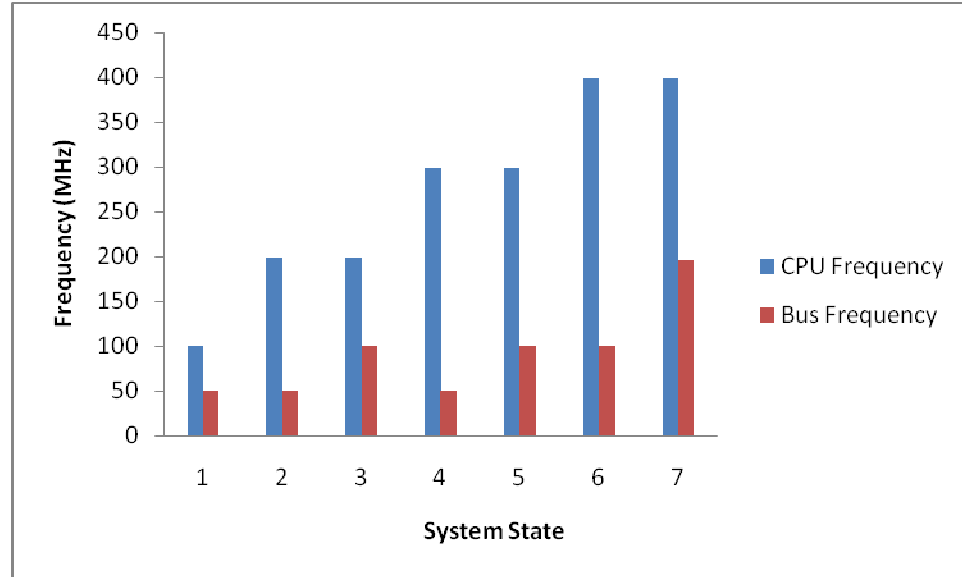


Fig 8.1: Plot showing variations in CPU and Bus frequencies for different system states

We conduct our resource consumption evaluations for all of these seven different system states for each level of database complexity for the recognition algorithm, varying k , the complexity factor from two to seven. The experiments provide us with two sets of resource usage values, one for execution time and the other for energy consumption for 42 different operating points.

In Tables 8.1 - 8.3 we list the average values for execution time, power consumption and energy consumption values for various system states and database complexity levels. Each row in the table represents a different system state identified by the state numbers in table 8.1, while each column identifies the database complexity for the recognition task. Fig. 8.2 – 8.3 represent the same data in the form of Line plots and Bar Graphs.

Table 8.2: Average Execution time values in Seconds for different recognition complexities and system states

System State	Recognition Database Complexity (k)					
	k=2	k=3	k=4	k=5	k=6	k=7
1	2.17	5.11	18.71	79.05	421.96	1757.26
2	1.77	3.53	11.72	48.04	282.22	1184.60
3	1.62	3.12	10.06	40.89	235.09	984.23
4	1.63	3.01	9.68	40.13	250.73	1053.48
5	1.49	2.69	7.68	31.24	193.71	805.7
6	1.45	2.53	6.81	26.74	170.02	707.78
7	1.42	2.34	6.52	24.23	157.31	636.43

Table 8.3: Average Power consumption values in Watts for different recognition complexities and system states

System State	Recognition Database Complexity (k)					
	k=2	k=3	k=4	k=5	k=6	k=7
1	2.11	2.13	2.14	2.11	2.12	2.13
2	2.20	2.27	2.26	2.25	2.26	2.25
3	2.42	2.44	2.39	2.41	2.35	2.35
4	2.31	2.34	2.30	2.31	2.30	2.31
5	2.53	2.53	2.53	2.53	2.48	2.46
6	2.60	2.67	2.64	2.66	2.62	2.61
7	2.80	2.83	2.82	2.79	2.65	2.65

Table 8.4: Average Energy consumption values in Joules for different recognition complexities and system states

System State	Recognition Database Complexity (k)					
	k=2	k=3	k=4	k=5	k=6	k=7
1	4.59	10.88	40.12	167.08	896.17	3750.13
2	3.89	8.00	26.45	107.94	637.03	2661.73
3	3.91	7.62	24.00	98.38	552.35	2317.54
4	3.76	7.04	22.25	92.85	576.24	2437.36
5	3.78	6.80	19.42	79.01	479.97	1983.96
6	3.77	6.75	17.96	71.19	444.82	1848.13
7	3.97	6.61	18.40	67.49	417.22	1684.68

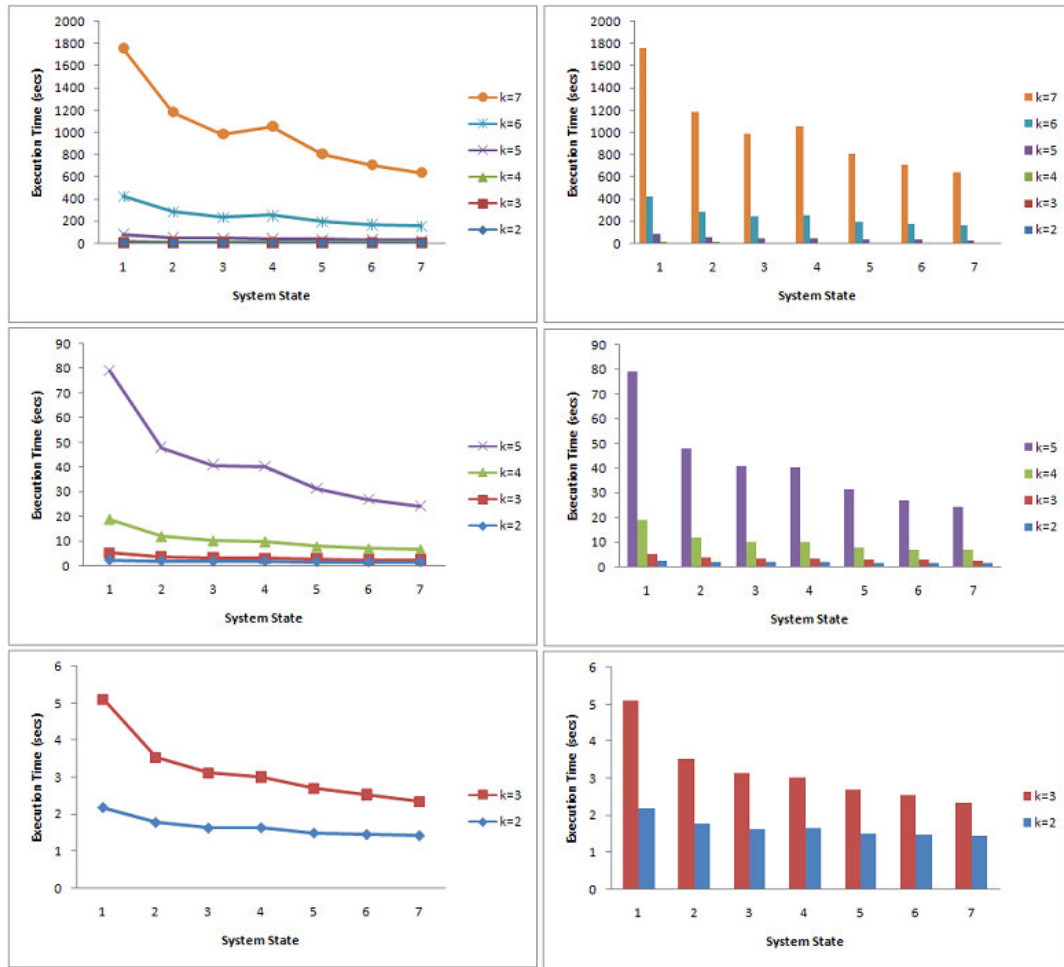


Fig 8.2: Line and Bar plots for Execution times in seconds for different complexity (k) levels and system states

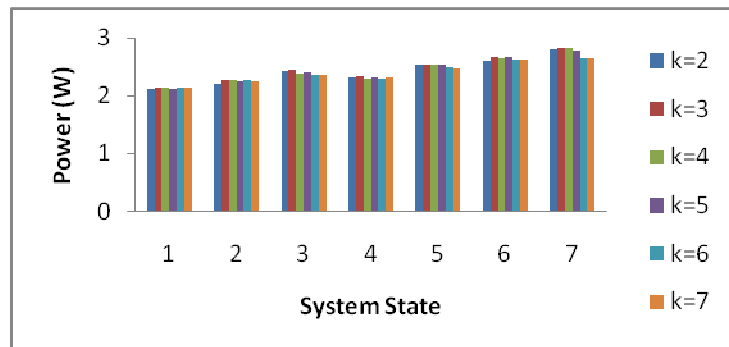


Fig 8.3: Power consumption (Watts) plot for various levels of database complexity (k) and system states

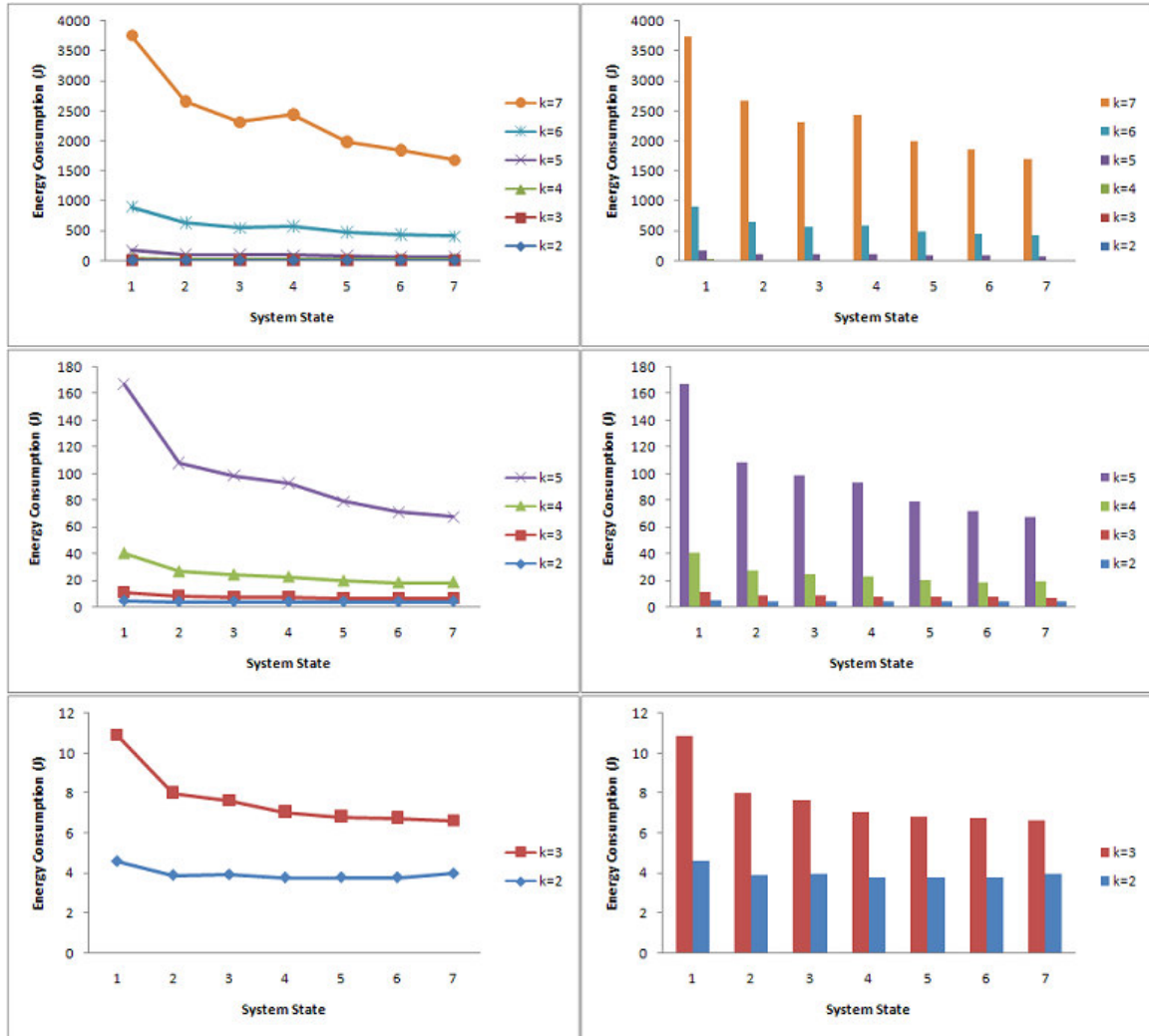


Fig 8.4: Line and Bar plots for Energy Consumption (Joules) for different complexity (k) levels and system states

It can be clearly seen from the execution time and energy consumption plots that both these quantities decrease with increasing levels of processor speeds except for system state four, where we see an increase in resource consumption when compared to system states three and five, at higher levels of recognition complexity. This highlights the effect of the bus frequency on the system performance. In general, we observe that for the same CPU frequency, lower bus frequency settings result in higher time and energy usage.

This effect is most pronounced at system state 4, where the CPU to bus frequency ratio is the highest in the system. The deviant behavior at system state 4 can be attributed to the substantial difference in CPU and bus speeds for that setting, which can introduce additional time lags due to a substantially higher number of cache misses at higher levels of database complexity where the amounts of data lookups from the database are substantially higher.

We also observe that energy consumption closely follows the execution time plots for all levels of database complexity. This can be explained by the nature of the power consumption profile, which is fairly flat for each system state, with slight increases as system frequencies are increased. The energy profile thus follows the pattern of the time curves as energy is a product of power and execution time.

Examining the time and energy curves for a fixed set of system frequencies identified by a particular system state, we see that resource consumption generally increases with increase in database complexity levels. Interestingly, we also observe that for a given set of frequencies, resource usage does not always scale according to the number of templates processed at each level, which increase by a factor of four for each complexity level. At lower values of complexity level k , for $k=2$ and $k=3$ we see that increases in time and energy are much less pronounced. This indicates that at these values of database complexity, the time spent in processing the models are comparatively less than other overheads associated with the system. Between complexity levels 4 and 5 we however see time and energy values increase by approximately a factor of four for every fixed system state. This relationship is again lost as we progress from complexity level five to level six, where we see resource usage increase by more than a factor of four. Comparing

levels six and seven restores the resource usage ratios approximately to a value of four once again. This behavior indicates that the nature of the recognition tasks change as we increase the level of complexity of the recognition problem, progressing from a CPU bound task at lower complexities to a memory bound task at higher values of k . This can be explained by the number of database recognition templates increasing by a factor of four at each level accompanied by an increase in the number of relevant pixels to be processed for each recognition template. At the highest levels of database complexity, the application has to deal with a substantially higher amount of memory lookups from the system RAM along with an increase in data cache misses, resulting in the CPU sitting idle for increased amounts of time. This adds to overall execution time and as a consequence, driving energy usage higher.

In conclusion, it is seen that the lowering of frequency settings do not provide any benefits during the execution of the ATR algorithm and we are almost always better off by operating the system at its highest performance state for all levels of recognition database complexity. The minor savings in power consumption that we achieve by operating the system at lower frequencies is offset by substantial increases in execution time that accompany the lower system settings. The power consumption values also point to a substantial amount of constant power in the hardware system which offsets any gains in power savings at the application level for lower system frequencies.

8.4 Resource Consumption Model Evaluations

In this section we evaluate the validity of our resource usage models developed in Section 8.1 and present the results of the model fitting exercise. The primary measures of

resource consumption that we are interested are the average execution times for a recognition task and the energy expended in performing a recognition exercise for a single test image. The expressions for these two measures have been presented in section 8.1 and are repeated below for reference.

$$T_{TOT}(F_p, F_b, k) \approx \left(\frac{I}{F_p} + \frac{J}{F_b} + H \right) \cdot 4^k$$

$$E_{TOT}(F_p, F_b, k) \approx \left(L F_p^2 + M F_p + \frac{N}{F_p} + \frac{Q}{F_b} + R \right) \cdot 4^k$$

In our initial model fitting exercise we determine the values of the coefficients in these two expressions and then proceed to estimate the energy consumption and execution time with the resulting coefficient values for various combinations of CPU and bus frequencies for different levels of recognition database complexity, k . We take a constrained least squares error minimization approach to determine the values of these coefficients under the assumption of all coefficients being non-negative. The regression analysis that we undertake is done against the data collected in our ATR testbed and presented in the earlier section. The regression exercise is done for every value of k , resulting in independent sets of time and energy models for different levels of database complexity. We take this approach of local regression against the idea of a global model because of inferences we drew from observing the resource consumption and ATR algorithm characteristics at different levels of recognition database complexity. We present the results of the regression exercise in the form of the coefficient values for the time and energy models in Tables 8.5 and 8.6.

Table 8.5: Coefficient values of execution time model variables for different levels of database complexity

Recognition Complexity(k)	Coefficient Values		
	I	J	H
2	4.948	0.735447	0.070627
3	4.7628	0.491057	0.021513
4	5.2892	0.595643	0.007574
5	5.7303	0.724127	0.004603
6	6.364	1.187197	0.014595
7	6.5597	1.321534	0.014365

Table 8.6: Coefficient values of energy consumption model variables for different levels of database complexity

Recognition Complexity (k)	Coefficient Values				
	L	M	N	Q	R
2	1.95E-07	0	9.842301	0	0.186024
3	8.92E-08	0	9.862092	0.417054	0.061935
4	6.11E-08	0	11.26268	0.836285	0.026413
5	0	2.37E-05	11.72771	1.120781	0.020465
6	0	4.76E-05	13.29469	2.177833	0.036714
7	0	3.94E-05	13.47282	2.465803	0.040036

A careful look at the values of the coefficients clearly brings out the dominance of the effect of CPU frequency, followed by the bus frequency in both the models. The values of the constants H and R also assume significant values when multiplied with the factor 4^k indicating substantial contributions to the time and energy usage. The values for I, J, N and Q increase with increasing levels of database complexity which lead us to infer that the influence of system frequencies on resource consumption increase with higher database complexities.

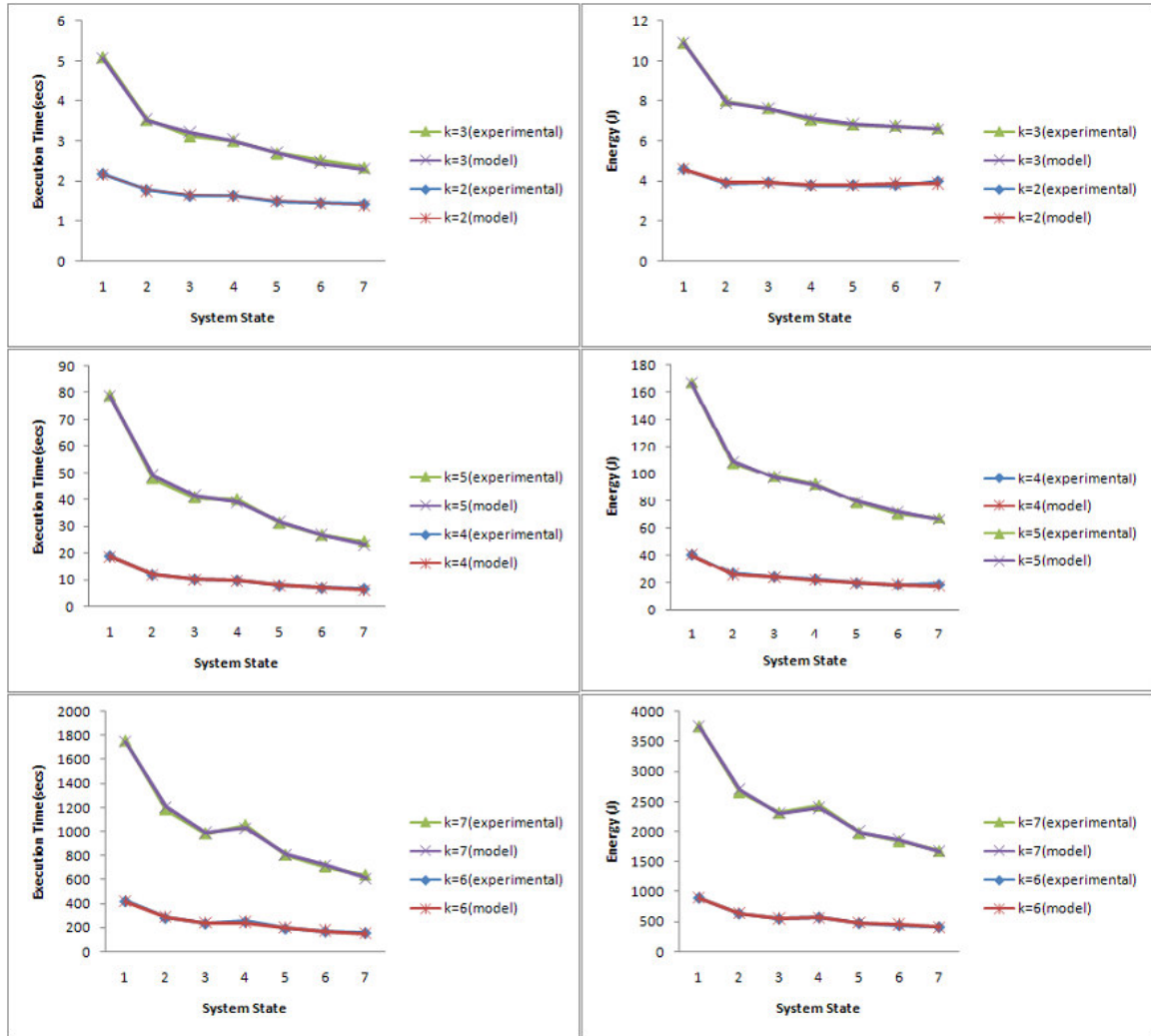


Fig 8.5: Comparison of experimental and modeled values for execution time and energy consumption

In Fig 8.5 we depict a comparison of the experimental values with the values predicted by the time and energy models after substituting the coefficients with values derived from the model fitting exercise. In tables 8.7 and 8.8 we list the residual errors for the regression analysis in terms of percentage differences between model predicted and experimental values.

Table 8.7: Residual errors for execution time model

System State	Execution Time Residual Errors					
	k=2	k=3	k=4	k=5	k=6	k=7
1	-0.41%	-0.81%	-0.61%	-0.68%	-0.70%	-0.50%
2	-0.40%	0.18%	0.59%	1.99%	1.99%	1.95%
3	1.58%	3.22%	2.07%	1.79%	1.87%	0.86%
4	0.03%	0.54%	-1.64%	-2.39%	-2.62%	-2.45%
5	1.55%	0.87%	4.07%	1.81%	1.14%	0.89%
6	-0.20%	-2.91%	0.91%	0.61%	2.36%	2.10%
7	-2.24%	-1.61%	-6.57%	-4.28%	-4.82%	-3.35%

Table 8.8: Residual errors for energy consumption model

System State	Energy Consumption Residual Errors					
	k=2	k=3	k=4	k=5	k=6	k=7
1	0.08%	0.17%	0.14%	-0.04%	-0.08%	-0.09%
2	0.12%	-1.39%	-1.19%	1.02%	0.64%	1.46%
3	-0.60%	0.13%	0.07%	-0.76%	0.00%	-0.80%
4	0.50%	1.12%	-0.70%	-1.62%	-1.20%	-1.73%
5	0.15%	0.76%	2.70%	1.17%	0.14%	0.50%
6	2.45%	-0.39%	3.63%	1.54%	2.12%	1.35%
7	-2.69%	-0.36%	-4.68%	-1.30%	-1.68%	-0.67%

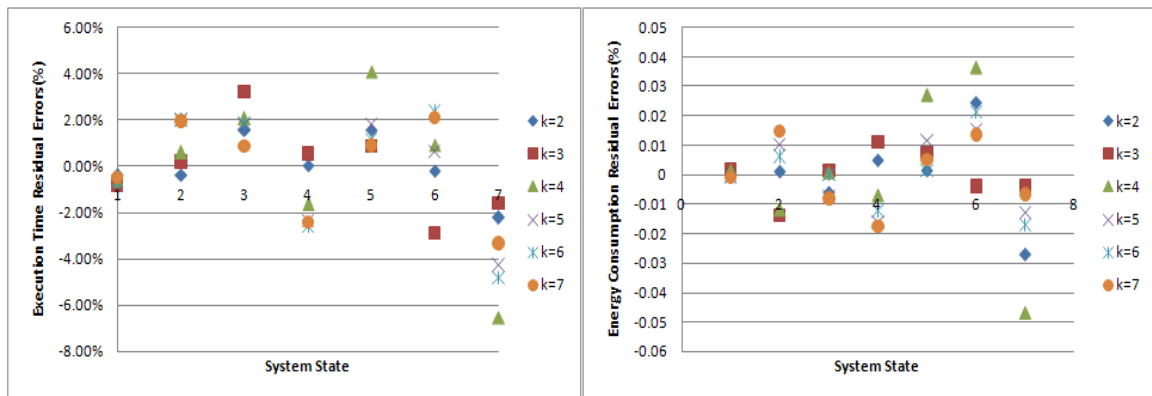


Fig 8.6: Plot of Residual errors for execution time and energy consumption

In Fig 8.6 we depict a plot of the residual errors for all system states and levels of complexity. It can be observed that the models fit the experimental data with a relatively high degree of accuracy with the highest absolute error for the execution time model being 6.5% and an average absolute error of 0.1%. The model for energy consumption yields an average absolute error of zero and maximum absolute error of 4.68%.

8.5 Resource Consumption and Recognition Accuracy Tradeoffs

In this section we briefly touch upon the relationship between recognition accuracy and resource consumption on the ATR system by relating the results from our accuracy estimation exercise in Chapter 7 along with the results presented in this Chapter. We have previously stated that recognition accuracy is solely dependent on the database complexity levels and algorithms employed and do not depend on the hardware characteristics of the ATR system. However, it has been explicitly demonstrated with the results in this Chapter that increase in levels of database complexity adversely affects resource consumption on the hardware subsystem. In Figs 8.7 and 8.8 we illustrate this relationship between the hardware and recognition subsystem performance which provides a unique view into the multidimensional aspects of ATR system performance enabling a system designer to choose a desired set of operating points based on the constraints that he has in hand.

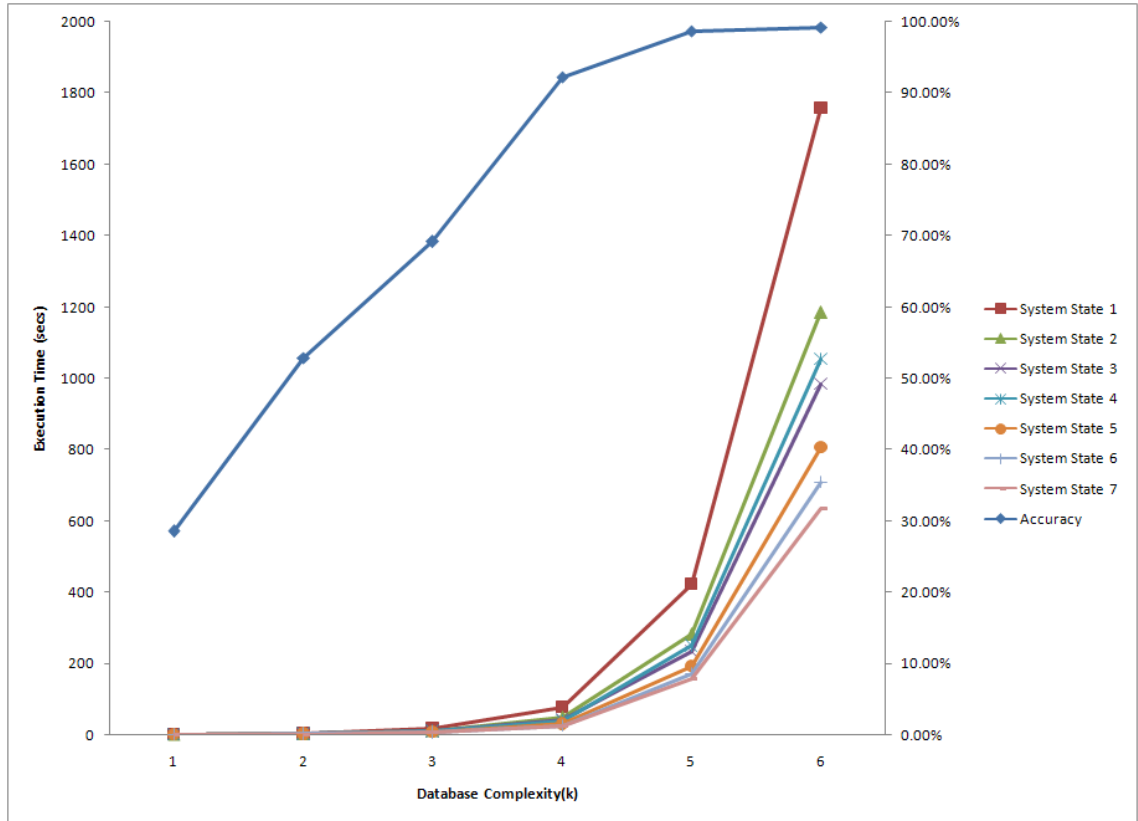


Fig 8.7: Plot of Recognition accuracy and average execution time for various complexity levels and hardware system states

We can infer from these plots that with increasing levels of database complexity, the gains in recognition accuracy start decreasing while the resource demands from the hardware subsystem increase by very large amounts. This effect is best illustrated at the highest levels of database complexity where increasing the database complexity from level 5 to level 6 results in an increase of around 6.5% in recognition accuracy but time and energy consumption increases by about 400%. Between levels 6 and 7, this tradeoff between accuracy and resource consumption is further magnified with an increase of less than half a percent in accuracy between the two levels for an increase of around 4 times in energy and average execution time demands for a classification result.

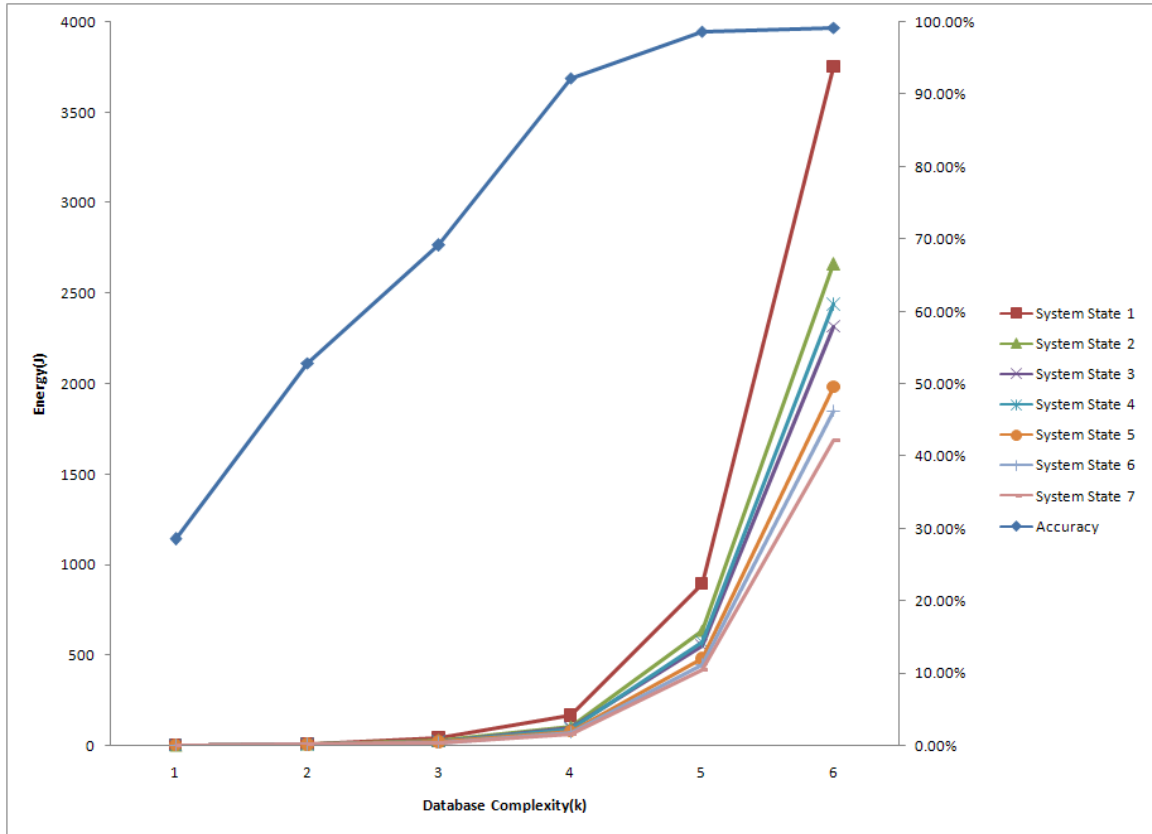


Fig 8.8: Plot of Recognition accuracy and average energy consumption for various complexity levels and hardware system states

These performance curves provide the necessary view to understand the behavior of the ATR system in terms of the available controls, enabling a system designer to critically examine system constraints and choose optimal operating points for the ATR application. The results from this exercise can also form the basis for intelligent ATR system design where the application incorporates the models for performance and decides on optimal operating points at runtime based upon constraints on time, energy, accuracy or any combinations of these performance criteria.

Chapter 9

Conclusion

This thesis presents a unified performance characterization and modeling methodology for generic ATR systems. We introduce a holistic systems evaluation method to analyze all aspects of ATR system performance encompassing recognition methodology as well as hardware system configurations. A method for characterizing generic ATR systems with the help of system control vectors is introduced to identify independent groups of variables that affect different aspects of ATR system performance. The idea of ATR system performance characterization is approached with the design of a performance vector that can potentially depict ATR system performance in multiple dimensions using an integrated approach.

We demonstrate the application of these ideas with the help of an actual ATR system implementation on a distributed platform of mobile processing devices by developing models for accuracy and hardware resource consumption and examining the validity of those models against experimental data. The ATR system that we develop incorporates and implements all the sub-components of the generic ATR system model that we propose to examine, namely, the recognition, database and hardware subsystems. With the help of this experimental setup, this work also demonstrates the actual tradeoffs involved in an ATR system design by characterizing accuracy and resource consumption together.

While existing research in the performance evaluation of recognition systems have primarily focused on developing accuracy measures for various recognition methodologies and algorithms, we present a model that ties together hardware performance measures to measures of accuracy. This type of modeling enables a system designer to quantitatively understand the implications of design choices in terms that are meaningful at the implementation level.

A general use of this modeling methodology can enable a system designer to optimally select an operating point for a given set of requirements. Optimal energy, throughput and accuracy configurations for a given set of constraints can then be searched over the performance surface generated by the model using suitable optimization methods. Future work in this area may involve the framing and solution of such an optimization problem utilizing the modeling framework presented in this work, which would then present the designer with optimal choices in system configuration and design. This work also opens up avenues for intelligent and self-aware ATR system design. Will it be possible to design a system that incorporates instrumentation and logic to determine performance characteristics and choose optimal operating points on its own? It might very well be possible for a system to evaluate the time and energy costs associated with processing but will it be possible to determine accuracy measures by the system itself? The resource consumption models presented as part of this work are simple enough to be incorporated into the framework of an ATR system to predict energy and throughput values at runtime or when need arises. In the occurrence of dynamic reconfigurations within the framework of the ATR system, such as when system resources suddenly become unavailable, the resource consumption models can easily be

modified to obtain new performance characteristics. The accuracy models however require extensive computation and simulation and provide opportunities for further analysis and simplification.

BIBLIOGRAPHY

1. John W. Boyse, David R. Warn. A Straightforward Model for Computer Performance Prediction, ACM Computing Surveys (CSUR), Volume 7 Issue 2, June 1975
2. <http://www-unix.mcs.anl.gov/mpi/>
3. Prete, C.A.; Prina, G.; Ricciardi, L. A trace-driven simulator for performance evaluation of cache-based multiprocessor systems, Parallel and Distributed Systems, IEEE Transactions on , Volume: 6 , Issue: 9 , Sept. 1995 Pages:915 – 929
4. P.Heidelberger, S.S. Lavenberg,; Computer performance evaluation methodology ,Computers, IEEE transactions on, volume 33, Dec 1984
5. Larson, E.; Chatterjee, S.; Austin, T.;MASE: a novel infrastructure for detailed microarchitectural modeling, Performance Analysis of Systems and Software, 2001. ISPASS. 2001 IEEE International Symposium on , Nov. 4-6, 2001 Pages:1 – 9
6. Parallel program performance prediction using deterministic task graph analysis. Source ACM Transactions on Computer Systems (TOCS) archive,Volume 22 , Issue 1 (February 2004) Pages: 94 – 136
7. Peterson, G.D.; Chamberlain, R.D.; Beyond execution time: expanding the use of performance models. Parallel & Distributed Technology: Systems & Applications, IEEE, Volume: 2, Issue: 2, Summer 1994 Pages:37 – 49
8. Nelson, R.; Towsley, D.; Tantawi, A.N.; Performance analysis of parallel processing systems, Software Engineering, IEEE Transactions on , Volume: 14 , Issue: 4 , Apr 1988
9. Madala, S.; Sinclair, J.B.; Performance of synchronous parallel algorithms with regular structures,Parallel and Distributed Systems, IEEE Transactions on , Volume: 2 , Issue: 1 , Jan. 1991
10. Lui, J.C.S.; Muntz, R.R.; Towsley, D.; Computing performance bounds of fork-join parallel programs under a multiprocessing environment, Parallel and Distributed Systems, IEEE Transactions on , Volume: 9 , Issue: 3 , March 1998
11. Balsamo, S.; Donatiello, L.; Van Dijk, N.M.; Bound performance models of heterogeneous parallel processing systems, Parallel and Distributed Systems, IEEE Transactions on, Volume: 9, Issue: 10, Oct. 1998
12. Linguo Gong; Xian-He Sun; Watson, E.F.; Performance modeling and prediction of nondedicated network computing, Computers, IEEE Transactions on, Volume: 51 , Issue: 9 , Sept. 2002 Pages: 1041 – 1055

13. Yen, T.Y.; Wolf, W.; Performance estimation for real-time distributed embedded systems, *Computer Design: VLSI in Computers and Processors*, 1995. ICCD '95. Proceedings., 1995 IEEE International Conference on , 2-4 Oct. 1995, Pages:64 – 69
14. Wolf, W.H.; Hardware-software co-design of embedded systems [and prolog], *Proceedings of the IEEE* ,Volume: 82 , Issue: 7 , July 1994, Pages:967 – 989
15. Wolf, W.; A decade of hardware/software codesign *Computer*, IEEE Transactions On, page(s): 38- 43, Volume: 36, Issue: 4, April 2003
16. Li, Y.-T.S.; Malik, S.; Performance analysis of embedded software using implicit path enumeration, *Computer-Aided Design of Integrated Circuits and Systems*, IEEE Transactions on, On page(s): 1477-1487,Volume: 16, Issue: 12, Dec 1997
17. Asaad, S.; Bapty, T.; Performance modeling for adaptive parallel embedded systems. *Performance, Computing, and Communications Conference*, 2002. 21st IEEE International , 3-5 April 2002, Pages:57 – 64
18. Luca Benini; Giovanni De Micheli;System-Level Power Optimization:Techniques and Tools. *ACM Transactions on Design Automation of Electronic Systems*, Vol. 5, No. 2, April 2000,Pages 115–192.
19. Ishihara, T. AND Yaasura, H. 1998. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED '98, Monterey, CA, Aug. 10–12)*, A. Chandrakasan and S.Kiaei, Eds. ACM Press, New York, NY, 197–202.
20. Johan Pouwelse, Koen Langendoen, Henk Sips; Dynamic voltage scaling on a low-power microprocessor.. July 2001 *Proceedings of the 7th annual international conference on Mobile computing and networking*
21. T. K. Tan, A. K. Raghunathan, G. Lakishminarayana, N. K. Jha High-level software energy macro-modeling. June 2001, *Proceedings of the 38th conference on Design automation*
22. Intel Corporation. Intel Xscale Microarchitecture for the PXA255 Processor: User's Manual, March 2003.Order No. 278796.
23. Thomas D. Burd, Trevor A. Pering, Anthony J. Stratakos, and Robert W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid State Circuits*, 35(11):1571{1580, November 2000.
24. Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the International Conference on Mobile Computing and Networking*, pages 251{259. SIGMOBILE:ACM Special Interest Group on Mobility of Systems, Users, Data, and Computing, ACM Press, 2001.

25. Micron Technology. Calculating memory system power for DDR. Technical Report TN-43-03, Micron Technology Inc., May 2001.
26. Transmeta Corporation, Santa Clara, CA. Crusoe TM5800 Version 2.1 Data Book, June 2003.
27. Wissinger, J., Washburn, R., Morgan, D., Chong, C., Friedland, N., Nowicki, A., and Fung, R. (1996) Search algorithms for model-based SAR ATR., In Proceedings of SPIE Conference on Algorithms for Synthetic Aperture Radar Imagery III, 2757, 1996,
28. Castanon, D.A.;Optimal search strategies in dynamic hypothesis testing Decision and Control, 1993., Proceedings of the 32nd IEEE Conference on 15-17 Dec. 1993 Page(s):265 – 270 vol.1
29. Michael Boshra and Bir Bhanu. Predicting performance of object recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(9):956-969, September 2000.
30. John C. Mossing and Timothy D. Ross. An evaluation of SAR ATR algorithm performance sensitivity to MSTAR extended operating conditions. In Edmund G. Zelnio, editor, Algorithms for Synthetic Aperture Radar Imagery V, Proc. Of SPIE, volume 3370, pages 554-565, 1998
31. Leslie M. Novak, Gregory J. Owirka, and William S. Brower. Performance of 10- and 20-target MSE classifiers. IEEE Transactions on Aerospace and Electronic Systems, 36(4):1279-1289, October 2000.
32. Cory Myers, Lawrence R. Rabiner, and Aaron E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(6):623-635, December 1980.
33. Constantinos Boulis, Mari Ostendorf, Eve A. Riskin, and Scott Otterson. Graceful degradation of speech recognition performance over packet-erasure networks. IEEE Transactions on Speech and Audio Processing, 10(8):580-590, November 2002.
34. Michael D. DeVore, Roger D. Chamberlain, George L. Engel, Mark A. Franklin, and Joseph A. O'Sullivan. Tradeoffs between quality of results and resource consumption in a recognition system. In Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors, pages 391-402, 2002
35. Michael D. DeVore, Joseph A. O'Sullivan, Roger D. Chamberlain, and Mark A. Franklin. Relationships between computational system performance and recognition system performance. In Firooz A. Sadjadi, editor, Automatic Target Recognition XI, Proc. Of SPIE, volume 4379, 2001.

36. Contreras, G., Martonosi, M., Peng, J., Ju, R., and Lueh, G. 2004. XTREM: a power simulator for the Intel Xscale® core. *SIGPLAN Not.* 39, 7 (Jul. 2004), 115-125.
37. Brooks, D., Tiwari, V., and Martonosi, M. 2000. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual international Symposium on Computer Architecture* (Vancouver, British Columbia, Canada). ISCA '00. ACM Press, New York, NY, 83-94.
38. Kotecha, J.H.; Ramachandran, V.; Sayeed, A.M. Distributed multitarget classification in wireless sensor networks. *Selected Areas in Communications, IEEE Journal on*, Vol.23, Iss.4, April 2005 Pages: 703- 713
39. Harry L. Van Trees. *Detection, Estimation and Modulation Theory Part I*. John Wiley and Sons. 1968.
40. Compact Flash Association. CF+ and CompactFlash Specification Revision 3.0. Dec 2004.
41. Laura Marie Feeney and Martin Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. *Proceedings of IEEE INFOCOM*. 2001.
42. Todd Austin, Eric Larson, and Dan Earnst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35(2):59-67, February 2002.

APPENDIX A

Image Rendering and Noise Simulation

In this discussion we elaborate on the methodology and approaches used in implementing a framework for generating synthetic optical imagery and using such imagery to build recognition templates for the purposes of classification in a ATR system. These templates act as a repository of target recognition models for different target classes, model complexities and target pose. The rest of the chapter further elaborates on what we mean by such characteristics as model pose and complexity.

The recognition models thus derived are intended to adequately characterize the surface properties and shape of a target for a given class and target pose. Our primary intention for generating such models is to generate and maintain a minimal set of statistical measures to classify a target accurately. The data sets that we generate should be optimal with respect to achieving desired recognition accuracy and should lend themselves for utilization in a set of algorithms for accurate target classification. We intend to strike a balance between the complexity of the models employed and computational complexity of the algorithm itself. The computational costs that are of interest include the costs associated with the generation of the target models and the runtime costs associated with the recognition algorithm that utilizes these models. While the costs associated with generating the models are a one-time expenditure, the run-time or execution-time costs associated with the recognition algorithm are recurring costs and are of primary interest to us and this is the cost we intend to minimize. However, the accuracy of the recognition algorithms is largely dependent on the modeling effort and

also on the fit between the algorithms used and the data being modeled. Our objectives as part of this work have been to find the least complex of data models and algorithms that render very high rates of accuracy, thereby minimizing both model building costs and run-time costs associated with utilizing such models for classification.

3-D Representation of physical objects

3-D models are mathematical representations of three dimensional physical objects which can serve as inputs to various programs to be displayed as two-dimensional images through processes of image rendering or ray tracing. Such models are extensively used in the computer graphics industry to generate photo-realistic simulations of physical phenomenon as well as in the production of computer games or other animations.

3-D models usually consist of collections of various three dimensional geometric objects along with their associated optical properties. As an example, a simple 3-D model for a sphere may simply consist of two elements, a vector value for the center of the sphere in Cartesian co-ordinates and a value for the radius of the sphere. Additionally, other elements may specify the surface color or various object properties of the sphere. As an example, Fig A.1 shows the syntax for describing a sphere in POV-Ray SDL, a language for describing three dimensional scenes and objects.

```
SPHERE:
sphere
{
    <Center>, Radius
    [OBJECT_MODIFIERS...]
}
```

Fig A.1 Syntax for describing a sphere in the POV-Ray scene description language.

To generate a simulated image from a 3-D model, several other additional elements have to be added to complete a scene. Addition of basic elements such as a camera and a light source usually suffice to generate a two dimensional image from a simple 3-D model. The union of such elements along with description of a 3-D object is known a 3-D scene. The following example describes a basic 3-D scene in the POV-Ray SDL. The scene describes a three dimensional representation of a sphere along with a camera and light source. The two dimensional image generated from the scene through ray tracing is given in the following figure.

```
#include "colors.inc"

global_settings {max_trace_level 5}

camera {
    location <-5, 10, -15>
    look_at <0,0,0>
}

light_source {< -50, 25, -50> color White
}

sphere { <0,0,0>, 3.75
    pigment {color rgb < 0.5, 0.2, 0.5>}
}
```

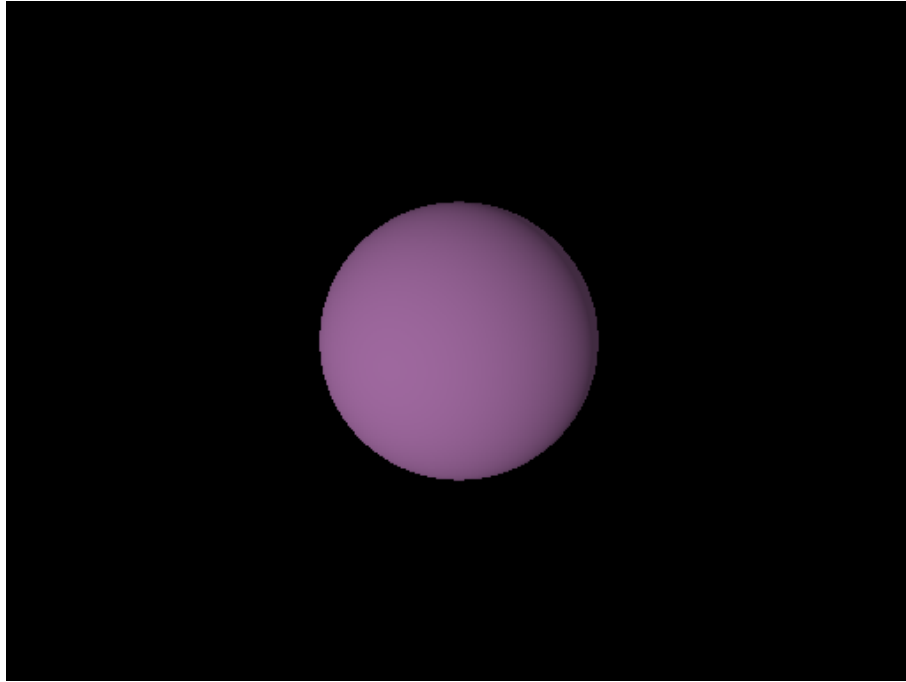


Fig A.2: POV-Ray Syntax and Resulting image for a simple sphere

Representation of Complex Objects

Three dimensional models of practical use, however, can rarely be defined with simplicity as illustrated in the previous section. A complex mechanical or physical object, for practical use in computer aided design or engineering is usually modeled as a collection of various components, with each component consisting of many subcomponents in many cases. As an example, a three dimensional physical model of a vehicle may consist of various sub-components such as the vehicle body, doors and windows, and even models for smaller elements such as wiper blades and door handles. Each sub-component might further be modeled as a collection of basic geometric objects such as triangles, spheres, ellipsoids etc. 3-D model representations also allow us to assign surface properties to complex as well as simple objects. As an example, we can decide to paint the whole surface of a vehicle model with a single uniform color or

provide different color assignments to individual subcomponents. Similarly a host of other surface attributes such as reflectivity, opacity, color gradients can be assigned to surfaces at any level of complexity.



Fig A.3: POV-Ray output from 3D model of a vehicle

Triangulation of Surfaces and Noise Induction

It is a common practice in models of high complexity and details to approach the problem of modeling with the help of triangulated surfaces or meshes. In this approach, each component and subcomponent of a three dimensional model is broken down into pieces constructed entirely of collections of triangles, collectively defining object surfaces. The following figure illustrates a triangulated surface of a vehicle after being rendered in 3-D modeling software.

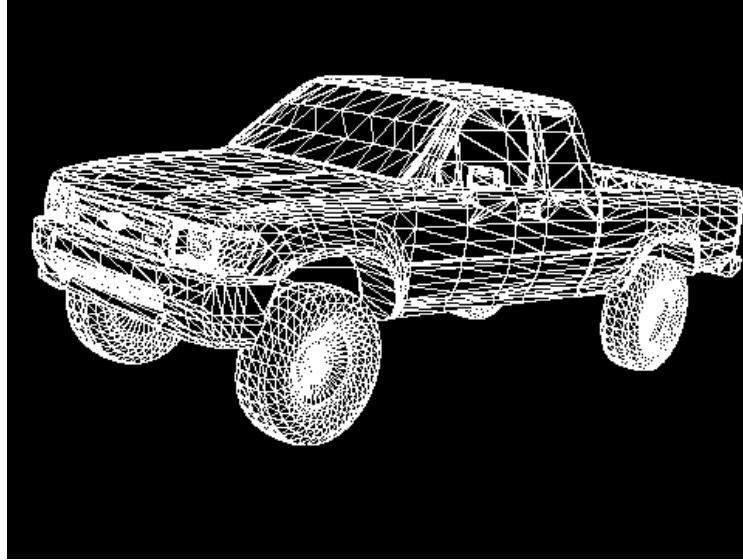


Fig A.4: Mesh diagram of a triangulated vehicle surface

Triangulated surfaces also provide benefits in terms of providing the ability to manipulate object models and in generating point cloud representations of physical objects. Such representations can be useful in simulating noise on the surfaces of physical objects and also in animating object motion by collectively displacing selected subcomponents or surfaces of objects.

During the course of this work we have developed a method for extracting point clouds from 3-D models of various objects and then injecting slight perturbations into the location of such points to provide a noisy surface to work with. The reconstruction of a triangulated surface with their vertices displaced from their original positions gives rise to a surface that loses its smoothness but retains continuity. The process of determining the point cloud and the reconstructing a noisy surface from a POV-Ray description file consists of the following steps:

- i) Extraction of all vertices of the global list of triangles for an object.

- ii) Extraction of a set of unique points from the global list of vertices. This reduces the data set because several points serve as a vertex for multiple adjoining triangles.
- iii) Add random amounts of displacements to all the unique points in the global list of vertices and replace the original vertices with the new noisy vertices in the triangulated surface.

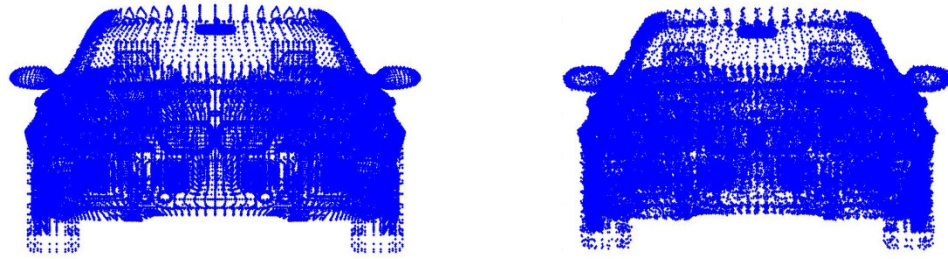


Fig A.5: Transformation of vehicle surfaces by noise addition

Figure A.5 illustrates the transformation of a smooth vehicular surface to a noisy but continuous surface without losing the basic characteristics of the vehicle shape. The final step in generating an image from a noisy surface requires the introduction of these noisy triangular objects into a POV-Ray description file and adding camera, lighting and surface properties to generate a series of images for various vehicle poses. We utilize this repository of noisy images to statistically model the representation of various vehicles classes and vehicle poses to serve as an input to our recognition algorithms.

Generation of Imagery and Recognition Model Building

The generation of a repository of vehicle images is the first step in building the recognition database that we will often traverse to solve our recognition problem. The recognition database consists of templates of various complexities for multiple vehicle

classes. We differentiate recognition complexity into seven consecutive levels, with each increase in the complexity level increasing the number of recognition templates by a factor of four for each class and simultaneously also increasing the resolution of the recognition templates several times.

A better understanding of the notion of recognition template complexity can be developed with visualizing an imaginary hemisphere drawn around a vehicle, with the geometric center of the vehicle body lying on the line joining the apex of the hemisphere to the center of the circular base. All images of the vehicle are then generated by positioning the camera on the hemispherical surface, with the focal plane of the camera forming a tangent to the surface of the hemisphere. This ensures that the focal plane of the camera is always perpendicular to the center of the base of the hemisphere and is also at the same distance from the center for all images. In a hypothetical scenario, an infinite number of images can be generated by placing the camera on every point on the hemispherical surface and spanning the entire surface that is available. Such an exercise will however be computationally impractical and cannot be feasibly implemented.

To overcome this inability to have a representation of the object for an infinite array of poses we can introduce some amount of segmentation where the hemispherical surface around the target object can be divided into several regions of interest. We then develop statistical representations from a finite number of images from each region and come up with measures that can satisfactorily represent image behavior for all camera positions contained within a region.

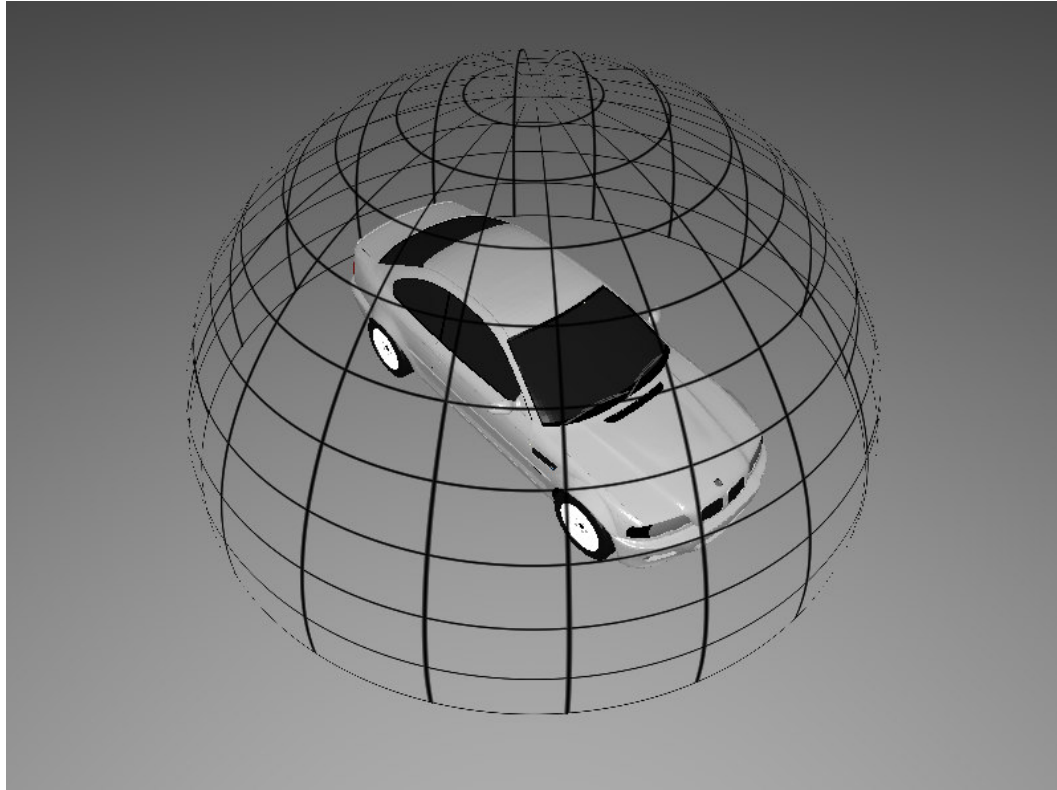


Fig A.6: Imaginary hemisphere around a vehicle and its associated segmentation

As an example, Fig. A.6 illustrates the segmentation of the hemispherical surface around a target object into sixteen regions of equal areas. As a method of representing the imagery corresponding to each region we estimate the mean and variance of pixel values from all the images generated by camera positions within each region. Figure A.7 provides examples of mean images generated from each region when the hemisphere is divided into sixteen equal parts.

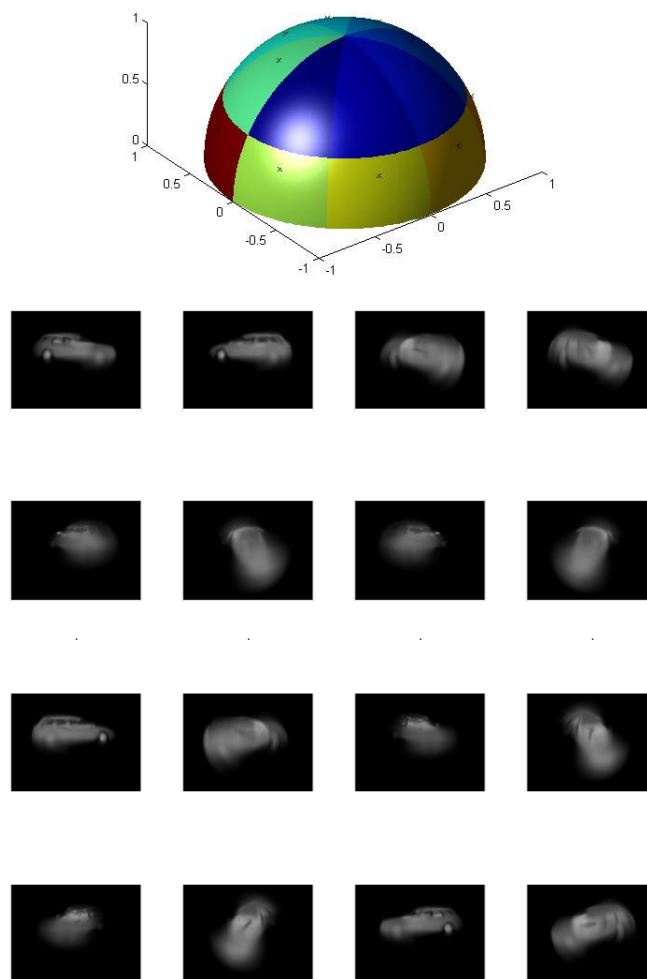


Fig A.7: Mean images of vehicle from 16 segmented regions of the hemispherical surface

The positioning of the camera on the imaginary hemispherical surface can be better understood by describing the location of the camera with the help of spherical coordinates in a three dimensional space. Consider the location of the camera at a point P in figure A.8. The location of the camera on the hemispherical surface can then be described with three variables, the azimuthal angle θ which lies in the x - y plane going anticlockwise from the x -axis with $0 \leq \theta < 2\pi$, the elevation angle φ to be the polar angle from the x - y plane and r , the distance (radius) from any point on the spherical surface to the origin or the center of the sphere.

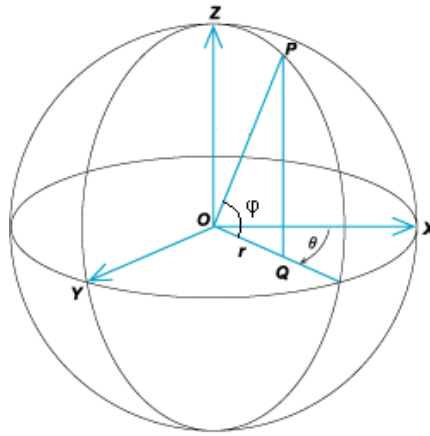


Fig A.8: Graphical Representation of a camera position P, in a spherical coordinate system

In the next section we describe a scheme for dividing the hemispherical surface to generate recognition templates for a variety of vehicle poses. As we move the camera on the hemispherical surface we group together images for variable azimuth and elevation spans according to the level of complexity we intend to achieve in our modeling process. This gives rise to a variable number of sub-models or recognition templates for different levels of complexities.

Recognition Database Complexity

In order to represent the statistical properties of object imagery across a vast array of poses we introduce the concept of segmenting the surface area around the vehicle in the form of segmented areas on a hemispherical surface. The segmentation of this area is done in an ordered fashion to attain various levels of representation in terms of resolution. As an example, the segmentation of the hemisphere into four equal parts will result in a mean image that is quite blurry and will have high variances in pixel values when compared to the same generated from segmentation of the sphere into sixteen equal parts. The clarity of the mean and variance images will keep on increasing with successive increases in levels of segmentation till we reach a point where further segmentation fails to yield any substantial and usable information. Each level of segmentation yields a set of mean and variance images which constitute the database of recognition templates for that particular level. In our recognition methodology we try to classify an object for various levels of complexities that correspond strictly to the set of templates generated from different levels of segmentation of the geometric space around the objects we have modeled and generated imagery using various camera positions. At different levels of complexity, the recognition algorithm switches the recognition database to a different set of templates with increasing complexities leading to an increased number of templates to match and also increasing the amount of information that is found in the templates.

The criterion that we use to segment the space around the object is based on the division of the hemispherical surface around an object into equal areas. The hemispherical surface around the object of interest can be defined with the co-ordinate

position of the center of the hemispherical space and the radius of the hemisphere. The location of the camera on the surface plane can be defined in co-ordinate space with the distance from the center of the sphere and the angles of azimuth and elevation. As we can assume a constant radius for a hemispherical surface, the only variables in the position of the camera are the angles of azimuth and elevation. However, if we apply an equal area criterion for the segmentation of the hemisphere, such areas do not translate into equal divisions of elevation angles but do so for azimuth.

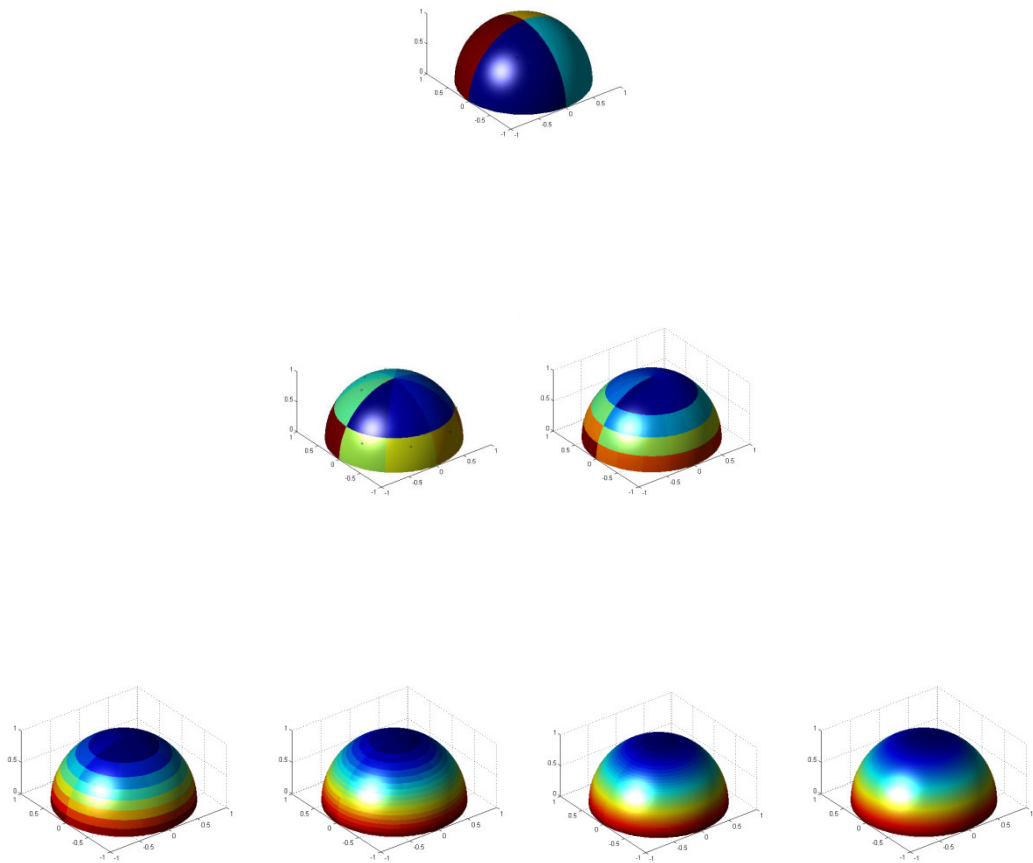


Fig A.9: Segmentation of the hemispherical surface for different levels of complexity

Table A.1: Listing of number of templates per vehicle for each complexity level

Level of Complexity	Number of Templates per vehicle
1	4
2	16
3	64
4	256
5	1024
6	4096
7	16384

At the lowest level of segmentation, we divide the surface traversed by the camera into four equal regions, which translates into regions that span ninety degrees in azimuth and ninety degrees in elevation. With each successive level of segmentation, we increase the number of divisions by a factor of two for both the azimuth and elevation angles. As a result the number of regions of segmentation increases by a factor of four for each jump in the level of complexity. We model each object for seven consecutive levels of complexities, with the least complex model having four templates for each object while at the highest level of complexity, we divide the hemisphere into 16384 regions of equal areas and correspondingly generate a template for each segment. Table A.1 depicts the number of templates for each level of recognition complexity and also depicts the segmentation of the hemispherical surfaces for each level.

Surface Segments and Camera Positioning

In the previous sections we had discussed the methodology of positioning our simulated camera on the surface of an imaginary hemisphere placed around the target to generate imagery for building our recognition templates. We had also discussed methods for

segmenting the sphere into various regions of equal areas and how we utilize imagery from a given area to obtain mean and variance models to build our recognition templates. In this section, we look into the actual positioning of our simulated camera inside the various regions and the number of images we utilize to generate templates for each level of complexity.

In figure A.10 we represent a section of the hemispherical surface as a two-dimensional square for the sake of simplicity. This surface has been divided into sixty four equal areas with the smallest division representing the highest level of complexity in our modeling process. As a result of this partitioning, this arbitrary region on the hemispherical surface will give rise to sixty four templates at the seventh and highest level of complexity, sixteen templates at the sixth level, four at the fifth level and one for level four. The illustration also shows the areas covered by the templates at the different levels with the help of dark borders. In table A.2 we list the azimuth and elevation spanned by the camera for modeling a single template at various levels of complexity along with the number of templates generated for each level.

The image generation exercise that we conduct to produce our recognition templates involve generating images for vehicles under consideration by placing the camera at various locations inside the each area partition and then analyzing pixel values from such images to estimate our pixel means and variances for each region. As a result of the proposed partitioning scheme, imagery generation can be done at the highest level of complexity and the same images can be processed at successively decreasing levels of complexity to achieve our template generation goals. Recognition templates at each level thus differ in the number of images that are processed, with the least complex models

assimilating data from a very large number of images spanning almost a quarter of the hemispherical surface while the most complex models incorporate few images from a very small region.

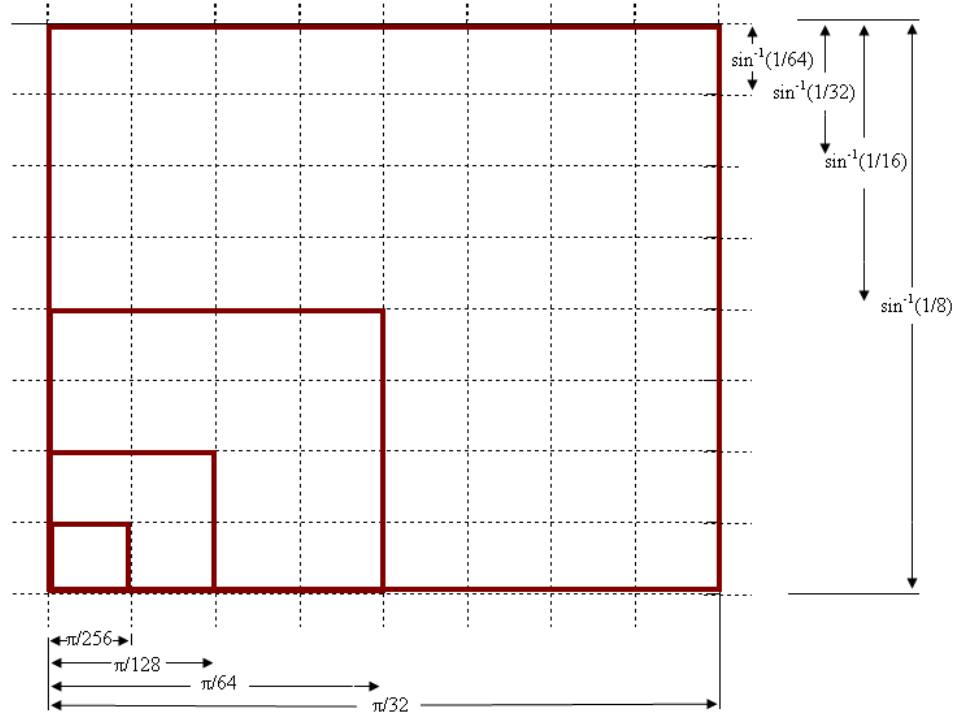


Fig A.10: Representation of a section of the hemispherical surface for illustrating camera positioning

Table A.1: Listing of number of templates per vehicle for each complexity level

Level of Complexity	Span in Azimuth	Span in Sine of Elevation
1	$\pi/2$	1
2	$\pi/4$	$\frac{1}{2}$
3	$\pi/8$	$\frac{1}{4}$
4	$\pi/16$	$\frac{1}{8}$
5	$\pi/32$	$\frac{1}{16}$
6	$\pi/64$	$\frac{1}{32}$
7	$\pi/128$	$\frac{1}{64}$

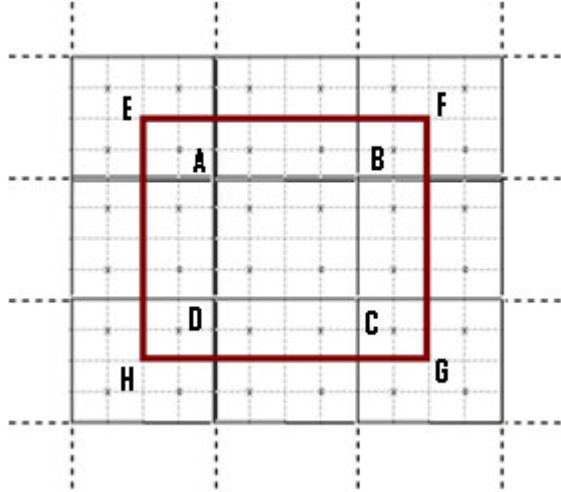


Fig A.11: Camera positioning for generation of a single recognition template at the highest level of complexity

Figure A.11 illustrates the approximate camera positions for the images that are generated for a region marked by the quadrilateral $ABCD$ on the hemisphere. The span of this region corresponds to the area covered by a single template at the highest level of complexity for the purposes of this example. Within this region, four images are generated at the locations marked by crosshairs. The camera positions are so chosen such that the locations are uniformly distributed in both azimuth and in the sine value of the elevation. A total of 65536 images are thus generated for a single vehicle spanning all the 16384 sub-regions existing at the highest or seventh level of complexity, with four images for each region.

Recognition template generation for the region $ABCD$ however includes the four images within the region as well as bordering images from the adjoining regions. A total of 16 images are processed, as enclosed by the dark border $EFGH$. This introduces some

amount of information overlap between templates generated from adjoining areas but results in a more robust modeling exercise by taking care of spaces between camera positions on the hemisphere. An exception to this method is made for templates generated from the lowest regions in elevation where there are no images available for elevations below zero degrees. In such a scenario, we process bordering images from five adjoining regions, two from each side and the top and one image each from the upper corners. A special case also arises for the topmost elevations where we generate and include images from the apex of the hemisphere in the mean and variance estimates.

For the next lower level of complexity, we increase the region of interest by a factor of four. At this level each template covers an area that is four times that of the area covered by *ABCD* in the previous figure. We illustrate this with the help of Fig A.12, where the region *IJKL* represents an area represented by a template at the sixth level of complexity. This area then gives rise to a single template at the sixth level utilizing the same set of images that were used to generate four templates for the next higher level. On similar lines to the previous example we process additional images from the bordering regions in addition to images contained within the region of interest. A total of thirty-six images are processed for each template at this level, which are enclosed by the quadrilateral *MNOP*. In Fig A.13 we illustrate the processing of images from four adjoining areas at level six. The images processed to generate recognition templates are marked by four distinct overlapping regions, with each region enclosing thirty six camera positions.

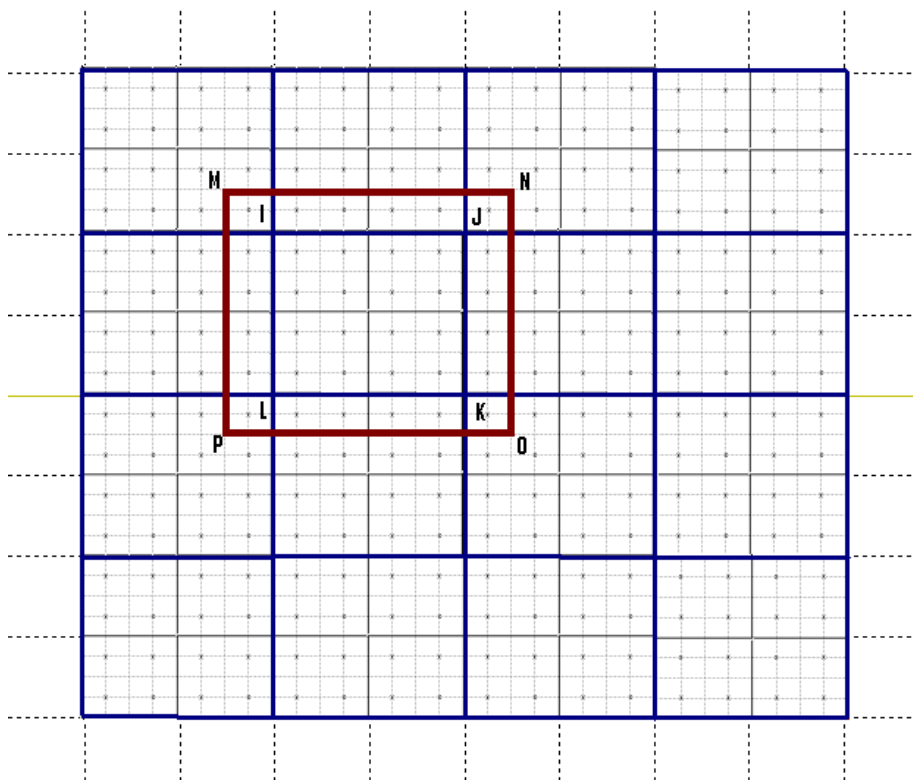


Fig A.12: Camera positioning for generation of a single recognition template at the second highest level of complexity

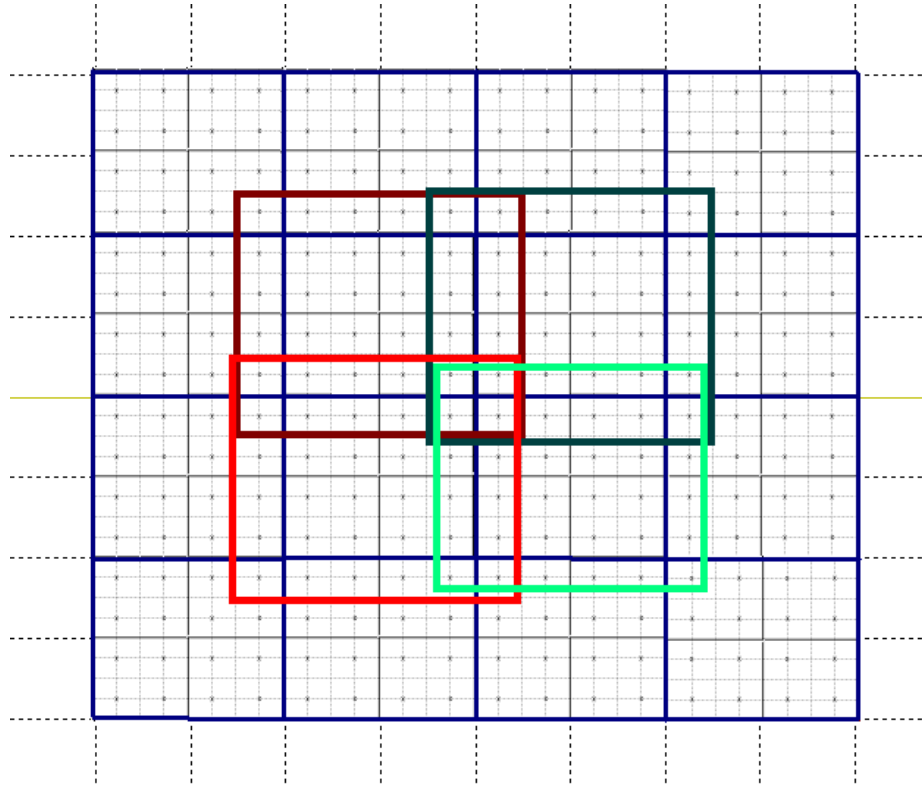


Fig A.13: Camera positioning for generation of a single recognition templates for adjoining segments

Using a similar scheme the recognition templates are built using images generated with a uniform distribution of camera positions for various complexity levels. In general with every increase in the level of complexity, the number of images processed for each template and the span or the area traversed by the camera decreases by approximately a factor of four.

Experimental Datasets and Software Tools

The ideas for recognition template generation developed as part of this thesis are implemented to develop a database of recognition models for four different vehicles. The 3-D models that were utilized for generating artificial imagery have been procured from Mesh Factory Inc. (www.meshfactory.com). The 3D model files procured from Mesh

Factory conform to the popular 3ds format and lend themselves to manipulation by a variety of software.

The ray-tracing software used extensively for imagery generation as part of the experiments during the course of this thesis was POV-Ray, a high quality free tool for creating three dimensional graphics on various operating systems. POV-Ray however does not support the 3ds format for image rendering and scene manipulation. As the first step in our image generation process we use Nugraf®, a 3D data translation, rendering and scene composition software for windows, to translate the 3ds model files to the POV-Ray scene description language. The POV-Ray SDL provides great flexibility for camera position manipulation and noise addition to the 3D models, processes which form a critical part of our template generation process.

An automated process for the generation of simulated optical imagery was established using a combination of Unix shell scripts and C programs. Preprocessing of the POV-Ray SDL files were done to split the camera position and triangle vertex information into two separate files. For each image generated the standard MATLAB rand() function was used to add noise to each vertex location and camera positions were updated according to the scheme developed for utilizing an entire hemispherical surface around the vehicle for camera locations. The updated camera position and vertex information were included in the main POV-Ray file before rendering an image for each camera location. A single point source of white light was added at the camera location for illuminating the scene. Additionally, the camera plane was always positioned to look at the centre of the geometric reference frame, i.e. the center of the base of the hemispherical surface on which the camera was positioned.

Images were rendered in 16 bit grayscale PNG format with a size of 320×240 pixels. A clear noise free uniformly black background was chosen to describe the rest of the scene. Fig A.14 provides an example of a vehicle image from the repository. The scene illustrates a significant amount of noise on the surface of the vehicle body, and the clear black background accompanying the vehicle image.



Fig A.14: Sample image from image repository

The generation of recognition templates at each level of complexity was done using MATLAB by calculating the statistical estimates for pixel values for groups of images pertaining to each recognition template. Recognition templates were stored as integer values in binary files using a format developed for utilizing the data in a mobile embedded platform.

Vehicle models

The four specific vehicles that we chose as part of the experimental implementation include a sedan, a two door coupe, a hatchback sedan and a sport utility vehicle. Fig A.15 depicts the representative images of these vehicles and Table A.3 lists the geometric properties of the vehicle models.

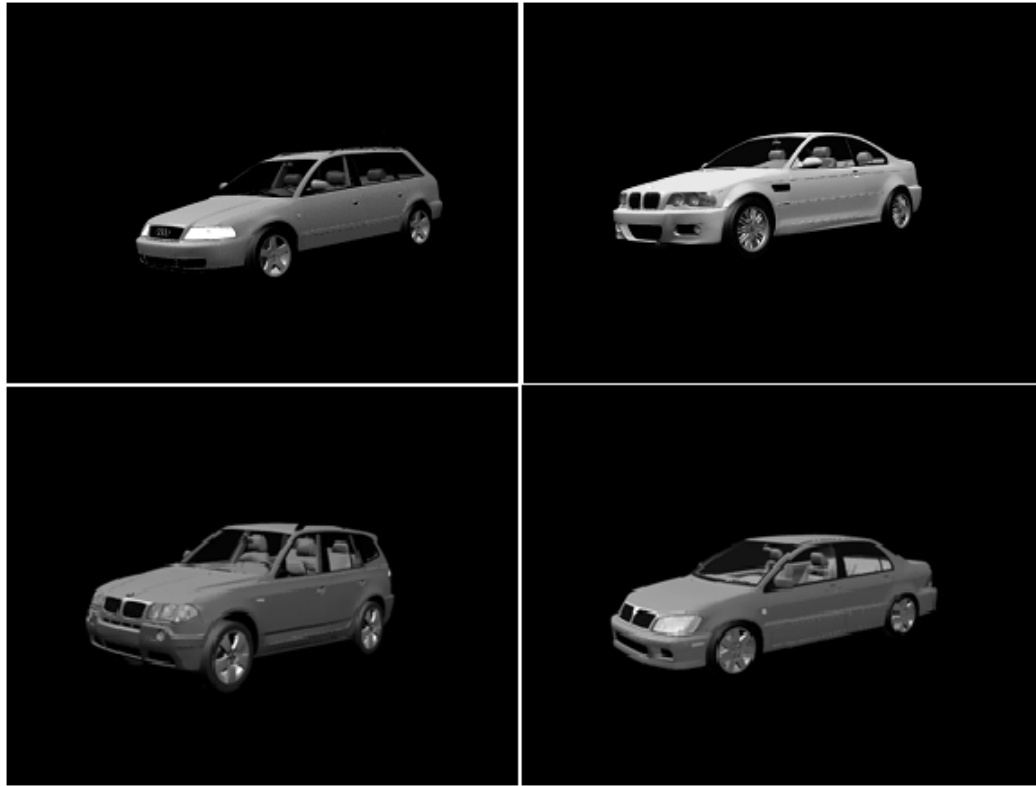


Fig A.15: Vehicle models used in implementing recognition subsystem. Clockwise from top left: Audi A4 Sedan, BMW M3 Coupe, Mitsubishi Lancer Sedan, BMW X3 SUV

Table A.1: Vehicle models and their associated 3-D model characteristics

Model identifier	Vehicle Class	No of surface points	No of surface triangles
Audi	Hatchback	144418	278822
Bmw	Coupe	147017	283872
Mlance	Sedan	60365	114426
bmx3	SUV	44205	77782

APPENDIX B

System Implementation

In this discussion we describe the system implementation of our experimental ATR system in terms of the hardware and software configurations.

Hardware Platform

The recognition system is deployed on a network of identical hardware nodes built on the Stargate high-performance processing platform manufactured by Crossbow Inc. The Stargate is built around the Intel PXA255 processor, which is commonly found in various handheld devices in the market. The Stargate platform is intended for use in sensor, signal processing, control and wireless sensor networking applications. Table B.1 lists the various characteristics of the Stargate Platform from the manufacturer's datasheet. Figs B.1 and B.2 show some photographic images of the Stargate platform.



Fig B.1: Top and Bottom views of the Stargate main board



Fig B.2: View of the Stargate daughter card

Table B.1: Stargate Specifications

Specifications	Remarks
STARGATE Processor Board	400 MHz, RISC Processor
Intel PXA255, Xscale®	Multiple I/O Companion Chip
Intel SA1111, StrongARM®	
Memory	
64 MB SDRAM	
32 MB FLASH	Linux Software < 10 MBytes
Communications	
PCMCIA Slot	Type II
Compact Flash Slot	Type II
51-pin MICA2/GPIO	UART, SSP via MICA2 Connector
Optional I2C Port	Installable Header
Optional Serial Port (2)	Installable Header
General	
Li-Ion Battery Option	
Watch Dog Timer (WDT)	Configurable up to 60 seconds
Battery Gas Gauge	
LED and User Application Switch	
Power Switch	
STARGATE Daughter Card	
Communications	
10 Base-T Ethernet Port	RJ-45 Connector
RS-232 Serial Port	DB-9 Connector
JTAG Debug Port	
USB Host Port	Version 1.1
General	
A/C Power Adaptor	5-6 VDC, 1 Amp
Reset Button	
Real-Time Clock	
Physical	
Processor Board (in)	3.50 x 2.49 x 0.73
Weight (oz)	1.68
Daughter Card (in)	2.49 x 2.00 x 0.60
Weight (oz)	1.42
Environmental	
Operating Temperature	0 to +70 (°C)

The Stargate main board takes in 3.8 – 5 V, which is fed into a linear dropout regulator to bring the voltage down to 3.3 V, which is then fed to all peripherals. The core processor takes a voltage of 0.8 – 1.3 V through a DC – DC converter. Fig B.3 shows a block diagram of the stargate platform. The daughter card can be used for debugging purposes and the platform can be powered by AC-DC converter if the daughter card is attached.

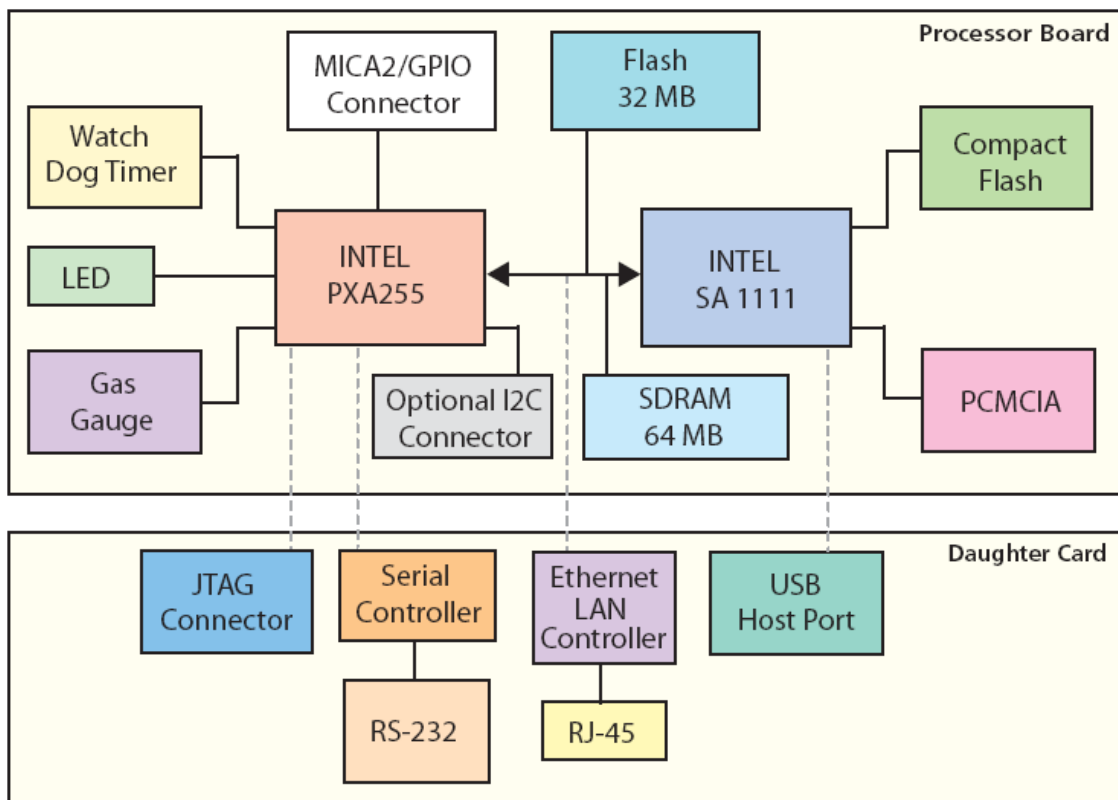


Fig B.3: Stargate Block Diagram

The final system implementation of a processing node consists of the basic stargate platform along with a 802.11 wireless networking card on the PCMCIA interface and a 2-Gigabyte flash memory card as the storage device for our recognition models and applications.

ATR System Software Architecture

The ATR application implements a client-server model with one manager node and several server nodes. The manager node acts as a client of the server nodes which run the recognition application. The operating system on which the manager and server nodes run is a variant of linux employing the kernel version 2.4.19. The system software is available as an open source distribution at <http://platformx.sourceforge.net>. The system software package includes various Familiar and Debian utilities. The filesystems supported by the platform include jffs2, ext2, ext3, vfat, and msdos file types along with support for POSIX. The platform also includes java and perl runtime environments.

Manager Node

The application software on the manager node consists of a client application that communicates with the server nodes to enquire on their availability and distributes images to the server nodes when a recognition task is initiated. The manager node waits for recognition results from the server nodes and selects the best result among all nodes to arrive at a recognition decision. A flowchart depicting the logical architecture of the manager node is depicted in Fig B.4.

The modules installed on the manager node are designated as *sendfile* and *mserver*. A separate script named *sendfile.sh* initiates the recognition process at the manager node. Bringing up the manager consists of powering up the Stargate Platform and then initiating the 802.11 wireless interface. The final step consists of mounting a compact flash card with the manager applications on the */dev/hda1* mount point. The resulting application files can then be found in the */mnt/cf1/recognition* directory.

The initiation of a recognition task is achieved by executing the *sendfile.sh* script with the binary image data file as its argument. Upon initiation the *mserver* application is triggered and keeps waiting for results from the server nodes. After receiving the final results from all nodes the *mserver* application displays the results for the recognition task and also stores the results in a log file named *results.log*.

Server Nodes

The server nodes deploy a multi-threaded server application, which performs the recognition task upon receiving an image for classification from the manager node. The server node application is also dependent on a local repository of statistical image models, which it utilizes to perform a recognition task. Potentially the server nodes can take up multiple processing tasks at the same time which enable them to provide services to one or more than one manager nodes in the network.

The programmatic flow of the server node application consists of waiting for an image from any connecting client and then initiating the recognition process upon receipt of an image. On completion of all processing the server node transmits back the results to the appropriate manager node. The server nodes can simultaneously wait for an image and continue working on an existing problem at the same time.

The software installation of the server node consists of copying the *nserver* executable file on the */mnt/cfl/recognition* directory and also the presence of a file named *argfile*, in the same directory. The second file contains static information about the file system of the server node database, the identities of the vehicle databases that are available in the repository of the server node and also the level of complexity at which a server node must operate while processing an image for classification. The server node application is

triggered by executing the *nserver* program in background mode. The application then keeps listening on a pre-designated port for service requests from the manager node. Upon receipt of a service request from the manager node, the application starts looking for the best match in its database. The best results are then printed into a log file, which is communicated back to the manager for processing.

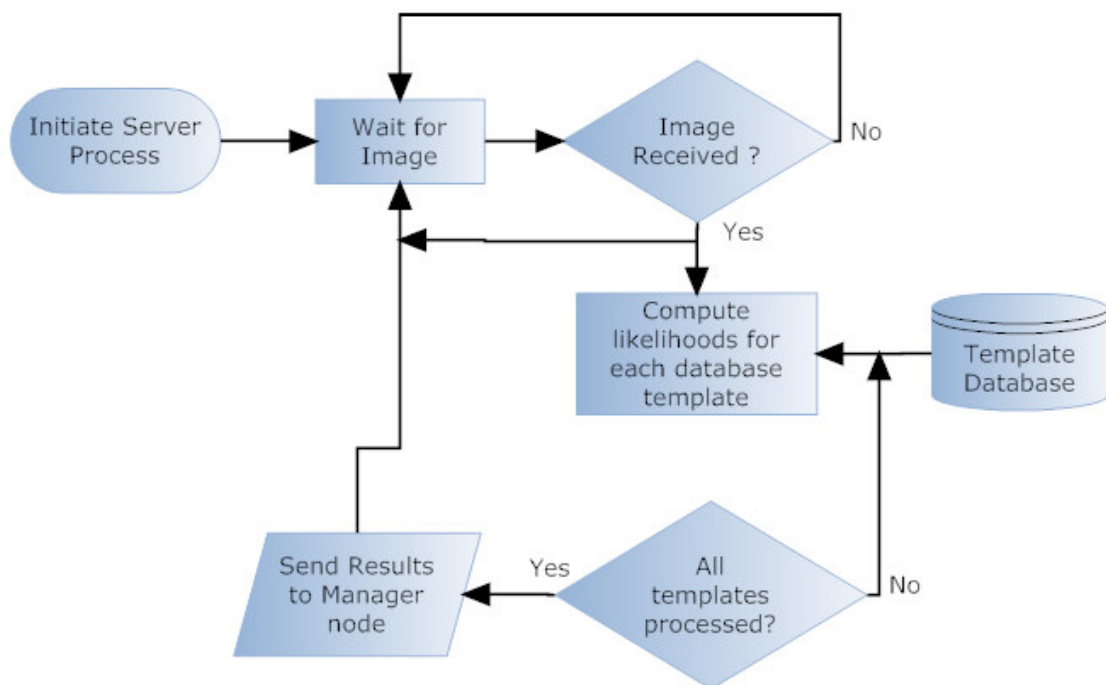


Fig B.4: Server Node Flow Diagram

Server Database

The server database consists of binary files that store mean and variance information of 16 bit integer grayscale pixel intensities pre-calculated for various poses of vehicles. The header information of the binary files consists of 9600 bytes consisting of pixel identifiers, which are relevant to the model. Each bit in the first 9600 bytes corresponds

to a pixel in a 320 X 240 image, with a value of 1 indicating that the pixel is to be included in the likelihood computation. The binary values map to a 320 X 240 image by storing data serially, with the first 30 bytes corresponding to 240 pixels in the first row. The next 30 bytes correspond to the second row in the image and so on. The rest of the bytes following the header store the actual mean and variance information for the pixels for which the header bits are set to 1. In our experiments we have dealt with uniform variances with variance values being set to the number of pixels of interest for which we encounter non-zero mean values in the recognition templates. In such a case we cease to store variance information in the data files but calculate the variance by summing up the number of non-zero bits in the header portion. As discussed earlier, for each level of recognition complexity k , we have 4^k files stored in the database located on the CF card. On the CF card file system, we separate the database files corresponding to different levels and models by storing them in separate directories. The directory structure is defined in a file titled *argfile* which is provided as an input to the recognition server so that it is able to locate the appropriate files at runtime.

Experimental Setup

A major part of this work involved measuring current usage and voltage information for the stargate during the execution of recognition algorithms on each node. Voltage and current measurements were done by tapping into the main power supply to the stargate node. This provides sufficient information to characterize the overall power consumption of the entire node, which includes power consumed in the peripheral devices as well as the processor. This approach was taken as it was observed that the idle power consumed by the peripheral devices constituted a significant portion of the power consumption even

at runtime. Any power model that aims to characterize the behaviour of the stargate platform will prove futile if only the processor was modelled, especially in cases where applications involved heavy peripheral usage. A typical ATR application accessing the PCMCIA wireless interface for communication with the manager node and intensively looking up a recognition database on an external CF card will consume a considerable amount of power in addition to power consumption by the processor.

Fig B.5 shows the typical experimental setup used for current and voltage measurements. A digital multimeter was placed in series with the main power supply to measure current usage by a server node. Voltage measurements were taken by a parallel setup at the main power supply to the daughter card. An RS-232 interface is available at the multimeter output for interfacing to a PC program which records current data at fixed one second intervals.

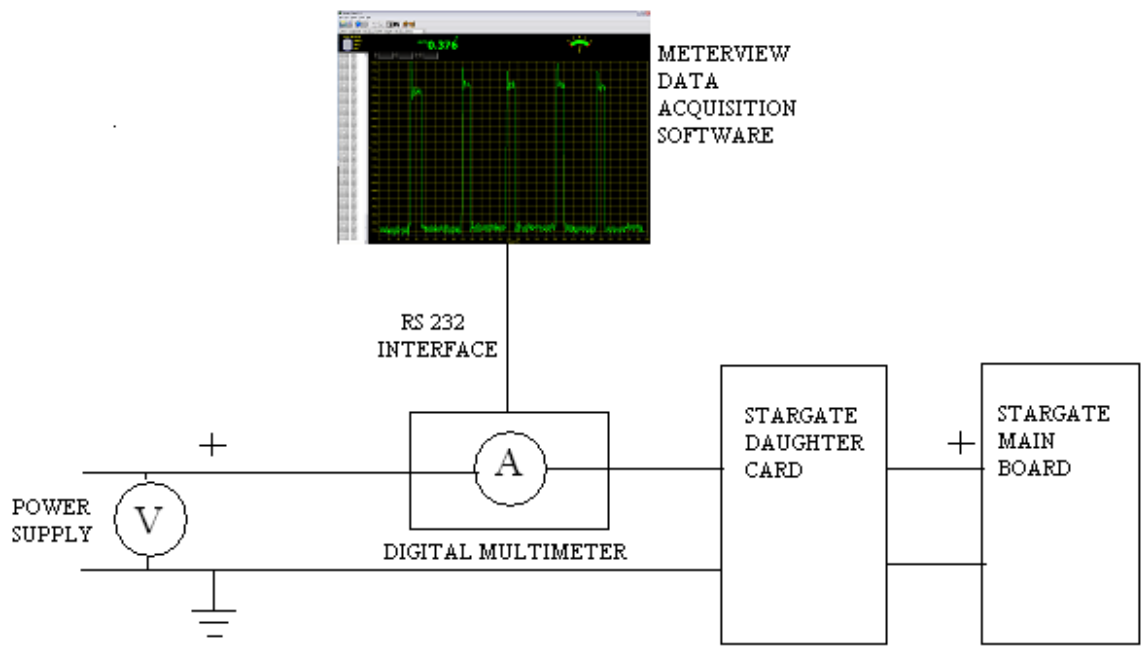


Fig B.5: Experimental setup for current and voltage measurements

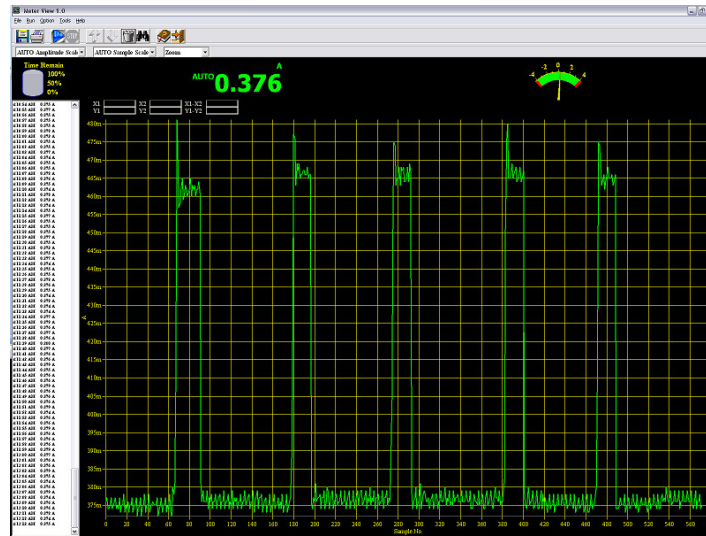


Fig B.6: Meterview software interface for current and voltage data acquisition

Data acquisition for power measurements on the stargate is done using the Meterview program on Windows PC. A serial port interface is needed for the program to gather data from the digital multimeter. The program interface is depicted in Fig B.6. The program supports real time display of sampled data and also provides a facility for saving historical data in ASCII text format that can be loaded into Microsoft Excel for later analysis.

We estimate the average power consumption of the ATR system at runtime by multiplying the average current consumption with the supply voltage, which is set at 5.13 V. The energy usage is determined by the product of the average power consumption and the time taken by the ATR system to process an input image to arrive at a recognition result. Data acquired for individual runs at each complexity level and processor state is stored along with execution time data for regression analysis which is undertaken offline using MATLAB.

At higher levels of database complexity, individual runs provide adequate data to measure steady state power consumption accurately. However, for lower levels of complexity, individual runs are very short in duration and often fail to yield any substantial data with a sampling interval of one second. To circumvent this problem, multiple numbers of images are queued to the server nodes to keep them continuously busy for each measurement interval. The average of the current consumption is then multiplied by the source voltage to yield the power consumption for each level of database complexity and processor state.

Frequency Scaling on the Intel PXA 255

The Intel PXA 255 processor lies at the core of our hardware implementation and the various experiments that we perform to characterize the performance of our ATR system involves measuring time and power consumption values for different frequency settings of the core CPU and bus frequency in this platform. In this section we briefly describe the methodology to obtain various processor and bus frequencies.

Support for frequency scaling comes inbuilt in the PXA255 processor and can be executed at the system level by following the procedures laid down in Developer's Manual and the Intel Xscale Microarchitecture User's Guide for the PXA255 processor. The execution of a frequency change operation requires privileged mode access and cannot be done with an application program running in the user space. We implement this mechanism by writing a kernel module that is plugged into the operating system kernel to provide us privileges to initiate frequency changes on the processor.

A frequency change sequence is initiated by accessing the core clock configuration register (CCLKCFG) which resides on Coprocessor 14. Fig B.7 below gives the layout of the register from the Intel PXA255 Microarchitecture Manual. Bits 3:0 of the register can be modified to enter a frequency change sequence as shown in Fig B.7.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCLKCFG																															
reset value: unpredictable																															
Bits		Access		Description																											
31:4		Read-unpredictable / Write-as-Zero		Reserved																											
3:0		Read / Write		Core Clock Configuration (CCLKCFG) 0b0001 - Enter Turbo Mode 0b001x - Enter Frequency Change Sequence (Turbo Mode bit may be set or cleared in the same write) Other values are reserved																											

Fig B.7: Layout of the CCLKCFG register(CP14, register 6) and associated instruction values

Once a frequency change sequence is entered, the desired system frequencies may be obtained by writing desired values to the core clock configuration register (CCCR), which is a memory mapped register with a physical address of 0x41300000. The different frequency settings that can be realised is given in Table B.2, along with the associated values to be written into CCCR.

Table B.2: CCCR values and the associated system frequency values

Mode	CCCR Value	TURBO Bit Value	Core Frequency(MHz)	Internal Bus Freq(MHz)	Memory Freq(MHz)
1	0x00000121	0	99.5	50	99.5
2	0x00000221	1	199.1	50	99.5
3	0x00000141	0	199.1	99.5	99.5
4	0x00000321	1	298.6	50	99.5
5	0x000001C1	1	298.6	99.5	99.5
6	0x00000241	1	398.1	99.5	99.5
7	0x00000161	0	398.1	196	99.5

