A Visual Servoing System for an Amphibious Legged Robot

Junaed Sattar

Master of Science

School of Computer Science

McGill University

Montréal,Québec

August 2005

A Thesis submitted to McGill University
in partial fulfilment of the
requirements for the degree of
Masters of Science

Your file Votre référence
ISBN: 978-0-494-24792-1
Our file Notre référence
ISBN: 978-0-494-24792-1

DEDICATION

This thesis is dedicated to Rafa, for turning my life around.

ACKNOWLEDGEMENTS

This thesis has been made possible because of the inspiration and support of a wonderful group of people, and it would be greatly unfair not to acknowledge their invaluable input. First and foremost I am thankful to my research supervisor, Professor Gregory Dudek, for being a patient and steadfast guide and helping this research along across more than a few difficult hurdles, for inspiring me through difficult times by his friendly demeanor and his endless enthusiasm for research.

My gratitude goes out to my fellow researchers in the Centre for Intelligent Machines, particularly those in the Mobile Robotics and Mechatronics Locomotion Lab. I would especially like to thank Philippe Giguère for lending a big hand before the first AQUA servoing field trials in Barbados; Chris Prahacs and Shane Saunderson for working tirelessly behind the AQUA robot and giving me such a wonderfully reliable platform for my research; Paul Di Marco, Dimitri Marinakis, Matthew Garden, David Meger, Derek Johns and Daniel Burfoot for their invaluable suggestions; Ioannis Rekleitis and Eric Bourque for showing me the way around the lab in my early days; and Michelle Théberge for volunteering to carry the target during the sea trials. A special thanks goes out to Michel Langlois for translating the abstract to French.

Finally, I would like to thank my family for being so supportive and for encouraging me to embark on my endeavors.

ABSTRACT

We present a visual servoing system for an amphibious legged robot. That is, a monocular-vision based servoing mechanism that enables the robot to track and follow a target both underwater and on the ground. We used three different tracking algorithms to track and localize the target in the image, with color being the tracked feature. Tracking is performed based on the object's color, color distribution and color distribution with a probabilistic kernel. Output from the tracker is channeled to a proportional-integral-derivative controller, which generates steering commands for the robot controller. The robot controller in turn takes the steering commands and generates motor commands for the six legs of the robot. A large class of significant applications can be addressed by allowing such a robot to follow a diver or some other moving target. The system has been evaluated in the open water and under natural lighting conditions, and has successfully performed tracking and following of a wide variety of target objects.

ABRÉGÉ

Nous présentons un système d'asservissement visuel pour un robot amphibie muni de jambes. Il s'agit d'un mécanisme de vision monoculaire qui permet au robot de dépister et suivre une cible mouvante sous l'eau et sur la terre. Nous avons utilisé trois algorithmes différents pour dépister et localiser la cible dans l'image, la couleur étant la caractéristique examinée. Le trajet à suivre est déterminé par les couleurs de l'objet, la distribution de ces couleurs et la distribution des couleurs avec un kernel probabiliste. La sortie du traqueur est envoyée à un contrôleur de PID, qui génère des commandes directives. Ces commandes sont transmises au contrôleur du robot, qui relaie pour sa part des commandes motrices aux six jambes du robot. Permettre à un tel robot de suivre un plongeur ou une quelconque autre cible mouvante pourra aider à solutionner de nombreuses applications significatives. Le système a été évalué en eau libre et dans des conditions d'éclairage normales, et a suivi avec succès une grande variété d'objets-cibles.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

We describe the vision-based servo control of a swimming aquatic robot. We have developed and deployed a swimming robot that uses legged motion to swim and navigate underwater and which depends on vision as its primary sensing modality. While sonar is the predominant sensor used by underwater vehicles, vision has the potential to be highly effective underwater as it is on land. In this thesis, we examine the use of visual feedback to accomplish navigation tasks in an open water environment. This is accomplished by using a visual feedback mechanism to modify the swimming gaits of our underwater robot as it follows a moving target. A large class of significant applications can be leveraged by allowing such a robot to follow a diver or some other moving target. Performing these tasks underwater is complicated by the variable lighting and visibility in the water. In addition, the undulating motion of our vehicle and the exogenously driven motion of the vehicle in the open sea further complicate the process. In this thesis we describe the system architecture and approach to vehicle control, with discussions of the relevant theory behind the servoing mechanism.

Underwater robotics appears to be an application domain of rapidly increasing significance, rife with challenges of both scientific and pragmatic importance. While computer vision has matured enormously in the last few decades, the peculiarities of underwater (sub-sea) vision have been largely ignored, presumably due to the enormous logistic and pragmatic overhead in examining them. It is akin to the manner in which the topography and zoology of the sub-sea environment has been ignored relative to the terrestrial analogs. In fact, vision can be as valuable a sensing medium underwater, and perhaps even more so than on land. Simple inspection of marina fauna demonstrates the ubiquity of eyes, and other optical sensors, in the marine environment and thus suggests its potential utility.

In our application we are particularly interested in tracking a diver as he swims either along the surface or under water using scuba apparatus. In this case we need a tracking technology that imposes a very limited cognitive load on the driver, which operates despite variations in lighting due to refractive effects and/or waves, which is immune to nearby wave action and which operates over a moderate range of distances.

In our applications, vision has the advantage of being a passive sensing medium and it is thus both non-intrusive as well as energy efficient. These are both important considerations (in contrast to sonar) in a range of applications ranging from environmental assays to security surveillance. Alternative sensing media such as sonar also suffer from several deficiencies which make them difficult to use for tracking moving targets at close range in potentially turbulent water.

1.1 Problem Statement

The task we want the robot to perform is that of following a particular moving object, either artificial or natural, under water and in an open water environment, thus achieving some degree of autonomy.

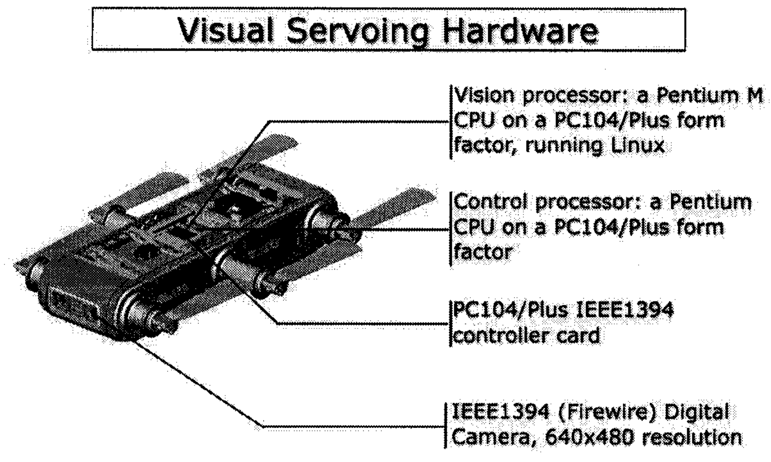

Visual Servoing Hardware
Vision processor: a Pentium M CPU on a PC104/Plus form factor, running Linux
Control processor: a Pentium CPU on a PC104/Plus form factor
PC104/Plus IEEE1394 controller card
IEEE1394 (Firewire) Digital Camera, 640x480 resolution

Figure 1–1: AQUA visual servoing hardware.

The robot houses three cameras, two in front and one in the back, as can be seen from Fig. 1–1. We use a color camera to track the target and utilize the tracking output to control the robot's trajectory in a closed-loop fashion. We require that the robot follows the target's trajectory as closely as possible. The robot must be able to maintain track in spite of change in lighting conditions and strong currents and other underwater forces that affect its hydrodynamics. One might even imagine the need for the robot to re-acquire the target in case of presence of false targets or a temporary failure in tracking. The robot should be trained with respect to the target (and its color properties) only once, and to rely only on that prior information to track. We also seek to utilize the servoing system to maneuver the robot in ways that would not be possible by using human input alone. In view of the real-time operating constraints, the entire mechanism of tracking, steering feedback and trajectory control needs to be fast as well as accurate. Evaluating the performance of the system quantitatively is quite a challenge; nevertheless we evaluate the servoing system in terms of the correctness and efficiency of their individual components – the tracker and the PID controller. Performance of the three different tracking methods used is discussed in detail, and the responsiveness of the controller given the tracking output is presented as well. This thesis focuses mostly on the experimental results, with the theory behind the different approaches being discussed to explain and justify their relevance. Complexity of the system is discussed both in terms of computation as well as the time required by the robot to perform the commands generated by the system. The results of the open– and closed-water trials are presented to show the success of the system and also to clearly establish directions for future work.

1.2 Approach

Our approach to servoing is two-fold: we use color cues to detect and track the target and use the tracking information in a feedback controller to generate commands for the robot's legs. Three different approaches are implemented for color tracking. In this work, target location is defined in image space, *i.e.* no pose estimation of the target is performed. We do not estimate the robot's pose either, but that information is available via an on-board inertial measurement unit. The robot "learns" the target color parameters at the start of the servoing run. This tuning step can be performed both on land or underwater, and is performed once. This is the only preprocessing step in the entire process. The tracking subsystem detects the target in the image coordinates and passes them on to the controller. The controller acts as a low-pass filter. We use a manually-tuned PID controller to eliminate oscillations and increase the responsiveness of the system. The PID controller has two control loops, one each for the yaw and pitch axes. The roll axis is not affected by servoing. An outline of the method is as follows:

- Preprocessing Stage

  i The tracker is tuned by placing the target at the center of the camera frame.

  ii The target color parameters are extracted. Depending of the tracker being used, RGB thresholds, histograms or histogram distributions are saved.

- Tracking

  i The target is tracked using one of three following approaches:

    a Color Segment or "Blob" Tracking, where image segmentation is applied to localize the target.

    b Color Histogram Tracker, where a model color histogram of the target is matched to other candidate histograms to detect target location.

    c Kernel-based Feature Tracker, where we use a radially symmetric monotonically decreasing kernel to extract target features. Candidate locations are generated by using a mean-shift tracking mechanism.

- Control

  i The goal of the controller is to minimize an error function, which we define as the Euclidean distance between the centroid of the target and center of the image frame. At each iteration, pitch and yaw commands are generated with the aim of reducing this error function. The gains described below are manually-tuned.

    a The proportional gain K_p contributes to the error correction amount since it is multiplied with the error signal (hence the name proportional gain).

    b The integral gain K_i contributes to the error correction amount since it is multiplied with the error accumulated over a period of time.

c The derivative gain K_d contributes to the error correction amount since it is multiplied with the derivative of error signal.

ii The pitch and yaw commands are sent to the robot gait controller. The gait controller generates commands for the robot's six legs and achieves the desired change in motion.

The loop continues, with every change in the target's position generating a (possible) change in the robot's pose. Figures 1–2 and 1–3 show outlines of the entire process.

1.3 Applications

Even mundane activities underwater pose problems for humans in terms of logistics, cost, efficiency and safety. As such, underwater environments represent a substantial area in which robotics can make a natural contribution. A range of applications can be identified for which simple inspection even in moderately shallow water can prove useful. These applications include underwater search and rescue, coral health monitoring, monitoring of underwater establishments (*e.g.* oil pipelines, communication cables) and many more. Specifically, we are interested in environmental assessment tasks in which visual measurements of a marine ecosystem must be taken on a regular basis. While automatically selecting regions of interest is beyond the scope of present technologies, once a biologist identifies areas of interest we believe a robot may be capable of collecting supplementary data or even independently executing inspection tours. It is in this context that the present work is framed.

1.4 Outline

This thesis discusses an approach to visual servoing an underwater legged robot. We discuss theoretical and practical aspects of this work and the problem in general. In Chapter 2, previous work in the area and individual problems associated with servoing are discussed, with an emphasis to those that relate to this particular work. We describe the overall system architecture in Chapter 3. Chapters 4 discusses tracking algorithms and control methods. The implementation details of the tracking and control systems are discussed in Chapter 5. We present some experimental results obtained from the system as a whole, as well as the performance results from the trackers used in Chapter 6. We conclude in Chapter 7 by discussing the results, and also identifying areas for improvement in future work.

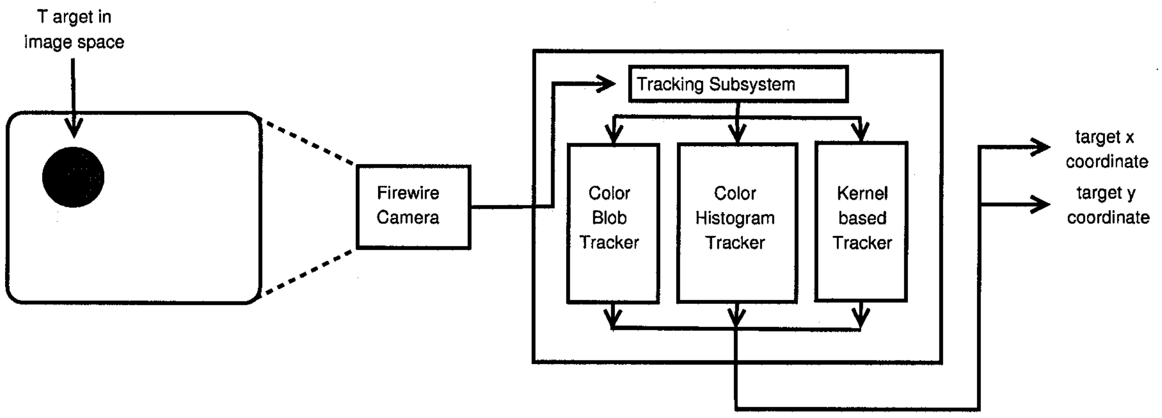


Figure 1–2: Tracker subsystem in AQUA visual servoing.

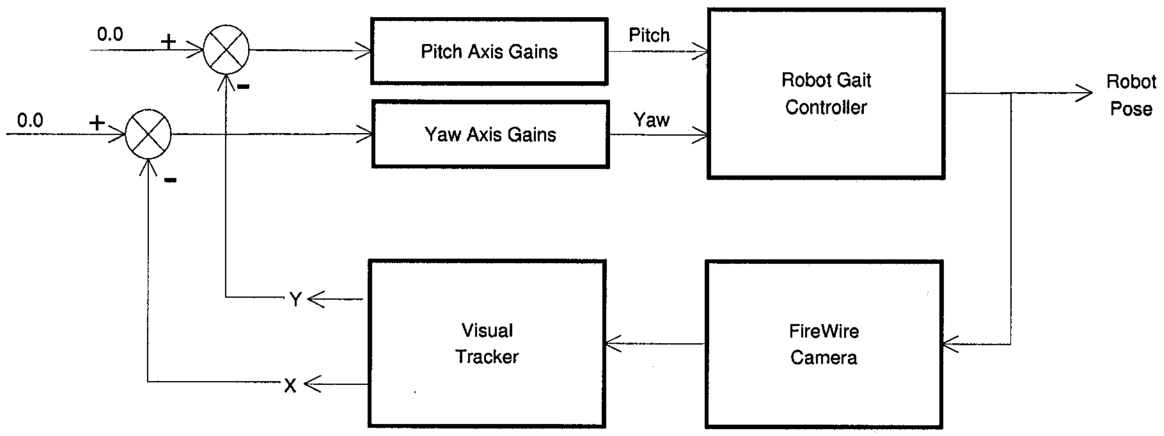


Figure 1–3: Control loops in AQUA visual servoing.

CHAPTER 2
Previous Work

This chapter surveys past work in the field of visual servoing and underwater robotics. Visual servoing is a complex task, involving a multitude of other disciplines; computer vision, image processing, control and tracking being the primary ones. Visual tracking is an important mainstay of servoing, and we discuss several different tracking methods used in the field. For an embedded platform like AQUA, real-time performance is of paramount importance, therefore we keep the focus on algorithms that have performed reasonably well within limited time and computational resources. In the end, we discuss AQUA visual servoing software architecture to put everything together and compare it with some other software architectures used in real robots over the years.

2.1 Visual Tracking

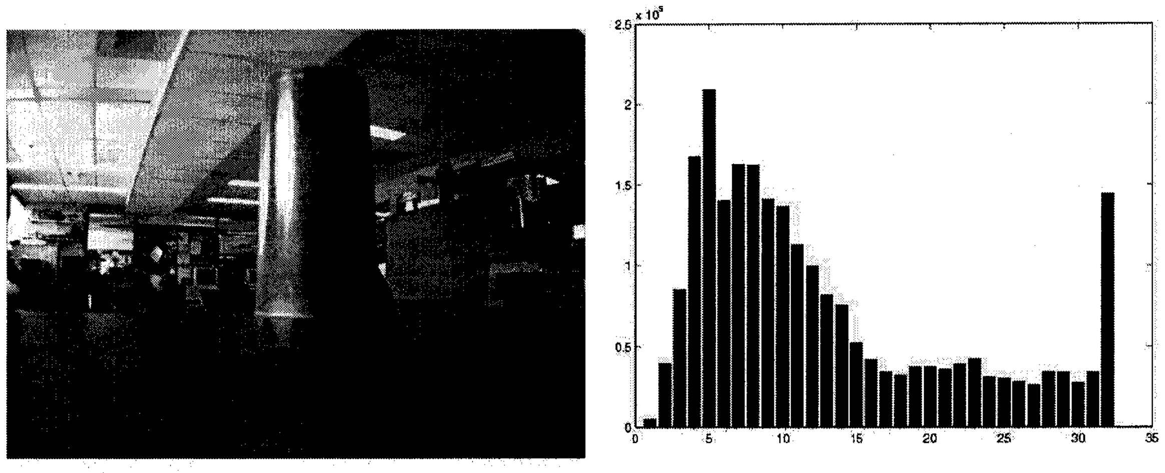
In computer vision, visual tracking is the process of repeatedly detecting a feature or sets of features in a sequence of input images. Choosing features to track can be a complicated problem, since noise in the sensor (*i.e.* camera), lighting and visibility changes, refraction and appearance of multiple similar objects in the image frame, among others, can create unforeseen problems. Since tracking is primarily an online, real-time application of vision, a tracking algorithm must be fast, as well as accurate. A tracker also needs to be robust, so that effects of false targets and occlusions are minimized.

Choosing a feature to track is an important step in tracking algorithm design, as already stated above. Over the years, a large amount of work has been done on tracking algorithms that track features ranging from shape and motion to color and grayscale intensities. The following are examples of approaches that have been proven to work best among these algorithms.

Techmer [43] uses object contours to detect motion and hence track targets. Freedman and Brandstein [13] have investigated detecting object contours in cluttered environments. Isard and Blake [23, 24] introduced the "condensation" or Conditional Density Propagation algorithm for stochastically tracking curves or contour shapes in a clutter. This is also known as Tracking using Particle Filters. Tracking contours usually involves an iterative scheme that converges to the shape being tracked after a finite number of iterations, and uses a probabilistic approach to converge to the best solution at the instant. While they can be quite accurate, contour trackers rely heavily on a clear view of the target that shows object boundaries distinctly compared to the background.

Tracking objects by their color has been extensively studied in the past. Color blob tracking is one of the simplest approaches. Color-based segmentation or "blobs" have been applied to not only tracking but also object recognition [19] and image retrieval [18]. Color blob trackers segment out sections of the image that match a threshold level for the given target[25] and based on the segmentation output, tracks the shape, size or centroid of the blob, among other features. The color space chosen for segmentation has a major impact on the performance of the algorithm. A detailed discussion of different color spaces and their impact on tracking can be found in [27]. We discuss the RGB and HSV color spaces briefly in Chapter 4, since these two spaces are used in our approach.

Color histograms are a measure of color distribution over an image, and are n-dimensional histograms over a neighborhood. The range of the possible color values depends on the color space used in the histogram algorithm. The possible color ranges are subdivided into a discrete number of 'buckets' or 'bins'. Each bin holds the number of pixels that has color values which fall between the upper and lower limit of the bin. An image and its corresponding 32-bin histogram is shown in Fig. 2–1.



(a) Sample Color Image (b) Sample Color Histogram

Figure 2–1: Sample color image with its color histogram.

Color histograms have been applied in color-based image retrieval applications and video indexing [14] by matching histograms of the source image with candidate images in a database. Swain and Ballard [39] demonstrate the use of histograms in their landmark paper. The same concept of histogram matching has been used in detecting and tracking targets in computer vision. Rubner and Tomasi [37] discuss different approaches to histogram based image retrieval, with a focus on a variety of methods for measuring distances between histograms. In their work, they introduce a measure called the Earth

Mover's Distance or EMD for histogram similarity measurement. Other measures for histogram distances exists and are widely used for histogram matching. We discuss more about these different methods in Chapter 4.

Some of the above methods of tracking are combined with statistical methods to provide more accurate results, albeit at the cost of increased processing time. Particle filters and Kalman filters are widely used statistical approaches to tracking. In recent work, the mean-shift process [6] has been successfully used for tracking in conjunction with color cues. Mean-shift tracking algorithms attempt to maximize the statistical correlation between two distributions. where the correlation between the two distributions are measured using the Bhattacharyya Distance [44]. Statistical distributions can be built using any characteristic discriminating to a particular object of interest. A general model might use color, or texture or include both. Zivkovic and others [48] have used machine learning approaches with color histograms for target tracking. Xu et al. [47] discusses a robust mean shift tracking algorithm by applying fast color thresholding. A more detailed discussion about mean-shift tracking and similarity measures can be found in Chapter 4.

2.2 Control

For every change in the target's position, a corresponding change in the robots pose may or may not be required. Responding to every change in tracking output will result in a very unstable and ill-behaving robot. Thus, a mechanism is required for relating the changes in target position to changes in actuator input in a stable and smooth manner. Control theory defines the laws by which this can be achieved. To achieve stable control, the output of a system is related to the input via a *Transfer Function.* In *Open-Loop Systems*, the input to the system does not rely on any feedback from the output, as shown in Fig. 2–2. In *Closed-Loop Systems*, the input of the system depends on the current state of the output as well as the new input; *i.e.* the system relies on **feedback** of the output. An outline of a closed-loop control system is shown in Fig. 2–3. A brief tutorial on the basics of control theory can be found in [34]. In Chapter 4, we briefly define some concepts of control theory that relates to this work.

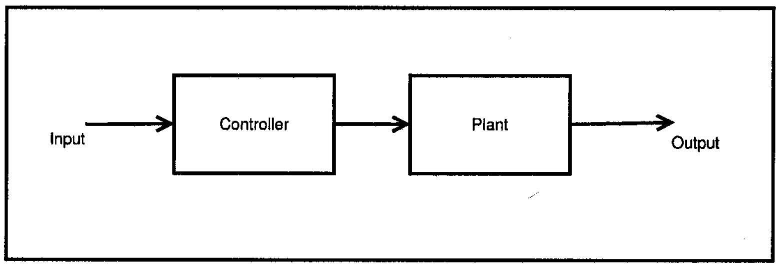


Figure 2–2: Open-loop control.

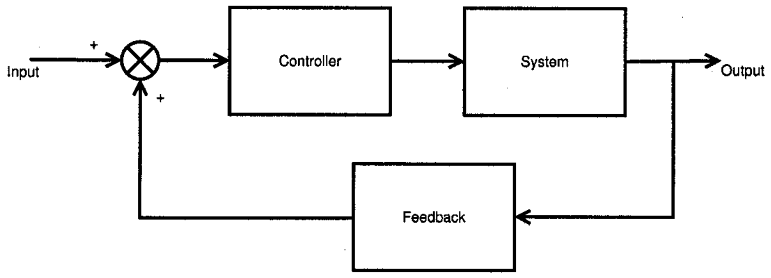


Figure 2–3: Closed-loop control.

2.3 Visual Servoing

A substantial amount of work has been done on visual servoing over the past fifteen years. Hutchinson, Corke and Hager's seminal paper [22] outlines many of the methods used in practice today as well as the basic foundations of servoing. This paper is an excellent tutorial on visual servo control methods for robotic manipulators. The authors classify servo methods based on the hierarchy of the control system and the domain of the error signal. Based on control hierarchy, servo systems are of two classes; namely, direct visual servo and dynamic look-and-move systems. In dynamic look-and-move, a hierarchical control system uses vision to to provide set-point inputs to the joint-level controller to internally stabilize the system. In direct visual servoing, a visual servo controller computes joint inputs directly instead of an intermediate joint controller. Again, based on the domain of error signals (*i.e.* if its is in image or task space) , the servo systems can be classified as position-based servo and image-based servoing systems. In image-based systems, control values are computed from the features in the images directly, whereas in position-based servo, image features and geometric model of the target is used to generate control values.

A variety of approaches has been adopted for visual servoing in the recent past. Cowan and Koditschek [8] show that visual servoing can be approached as a robot navigation problem. Hager discusses the use of stereo vision for robust positioning in [20]. Planning camera motion is an important aspect of visual servoing, specially in image-based servoing since target locations are specified in image coordinates. An image-based servo mechanism has to take into account issues like maintaining the target in the field of view and obstacle avoidance, among others, A treatment of such issues can be found in Marchand and Hager [32].

2.4 Underwater Robotics

Underwater robotics research has been one of the more challenging domains of robotics science. The underwater domain poses certain unique challenges that render a lot of the principles of terrestrial robotics problematic. An underwater robot has six degrees of freedom, and maneuvering with six degrees of freedom creates serious complications. The three axes of control for the AQUA robot can be seen in Fig. 2–4. A computationally straightforward task of pose maintenance on land becomes far more challenging under water, because of strong currents in marine environments. Infra-red sensors lose some of their effectiveness in water as well. Wireless radio communications are also impossible over a large distance in water compared to ground based control. All these issues make underwater robotics problems more difficult than terrestrial robotics. To a degree, interplanetary space rovers like the Mars rovers Spirit and Opportunity [28] face a less daunting computational task than an underwater autonomous vehicle.

In spite of all the hindrances, substantial progress has been made in designing the hardware and algorithms for underwater robots, and much of the research is directed in creating an autonomous underwater vehicle (AUV) for operator-independent exploration of underwater environments. Robotics researchers have taken a number of approaches in creating underwater robots. The traditional approach to propel undersea vehicles is by using propellers or thrusters. Although simple by design, these vehicles lack the maneuverability and agility seen in fish and other marine species. For an AUV, efficient energy consumption is critical, and thrusters are not an energy efficient approach to station keeping underwater [16]. Among other efforts to proper underwater vehicles, the RoboTuna project at Massachusetts Institute of Technology (MIT) is well known. The RoboTuna project [42] attempted to create a fish-like underwater vehicle, with a propulsion system mimicking those found in fish, hence creating an example of Biomemetic Robotics applied in underwater environments. Figure 2–5 shows the RoboTuna shell and the skeleton.

The MIT Sea-Grant Program has an extensive program to create new underwater platforms for deep ocean explorations, including AUVs. The flapping-foil fish robot [29]

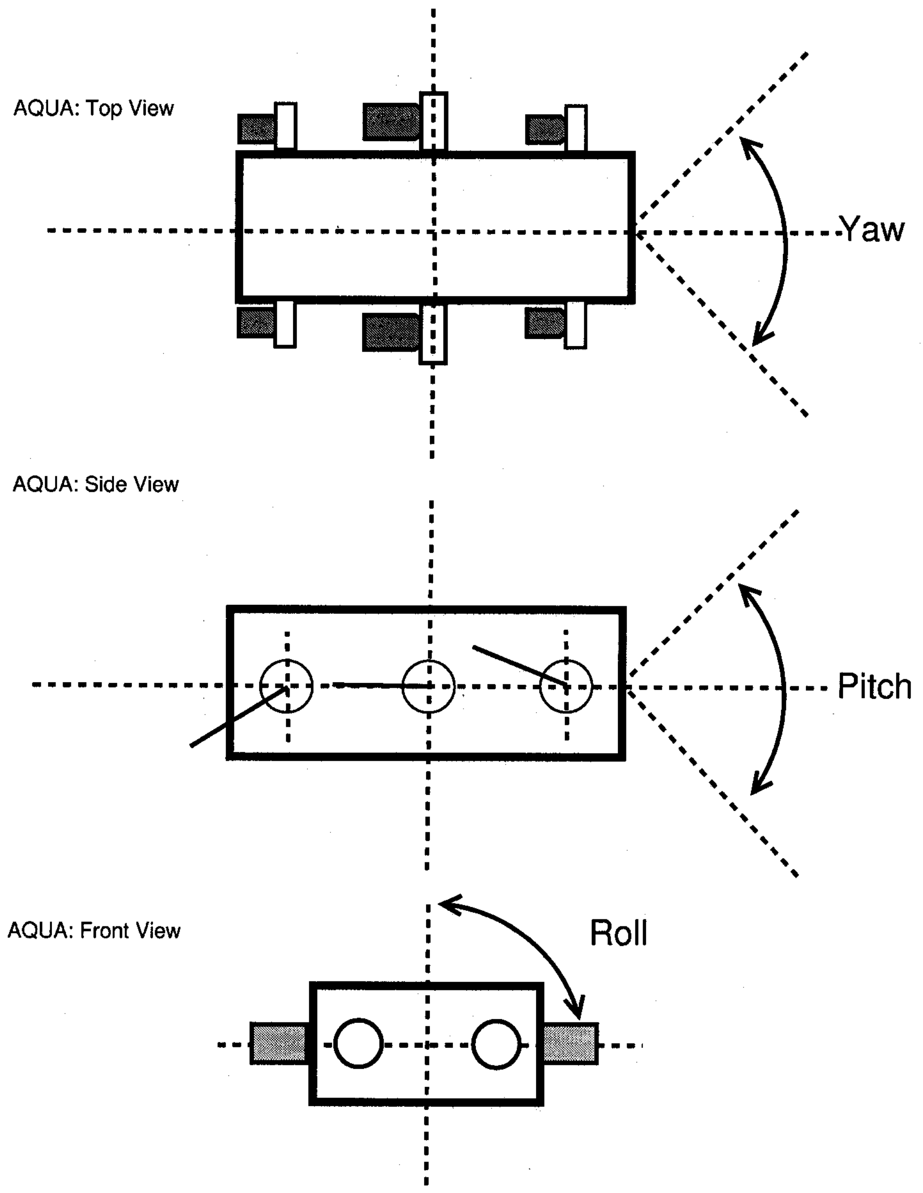


Figure 2–4: AQUA pitch-roll-yaw axes.

is an example of an experimental, high-maneuverability robot created by the Tow Tank Lab under the Sea Grant Project.

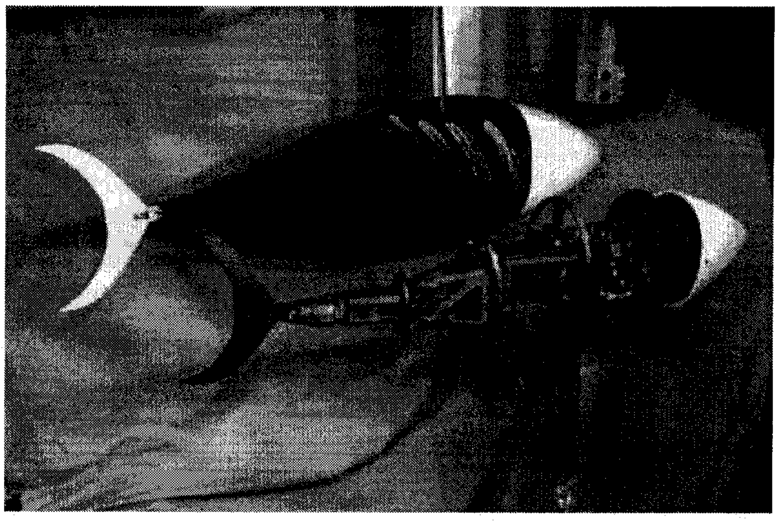


Figure 2–5: The MIT RoboTuna underwater robot. ©MIT RoboTuna project.

Vision in underwater environments is an attractive sensing platform, due to its passive and unobtrusive features; but it has been examined rarely due to the complications involved. One of the newer applications of vision sensors is in Simultaneous Localization and Mapping or SLAM problems, which is referred to as Visual-SLAM or VSLAM [9]. Underwater vehicles have a potential to be used for underwater terrain mapping and surveying, and applying VSLAM methods is an attractive approach. VSLAM has been applied to map underwater reef environments at the Great Barrier Reef in Australia [46]. Another notable use of vision for mapping is the inspection of the wreckage of HMS Titanic[11], where visual data was bolstered with information from an inertial sensor, thereby increasing reliability of the VSLAM process. Apart from mapping, vision-based vehicle navigation and station keeping have also been attempted. Hamel and Mahony[21] use an image-based approach for visual servoing an "eye-in-hand" robot configuration underwater. An approach to underwater station keeping using visual servoing can be found in [30], where feature point extraction from unmarked objects are used to maintain the robot's pose and station underwater.

2.5 Robot Software Architectures

Most robots in existence today are made up of complex hardware architectures, controlled by somewhat equally complex systems of software [7], that provide the robot with low- and high-level behaviors and commands. Traditionally, each robot or family of robots have been operated by a particular software system architecture, instead of having a generic architecture suitable for all robots. Two and three layers of software abstractions [15] are seen in robotics systems. The software system is dictated by the hardware architecture and the demands of real-time operation, which a generic architecture is unable to provide. This is evident in several robot architectures seen commercially or in academic and scientific projects. We discuss a few of the more prominent robot architectures in the subsections below.

2.5.1 Higher-level Architectures

One of the earliest works in robot software architecture was Brooks' Subsumption architecture [5] where multiple layers of control can coexist, with higher priority behaviors 'subsuming' or taking over lower priority tasks at different layers. A related architecture to Subsumption is Parker's ALLIANCE [36], a fault-tolerant architecture designed for multi-robot cooperation. The SAPHIRA architecture [26] is a client-server architecture aimed at achieving autonomous behavior and capable of interfacing with visual and speech sensors, mapping and task-based operation. Architectures like the CAMPOUT [35] (Control Architecture for Multi-robot Planetary Outposts) and CLARAty[45] (Coupled Layer Architecture for Robotic Autonomy) have been developed with multi-robot exploration being the primary concern. These two architectures were developed by NASA as a control system for their interplanetary rovers, which have different hardware architectures.

2.5.2 Dual-mode Architectures: Simulators and Operators

Certain software tools have been developed to work as both a simulation platform and a robot interface through robot software "drivers". These toolkits allow for software simulation of new concepts and algorithms, as well as driving real robots instead of the simulator just by replacing the 'back-end' with the proper robot hardware interface code. Tools like Player/Stage/Gazebo [17] and the Carnegie Mellon Navigation Toolkit (CARMEN) [33] fall under this category. The RoboDaemon package [10] provides an interface for point-and-click robot navigation for real and simulated robots, as well as an API for programming higher level behaviors for real robots. It is a part of the McGill Mobile Robotics Architecture (MMRA) package. The ORCA suite is one of the newer dual-mode software packages to appear in the field. Developed at the Royal Institute Technology in Sweden, University of Technology at Sydney and The Australian Centre for Field Robotics, ORCA[4] is an open-source framework for creating component based robotic systems, and is closely related to the Player/Stage/Gazebo architecture.

2.5.3 Low-Level Operational Software

Low level software architectures provide the lower levels of robot control, without sophisticated behaviors. The RoboDevel[38] library is an example of such a system. RoboDevel (formerly known as RHexLib) is the operating library for the RHex[1] family of hexapod robots. The library is written in C++ and has method calls for directly accessing the robot's actuators and other hardware. RoboDevel implements inter-process and inter-module communication by maintaining a central database or ”blackboard” where all modules store information that are considered public.

In the next chapter, we discuss the system architecture of the AQUA robot in some details. We present the RoboDevel software system and software control of AQUA in greater detail in Chapters 3 and 5.

CHAPTER 3
System Architecture

We describe in this chapter the overall hardware and software systems layout put in place for visual servoing with the AQUA robot. The hardware design of the AQUA robot is introduced, with emphasis on the electronics rather than the mechanical design. The software for visually guiding AQUA is split into two logically different sections: one for visual tracking of the target and generating robot commands, the other for taking those commands and transforming them to actuator commands. We describe both software systems in detail.

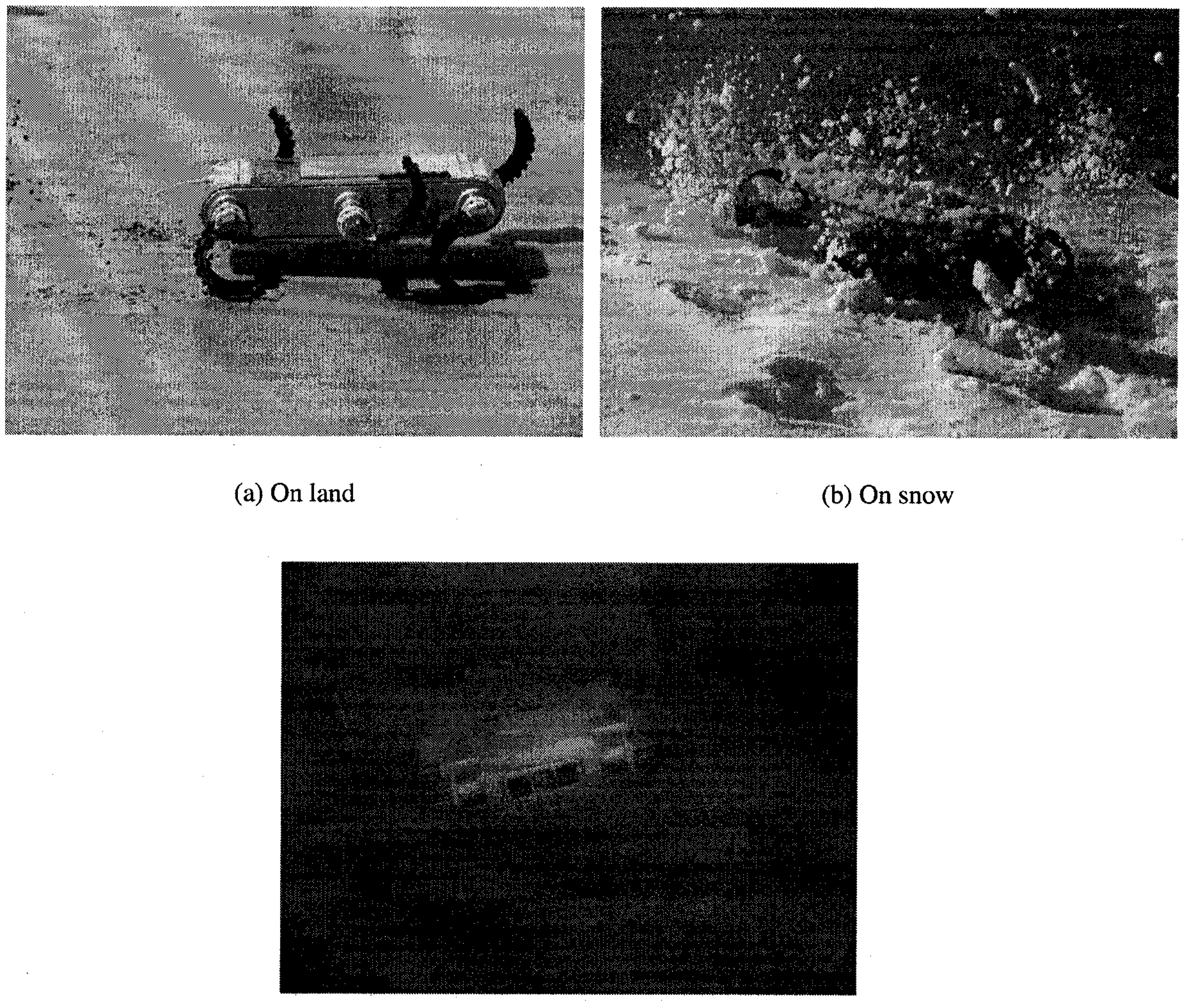
3.1 The AQUA Robot

The AQUA robot [16] is designed as an aquatic swimming robot that is capable of operating both on land as well as under water. A direct descendant of the RHex hexapod robot [1], AQUA was built with underwater applications in mind, one of which was monitoring of marine life (i.e. coral reef, fish population). The robot has a waterproof aluminum shell inside which the electronics and sensors are housed. Figure 3–1 shows the AQUA robot underwater, on land and on snow, demonstrating its ability to operate in different environments and different terrain conditions.

3.1.1 Hardware

Propulsion

The AQUA robot uses six legs or paddles to swim underwater or walk on the ground. These legs give the robot the ability to turn sideways (yaw), change depth (pitch) and rotate



(a) On land

(b) On snow

(c) Underwater

Figure 3–1: AQUA in different environments.

on its horizontal axis (roll). There is only one actuator per leg, significantly reducing power required for operating the legs. For our purposes, each leg has three main controllable parameters associated with it: leg *amplitude, offset and phase*. The *amplitude* parameter governs the distance the legs sweep along the spherical arch during each cycle. *Offset* dictates the relative starting orientation of the legs to each other at the beginning of the cycle. Direction of the leg motion is controlled by the *phase* parameters of each leg. The legs generate thrust by moving according to preset gaits. Gaits are a combination of leg parameters that generate a fixed motion for a fixed set of parameters. Depending on whether the robot is swimming or walking, there are several different table-driven gaits that can be used to drive the robot forward. Different gaits have different power consumption rates, and also effect the stability of the robot in different ways. For operation on land and in water, different sets of legs are usually used, although a new compliant design is being tested that can be used equally well in both environments.

Sensing

AQUA is primarily a submerged vision platform, with three cameras being the principal sensing devices on the robot. An Inertial Measurement Unit (IMU) has been installed on board for orientation and acceleration sensing. Two cameras are mounted in the front and one in the back. One of the front cameras is a IEEE1394 (*aka* FireWire) digital camera from Point Grey Research conforming to the Industrial and Instrumentation Digital Camera (IIDC) standard, and it interfaces with the vision processor for visual servoing. The other two cameras are analog, and provide streaming video for remote operator control. There are internal sensors for monitoring the current state of robot health; these include battery power and power consumption levels for the leg motors.

Power

AQUA is a self-sustaining robot, powered internally by two NiMH batteries. These batteries can power the robot continuously for over three and a half hours.

Computing

AQUA has two computers on board, one for gait control and the other for vision-related processing. Both computers are of the PC104/Plus form factor, due to the space restrictions inside the robot. These two computers, along with the additional port and interface circuit boards stacked on top of each other, connect via the ISA and PCI buses. As such, these will be referred to as the *control stack* and *vision stack* throughout this dissertation.

The control stack has a Pentium III processor, 256MB of RAM and a 256MB CompactFlash card for secondary storage. Due to its real-time requirements, It runs on the QNX real-time operating system (RTOS). The control stack is tasked with controlling robot motion by manipulating the leg actuators in real-time.

The vision stack is responsible for processing visual data. Currently it is being used for visual servoing only, but future goals are to use this processor for underwater stereo algorithms and VSLAM with AQUA. It is powered by a Intel Pentium-M processor with a maximum clock speed of 1.4 GHz. The board has 2MB of on-chip cache memory and 1GB of RAM, which contributes to faster vision processing. A 512MB CompactFlash card is being used as secondary storage. We are using a custom-built version of the Linux operating system on the vision stack. The servoing code executes under this environment. Taking advantage of advanced power management features of the Pentium-M processor, we designed the vision stack operating environment to be capable of scaling the

CPU clock frequency to preserve battery power. This particular model of the Pentium-M processor can be scaled from 1.4GHz down to 150MHz. During idle periods, the CPU slowly scales down to the lowest clock setting, but jumps to the highest speed instantly on-demand. A PC104/Plus FireWire interface board enables the vision stack to interface with the FireWire cameras.

Both the control and vision stacks have on-board serial and Ethernet ports. These ports are used for the IMU and communication between the stacks, respectively. Power to the boards are supplied using a custom-designed hardware controller board known as the RHIO card (RHex Input/Output).

Communication

There are several communication channels in the AQUA robot. Communication from the camera to the vision processor is over the FireWire bus, as already stated in the previous section. The vision stack communicates with the control stack via the Ethernet ports, utilizing the UDP protocol. Outputs from the two analog cameras, IMU readings. robot control and logged data are communicated from the robot to the operator platform on the surface over a fiber optic tether. The *Operator Control Unit* (OCU) is connected to this fiber and provides the operator with the visual data required for teleoperation.

A cut-away section of the robot with interior components are shown in Fig. 3–2. The hardware and software systems are explained in detail in the following two subsections.

3.1.2 Software

The task of visually guiding the AQUA robot is a two stage software process. The first stage is performed in the vision stack, where the target is tracked and robot pitch and yaw commands are generated. The second stage accepts these yaw and pitch commands

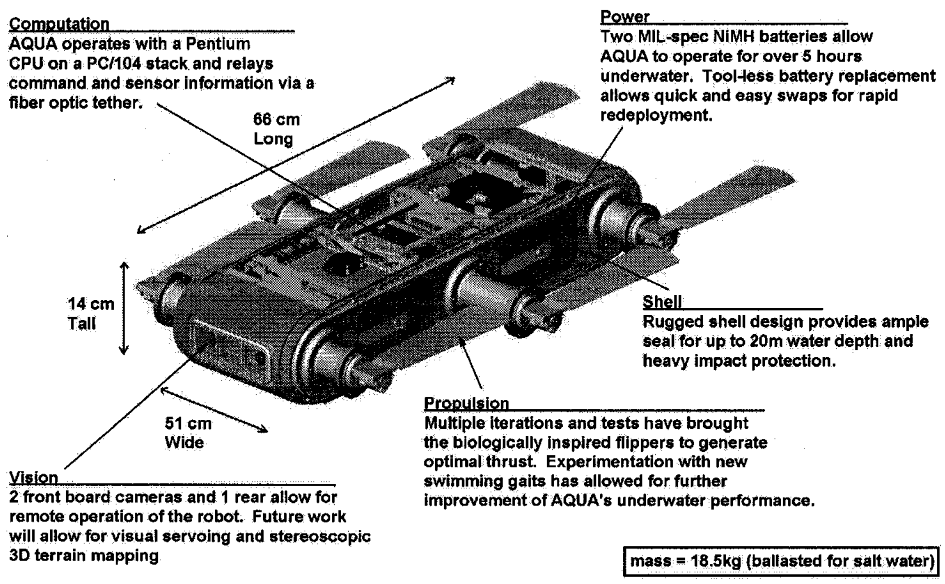

Computation
AQUA operates with a Pentium CPU on a PC/104 stack and relays command and sensor information via a fiber optic tether.
66 cm Long
Power
Two MIL-spec NiMH batteries allow AQUA to operate for over 5 hours underwater. Tool-less battery replacement allows quick and easy swaps for rapid redeployment.
14 cm Tall
Shell
Rugged shell design provides ample seal for up to 20m water depth and heavy impact protection.
51 cm Wide
Propulsion
Multiple iterations and tests have brought the biologically inspired flippers to generate optimal thrust. Experimentation with new swimming gaits has allowed for further improvement of AQUA's underwater performance.
Vision
2 front board cameras and 1 rear allow for remote operation of the robot. Future work will allow for visual servoing and stereoscopic 3D terrain mapping
mass = 18.5kg (ballasted for salt water)

Figure 3–2: AQUA hardware components.

and generates leg actuator commands that actually enables the robot to perform the requested maneuver. In both stages, the software is written in C++, with the emphasis on small footprint and fast-executing binaries suitable for an embedded system. The software system of these two stages are discussed in the following two sections.

Visual Servoing Software

The vision software based on an open-source vision library called VXL (Vision "something" Libraries)[1] . VXL is a suite of packages designed for creating efficient and fast programs for computer vision related applications. VXL includes libraries for numerical

[1] www.sourceforge.net/projects/vxl

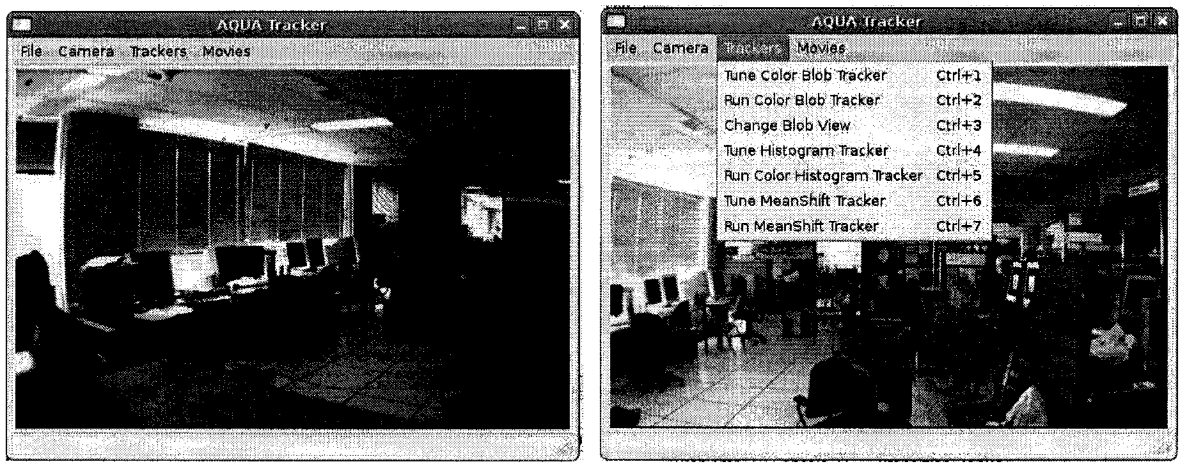
algorithms, image processing, coordinate systems, camera geometry, stereo, video manipulation, structure recovery from motion, probability modelling, GUI design, classification, robust estimation, feature tracking, topology, structure manipulation and 3d imaging, among others. VXL also provides system-independent toolkits for cross-compiler compatibility.

We looked at creating a modular, extensible software base so that enhancements and integration of advanced features in the future would be significantly easier. The object inheritance capabilities of the C++ programming language provided us with tools for achieving that goal. From a functional point of view, the code base is made up of visualization and user-interface code for development and testing of algorithms offline, off the robot. A subset of that code is compiled and installed into the robot before actual visual servoing runs. This model allows for shorter develop-test-deploy cycles. Screenshots of the tracking testbed running with full graphical user interface can be seen in Fig. 3–3.

More details about the visual servoing software and experimental setup will be given in chapters 5 and 6.

Control Software

Control of robot motion is supervised by code running in the control stack. In all robots derived from the original RHex, control software is written using a library called RHexLib (RHex Library). As the number of RHex derived robots increased, the RHexLib library grew into a package of robot-independent and robot-dependent code, with code specific to each robot having their separate space, unrelated to other robot codes. This package is known as the RoboDevel suite. AQUA control code has been derived from the


AQUA Tracker
File Camera Trackers Movies
AQUA Tracker
File Camera Trackers Movies
Tune Color Blob Tracker Ctrl+1
Run Color Blob Tracker Ctrl+2
Change Blob View Ctrl+3
Tune Histogram Tracker Ctrl+4
Run Color Histogram Tracker Ctrl+5
Tune MeanShift Tracker Ctrl+6
Run MeanShift Tracker Ctrl+7

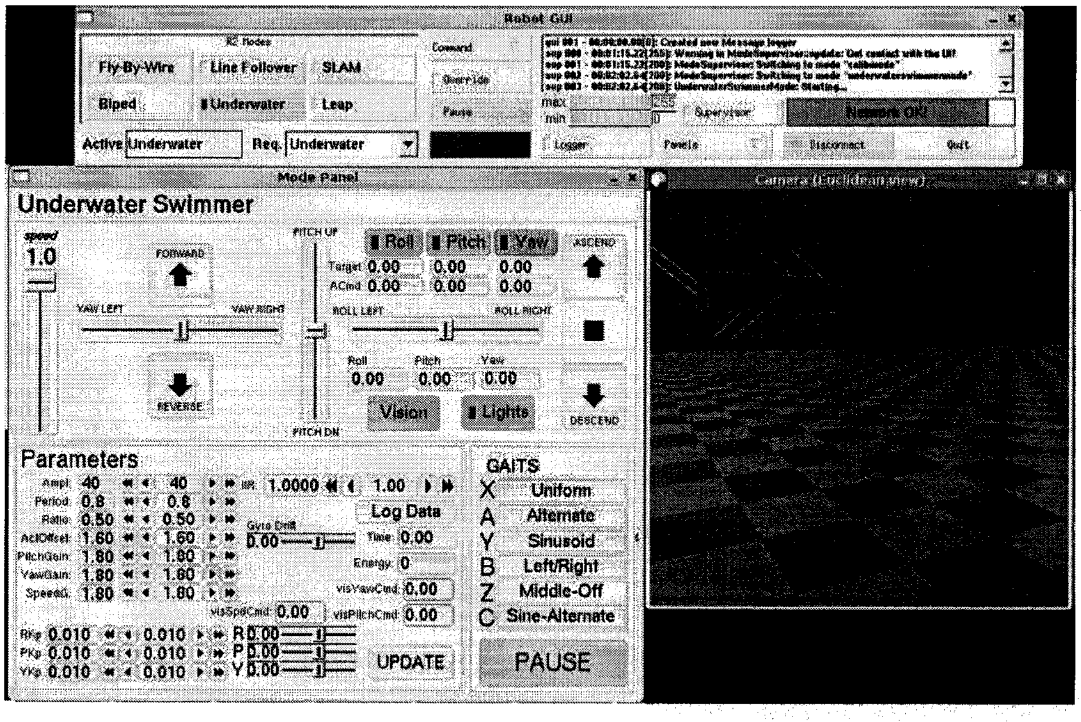
(a) Main Interface

(b) The Tracker Control Menu

Figure 3–3: Screenshots of the VGUITracker application.

RHex codebase, but a large number of enhancements as well as newer innovations have been made in the AQUA code that distinguish it from the original RHex software.

The structure of control code in AQUA is similar to a client-server architecture, with the robot running the RoboDevel **"supervisor"** code, and the robot control machine running the **"operator"** code. RoboDevel also comes with a simulator for RHex robot visualization, called SimSect. In case where the real robot is being operated, the supervisor communicates at a very low-level to the robot hardware; legs, inertial sensors and health monitors. The operator communicates with the supervisor using a point-to-point protocol over a serial link that runs through the fiber optic tether. A full-featured graphical user interface is available at the operator terminal for easy access to all the robot control parameters as well as health and system monitors. The control GUI can also be operated via a wireless gamepad to assist the robot driver. A screenshot of the GUI during a simulated underwater operation of the robot is shown in Fig. 3–4 below. Note the displays for visual servoing commands in the middle, the IMU readings and controls on top and the gait controls at the bottom right, among other things. The drawing of the robot to the right is an output from SimSect, via the Geomview[2] tool.



```
gui 001 - 00:00:00.00[0]: Created new Message logger
sup 000 - 00:01:15.22[255]: Warning in ModeSupervisor::update: Got contact with the UI!
sup 001 - 00:01:15.22[200]: ModeSupervisor: Switching to mode "calibmode"
sup 002 - 00:02:02.6[200]: ModeSupervisor: Switching to mode "underwaterswimmermode"
sup 003 - 00:02:02.6[200]: UnderwaterSwimmerMode: Starting...
```

max 255
min 0

Active: Underwater Req.: Underwater

speed 1.0

Target 0.00 0.00 0.00
ACmd 0.00 0.00 0.00

Roll 0.00 Pitch 0.00 Yaw 0.00

Ampl:	40	40
Period:	0.8	0.8
Ratio:	0.50	0.50
ActOffset:	1.60	1.60
PitchGain:	1.80	1.80
YawGain:	1.80	1.80
SpeedG:	1.80	1.80
RKp	0.010	0.010
PKp	0.010	0.010
YKp	0.010	0.010

IIR: 1.0000 1.00
Gyro Drift 0.00
Time: 0.00
Energy: 0
visYawCmd: 0.00
visSpdCmd: 0.00
visPitchCmd: 0.00
R 0.00
P 0.00
Y 0.00

X Uniform
A Alternate
Y Sinusoid
B Left/Right
Z Middle-Off
C Sine-Alternate

Figure 3–4: AQUA operator graphical user interface.

The robot supervisor and the visual servoing software communicate via the UDP protocol, using a custom communication class. Communication between the two software

[2] http://www.geomview.org

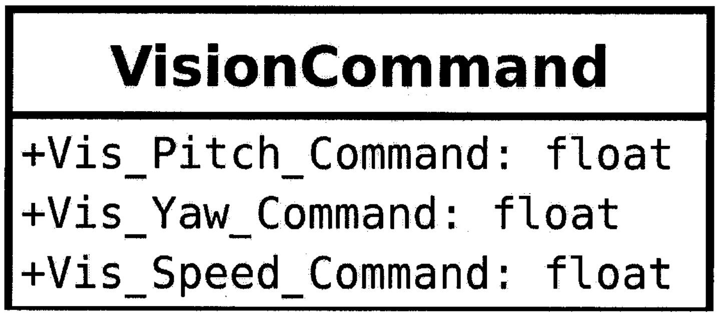


Figure 3–5: Vision command data structure.

modules are uni-directional, with the servoing software sending pitch, yaw and speed commands to the robot supervisor. The structure of the vision command packet is shown in Fig. 3–5.

We opted for the the UDP protocol, since for TCP connections, there is an added overhead of connection setup, because of its state-oriented nature. UDP, on the other hand, is stateless and provides no guarantee of packet delivery, but at a high rate of packet transmission over a short distance, there is virtually no packet loss. Our application is robust enough to remain stable in case a packet or two fails to reach the control stack. The vision stack command packets are sent at the rate each frame is processed from the camera. We currently use a rate of 15 frames/second from the FireWire camera.

The diagram in Fig. 3–6 shows a UML class diagram for the software components residing in the vision stack that performs tracking and robot command generation. The complete software architecture is demonstrated in Chapter 5, in Fig. 5–6.

3.1.3 The System Environment

For the vision stack we did not require hard real-time capabilities, hence a custom-tailored version of the Linux Operating System is used as the environment for the visual servoing code. We built the OS from the source, compiling all the binaries for the target PC104/Plus platform to optimize runtime performance and reduce size of the executables. Some salient features of this OS are listed below.

- **Storage**: The storage medium is a 512 megabyte CompactFlash card, with very little power requirement as compared to micro-size hard drives. The resistance to shock impact and vibration is also far greater for CompactFlash cards (2000G's for CF against 12G's for MicroDrives). The downside is the limited number of write cycles that can be performed on a CompactFlash card. To extend the working life of the CF card, we stored all the temporary system logs and cache files on RAM disks created during the Linux bootup. The entire storage space was partitioned into two sections; one read-only for the operating system of roughly 100 MB and the other around 400MB, for storing images and video from the FireWire camera.
- **Kernel** A monolithic kernel (version 2.6.11, latest at the time of the experiments) is used, with all device drivers and modules built right into the kernel, to reduce the latency in activating devices. This increased the kernel size, but reduction of module files and other unnecessary components more than compensated for that increase.
- **Startup** To ensure a fast and responsive system startup, we start a very small number of services (or daemons), The boot partition is mounted read-only, so we disable file system checks at the startup. All system logs and temporary files are stored in the memory, in temporary **RAM Disks** , which allows us to use a read-only boot partition.
- **Network** The vision stack uses Ethernet connection to communicate with the control stack, which is setup with a static IP address. We also have provisions for *Ethernet*

over FireWire, which enables the use of Ethernet protocol over a FireWire link. The vision stack also supports both SSH and Telnet connections for remote logins.

In the next chapter, we discuss the tracking algorithms we used in our problem. The theory behind each of these algorithms are explained, as well as the advantages and disadvantages of each method, as relevant to our application. The application of these methods in the AQUA visual servoing application is discussed in detail in Chapter 5.

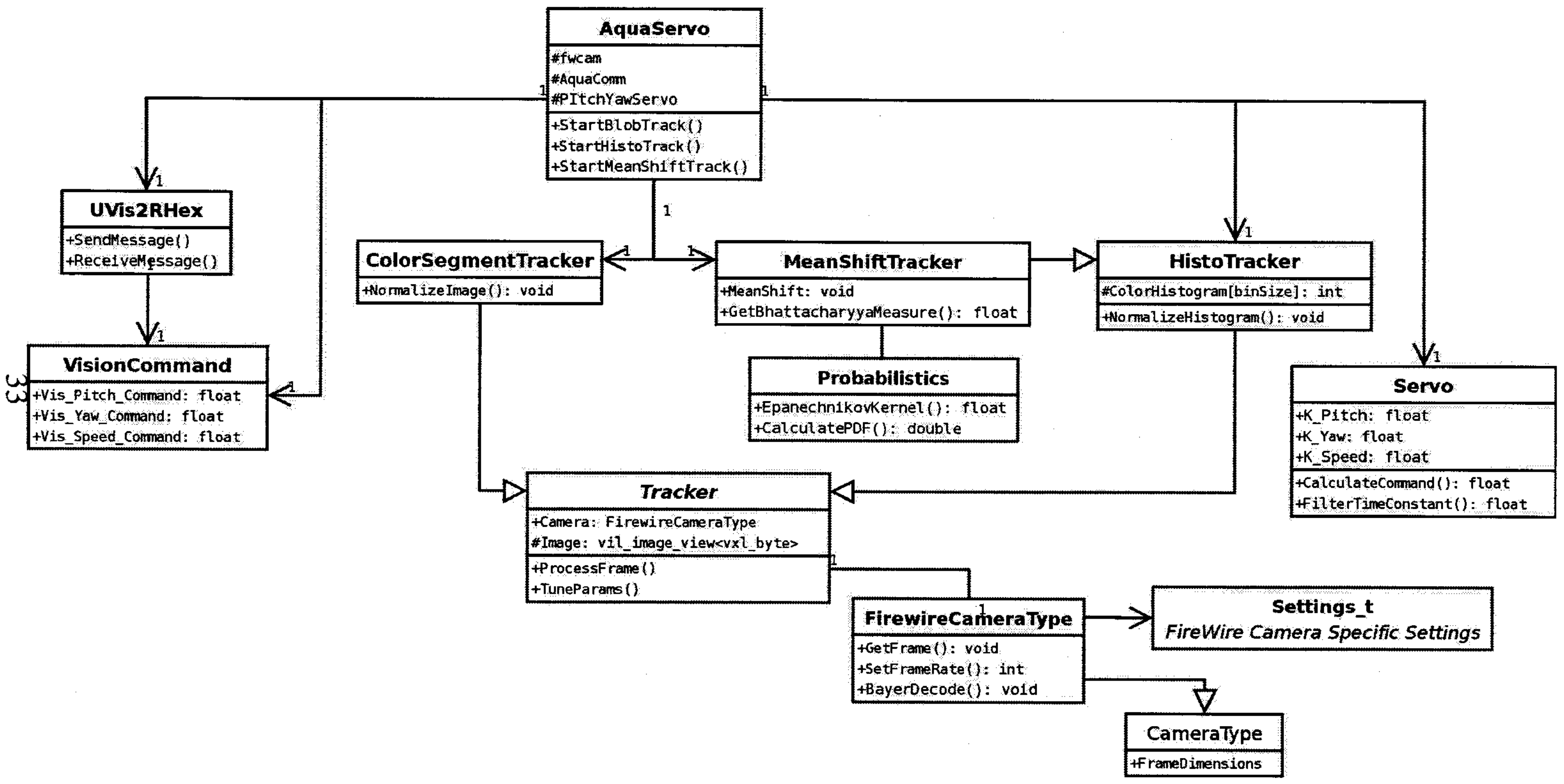


Figure 3–6: Visual servoing software structure.

CHAPTER 4
Visual Tracking and Control

We discuss the theory behind the tracking methods used in our work in this chapter. All our tracking algorithms are based on color cues; hence we focus on color space and impacts of lighting on color. The three tracking methods used have increasing computational cost and complexity. We justify the need for using three tracking methods and point out the advantages and disadvantages of each. The chapter concludes with a comparative discussion of the three approaches. Implementation details for these tracking algorithms are explained thoroughly in the following chapter. We begin, however, with a short discussion of the physics of light in underwater environment.

4.1 Underwater Light and Vision

In underwater environments, illumination depends on depth, refraction, scatter and absorption of the water medium. Of all the sunlight reaching the surface, only 18% reaches a depth of 18 meters, and 1% reaches 100 meters [41]. The most important reason for reduced visibility underwater is the small difference between refractive indices of the human eye and the water medium. On air, refraction index of light is almost close to 1, and that of the human eye is 1.38. This difference is sufficient to form images on the retina. In water, the refractive index is 1.34, reducing the difference greatly and forming images far beyond the retina. Divers tend to use masks or goggles to form a layer of air between the eye and the water outside to ensure visibility is not hampered due to this lack of refraction.

The three phenomenon effecting light underwater are briefly described below, and are illustrated in Fig. 4–1.

- **Refraction** : Refraction is the effect which causes light rays to bend while passing from one medium to another. This effect happens when light passes from water to the camera or human eye underwater. Refraction can change the perception of distance of objects underwater, making objects appear at three-fourths the distance than they actually are. At greater distance this effect might be reversed; causing objects to appear further away than they are. The more turbid the water, the less the distance at which this effect or over- or underestimation of distance can happen. Again, when light passes from air into the water through the surface, the water causes rays to enter at various angles due to waves, and also through the water column due to variable levels of salinity.
- **Scatter** : Scatter occurs when individual photons of light are deflected or diverted when they encounter suspended particles in the water. Although scattering also occurs in air, it is of much greater concern under water because light is diffused and scattered not only by the water molecules themselves, but also by all kinds of particulate matter held in suspension in the water, and by transparent biological organisms. Normally, scatter interferes with vision because it reduces the contrast between the object and its background, which is why vision is so much more restricted in water than in air; for the same reason even large objects can be invisible at short viewing distances. Even more problematic is the fact that this scattering can be wavelength dependent and non-uniform, affecting transmission of color hues. In addition, acuity or perception of small details is generally much poorer in water than in air, despite

the fact that the optical image of an object under water is magnified by refraction. The deterioration increases greatly with the distance the light travels through the water, largely because the image-forming light is further interfered with as it passes through the nearly transparent bodies of the biomass, which is composed of organisms ranging from bacteria to jellyfish.

- **Absorption** : Light is absorbed as it passes through the water, and much of it is lost in the process. In addition, the spectral components of light, the wavelengths that give rise to our perception of color, are differentially absorbed. Transmission of light through air does not appreciably change its spectral composition, but transmitting light through water, even through the clearest water, does, and this can change the resulting color appearance beyond recognition. In clearest water, long wavelength or red light is lost first, being absorbed at relatively shallow depths. Orange is filtered out next, followed by yellow, green, and then blue. Other waters, particularly coastal waters, contain silt, decomposing plant and animal material, and plankton and a variety of possible pollutants, which add their specific absorptions to that of the water.

4.2 The Visible Color Space

Using color features in visual tracking is an attractive option because of its simplicity and robustness under partial occlusion, depth and scale changes. Tracking color cues helps one avoid using complicated and computationally expensive feature trackers that may well be infeasible in real-time applications. In spite of the apparent advantages of color tracking, there exists some significant problems that need to be addressed to design a robust and accurate color tracker. The biggest problem existing with color cues is color

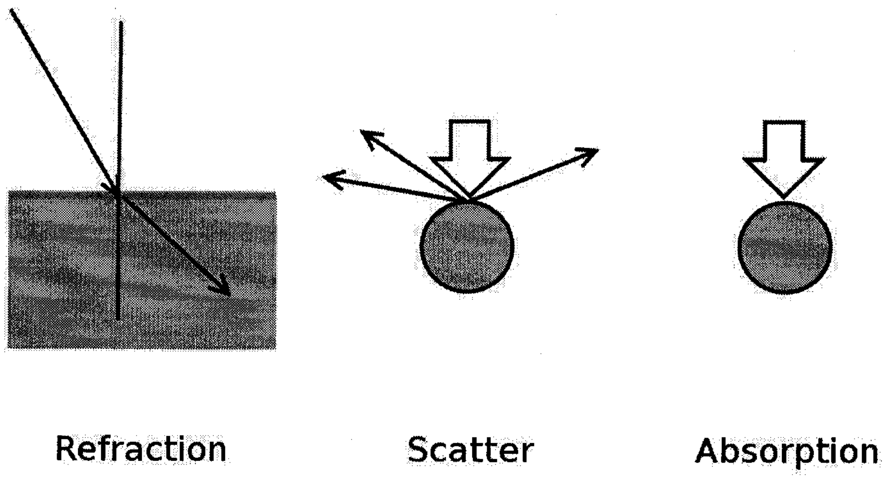


Figure 4–1: Phenomena effecting light underwater.

constancy [12]. Color constancy is defined as the removal of color bias due to effect of illumination. Issues like shadows, change in illumination and camera characteristics effect the phenomenon of color constancy. Keeping in mind the real-time performance demands from the tracker, we seek a robust and efficient representation of the object colors, resulting in faster and accurate computation.

In the following subsection, we first describe the RGB color space, which is a basic color space widely used to represent color images in image processing systems. We show the properties of the RGB space and the shortcomings, which influenced our decision to move to the more robust HSV color space for our tracking applications. Both color spaces are presented briefly, as a precursor to understanding the application of the three tracking algorithms we used in AQUA.

4.2.1 RGB Color Space

The Red-Green-Blue or the RGB color space represents the basic approach in representing color digitally. The RGB space uses a Cartesian coordinate system and forms a unit cube as shown in Fig. 4–2.

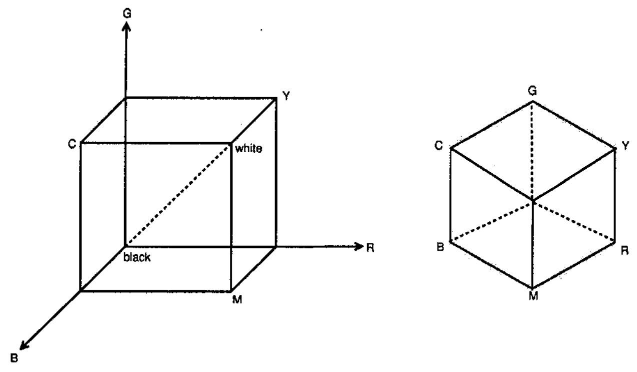


Figure 4–2: The RGB color space.

Each corner of the cube lying on an axis represents the point where the color represented by the axis is maximum, with other colors absent. The origin represents black, as all color amounts are zero. The diagonal emanating from the origin to the top-right corner of the cube (representing white) is the locus of points with equal amounts of each color. This is also referred to as the *gray diagonal.* Viewed from top, with the white-corner in the center, the cube on the left can be seen as a two-dimensional equilateral hexagon, with the white-corner overlaying the black.

4.2.2 HSV Color Space

The Hue-Saturation-Value or HSV model was suggested to capture the artistic ideas of hue, tint, shade and tone. Also referred to as the hexcone model, as shown in Fig. 4–3,the HSV model uses hue, saturation and value or brightness as the three dimensions for describing color, instead of RGB values.

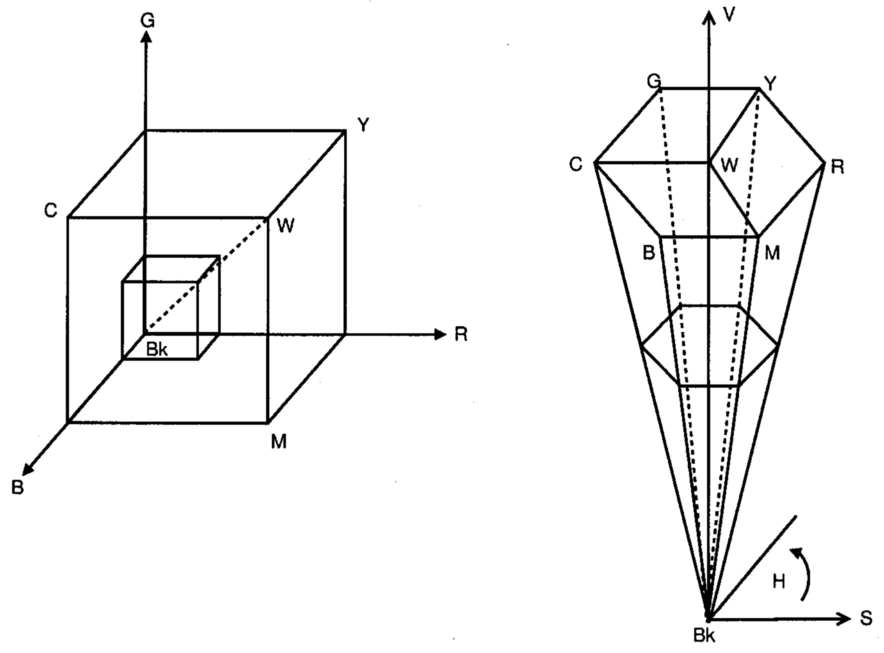


Figure 4–3: The HSV color space.

Briefly, hue is the dimension on which the principal color points lie. Saturation measures the distance of a color point from the white or gray value, also known as the *achromatic*. Value, on the other hand, measures the distance of color from the color black.

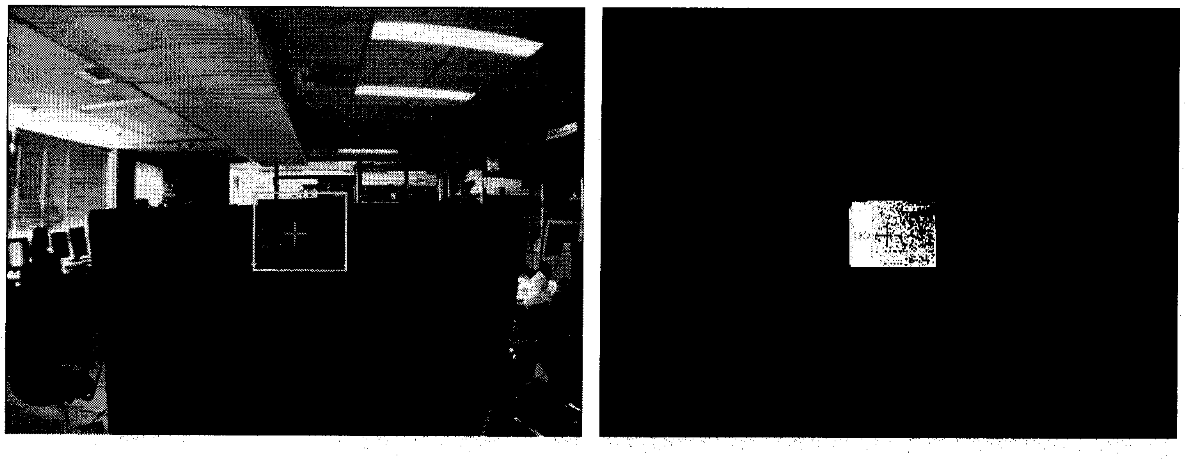
The relationship between the RGB model and the HSV model can be seen from figures 4–2 and 4–3. The orthogonal projection of the top surface of the RGB cube along the gray diagonal from white to blue corresponds to a plane in the HSV hexcone with constant V.

4.3 Color Blob Tracking

The simplest approach to color based tracking is using a segmentation algorithm to detect objects of interest using their color features. The output of the segmentation algorithm is (possibly disconnected) regions in a binary image that match the color properties being tracked. These regions are termed 'blobs', and hence the approach is known as color blob tracking. We attempt to form these blobs through a *thresholding* process. By thresholding, we refer to the operation where pixels are turned 'on' if and only if their color values fall within a certain range and turned 'off' otherwise.

The basic color blob tracker is a straightforward algorithm. The tracker is initialized with the target's color properties; in case of the RGB space, the tracker has to be aware of the red, green and blue color values of the tracked object. Next, sequential scanning is performed on the image, pixel-by-pixel. The pixel falling within the threshold of the color values of the target are turned on in the output image, and other pixels are turned off. Figure 4–4 below shows the segmentation output of tracking a red object. The target is framed by a yellow rectangle for clarity.

The tracker was tuned beforehand to the red rectangular target in Fig. 4–4(a). The segmentation produced the image in Fig. 4–4(b). The tracking algorithm detects this blob in the binary image in every frame, and calculates its centroid. This centroid is taken as the new target location. This process iterates over every frame, and the target is localized in the image frame.



(a) Tracking red (b) Segmented output showing the blob

Figure 4–4: A color blob tracker tracking a red-colored object.

The obvious downside to using a naive color blob tracker as explained above, is the presence of duplicate targets. For example, in Fig. 4–4(a) above, if any other red colored object appears in the scene, the segmentation process will generate another blob for that object. This second blob will effect the calculation of the center of mass for the tracker; the effect will be more prominent if the two generated blobs are disconnected; *i.e.* further away in the image frame. Therefore, the tracker works only accurately when there are no similarly-colored object in the camera's field-of-view.

Several approaches to address this problem have been suggested. One way to avoid tracker confusion in the presence of duplicate objects is to use shape cues to identify the proper object. This, of course works only when the 'false' target is of a different shape from the 'real' one. Also, incorporating shape detection is a computationally expensive process, and shape trackers can be confused by lighting variations, and lock on the wrong target.

Another way of making a a blob tracker more robust is to use a statistical approach to choose the proper blob to track, for example by using Monte-Carlo based methods like the Particle Filter. Particle filters have been extensively used in computer vision and tracking applications, first introduced as the Condensation algorithm by Isard and Blake [23]. The Condensation algorithm works well with contours and blob-trackers, but it is an iterative process, which would increase computational load significantly.

In the following subsection we look at a more robust approach than the naive color blob tracker, which applies color *distribution* matching instead of a single color range.

4.4 Color Histogram-based Tracking

A color histogram represents the distribution of color in an image or a region of an image. Color histograms are useful for characterizing the color content of a given image or image sub-window, and have been applied for image retrieval applications as well as video indexing and lookup. Formally, a *histogram* h_i is a mapping from a d-dimensional integer vector i to the set of non-negative reals. These vectors can be thought of as 'bins'; each bin represent the center of a region in a fixed partitioning of the underlying feature space. For a grayscale image, the histogram is one dimensional, *i.e* $d = 1$. Similarly for a color image represented using the RGB space, the histogram would be 3 dimensional. In both cases, the set of possible color values for each dimension is split up into N equally-spaced partitions. h_i contains the number of pixels that has color values which fall between the interval specified by the index i. Computationally, the color histogram is formed by discretizing the colors within an image and counting the number of pixels of each color.

The size (and number) of the bins and the range of values the histogram counts are important parameters since these control the effectiveness of the histogram in representing the color distribution of the underlying space. A *coarse histogram* has lesser number of bins than a *fine histogram.* Coarse histograms are not a good choice when the underlying image has a multitude of hues. On the other hand, for an image with a few colors, a fine histogram would be an over-representation, and would also lead to waste of storage as most of the bins would be empty.

The color histogram tracker works as follows. First, a histogram of the target to be tracked is created. This histogram is stored as the *target model histogram.* This is the preprocessing stage. During the tracking stage, every incoming frame from the camera is divided into rectangular regions and their histograms are calculated. The similarities between the new *candidate* histogram and the target model histogram is calculated following one of several possible distance measures (to be discussed below). The subwindow with the highest match is chosen as the probable subwindow containing the target. The pattern of scanning the image for the target can be done sequentially,or in a spiral pattern starting from the location of target found in the previous frame. Depending on the application, the size and shape of the subwindow can also be made to change dynamically, although that makes the tracker computationally slightly expensive.

The following sequences of images in Fig. 4–6 shows the operation of a histogram tracker. The target to be tracked is the show in Fig. 4–6(a). A one-dimensional color histogram of the target is shown in Fig. 4–6(d).

Figure 4–5 shows the tracking stage, with fixed size subwindows and sequential region search. The window where the tracker detects the target is shown in Fig. 4–6(b) and its histogram is in Fig. 4–6(e). A subwindow not containing the target and its color

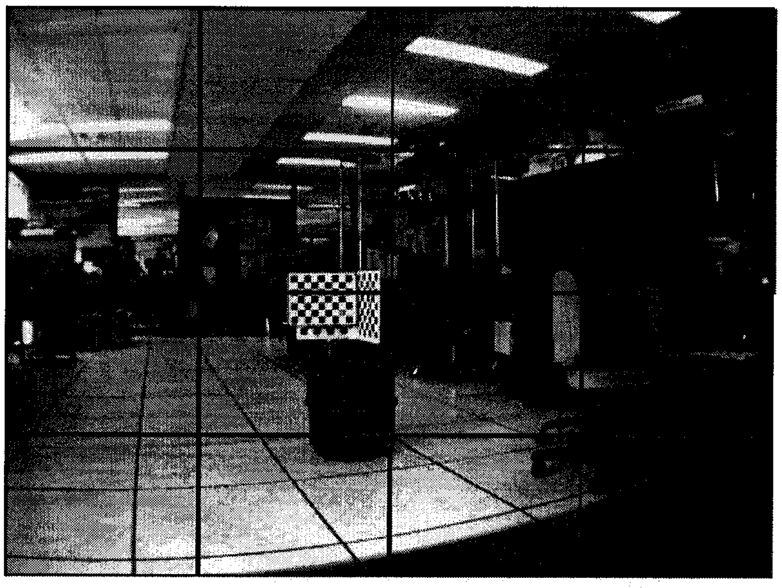
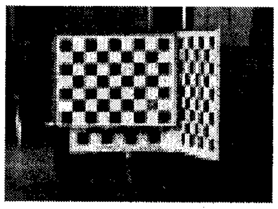


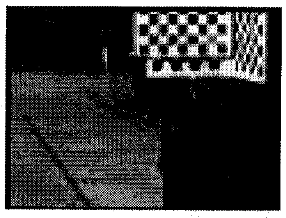
Figure 4–5: Rectangular search windows.

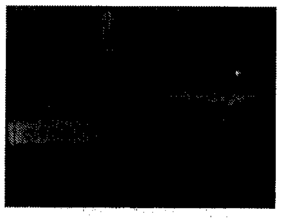
histogram are shown in Figures 4–6(c) and 4–6(f) respectively, for comparison with the target histogram shown in Fig. 4–6(d).

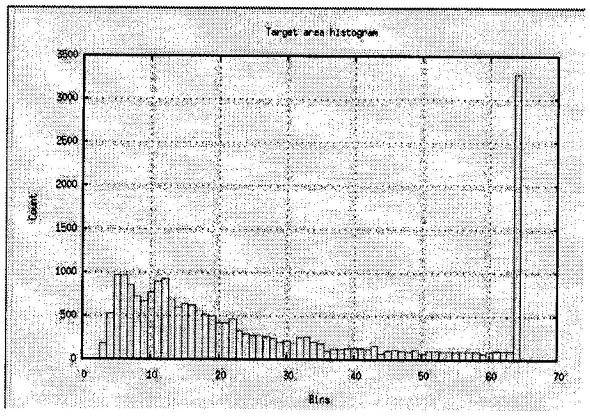
4.4.1 Measuring Similarity between histograms

At the heart of the histogram-based tracker lies the similarity measurement metric for comparing two histograms. The similarity measure is a function of the two histograms that returns a scalar value indicating the amount of similarity between the two histograms. Based on the way the bins are compared, these measures are of two fundamental types. One approach is to only compare corresponding bins in the two histograms, in a 'bin-by-bin' fashion. The other method does not limit the comparison between similar bins in the two histograms, but extends the comparison in a 'cross-bin' approach. We discuss a few of the measures available among many; the ones discussed have been used on board the AQUA robot for visual tracking with color histograms. One important point to note

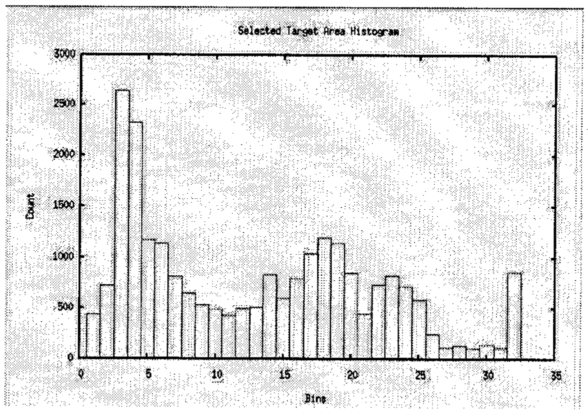

(a) A Target region


(b) Selected target region

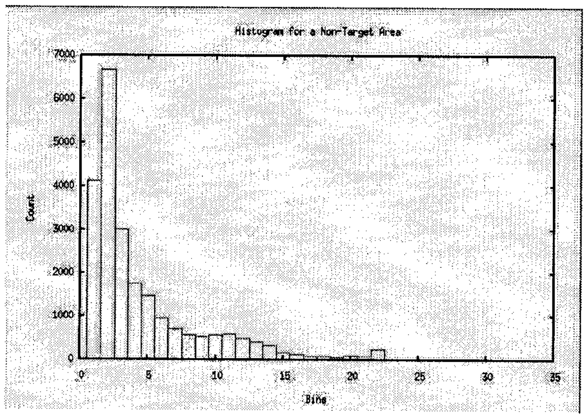

(c) Region without target



(d) Target region histogram



(e) Selected target region histogram



(f) Non-target region histogram

Figure 4–6: Comparison between histograms of different subwindows in an input frame.

is among the similarity measures discussed, there is not one which is clearly superior; instead, we stress the fact that the selection of the measure is very much application dependant.

The following subsections discuss the issue of measuring similarities between two histograms H and K, both having the same number of bins, N. These measurements only compare the corresponding bins of both histograms; *i.e.* they compare h_i and k_j for $i = j$, where h_i and k_j are the i-th bins of histograms H and K respectively..

Histogram Intersection Measure

The *histogram intersection similarity measurement* is calculated using the following formula:

$$d_{\cap}(H, K) = 1 - \frac{\sum_i min(h_i, k_i)}{\sum_i k_i} \tag{4.1}$$

This measurement has been proven useful in comparing histograms of different sizes [37].

The χ^2 (Chi-Squared) Measure

The χ^2 (chi-squared) metric is a measurement of the probability that one distribution was drawn from the other. The χ^2 measure is calculated by:

$$d_{\chi^2}(H, K) = 1 - \sum_i \frac{(h_i - m_i)^2}{m_i} \tag{4.2}$$

where,

$$m_i = \frac{h_i + k_i}{2}$$

The χ^2 metric does not permit the data in the underlying distributions to be percentages; they must be raw data. Also, the measured values have to be independent and observed frequencies must not be too small.

The Bhattacharyya Distance Measure

The Bhattacharyya coefficient has a direct geometric interpretation with respect to two distributions; for two m-dimensional unit vectors p and q, it is equal to the cosine of the angle between them. The Bhattacharyya distance between two histograms can be found using the following expression:

$$\rho_{Bhattacharyya}(H, K) = \sum_{i=1}^{m} \sqrt{k_i h_i} \tag{4.3}$$

Jeffrey's Divergence

Jeffrey's Divergence has been derived from the Kullback-Leibler (K-L) divergence. The KL divergence measure is an information theoretic measure that can be interpreted as the inefficiency of transforming one distribution to the other using a code book. The KL measure, however is sensitive to quantization effects in the histogram computation (*i.e* bin size). Jeffrey's divergence is an empirically derived divergence that is numerically stable, insensitive to histogram binning and also robust in the presence of noise. The Jeffrey's divergence measure of similarity is calculated as follows:

$$\rho_J(H, K) = 1 - \sum_{u=1}^{m} (h_i log\frac{h_i}{m_i} + k_i log\frac{k_i}{m_i}) \tag{4.4}$$

where

$$m_i = \frac{h_i + k_i}{2} \tag{4.5}$$

4.5 Mean-shift Tracking

Mean-shift tracking performs visual tracking by attempting to maximize the correlation between two statistical distributions. The correlation between the two distributions is expressed as a measurement derived from the Bhattacharyya coefficient described in the preceding section. Mean-shift trackers have been used to track objects based on color or texture, by building a statistical distribution of the feature being tracked. We introduce the basic concepts behind mean-shift tracking in the next paragraphs.

For any density function, the mean $\bar{x}$ of a set of samples x tend to be biased towards a local mode, *i.e.* a local maxima. The **mean-shift** vector, $\bar{x} - x$ points towards this local maxima. Based on this mean-shift property, tracking is performed as described in the following steps:

I A probability distribution model of the target T is built, based on color, texture or any other feature. We call this model p. This model is overlayed with an isotropic kernel with a convex and monotonically decreasing kernel profile. This step assigns weights to each pixel in the target region, with center pixels having more weights than those on the periphery. This distribution of weights increases robustness against occlusion, since the pixels on the boundary do not have large weights assigned.

II Starting from the previous location k_0 of the target, a new model of the target is generated at the same location. This candidate model at location k_0 we denote as $q(k_0)$.

III The weights, as described in step I, are calculated.as described in step I

IV The new location k_1 of the target is calculated recursively, using the mean-shift procedure.

V If the shift amount between the new and old locations is smaller than an arbitrary small constant ϵ, the algorithm stops. Otherwise, we assign $k_0 \leftarrow k_1$ and iterate from step III.

4.6 A Comparison

Each of the three trackers in the system have desirable features, as well as some potentials caveats, as we will discuss in detail in Chapter 6. The color blob tracker is inherently simple, easy to implement, and has a running time directly proportional to the frame size. This running time emerges from the need to raster scan the image pixel-by-pixel to check for color matches with the target object. For a high frame-rate and a large frame size, the computational costs increase significantly, although we have used medium resolution (*i.e.* 640×480) size frames at 15 frames/second to keep the computational overhead low. Intensity normalization of the input frame also reduces the effect of lighting changes. The appearance of a similar-colored object in the camera field of view can confuse the color blob tracker.

The color histogram tracker, unlike the blob tracker, is suitable for tracking multi-hued objects. We use fixed bin sizes for histograms, and compare the target region with fixed size windowed regions in the image. The advantage of the histogram tracker over the color blob tracker is in the ability to track a variety of colored objects, both multi and single colored. The algorithm also has a runtime proportional to the frame size.

We use normalized histograms for our application, which reduces the effect of brightness changes on color matching. The tracker is more robust to tracking in the presence of multiple objects, since two objects has to have the same distribution of color to confuse the tracker. The computational cost, although linear, is significantly more than that of the

Table 4–1: Tracker comparison table.

Tracker	Computational Cost	Target Color Properties	Effect of Lighting Variations
Color Blob	Low	Single color only	Moderate
Color Histogram	Moderate	Single and Wide range of colors	Moderate
Mean Shift	High	Single and Wide range of colors	Low

color blob tracker. This cost is incurred due to the computation of color histograms of every rectangular region we scan for the occurrence of the object.

The mean-shift tracker is the most computationally expensive of the three trackers. It is also the most robust. The mean-shift vector tracks changes in underlying color distribution and follows the target in successive frames. The process is localized, in a small region of the image, and no raster scan of the image is performed. However, the color distribution of the target and candidate regions have to be computed, as is their probability density functions. These computations increase the overall running time for the algorithm. On the other hand, the mean-shift tracker is more robust to changes in lighting and appearance of duplicate objects in the frame. We have found this tracker to work well under partial occlusion and cluttered environments as well. We discuss the performances of each in detail in Chapter 6.

Table 4–1 presents a summary of the three tracking algorithms, comparing the degree of main desirable and undesirable features.

4.7 Preliminary Control Theory

This section introduces several key concepts relating to Control Theory. Control theory deals with the behavior of dynamical systems over time, *i.e.* systems that change properties over time. When one or more output variables of a system requires to be set at a certain value, the Controller attempts to manipulate the input of the system to adjust the output to the desired value. Visual servoing of the AQUA robot qualifies the system as a dynamic system, since continuous visual tracking of a target attempts to control the motion of the robot. We present some fundamental definitions to aid understanding of the control process of the AQUA visual servoing system. A couple of the following sections briefly discuss filters and PID control, with relevance to the current work.

4.7.1 Definitions

We present some definitions to begin the chapter. These are common terms in Control Theory literature, and are only presented here to familiarize the reader with the basic concepts.

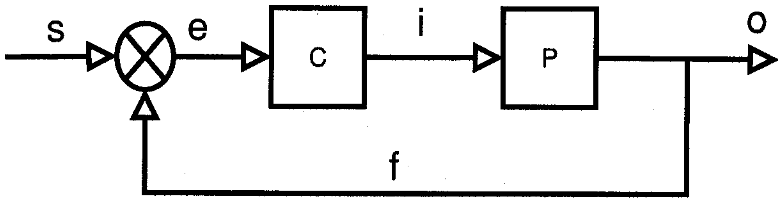


Figure 4–7: A simple feedback loop.

- The **Reference Variable** is the final output variable of the system.
- The **Plant** is the system being controlled.
- The **Controller** is responsible for manipulating the Plant.

- The **Setpoint** is the target value of the reference variable which the controller attempts to maintain at all times.
- In **Open-Loop Control**, the Controller has no feedback from the Plant as it attempts to control the reference variables. Open-loop systems have no sensitivity to the dynamics of the system being controlled.
- In **Closed-Loop Control** , the problem of the Open-loop control described above is reduced by introducing feedback from the Plant output. The Controller receives as input both the reference value s and the output feedback f. It measures the difference between the reference and current output as the error e and changes the input i to the Plant accordingly, as shown in Fig. 4–7.
- **Stability** refers to the fact that for any bounded (*i.e* finite) input to the Plant, the output of the Plant is also bounded. Stability is essential for a well-controlled dynamical system. This is also referred to as the *Bounded-Input-Bounded-Output* or *BIBO* stability.

4.7.2 Proportional-Integral-Derivative Control

A *Proportional-Integral-Derivative* (or *PID*) Controller is a standard closed-loop control that tries to control one or more reference variable of a plant by "sensing" the output at a given time and adjusting the plant input accordingly. The error signal at time t is the difference between the setpoint and the output of the system at time t. The controller treats the error signal with three different multiplication constants or gains, as described below, justifies its name.

a The **Proportional Gain,** K_P is a negative term, which is multiplied with the error signal and the result is sent to the output. The proportional gain dictates the band over which the output of the controller is proportional to the error signal. This gain is responsible for making the controller react to the current value of the error signal.

b The **Integral Gain,** K_I is multiplied with the integral of the error signal over a (usually short) period of time, and added to the proportional output. K_I denotes the steady state error of the system, and attempts to remove errors that have persisted in the system over a period of time.

c The **Derivative Gain,** K_D is used to adjust the response of the controller to the changes in the system. The rate of change of the error signal (*i.e.* first derivative) is multiplied with K_D and added to the sum of the two outputs above. The larger the derivative gain, the faster the controller responds to changes in the plant.

Equation 4.6 below shows the form of a PID controller.

$$Output = K_P\overline{\epsilon_t} + K_I\int\overline{\epsilon_t}dt + K_D\frac{\partial}{\partial t}\overline{\epsilon_t} \tag{4.6}$$

Here, $\overline{\epsilon_t}$ is the time-averaged error signal value at time t.

4.7.3 PID Controller Tuning

Tuning of a PID controller refers to finding the values for the proportional, integral and differential gains. Depending on whether the system can be taken offline or not, there are several standard approaches to tuning a PID control loop. In one approach, the system taken offline and is subjected to *step changes* in input and the output is measured as a function of time. The response of the system is used to find the optimal value of the parameters. If, on the other hand, the system cannot be taken offline, the values of K_I and K_D can be set to zero and the value of K_P is increased until the output loop oscillates. At this point, K_P is fixed and then K_I is adjusted to stop or substantially reduce the oscillation. Finally, the derivative gain K_D is used to reduce the response time of the system.

4.7.4 PID Controller Issues

In theory, a PID controller is robust and easy to tune for controlling a dynamical system. In practice, the algorithm suffers from a few drawbacks, which arise from the real-life imperfections of the systems being controlled, the model and environmental factors. One common problem a PID loop suffers from is the delay in getting the output of the system to ramp up to the desired setpoint. This is referred to as the *Integral Windup* phenomenon. Using a large initial value for the differential gain or preloading the system with a certain output value sometimes can reduce this effect.

A frequently changing controller output is not always desirable, as it might lead to mechanical wear or unstable vehicle control. A *deadband* value is introduced in PID controllers to prevent the controller from responding to small changes in the plant output. The deadband defines the range of change in output for which the controller will not respond at all. In effect, this allows the controller to respond to major changes in system output.

The differential gain can cause the controller to change its output by a large amount in the presence of a small amount of noise in the system. Passing the measurements through a low-pass filter helps reduce the effect of noise in measured values. In some systems, the differential gain is not used at all, resulting in a *PI* control loop.

4.7.5 Filters

Frequency domain filters are used in signal processing to block signals of certain frequencies from passing through. A certain type of filter allows only a certain band of signals to pass, *e.g.* low-pass filters only allow components with low frequency to pass through, blocking high frequency signals. Filters can be analog or digital, depending on the application domain. Digital filters can be used to implement any mathematical filtering application that can be expressed as an algorithm. We constrain our focus to digital filters in this section.

4.7.6 Infinite-Impulse Response Filters

Infinite-Impulse Response filters are digital counterparts of analog filters. When subjected to an impulse function, IIR filters produce a response which is non-zero over an infinite time period. IIR filters use feedback from the output to create a form of *recursive filtering* that results in an unending impulse response. The response is either exponentially decaying, growing or sinusoidal.

The next Chapter introduces the theory behind visual servoing and describes the system implementation specifics in details. The tracker and controller implementations as well as the systems environment components are explained in greater detail.

CHAPTER 5
Visual Servoing System Implementation

This chapter describes the implementation details of the visual servoing system in AQUA. We discuss in detail the tracker implementations, control loop design and the entire system with the communication between the vision and control stacks. The Linux operating environment for the vision code was built from the scratch for faster execution times and small memory requirements. Salient features of the Linux environment are described in this chapter as well.

5.1 Visual Servoing

Vision is an excellent sensing medium for robotic applications due to its passive property and very little power requirement. In a real time robotic system, vision can be applied to provide a closed-loop control of the robot manipulator or the entire robot as a whole. In context to our work, *Visual Servoing* refers to the task of controlling the robot's pose by using feedback from a vision sensor. Visual servoing has been used extensively in the manufacturing industry [31, 2] as well as vision-assisted vehicle control. Hutchinson, Hager and Corke [22] has an extensive discussion about the theories behind visual servoing and the classification of visual servoing methods. Based on whether the tracking is performed in image or task space, and the hierarchy of control mechanism, visual servoing approaches can be classified in four major categories. Figure 5–1 shows the major classes of visual servoing approaches as mentioned in [22].

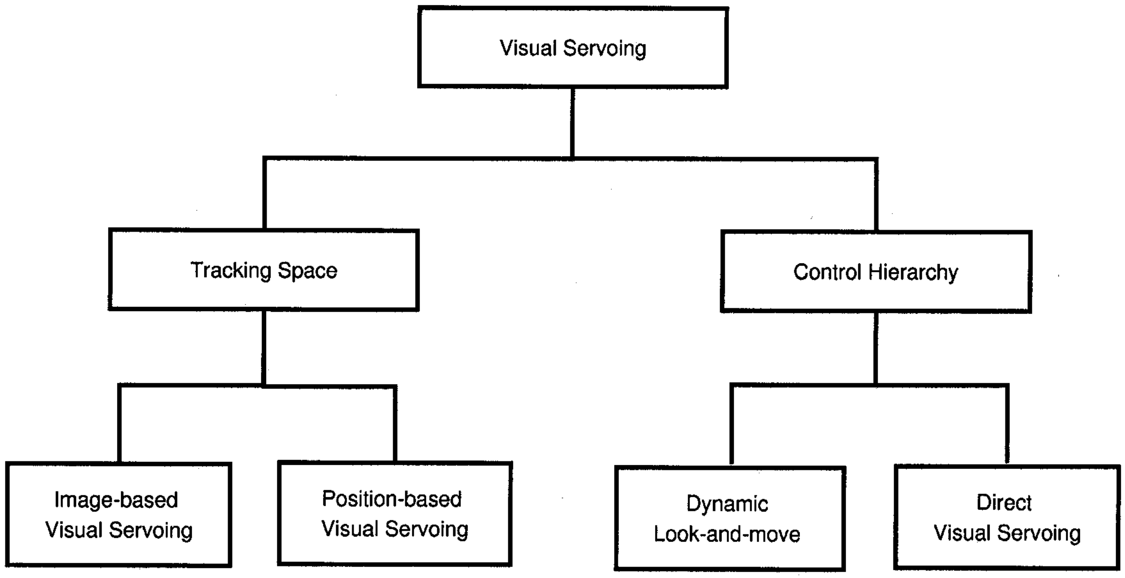


Figure 5–1: Types of visual servoing, as described in [22].

To elaborate further, servoing mechanism can be designed with an intermediate robot command generator that accepts feedback from the vision sensors and outputs robot motor control commands for controlling robot pose. These systems are known as *Dynamic Look-and-Move* systems. The systems where motor input commands are generated by the servo system directly are called *Direct Visual Servo* systems. In AQUA, the vision stack sends yaw and pitch commands to the control stack, which in turn generates motor commands for each of the six legs, effectively working as a dynamic look-and-move system. Our tracking and target detection is performed in image coordinates, which makes it an image-based servo system. In the following subsection we discuss a brief background of image-based visual servoing.

5.1.1 Image-based servoing

In image-based visual servoing, as discussed earlier in the section, the target object is tracked in image coordinates. The position of the object in each image frame is expressed in terms of image-space coordinates, not real-world or task-space locations. In our case we localize the centroid of the target in two-dimensional Cartesian coordinates, not three dimensional real-world coordinates. The error function ε, in this case is defined in $\Re^2$, as the Euclidean distance between the center of the frame and the center of mass of the target. The goal is to reduce ε such that $\varepsilon \longrightarrow 0$, as time $t \longrightarrow \infty$.

In spite of defining the error function in image space, the motor commands are specified in task space coordinates; *i.e.* the yaw and pitch commands are the same when the robot would be tele-operated by a remote controller. This necessitates a mapping of changes in the image coordinates to changes in the position of the robot. The **image Jacobian** is used to perform these changes.

Let f be a feature vector in the image space, r a vector of robot translational and rotational velocities, and $\dot{r}$ is the rate of change of these velocities. If k is the number of dimension of the image space and m is the number of dimensions of the task space, then the image Jacobian is defined as follows:

$$J_v(r) = \left[\frac{\partial f}{\partial r}\right] = \begin{bmatrix} \frac{\partial f_1(r)}{\partial r_1} & \dots & \frac{\partial f_1(r)}{\partial r_m} \\ \vdots & & \vdots \\ \frac{\partial f_k(r)}{\partial r_1} & \dots & \frac{\partial f_k(r)}{\partial r_m} \end{bmatrix} \tag{5.1}$$

The image Jacobian relates changes in the image space vector f to changes in the velocity vector r. The velocity vector r is also commonly referred to as the velocity screw in visual servoing literature. The mapping from changes in image-space to task-space is expressed by the following expression:

$$\dot{f} = J_v(r)\dot{r} \tag{5.2}$$

In image-based visual servoing, the interest is in finding the robot velocity $\dot{r}$ given the rate of change in image features $\dot{f}$. Hutchinson et al. shows [22] how to solve Eq. 5.2 to determine the velocity screw $\dot{r}$ to achieve a desired state of the image feature vector f.

5.2 Tracker Implementation

We discussed the theory behind the tracking algorithms used in AQUA in the previous chapter. This section and the included subsections explain in detail the implementation specific issues of each of these trackers.

5.2.1 Preprocessing

The color space chosen for blob tracking was the normalized RGB space, which is in effect an over-represented hue space. We chose the normalized RGB colorspace since the effect of lighting changes were minimum, and conversion from RGB to normalized RGB was not computationally expensive. The color frames obtained from the camera were 640×480 with 8-bits per pixel, with Bayer-encoded color information. The Bayer encoding scheme is a method of interpolating color information of a pixel using color values from the neighboring pixels, by using only 8-bits per pixel instead of 8-bits per red, green and blue channels. Bayer encoding reduces the overall data size by one-thirds, since only 8-bits per pixel are used, instead of 24, with a concomitant loss of color resolution. The Bayer pattern is specific to each camera and its CCD(Charge-Coupled Device, which is the equivalent of film in a 'traditional' camera). To retrieve color data, a conversion is required from the monochrome 8-bit Bayer-encoded image to a three-plane RGB image. This RGB color image is normalized by dividing each pixels RGB values with the sum of the RGB values. Examples of a Bayer-encoded frame, the RGB color image and a normalized RGB frame can be seen in images 5–2.

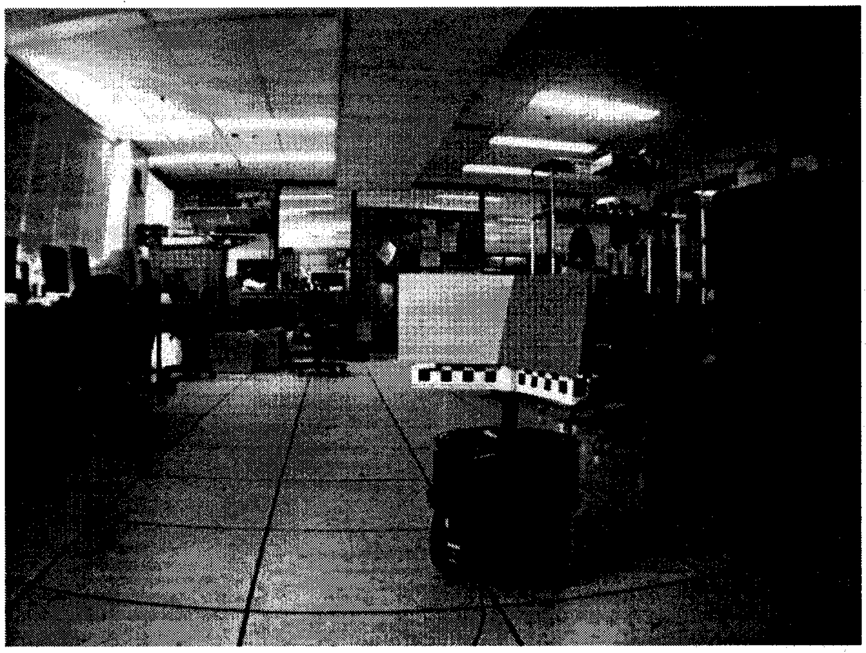


Figure 5–2: A raw frame from the camera.

These preprocessing steps are common to all the tracking methods described below.

5.2.2 Blob Tracker

The tracker is built on the principle of a Region of Interest operator. As discussed in the previous subsection, the RGB image is converted to a normalized RGB format. addition, a thresholding is performed on the absolute value. This is done in order to prevent the darker areas of the image from contributing to the region of interest. The parameters for finding the region of interest with the target color is given in normalized RGB values

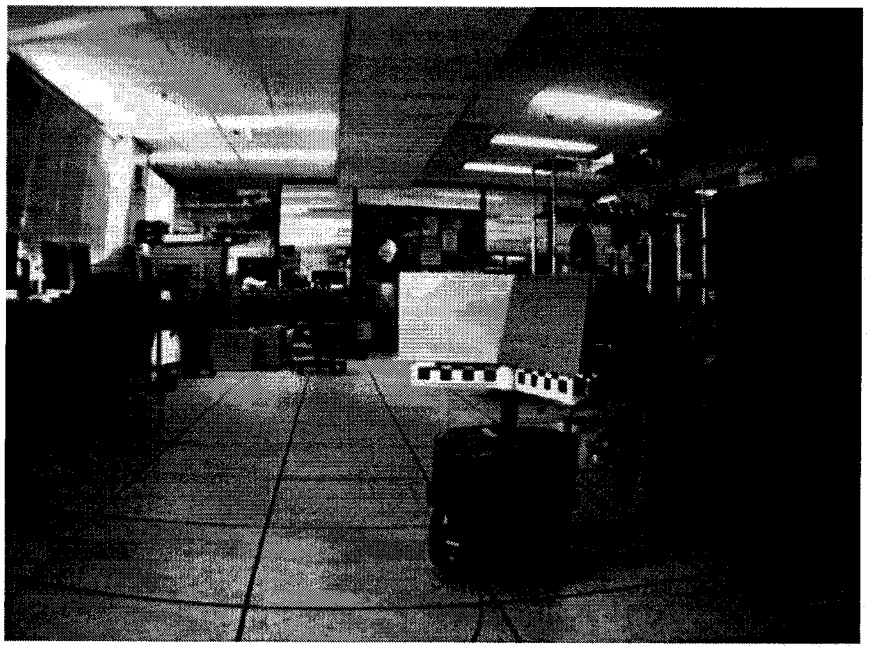


Figure 5–3: Bayer coded frames.

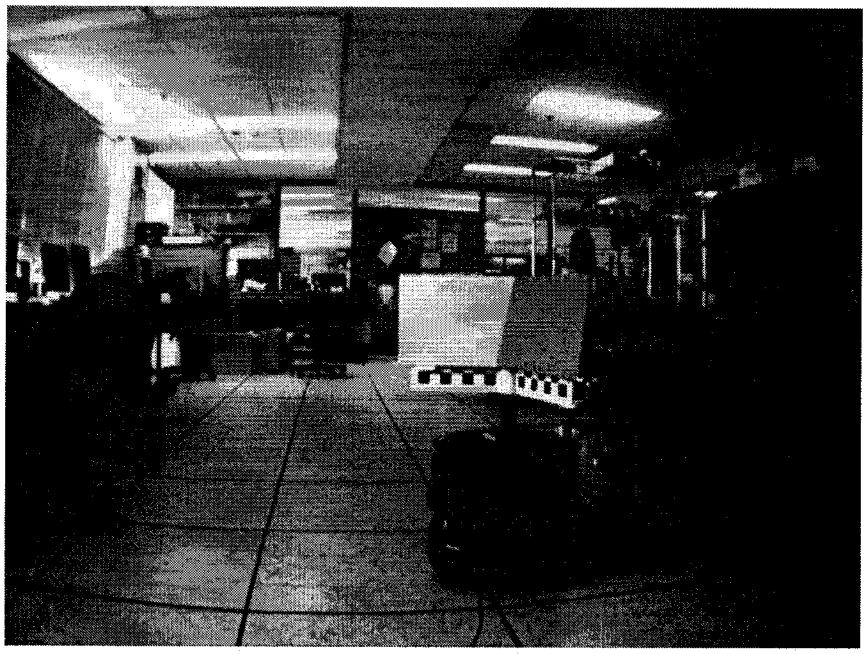


Figure 5–4: Normalized color frame.

as well (i.e hue). These values are manually tuned prior to the experiments. The image is then segmented and only those pixels whose RGB values fall within the thresholds are retained. To remove high-frequency (or shot/salt-and-pepper) noise[3], the median filtering algorithm[40] is used over the segmented image with either 5-by-5 or 7-by-7 pixel grids, with typical threshold values of 30%-40%. The center of mass of the blob is then calculated and the total mass of the blob is used as a confidence factor. If it is below a certain threshold (set *a priori* based on the object being tracked and lighting conditions), the measurement is ignored. Finally, the error signal is computed using the Euclidean distance between the centroid of the blob and the center of the image frame. Two error signals are used for pitch and yaw, and both these signals are then propagated to the PID controller.

5.2.3 Histogram Tracker

The histogram tracker compares rectangular regions of the input frame with the target region by comparing their corresponding color histograms. The target color distribution is stored by calculating a normalized histogram of a fixed number of bins, over the hue space. We used either 32 or 64 bins for histograms, depending on the target and the size of the image frame. The histograms are one-dimensional vectors that combine the multi-hue channel data. Similar histograms are computed for the searched sub windows. The subwindows have either one-eighth or one-sixteenth the dimension of the image frame. Similarity between histograms are computed by the measures discussed in the previous chapter. Since the histograms are normalized, the measures return values ranging from 0 to 1; higher values indicating higher degree of similarity. The minimum similarity measure is taken as 0.5; any measure below this threshold is not accepted as a valid target region.

The center of the chosen window is taken as the new target location. As in the case of the blob tracker, these coordinates are used by the PID controller to generate pitch and yaw commands.

5.2.4 Mean-shift Tracker

Color histograms are used as the underlying distribution in the mean-shift tracker. The histograms are three-dimensional arrays in this case, one each for the three RGB channels. We use 16 bins per channel for the mean-shift tracker. The target histogram are computed in a square window of sides equaling 100 pixels. The color model probability density function for the target is calculated by overlaying the subwindow by a kernel having the Epanechnikov profile. The weights for the mean-shift vector are calculated using the Epanechnikov kernel (See Appendix A). The tracker is initialized with the last known location of the target and the target PDF model. In each successive tracking step, the candidate window is created at the location of the last known target position, the candidate PDF model is calculated and the weights for pixel are calculated, leading to a new candidate position. We use 10 iteration steps for the mean-shift process to choose a new target location. Also, the target and the candidate window sizes are the same as well. The Bhattacharyya distance between the candidate PDF model and the target PDF model is calculated to quantify the similarity between the target and the new candidate location. The location with the minimum Bhattacharyya distance is chosen as the new target location. The method is based on that of Comaniciu, Ramesh and Meer [6].

5.3 Controller Design

The trackers are able to track the target at almost 15 frames/second, and therefore without filtering the control stack would be receiving possibly 15 different pitch and yaw command pairs from the visual tracker. Changing commands sent to the leg motors at such a high rate would yield a highly unstable swimming behavior. A PID controller is used to take these target locations and produce pitch and yaw commands at a rate to ensure stable behavior of the robot. The controller takes the error signals from the tracker and generates pitch and yaw commands for the gait controller. Given the input from the tracker at any instant, and the previous tracker inputs, the controller generates commands based on the following control law:

$$\Delta = K_P\overline{\epsilon_t} + K_I\int\overline{\epsilon_t}dt + K_D\frac{\partial}{\partial t}\overline{\epsilon_t} \tag{5.3}$$

where $\overline{\epsilon_t}$ is the time-averaged error signal at time t and is defined recursively as:

$$\overline{\epsilon_t} = \epsilon_t + \gamma\overline{\epsilon_{t-1}} \tag{5.4}$$

ϵ_t is the error signal at time t, K_P, K_I and K_D are respectively the proportional, integral and differential gains and γ is the error propagation constant.

We use two PID controllers for the pitch and yaw axes. Accordingly, there are two sets of gains for each of the three multiplication constants (proportional, integral and derivative). These gains are adjusted manually before visual servoing runs. The closed-loop control architecture of the visual servoing system is seen in Fig. 5–5. The servo control loops indicate the two different PID controllers for pitch and yaw as well as the different controller gains, denoted K_P for the pitch axis and K_Y for the yaw.

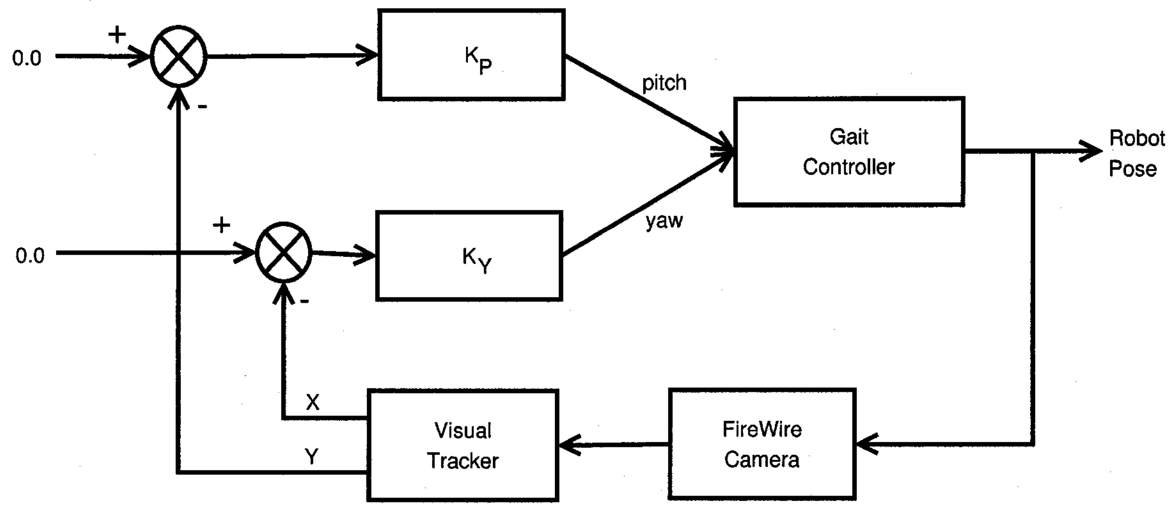


Figure 5–5: Closed-loops servo control.

For each of the pitch and yaw axes, the control loops work identically as follows. The controller acts as a low-pass filter, smoothing out fast changing pitch and yaw commands by averaging them over a period of time. The servo module implements a PID controller with a first-order IIR filter. We define a time constant for the low-pass filter for each PID controller. The controller gains are input manually, as mentioned earlier, with limits to truncate the gains. Each axis has a deadband limit applied to the error signal. This prevents the controller output from changing too frequently, by ignoring small changes in the error signal. We also introduce a *sleep time* between each iteration in servoing to reduce command overhead of the control stack.

5.4 Software System

As mentioned before in Chapter 3, the visual servoing code is based mostly on the VXL library. In particular, we have used VXL to provide the fundamental image processing algorithms and data structures, and our algorithms are implemented with these as the building blocks. The FireWire camera code uses another open-source library called CamWire[1] , by Johann Schoonees. CamWire encapsulates the low-level camera control code provided by the libdc1394 Linux library, making it easier to write and organize camera code. The tracking software has two modes of operation. During development and testing, we use a graphical user interface to interact with the trackers in real-time and experiment with the output. The GUI library significantly increases the size of the executable and is not suitable for deployment in the robot. The version on the robot has no GUI, but uses the Linux shell to interact in a command line environment. This reduces the size of the executable and speeds up execution. The tracking algorithms on both cases are same, the only change being the user interface.

The overall software architecture for the visual servoing task is shown below in Fig. 5–6 as a block diagram.

In the next chapter we present the results of the system we described so far. Servoing test results of the AQUA robot is discussed. We also look into the performance of the tracking algorithms, and how the choice of targets effect the trackers.

[1] http://kauri.auck.irl.cri.nz/ johanns/camwire/

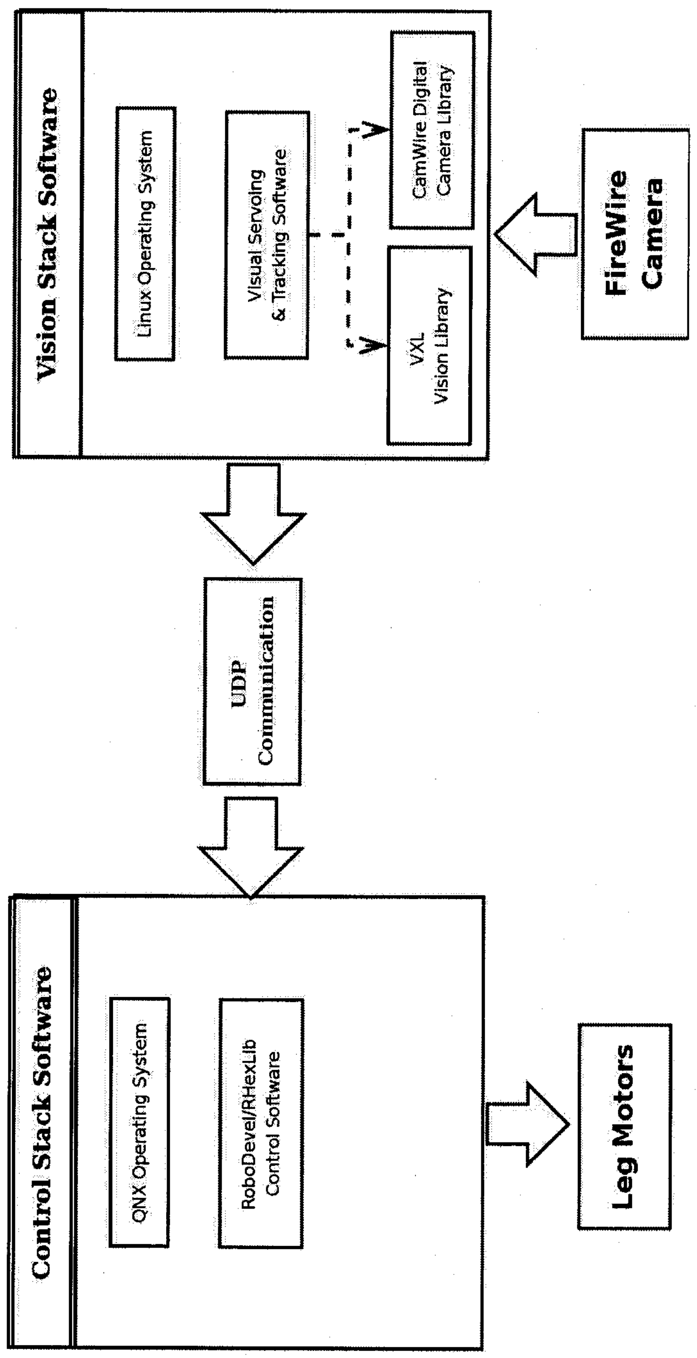


Figure 5–6: Complete software architecture block diagram.

CHAPTER 6
Experimental Results

We present the results of the visual servoing system described so far in this chapter. We evaluate the performance of the tracking approaches in terms of the accuracy and robustness, and also discuss types of targets each tracker is suitable for tracking. Performance of the PID controller is presented, while tracking an object travelling in a straight line during open-water trials. To gather ground truth data from underwater servoing experiments is an extremely complicated, if not impossible, task, given the unavailability of measurements that are standard for surface environments. This inability prevents us from presenting quantitative measurements of success for a target-following task, but we present results that demonstrate qualitatively the effectiveness of our system.

6.1 Tracking Performance

The three tracking algorithms are optimized for tracking objects with different color features, as discussed in Chapter 4. In this section we show the accuracy and robustness of each of the trackers while tracking different target objects. Situations which cause these trackers to fail to successfully track are also demonstrated.

6.1.1 Color Blob Tracker

As discussed in the previous chapter, the color blob tracker we used worked in the normalized-RGB color space. The blob tracker uses previously computed normalized RGB thresholds of the target object to segment out regions of an input frame in an attempt to locate the target. The following sequence of images demonstrate the operation of the

Table 6–1: Color blob tracker thresholds for a yellow target underwater.

Color Channel	Low	High
Red	0.398129	0.438129
Green	0.371813	0.411813
Blue	0.170058	0.210058
Selecting pixel (58, 129). Color R:143 G:134 B:65		

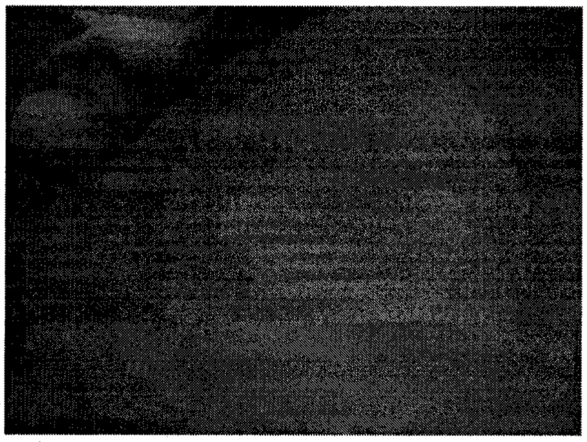


Figure 6–1: Yellow target during tuning.

color blob tracker during the tuning and tracking phases. The tuning operation and the resulting normalized RGB threshold values are shown in Fig. 6–1 and Table 6–1.

The result of the segmentation algorithm on this image produces Fig. 6–4(a) below. The crosshair in the middle indicates the centroid of the blob as calculated by the tracker.

As is clearly evident from figures 6–1 and 6–4(a), the color blob tracker performs as expected in localizing the target, based on the initial color thresholds tuned out.

The problems with the color blob tracker become evident when targets to be tracked have complex color characteristics. The sequence of figures in Fig. 6–4 demonstrates the blob tracker outputs for the checkerboard patterned object and clearly demonstrates its limits.

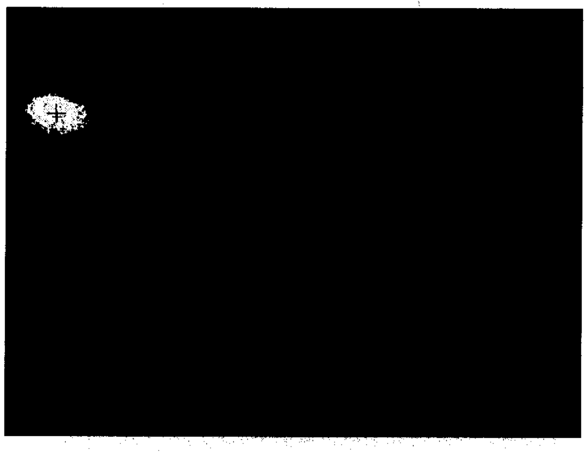


Figure 6–2: Result of Segmentation for the yellow ball.

Another of the major problems that we encounter with the color blob tracker, is the presence of duplicate targets. As seen in Fig. 6–5, even a surface reflection of the object creates a mirror image of the target object, which is sufficient distraction enough to confuse the tracker. The tracker, in such cases, calculates two blobs and the overall centroid is located on a line connecting the centers of masses of the two blobs. The tracker finds the location closer to the larger bob in such cases.

The case above demonstrates the need for using an object with very different color characteristics from the surrounding environment. We found yellow to be a color which is sufficiently unique, and also the hue transmitted for yellow underwater was much greater than any other single color we used.

6.1.2 Color Histogram Tracker

The color histogram tracker is a more enhanced tracking algorithm than the naive blob tracker. Using the target's color distribution as the feature to track, it achieves robustness compared to the blob tracker which easily fails in the presence of a duplicate target. As

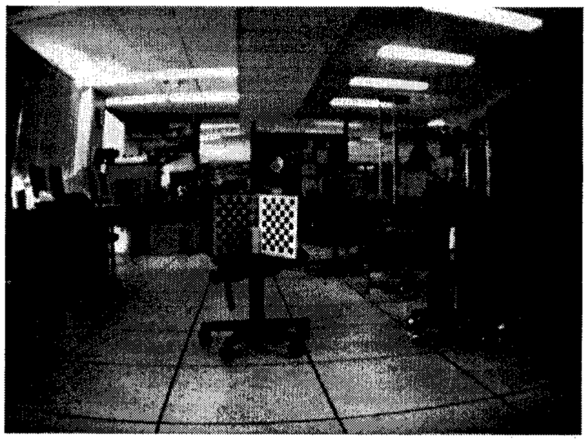


Figure 6–3: Checkerboard object for tracking.

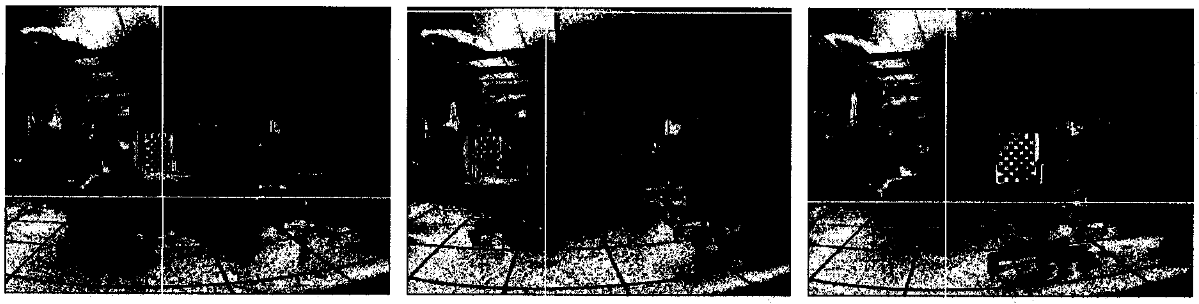

(a) Frame 1 (b) Frame 2 (c) Frame 3

Figure 6–4: Color blob tracking. Centroid of target at the crosshair.

such, the histogram tracker is suitable for tracking objects that have a variety of color, not just one. The operation of the histogram tracker during tuning and tracking is shown in the sequence of images in Fig. 6–7. We use one-dimensional histograms with 64 fixed-size bins, and each window is roughly $\frac{1}{16}$ (one-sixteenth) the width and height of the input image frame. Figure 6–6 shows the histogram distribution for the target shown in Fig. 6–6 with the target rectangle shown inside of a yellow border. The output of the tracker in another frame is the region outlined in red. For comparison with a region not containing

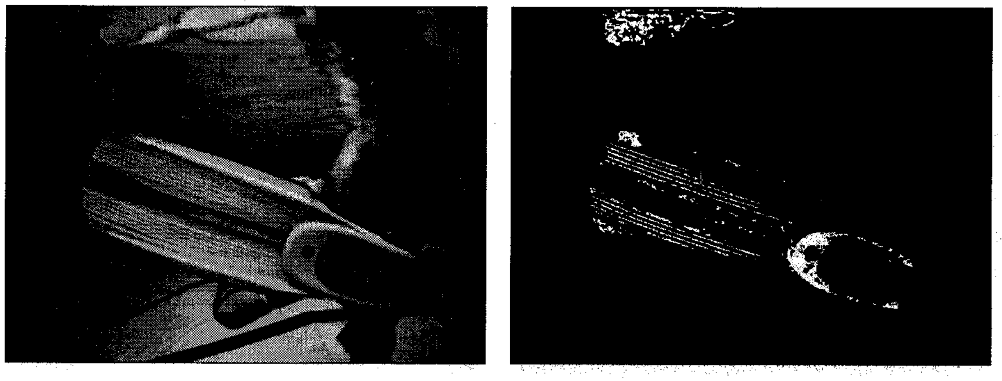

(a) Target with surface reflection (b) Segmented output, with the red circle indicating centroid

Figure 6–5: Effect of surface reflection.

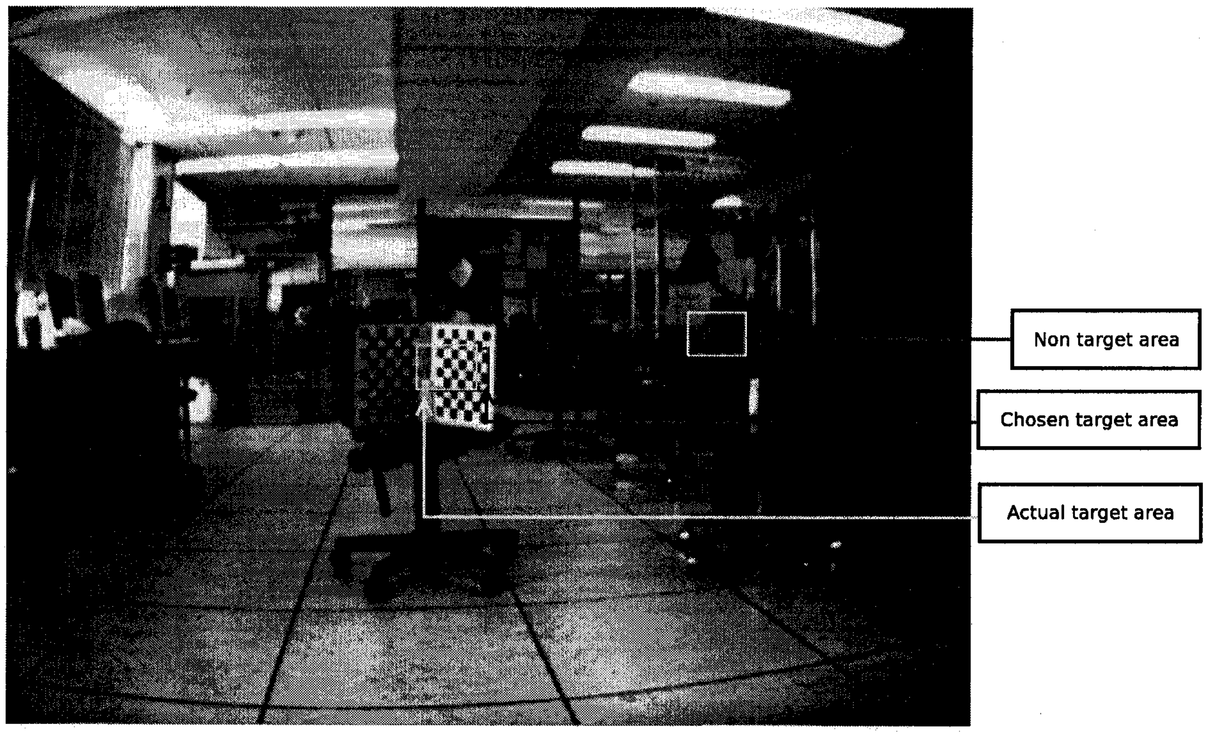
the target (*e.g.* the one outlined in white), we show the three histograms in Figures 6–6(b), 6–6(c) and 6–6(d) respectively, of the actual target, the tracker output and the non target area. The measured similarity value using the Bhattacharyya measure is given in table 6–2.

Table 6–2: Bhattacharyya measures.

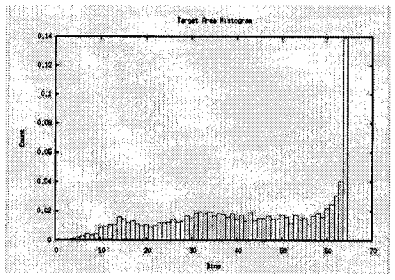
Target and Chosen	Target and Non-target
0.989718	0.192324

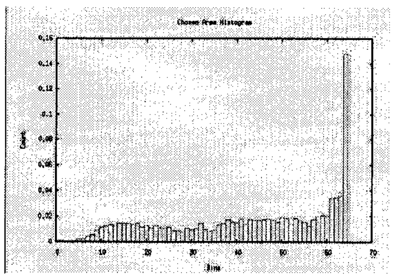
6.1.3 Mean-shift Tracker

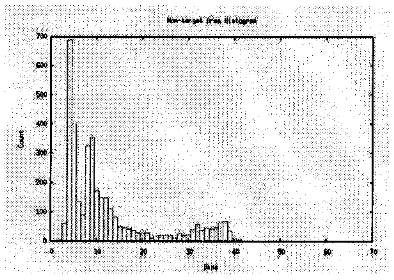
Robust as it is, the histogram tracker is accurate only as far as the size of the sub-windows in which it operates; that is, the center of the target will be located at the center of the rectangular search window, which may not be the actual centroid of the target. To achieve higher accuracy without sacrificing the robustness of the histogram tracking, the



(a) Input frame with regions



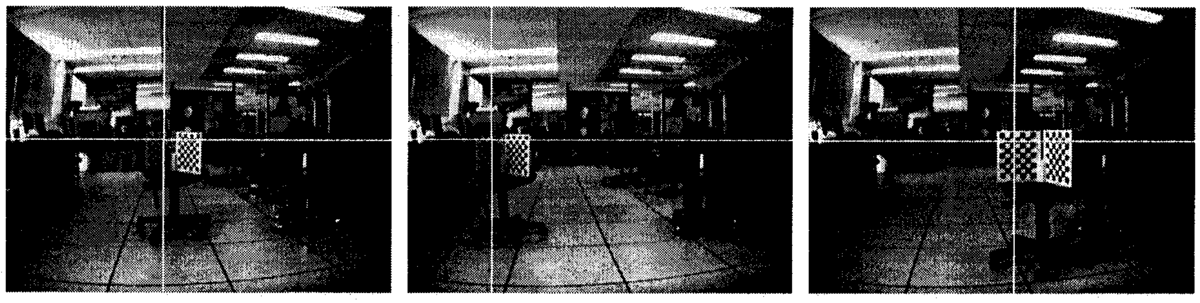




(b) Target Histogram (c) Chosen Histogram (d) Non-target Histogram

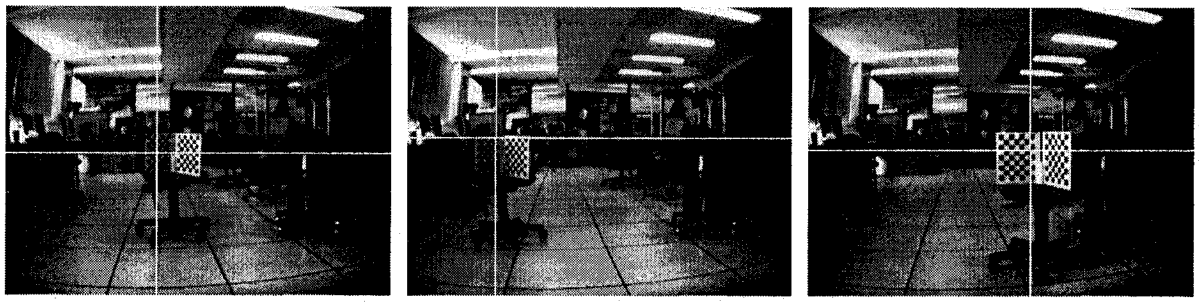
Figure 6–6: Histogram tracking results.

mean-shift tracker is used. Building on the color histogram property of the target object, the mean-shift tracker localizes the target using the direction of mean-shift vector as



(a) Frame 1 (b) Frame 2 (c) Frame 3

Figure 6–7: Histogram tracking output, with target at crosshair.



(a) Frame 1 (b) Frame 2 (c) Frame 3

Figure 6–8: Mean-shift tracking output, with target at crosshair.

a pointer to the next probable target location. The accuracy provided by the mean-shift tracker, however, does not come without the extra computational cost. This algorithm is the slowest of the three we used in AQUA, and assumes that the target does not change location by a large margin between consecutive frames. With this reasonable assumption, we present the results of the mean-shift tracker in operation in the sequence of images shown in Fig. 6–8.

Table 6–3: Pitch and yaw servo controller parameters; see Section 5.3.

Axis	K_P	K_I	K_D	$Limit_P$	$Limit_I$	$Limit_D$	Deadband	Time Constant	Command Limit
Pitch	1.0	0.0	0.0	1.0	0.3	1.0	0.2	0.35	1.0
Yaw	1.0	0.0	0.0	1.0	0.3	1.0	0.2	0.05	1.0

6.2 PID Controller

To tune a PID controller (*i.e.* adjust K_P, K_I and K_D) to ensure smooth, non-oscillating performance, the step response of the system has to be measured. For this work, we do not have the step response data available from the robot. This hampers our ability to tune the PID controller properly. Nevertheless, the use of the proportional gain alone has been sufficient to ensure periodic and bounded response from the servo system, as found from the open-water trials, discussed in the next section. For the servo trials, the PID gains were adjusted using empirical values, observing the robot's response to a change in the PID gains. A typical set of parameters for the yaw and pitch servo controllers are presented in table 6.2.

We observed a direct improvement in the yaw response as the yaw axis filter-time constant was reduced below 0.1 seconds. An IIR filter present on the control stack smoothed out oscillations in input roll, yaw and pitch commands from the operator, as well as the pitch and yaw commands coming from the vision stack. In our tests, reducing the filter-time constant to as low as 0.05 seconds had no adverse effect on the robot's stability.

The next section presents some quantitative results acquired from the open-water trials held at the Bellairs Research Centre of McGill University, at Barbados. The results from these tests demonstrate the success of the system, in spite of all the unknown parameters affecting the robot in an open-water environment.

6.3 Open-Water Trials

For the open-water trials at Barbados, a 15 centimeters diameter yellow ball was held by a swimmer in front of the robot at a distance of approximately 2 meters. The diver swam in a roughly straight line at a pace consistent with the robot's speed. (In fact, the robot could outpace the diver at full speed.) A sequence of successful trials was completed each for duration of roughly 100 seconds over a distance of 27 meters, giving the robot an average ground speed of approximately 0.3 meters/second. The tests were concluded when the robot reached the end of its fiber-optic telemetry cable. The frame rate used by the visual tracker was one frame per second, consistent with the low-pass control loop mandated by the oscillating swimming gait. During this trial, the color segmentation algorithm successfully found the target in all but one image. This gives a success rate of 99% for the visual tracker. Other trials had slightly less successful detection rate, but all were above 90%. Figures 6–9 and 6–10 show the relative tracked positions of the target for the x-axis (yaw) and the y-axis (pitch) over the duration of one trial. A value of zero indicates a centered target, whereas a value of +1.0 or -1.0 indicates the target has reached the boundary of the camera's field of view. Since only a proportional gain was used in the experiment, the command sent to the robot's gait controller was simply proportional to this relative position. The bounded periodic nature of the signal indicate that the tracker and control loop are functioning in a stable operating mode.

In order to maintain the target in the center of the camera's field of view, two feedback loops are necessary as shown in Fig. 5–5. Yaw commands are used to correct error in the image's x-axis, and pitch commands in the y-axis. Note that in the current experiment, the roll axis was left uncorrected. This would have required either using some form of

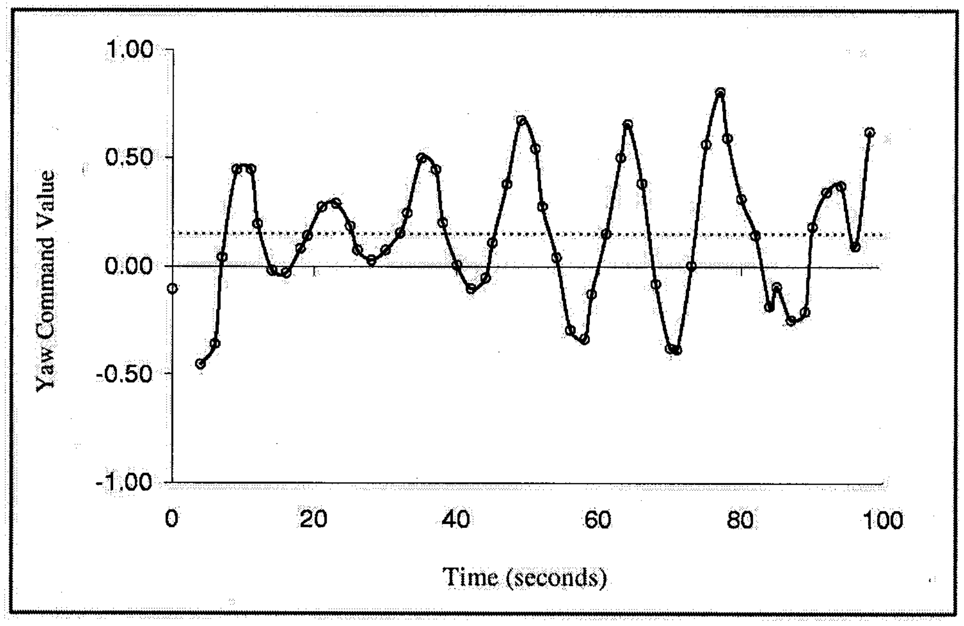


Figure 6–9: Response of yaw to visual servoing.

shape recognition and an asymmetric target, or to be able to establish the direction of the vertical axis by using an Inertial Measurement Unit. Provisions have been made to integrate such a device in future experiments. However, since no robot response data (such as step response) was available, the transfer function as well as the frequency response of the robot for each axis was unknown. It was therefore not possible to fully tune the PD controller beforehand. This limited our ability to find the optimal parameters for the controllers. See Fig. 6–9.

The average value for the yaw axis relative position was important (0.15), especially compared to the pitch axis' average position value (0.00095). This can be explained by the

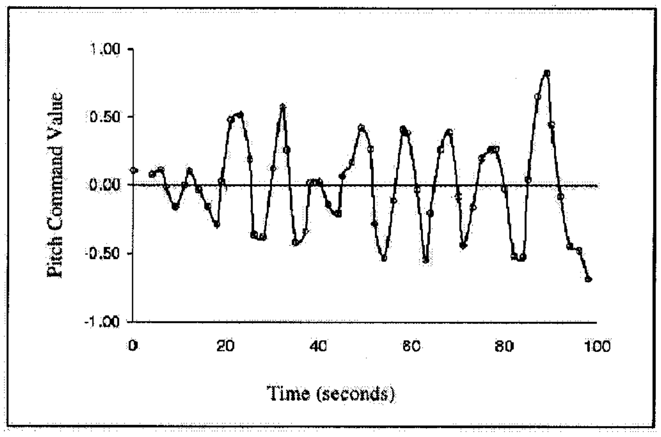


Figure 6–10: Response of pitch to visual servoing.

presence of a side-current in the open water condition. Since the diver holding the target was attempting to follow a straight line using visual cues from the substrate, a bias was needed to compensate for the side-current dynamics. No such bias was needed in the pitch direction.

In both figures an oscillation with a period of approximately 10 seconds can be noted. These are coming from the sub-optimal behavior of the proportional controller in the presence of the low-pass filter of the robot's gait controller, and to a minor extent to the inertia of the robot. The full use of a properly tuned PID controller as well as further refinement of the gait controller would help reduce and possibly eliminate those oscillations. Nevertheless, this sub-optimal tuning proves the feasibility of on-board visual servoing using AQUA's six flippers motion actuation.

CHAPTER 7
Discussions and Conclusion

7.1 Overview

This thesis presented a visual servoing system for an aquatic legged robot called AQUA. The approach to servo control is based on simple color tracking coupled with a control loop whose low-pass properties are tuned to eliminate the natural undulations caused by the robot's swimming gait. We discussed the underlying hardware and software components of the system and presented the data collected during open and closed-water trails of the system. The system is inherently simple and enables AQUA to achieve some degree of autonomy in navigating underwater. The recent sea trials of the system have proven to be very successful, and presents exciting new directions for future work.

7.2 Tracking and Underwater Vision

We have chosen tracking algorithms that have been shown to be successful in terrestrial, non-underwater vision, and applied them in the underwater vision domain to achieve a high degree of success. As mentioned earlier in this thesis, the marine environment poses unique challenges for vision systems to work effectively, as a number of assumptions that can be made for non-underwater systems are no longer realistic in this domain. The unsuccessful tracking trials have helped to reveal the limitations of these algorithms applied "as-is" in the underwater domain. We have investigated the effect of lighting variations on the underwater vision system as well. We hope the experiments will inspire future work in developing algorithms particularly designed for underwater vision.

7.3 Vision-based Vehicle Control

The main contribution of this work is not in any one of the individual components, but instead in demonstrating the applicability of vision in autonomous underwater vehicle control. The advantage of using vision to achieve autonomous navigation lies in the passiveness and low power consumption property of vision sensors. A simple PID controller has been shown to be sufficient to produce bounded motion of the robot during servoing. The nature of motion of an aquatic vehicle is complex, and whether or not to mimic biological motion is a question that needs to be investigated thoroughly before a satisfactory answer can be given. We have implemented a Linux-based environment for the vision processor in a limited storage that is also power conservative and robust to system failures. Each of these system components have played a vital role in the overall performance of the system.

7.4 Future Work

The visual servoing system described here has been proven to work in the real world, but there is room for many more enhancements and new provisions. Improvements can be made to the tracker as well as the overall control system, to build a more robust and stable visual servoing mechanism. To date, the tracker only looks for an object of a certain color or color distribution, without looking for an object of predefined shape. We plan to integrate shape and pattern matching with the tracker in the near future. Also, from the size of the tracked object on screen, the speed of the robot can be controlled; so that the robot can catch up with the moving target when it is about to lose track. Neither the tracker or the controller incorporates any learning scheme at the present time. All gains and parameters are tuned manually with the aid of data from previous trials. A probabilistic learning scheme incorporated with the servoing could greatly increase the robustness as well as provide for automatic object recognition and training of parameters and gains. An Inertial Measurement Unit (IMU) has been used as a stability augmentation system. We aim for tighter integration of the IMU with the vision system, making for a semi-dynamic look-and-move servoing architecture. Currently, the servo system has no control over the roll command of the robot. Using feedforward control, we can also compensate for the coupling between the axis controls.

It appears that a more flexible learning-based scheme for target acquisition and tracking would permit the system to operate more robustly. While we have not experienced serious tracking failures where illuminations prevents the target from being acquired, one might expect this to occur in the absence of on-line auto-calibration. More important, it appears that the tracking system can be "fooled" by distracting objects whose coloration matches the target of interest. While using supplementary shape-based cues would be a natural improvement to the tracker, the computing overhead, particularly in the robot's small form-factor make this a challenge. Under poor visibility conditions the range of available hues transmitted through the water is very limited. A particularly attractive option is the use of a tracker that explicitly models the motion of the target (for example the undulation of the diver being followed). This suggests several interesting avenues for future work.

Appendix A: The Epanechnikov Kernel

The **Epanechnikov kernel** has the following form:

$$\begin{aligned} t &= \frac{3}{4}(1-u^2); -1 < u < 1 && (7.1) \\ &= 0; otherwise. && (7.2) \end{aligned}$$

Here, $u = \frac{x-x_i}{h}$, where h is the window width and x_i are the values of the independent variable in the data, and x is the value of the scalar independent variable for which one seeks an estimate.

Also, the **profile** of a kernel K is defined as a function $k : [0, \infty) \rightarrow R$ such that $K(x) = k(\|x\|^2)$.

REFERENCES

[1] R. Altendorfer, N. Moore, H. Komsuoglu, M. Buehler, H.B. Brown Jr., D. McMordie, U. Saranli, R. Full, and D. E. Koditschek. RHex: A biologically inspired hexapod runner. *Autonomous Robots*, 11:207–213, 2001.

[2] K. Arbter, J. Langwald, G. Hirzinger, G. Wei, and P. Wunsch. Proven techniques for robust visual servo control. In *IEEE International Conference on Robotics and Automation, Workshop WS2, Robust Vision for Vision-Based Control of Motion*, pages 1–13, 1998.

[3] Alan C. Bovik, editor. *Handbook of Image and Video Processing*. Academic Press, 2000.

[4] Alex Brooks, Tobias Kaupp, Alex Makarenko, Anders Orebck, and Stefan Williams. Towards component-based robotics. In *IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, Edmonton, Canada, 2005.

[5] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):24–30, March 1986.

[6] Dorin Comaniciu and Peter Meer. Mean Shift: A robust approach toward feature space analysis. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 24(5):603–619, May 2002.

[7] Eve Coste-Maniere and Ried Simmons. Architecture, the backbone of robotics systems. In *IEEE International Conference on Robotics and Automation*, pages 67–72, April 2000.

[8] Noah J. Cowan and Daniel E. Koditschek. Planar image based visual servoing as a navigation problem. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 611–617, Detroit, Michigan, USA, 1999. IEEE.

[9] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, April 2000.

[10] Gregory Dudek and Robert Sim. RoboDaemon - a device independent, network-oriented, modular mobile robot controller. In *IEEE International Conference on Robotics and Automation (ICRA), Taipei, Taiwan*, volume 3, pages 3434–3440, Taipei, Taiwan, October 2003. IEEE Press.

[11] Ryan Eustice, Hanumant Singh, John Leonard, Matthew Walter, and Robert Ballard. Visually navigating the RMS Titanic with SLAM information filters. In *Robotics Science and Systems*, June 2005.

[12] Graham. D. Finlayson. Computational color constancy. In *International Conference on Pattern Recognition*, volume 1, pages 191–196, Barcelona, Spain, September 2000.

[13] Freedman and M. Brandstein. Contour tracking in clutter: a subset approach. *International Journal of Computer Vision*, 38(2):173–186, 2000.

[14] U. Gargi and R. Kasturi. An evaluation of color histogram based methods in video.

[15] Erann Gat. *Artificial Intelligence and Mobile Robots, Editors D. Kortenkamp, R. P. Bonnasso and R. Murphy*, chapter On three-layer architectures. AAAI Press, 1997.

[16] Christina Georgiades, Andrew German, Andrew Hogue, Hui Liu, Chris Prahacs, Arlene Ripsman, Robert Sim, Luiz Abril Torres-Mendez, Pifu Zhang, Martin Buehler, Gregory Dudek, Michael Jenkin, and Evangelos Milios. AQUA: An aquatic walking robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 4, pages 3525–3531, Sendai, Japan, September 2004. IEEE, IEEE Press.

[17] B. Gerkey, R. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics (ICAR)*, pages 317–323, Coimbra, Portugal, June 30 - July 3 2003.

[18] J. Geusebroek, D. Koelma, A. Smeulders, and T. Gevers. Image retrieval and segmentation based on color invariants. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 784–785, Hilton Head Island, South Carolina, June 2000.

[19] Theo Gevers and Arnold W. M. Smeulders. Color based object recognition. In *International Conference on Image Analysis and Processing*, pages 319–326, 1997.

[20] Gregory D. Hager. A modular system for robust positioning using feedback from stereo vision. *IEEE Transactions on Robotics and Automation*, 13(4):582–595, August 1997.

[21] T. Hamel and R. Mahony. Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach. *IEEE Transactions on Robotics and Automation*, 18(2):187–198, April 2002.

[22] S. A. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, October 1996.

[23] Matthew Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 1(29):5–28, 1998.

[24] Michael Isard and Andrew Blake. Contour tracking by stochastic propagation of conditional density. In *European Conference on Computer Vision*, volume 1, pages 343–356, Cambridge, UK, 1996.

[25] Xu Jie and Shi Peng-fei. Natural color image segmentation. In *International Conference on Image Processing*, volume 1, pages 973–976, September 2003.

[26] K. Konolige, K. L. Myers, E. H. Ruspini, and A. Saffiotti. The Saphira architecture: A design for autonomy. *Journal of Experimental & Theoretical Artificial Intelligence: JETAI*, 9(1):215–235, 1997.

[27] Vladimir Kravtchenko. Tracking color objects in real time. Master's thesis, University of British Columbia, Vancouver, British Columbia, November 1999.

[28] NASA Jet Propulsion Laboratory. Mars exploration rover mission. http://marsrovers.jpl.nasa.gov/home/index.html, June 2005.

[29] Stephen Licht, Victor Polidoro, Melissa Flores, Franz Hover, and Michael S. Triantafyllou. Design and projected performance of a flapping foil AUV. *IEEE Journal Of Oceanic Engineering*, 29(3):786–794, July 2004.

[30] J-F. Lots, D. M. Lane, E. Trucco, and F. Chaumette. A 2-D visual servoing for underwater vehicle station keeping. In *IEEE International Conference on Robotics and Automation*, pages 2767–2772, Seoul, South Korea, May 2001.

[31] Yi Ma, Jana Kosecka, and Shankar Sastry. Vision guided navigation for a nonholonomic mobile robot. In *IEEE Conference on Decision and Control*, 1997.

[32] Eric Marchand and Gregory Hager. Dynamic sensor planning in visual servoing. In *IEEE International Conference on Robotics and Automation*, pages 1988–1993, Leven, Belgium, May 1998.

[33] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) Toolkit. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.

[34] Regents of the University of Michigan. Control tutorials for Matlab.

[35] A. Trebi-Ollennu H. Aghazarian H. Das S. Joshi P. Pirjanian, T. Huntsberger and P. Schenker. CAMPOUT: A control architecture for multirobot planetary outposts. In *Proceedings of SPIE Conference. Sensor Fusion and Decentralized Control in Robotic Systems III, Boston, MA*, November 2000.

[36] Lynn E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[37] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The Earth Mover's Distance as a metric for image retrieval. Technical report, Stanford University, Stanford, CA, USA, 1998.

[38] Uluc Saranli and Eric Kavins. Object oriented state machines. Embedded Systems Programming, May 2002.

[39] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, November 1990.

[40] G. J. Yang T. S. Huang and G. Y. Tang. Fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1979.

[41] The ThinkQuest Team. The physics of diving: Light and vision. http://library.thinkquest.org/28170/35.html.

[42] Alexandra H. Techet, Franz S. Hover, and Michael S. Triantafyllou. Separation and turbulence control in biomimetic flows. *Flow, Turbulence and Combustion*, 71:105–118, October 2003.

[43] Axel Techmer. Contour-based motion estimation and object tracking for real-time applications. In *International Conference on Image Processing*, volume 3, pages 648–651, Thessaloniki, Greece, October 2001.

[44] G. T. Toussaint and B. K. Bhattacharya. Optimal algorithms for computing the minimum distance between two finite planar sets. *Pattern Recognition Letters*, 2:79–82, 1983.

[45] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The CLARATY architecture for robotic autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference, Big Sky, Montana*, March 2001.

[46] S. B. Williams and I. Mahon. Simultaneous localization and mapping on the Great Barrier Reef. In *IEEE International Conference on Robotics and Automation*, New Orleans, USA, April 2004. IEEE.

[47] Richard Y. D. Xu, John G. Allen, and Jesse S. Jin. Robust mean-shift tracking with extended fast colour thresholding. In *International Symposium on Intelligent Multimedia, Video Speech Processing (ISIMP2004)*, pages 542–545, Hong Kong, October 2004.

[48] Zoran Zivkovic and Ben Kr̋ose. An EM-like algorithm for color-histogram-based object tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, Washington D.C., June 2004. IEEE, IEEE Press.

KEY TO ABBREVIATIONS

BIBO: Bounded Input Bounded Output

HSV: Hue-Saturation-Value

IEEE: Institute of Electrical and Electronics Engineers, Inc.

IIDC: Instrumentation and Industrial Digital Camera standard

IMU: Inertial Measurement Unit

PID: Proportional-Integral-Derivative

RAM: Random Access Memory

RGB: Red-Green-Blue

TCP/IP: Transmission Control Protocol/Internet Protocol

UDP: User Datagram Protocol

UML: Unified Modelling Language