A Hybrid Motion Vision Guided Robotic System with Image Based Grasp Planning

by

Kevin Stanley

BASc. Simon Fraser University, 1997

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in the

School of Engineering Science

Simon Fraser University

August 1999

Approval

Name: Kevin Gordon Stanley

Degree: Master of Applied Science (Engineering)

Title of Thesis: A Hybrid Motion Vision Guided Robotic System with Image Based Grasp Planning

Examining Committee

Chair: Dr. John Jones

Dr. William A. Gruver
Senior Supervisor
Professor of Engineering Science

Dr. Kamal Gupta
Professor of Engineering Science

Dr. Jonathan Wu
Group Leader, Vision and Sensing
National Research Council of Canada

Dr. Ze-Nian Li
External Examiner
Professor of Computing Science

Date Approved: ______________________________

Abstract

The field of vision guided robotics has existed for nearly twenty years in academia, but has not made the transition to industry on a wide scale. During the last five years, the increase in computational power and the decrease in the cost of components have allowed vision-guided robotics to become a reality. This thesis describes the design; implementation and verification of a practical vision guided robotic system for handling a broad range of discrete parts.

There are two major methods for mapping the input image to robot motion: computed kinematics and visual servoing. We use computed kinematics to refer to any algorithm, which computes the transform between the image plane and the world frame, including calibration and neural network based methods. In computed kinematics, the pose of the target in world coordinates is calculated, then an intercept motion vector is generated for the robot. In visual servoing the error between the desired and current image state is regulated to zero using a control law. Computed kinematics requires only a single iteration, but is very sensitive to calibration errors, and cannot compensate for non-idealities in the image. Visual servoing is more robust, but requires more computation to reach a solution. To mitigate these factors, a hybrid system has been designed. The hybrid system uses an initial computed kinematics move followed by a visual servoing correction, thereby providing a good compromise between speed and accuracy. A linear approximation model and a neural network were used to approximate the kinematic transform between the image and world frames. The neural network has the advantage of non-linear approximation, while the model based linear approximation incorporates *a priori* knowledge. By carefully choosing input values the system is capable of moving the robot manipulator to stationary, rigid parts, for which good contrast can be obtained.

Part manipulation is performed using an image based grasp planner. This grasp planner performs a quadtree expansion of the image, examining sampled points for their suitability based on a fitness function and their validity based on a collision checker. The algorithm exits once a suitable grasp point is found. The purpose of the grasp planner is to find a suitable grasp point quickly, not to find an optimal grasp point. The integrated system is capable of grasping any object with a near planar surface.

The motion system was tested by comparing the hybrid system with a visual servoing system. The hybrid system showed a performance increase. The grasp planner was tested using real and synthetic images. The entire system was implemented in a sorting cell. The proposed system has commercial potential for both sorting and tele-operation.

Acknowledgements

I would like to thank Dr. Bill Gruver for his supervision and knowledge, Dr. Kamal Gupta for his insight on the grasp planning problem, Dr. Jonathan Wu and Dr. Ali Jerbi for their guidance, my family for their support, and most of all my fiancée Regan, who by the time this is published will be my wife.

Table of Contents

List of Tables

List of Figures

List of Symbols

Coordinate Systems and Variables

$\{\mathbf{C}_e\}$	External camera frame (Cartesian)
$\{\mathbf{C}_r\}$	End-effector mounted camera frame (Cartesian)
$\mathbf{f}$	A feature vector in $\{\mathbf{I}\}$
$\mathbf{f}_d$	A desired feature vector in $\{\mathbf{I}\}$
$\mathbf{f}_e$	A feature error vector in $\{\mathbf{I}\}$
$\{\mathbf{I}_r\}$	End-effector mounted camera image frame (planar)
$\{\mathbf{I}_e\}$	External camera frame (planar)
$\mathbf{J}$	Feature Jacobian
$\mathbf{J}^{-1}$	inverse feature Jacobian
$\mathbf{K}$	A PD control gain matrix
$\{\mathbf{O}\}$	World coordinate frame

$\{\mathbf{R}\}$	Robot end-effector frame
$\mathbf{r}$	Position of the robot end-effector in the world frame
$\mathbf{r}_{ce}$	Position of the robot end-effector in $\{\mathbf{I}_c\}$
$\mathbf{T}$	Transform matrix between $\{\mathbf{I}\}$ and $\{\mathbf{O}\}$
u	Position of the target in image space along the x axis
v	Position of the target in image space along the y axis
w	Difference between the vertical position of the target and end effector in $\{\mathbf{I}_c\}$, used as the z coordinate in image space.
$\mathbf{x}_t$	Position of the target in the world frame
$\mathbf{x}_{ti}$	Position of the target in image space
$\mathbf{x}_{cr}$	Position of the target in $\{\mathbf{I}_r\}$
$\mathbf{x}_{te}$	Position of the target in $\{\mathbf{I}_c\}$
$\mathbf{x}_{rt}$	Difference of $\mathbf{r}$ and $\mathbf{x}_t$
x	Position in Cartesian space (usually of the target) on the world x axis
y	Position in Cartesian space (usually of the target) on the world y axis

z	Position in Cartesian space (usually of the target) on the world z axis
θ	Orientation about world z axis in either image or Cartesian space
ω_z	Speed of rotation about world z axis

Image Processing

A	Area
A_{npoly}	Area of a regular n-sided polygon
F	Set of allowed grayscale values during thresholding
f_c	The focal length of a camera
f_c	Compactness
g_x	x coordinate of Sobel gradient operator
g_y	y coordinate of Sobel gradient operator
I_{ij}	Intensity of the pixel i,j
p	Perimeter

Grasp Planning

C_{be}	Cost of evaluating if a window contains a node
C_{bs}	Cost of evaluating the binary search
C_{cd}	Cost of calculating the collision detection algorithm
C_{d}	Cost of evaluating the distance from center of mass
C_{ij}	Cost of calculating the grasp planner for a level
C_{f}	Cost of evaluation the fitness function
C_{T}	Total cost of using the grasp planner
C_{θ}	Cost of evaluating the difference in surface normals
$C_{\sigma\theta}$	Cost of evaluating the curvature
d_{cm}	Distance from the grasp axis to the targets center of area
F	Force applied during a grasp
g_{i}	Number of grayscale levels
p_{g}	Number of pixels in the gripper window

p_i	Number of pixels in the window of the *ith* layer
W_{bi}	Number of boundary nodes at layer *i*
W_i	Number of nodes in the *ith* layer
W_{wi}	Number of winning nodes at layer *I*
μ	Frictional coefficient of a surface
θ_1-θ_2	Relative surface normal, used in grasp planning
σ_θ	Standard deviation of the gradient orientation intensity; a measure of curvature for grasp planning

Neural Networks

bias	The node bias
w_{ij}	Weight of the *ijth* connection
x_j	Node input
y_d	Desired output
y_{ij}	Output of the *ijth* node

y_{net}	Neural network output
$\Theta(\psi)$	Node activation function
ζ_i	Unweighted node output
α	Momentum coefficient
η	Learning rate coefficient
δ_{i}	Node error

1 Introduction

Vision guided robotics has a rich research history that dates back to the late seventies and early eighties. While many elements of vision guided robotics have been thoroughly researched, few vision guided robotic systems have found their way into industry. Most vision guided robotic systems were too slow, and too sensitive to the environment to be useful in an industrial setting. With the rapid rise in computing power and the drop in price of high quality robotic and vision systems, the application of vision guided robotic systems to an industrial setting is becoming a reality. However, there are still barriers and limitations to the production of generic, robust, and practical vision guided robotic solutions.

The broad nature of the research means we must draw from a wide body of previous research. Advances in visual servoing, kinematics, grasp planning, computer vision, and neural networks are all relevant. An overview of relevant articles is presented in the following paragraphs.

An overview of visual servoing is given by Hutchinson, *et al.* [1] who describes the research and fundamentals of geometric feature based visual servoing. Corke [2] has shown that the performance of visual servoing algorithms can be enhanced by incorporating the dynamics of the system in the model. Papanikopolous, *et al.* [3] have used adaptive control techniques to perform visual servoing.

To automate the derivation of the Jacobian, many researchers have used neural network approximators in vision guided robotics. Miller, *et al.* [4] used a neural network to demonstrate the feasibility of the approach. Hashimoto, *et al.* [5] used a two level self-organizing network to approximate the Jacobian for coarse and fine motion. Wu and Stanley [6][7] used a fuzzy decision network to manage a hierarchy of backpropagation networks to approximate the Jacobian over the entire workspace. Van Der Smagt, *et al.* [8] theoretically demonstrated that a neural network can achieve positioning of a vision-guided robot. All of the above researchers used geometric features for their input. A new target object required that a new feature set be chosen.

Other researchers have attempted to automate feature selection. Feddema [9] has shown that geometric feature selection can be aided by measuring the properties of the Jacobian. Hashimoto and Noritsugu [9] have derived a similar measure for feature sensitivity. On the other hand, some researchers have used appearance or pixel-based methods to create a feature vector. Instead of automating the feature extraction, Nayar [11] used principal component extraction (PCX) algorithm to find the eigenvectors of the image and servoed the robot based on these eigenvectors. Anguita, *et al.* [12] used an averaging compression to create a feature vector, which was used to train a neural network which approximated the inverse Jacobian. Wu and Stanley [13] have demonstrated the use of neural networks to perform the PCX and vector quantization compression on images for data input, removing the need for feature extraction entirely. Their research was used to automatically compute new features for different targets.

Calibration of the kinematic transforms between the image and the world coordinate system has been examined by many researchers. The calibration-based method has been based on determining the coefficients of the transform using captured from the robot camera pair. Wang [14] described in detail the relationships between the different frames and applied three different methods to approximate the transform, ranging from the known target and position case, to unknown target and position. Horaud, *et al*. [15] described the effect of perspective model on the accuracy of the approximation. Wei, *et al*. [16] outlined an approach for computing the transform based on active vision principles. Zhuang, *et al*. [17] described a system where both the robot and camera were calibrated simultaneously. Remy, *et al*. [18] simplified the estimation by employing Euler representations in the transform.

Nguyen [19] laid out the basic framework for grasp planning and stability in his seminal paper on grasping polygonal shapes. His work is the foundation for model based grasp planning. Ponce, *et al.* [20] expanded Nguyen’s work to curved two-dimensional shapes, and Montana [21] gave definitions of contact stability. Work in model based grasp planning has primarily moved to dexterous manipulation of parts using multi-degree of freedom hands (see Shimoga [22] for an overview). Sensor based grasp planning, which until recently has not received much attention, dynamically determines the grasp of an unknown object based on sensor input. Sensor based grasp planning is generally used in conjunction with visual servoing for vision guided part manipulation. Bard, *et al.* [23] proposed a grasp planning system using a voxel representation of the target. Their implementation placed a heavy computational load on the processor because of the large number of voxels required for accurate representation of the target. They used an octree data representation to reduce the complexity of the system. Janabi-Sharifi and Wilson [24] presented a feature based grasp planner and manipulation system that relies on global features to determine a grasp. Davidson and Blake [25], and Taylor, Blake, and Cox [26] have described a grasping planning system based on B-Spline interpretations of input images. Sanz, del Pobil and Inesta [27][28] have described a planner employing symmetry information to determine suitable grasp points. Stanley, *et al.* [29] have described a system similar to the Sanz planner; however, their system uses a quadtree expansion to determine samples for grasp point determination, and a different set of measurements for evaluating the quality of the grasp. Early stopping is used to generate faster response times.

Numerous researchers have attempted to meld vision guided manipulator motion with vision based path planning. Smith, Brandt and Papanikolopoulos [30] have described a system for manipulating targets based on stereo from motion. Sanz, del Pobil and Inesta [27][28] have created a vision guided robotic system to correspond with their grasp planner, and have demonstrated grasping in four degrees of freedom (DOF). They use assumptions about the workspace to limit the system to 4 DOF, and manipulate the object without a priori models of the targets. Janabi-Sharifi and Wilson [24][31] have created a complete system, which grasps and manipulates objects, building on earlier experiments in grasp planning and collision avoidance. Their system makes extensive use of object models and feature scheduling to perform full 6 DOF part manipulation. Our system is similar to the Sanz, del Pobil, and Inesta system in terms of motion generation and grasp planning.

In this thesis we create a hybrid robot guided vision system that uses generic vision input and grasp planning to manipulate a large class of objects in four degrees of freedom. We compare the results of linear approximation and neural network approximation of the kinematic mapping between the image frame and the world frame. We integrate the entire system to perform sorting of unknown objects. Unlike most previous research, our implementation is generic, model-free and image based.

The core problem in vision guided robotics is balancing the requirement for both accuracy and speed. Accuracy requires that many samples of the target position be taken throughout the motion of the robot. Speed constraints require that few iterations be taken before the robot reaches its target. While the cost of computers and robotic systems is rapidly decreasing it will still be some time before the cost of additional computing power is negligible with respect to the performance gains. To increase the speed of the response, it is necessary to reduce the amount of computation by reducing the number of iterations. A balance between the speed of execution and the accuracy of positioning must be obtained. We propose a system based on a hybrid computed-kinematics and visual servoing system.

The system is capable of performing different types of motions, a neural network based computed kinematics motion where the pose of the target in world coordinates is estimated from the measured coordinates in image space, and an intercept vector for the robot is generated in Cartesian space. Neural networks provide a method of non-linear approximation, which can be used to estimate the transform as a whole rather than the coefficients of the transform matrix. The robot can also perform servoing motions where Cartesian intercept vectors are generated based on a PD control law. The PD motions are broken into coarse and fine motions. Coarse motions are generated by an underdamped PD controller. Fine motions are generated by an overdamped controller.

A second challenge in vision guided robotics is generality, or applicability. The main advantage of vision guided robotics is its ability to adapt to sensed changes in the environment. If that change is a new part, common in many manufacturing applications, the robot should be able to adapt. In order to make the system generic with respect to different types of parts we attempted to design a system that would be valid for a large class of objects, subject to the conditions described in section 2.1. We achieved this by employing generic position measures, which is directly mapped onto the Cartesian coordinates regardless of image shape. This system allowed us to position the robot with respect to a large number of targets; however, to manipulate those targets, a grasp planner was required.

We developed a grasp planner because our target application was discrete part handling. The grasp planning algorithm, like image processing was designed to determine grasps for a class of objects. Furthermore, we attempted to create an efficient grasp planner based on quadtree resolution expansion of the image. By keeping the image in pixel form, we eliminate costly curve fitting algorithms. By employing a quadtree resolution expansion, we eliminate the requirement to consider every pixel on the boundary. The resulting grasp planner can quickly determine an acceptable grasp, but does not attempt to find a globally optimal grasp.

The complete system is capable of manipulating discrete targets in four degree of freedom. Generally, less than ten iterations are required to position the robot within the sensors' accuracy of the target.

1.1 Organization

Section 1 gives an overview of previous research, as well as the motivation and system overview including assumptions about the target functionality of the system. Section 0 describes the system and our assumptions. Section 3 through section 7 outline the theory and implementation of each of the system elements in detail, and give an overview of the overall system architecture. Section 8 describes the experimental procedures and results for various tests performed on the system. Conclusions are presented in section 9.

2 Problem Statement

The system is based on a discrete part manipulation scheme. The workspace is assumed to be locally planar, but the orientation of the plane is not assumed. Targets are assumed to be locally planar as well. A diagram of the system is shown in Figure 1.

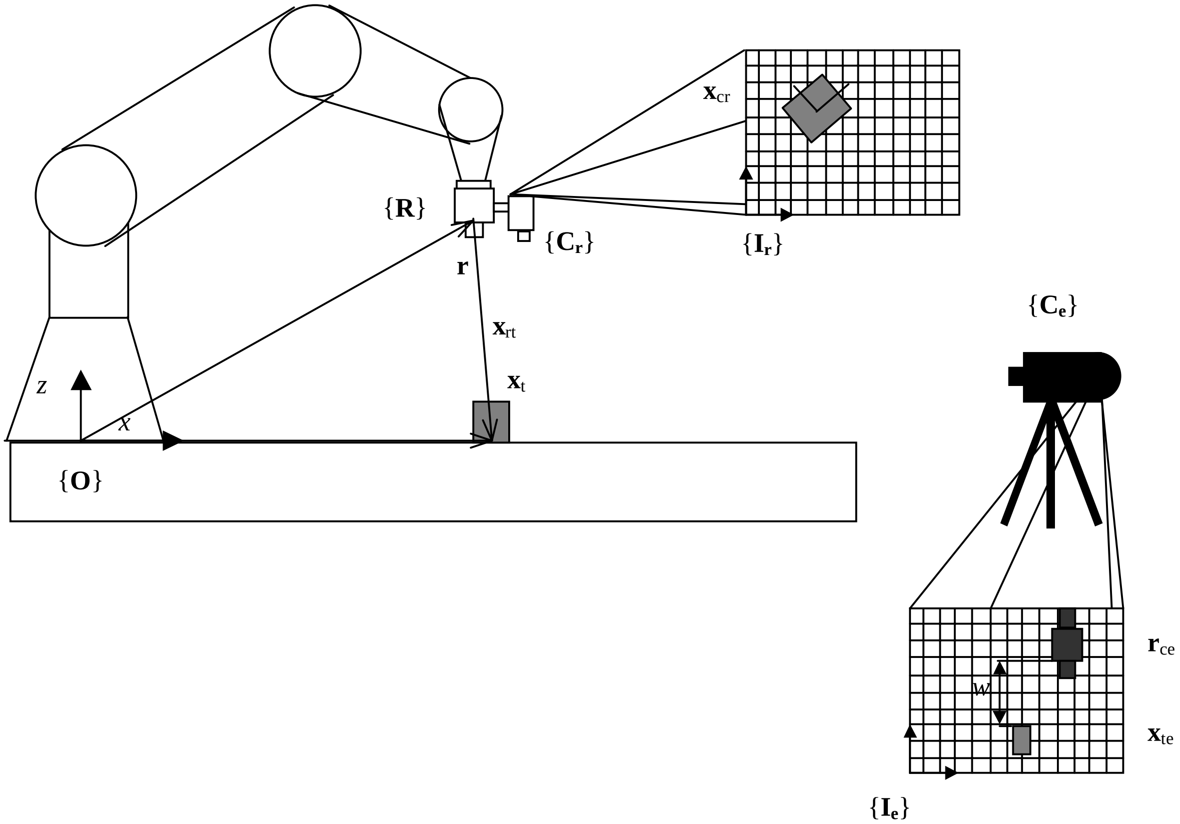


Figure 1: Camera and Robot Configuration for Test Cell

The above diagram shows the experimental configuration of the system, and the important frames and variables. The purpose of the system is to grasp a target at some unknown position $\mathbf{x}_t$ by positioning the end-effector of the robot to a position $\mathbf{r}$ with respect to the target. The position of the robot $\mathbf{r}$ is known from the encoder positions and the inverse kinematics. The target position, $\mathbf{x}_t$, must be estimated from the image systems. The *x*, *y*, and θ components of the target position, $\mathbf{x}_t$ are estimated by measuring the position of the target in $\mathbf{x}_{cr}$ in the end-effector mounted camera frame {$\mathbf{I}_r$}. Because targets may have varying heights, we must also estimate the distance along *z* in the world coordinate frame {**O**}. This measurement is made by examining the image frame of the external camera {$\mathbf{I}_e$}. The difference in y in {$\mathbf{I}_e$} between the observed robot position $\mathbf{r}_{ce}$ and the measured target position $\mathbf{x}_{te}$, generates *w*, the observed difference in world *z*. Using Hutchinson's [1] notation, let $\mathbf{x}_{cr}$ be represented by the three vector:

$$\mathbf{x}_{cr} = \begin{pmatrix} u \\ v \\ \theta \end{pmatrix} \tag{1}$$

Combining $\mathbf{x}_{cr}$ with w as measured in {$\mathbf{I}_e$} we can generate image space position of the target, denoted $\mathbf{x}_{ti}$, as:

$$\mathbf{x}_{ti} = \begin{pmatrix} u \\ v \\ w \\ \theta \end{pmatrix} \tag{2}$$

The problem then resolves to using $\mathbf{x}_{ti}$ to position the robot with respect to the target, such that grasp planning and grasping can occur. This motion can be generated using either a computed kinematics (also called look-and-move) or a visual servoing. In computed kinematics, the position of the target in the world frame, $\mathbf{x}_t$ is estimated using a transform **T** such that:

$$\mathbf{x}_{\mathrm{t}} = \mathbf{T}\mathbf{x}_{\mathrm{ti}} \qquad (3)$$

Determining the transform T is the key focus of [14]-[18]. Because look and move techniques are so sensitive to the calibration of **T**, we also use a visual servoing system. In visual servoing the image position of the target, $\mathbf{x}_{\mathrm{ti}}$, is regulated to a desired position $\mathbf{x}_{\mathrm{tid}}$, such that the three-dimensional position of the target relative to the robot in the base frame $\mathbf{x}_{\mathrm{rt}}$ is known.

$$\mathbf{x}_{\mathrm{rt}} = \mathbf{r} - \mathbf{x}_{\mathrm{t}} \qquad (4)$$

That is, if

$$\mathbf{x}_{\mathrm{ti}} = \mathbf{x}_{\mathrm{tid}} \qquad (5)$$

then Eq. (4) must hold. Given that the robot is positioned such that Eq. (4) holds, we can generate a grasp from the image as described in section 5, and manipulate the target.

2.1 Assumptions

In an effort to make our system generic, we have tried to limit the number of assumptions about the target, and eliminate target models completely. Our targeting system is based entirely upon sensed data. However, without specific object models to refer to, we needed some new boundaries to separate targets from background and noise. We made the following assumptions with respect to the target object:

1. *There is good intensity contrast between the target and the background.* Without specific models to guide us, we must assume that we can sense the entire object. Without adding too much complexity to the segmentation algorithm (which is a major research area in itself) we have assumed that the object can be distinguished from the background based on intensity alone.

2. *The target has limited contours on its top surface.* When performing the grasp-planning algorithm (described in section 5) we assume that the target is planar. This is not strictly required, but the variance in the top surface must be less than half the length of the manipulator fingers for grasping.

3. *Target dimensions less that (10 cm)^3.* This constraint is governed by the size of our manipulator and the camera's field of view. With a camera mounted on the end effector there is a certain maximum area on the target surface that is visible. If the target cannot be contained in a cube 10 cm to a side, the system must be accurately placed over the target initially to ensure that the object is at least partially within the field of view. The requirement for accurate initial positioning negates the purpose of vision guided robotics.

4. *Piecewise continuous boundary (a real object).* Essentially, the target must be a real object. Degenerate objects with single dimensional features are not permitted. An example of a disallowed object is shown in Figure 2.

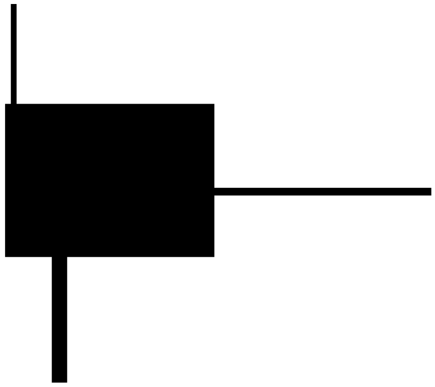


Figure 2: Disallowed Degenerate Target

5. *Only one part is visible at a time.* While it would not be difficult to create a system, which allowed sorting through multiple singulated targets, we felt that such development was outside the scope of the current project.

6. *Parts are rigid in projection.* While this assumption is not entirely necessary (results for targets that deform in projection are presented in section 8.1.3) we assume that the object does not change shape as the robot approaches it.

7. *Targets are stationary.* As of the time of this writing, the bandwidth of the system is too slow (~1.5 Hz) to perform successful interception tasks. It is possible to track an object, but not to intercept it. In section 9.2, plans to perform target interception are described.

Essentially, we have assumed that the parts will be small, discrete, rigid, and stationary. Objects commonly found in industrial assembly or office environments are all suitable targets.

3 Computer Vision and Image Processing

In image processing an image is captured and discretized as a set of picture elements or pixels. These pixels can then be processed to extract information from the target [32]. The key component of digital image processing is dimensionality reduction. A standard digital image is 640 pixels by 480 pixels with a color depth of 24 bits (8 bits per color for red, green, blue encoding). This leads to over 7 million bits of information for each frame. Of those seven million bits, only a tiny fraction of a percent, is actually relevant. For example, in servoing a robot, the number of input values must at least equal the number of degrees of freedom. For a six degree of freedom robot, sufficient accuracy can be obtained from 6, 32-bit variables, or 192 bits of information. This means that the input image must be reduced from over seven million to 192 bits or a compression of over 36000 times. Percentage wise, the input image must be reduced to approximately 0.003% of its original size. It is difficult to reduce the dimensionality using compression alone. It is much more efficient to isolate the target in the image, and express its position as a combination of the positions of its pixels.

In particular, in vision guided robotics we are interested in isolating an object (hereafter referred to as the target) in the image, and extracting its properties. Targets have two classes of properties that we are interested in: intrinsic properties and extrinsic properties. Intrinsic properties are features inherent to the object itself such as its area, color, and dimensions. Extrinsic values are the position and orientation of the target in space with respect to some base frame.

There are two major components to our vision system: target isolation and description, and boundary isolation and description. The target isolation component is used to determine the extrinsic and intrinsic properties of the target based on area measurements. The boundary description is used to extract and encode the shape of the target boundary, for determining stable grasps.

3.1 Region Segmentation

The first step in the analysis of the image is region segmentation. Region segmentation is used to determine which image areas are locally similar. Having segmented an image into regions, it may be possible to classify the regions as foreground and background. The most common method of segmentation is intensity-based thresholding.

Thresholding reduces the number of gray scales used to represent the image. The reduction of gray scale values can often eliminate extraneous information or noise from an image. A thresholded image is usually reduced to a binary form (black and white) to accentuate certain features and reduce the computation required for scene analysis. Thresholding is often based on the intensity or gray scale histogram of the image

The simplest thresholding operation is mapping an image from a gray scale image to a binary image. The image is scanned left to right and top to bottom. Any pixel with a gray scale value greater than the threshold is changed to white, and any pixel with a gray scale value less than the threshold is changed to black, as shown in Eq. (6). A more general version of the thresholding operation is shown in Eq. (7). The operation will turn a pixel white if it belongs to set *F* (the set of allowed grayscale levels) and black if it does not. This type of threshold is useful for applications where the image may contain more than one gray scale.

$$I_{ij} = \begin{cases} 1 & If\ I_{ij} > T \\ 0 & If\ I_{ij} < T \end{cases} \tag{6}$$

$$I_{ij} = \begin{cases} 1 & If\ I_{ij} \in F \\ 0 & otherwise \end{cases} \tag{7}$$

The thresholding operation is useful in a number of situations. If an object has a high degree of contrast with the background, a thresholding operation can effectively segment an image [33]. Thresholding can also be used to remove noise from an image after an edge detection algorithm has been employed.

After the image has been thresholded, there are several regions of light and dark areas in the image. A clustering or labeling algorithm must generate a list of objects from these labels. We use an 8 connected brushfire method of region generation [35]. The following algorithm outlines the brushfire expansion. The algorithm uses two buffers, one for pixels, which are part of the object but not analyzed, and another for pixels, which are part of the object and have been analyzed.

```
While there are still pixels in the image.
        Find the first non-background pixel and add it to the current buffer and create a new
        image buffer.
        While there are still pixels in the pending buffer.
             Take out a pixel and add it to the object buffer
             For each neighbor of the current pixel
             If the neighbor is foreground, add it to the pending buffer and change the
             original pixel in the image to black.
```

The result of this algorithm is an array of buffers containing sets of pixels that are eight connected. Intrinsic and extrinsic features are then calculated for each of these buffers.

In the system, we use a two level approach to region segmentation and target identification. The image is first thresholded to eliminate the background, and then a blob analysis is performed to generate regions of interest. From these regions of interest, three targets must be identified: the target in the robot frame, the robot, and the target in the external frame. Three methods are used to isolate the target:

1. *Size Filter:* All regions less than a predefined area are considered noise and ignored.

2. *Shape Filter:* Often specular reflection from ambient light can create artifacts large enough to pass the size filter. These artifacts are eliminated by looking at the perimeter to area ratio of the target. Artifacts tend to be noisy, and have chaotic boundaries,

which give rise to large perimeter to area ratios. Actual targets tend to have much cleaner lines.

3. *Position filters:* In the case of the robot end effector, we have a secondary measurement of its position, the encoder position of the end-effector, **r**. We can feed this information back to the image processing system to estimate the current robot position, $\mathbf{r}_{ce,}$ in the external camera frame {$\mathbf{I}_e$}. The targets outside this estimated region can be eliminated [33].

3.2 Feature Extraction

Once the regions have been segmented, the properties of each region must be determined. Each region of connected pixels is analyzed for intrinsic and extrinsic features. The extraction of the characteristics of each blob or region is generally referred to as feature extraction. The following features were extracted for each foreground region:

1. *Area*: the number of pixels in the region.

2. *Perimeter*: the number of 8-connected pixels in the boundary.

3. *Compactness*: the ratio of Area/Perimeter.

4. *Center of Area X*: the *x* coordinate of the first moment of the region in the image frame.

5. *Center of Area* the *y* coordinate of the first moment of the region in the image frame.

6. *Inertial Angle*: The angle of equal area passing through the center of the area.

7. *Left coordinate of the bounding box.*

8. *Right coordinate of the bounding box.*

9. *Top coordinate of the bounding box.*

10. *Bottom coordinate of the bounding box.*

11. *The distance between the part and the end-effector (*w*) in the external frame* $\{\mathbf{I}_e\}$.

These features are extracted and then placed in a feature vector. The area, perimeter and compactness are intrinsic features that are used to identify targets, and distinguish them from background noise. The area is used to determine if a region corresponds to a target or background noise. While we have endeavored to make a generic system, we do make some simple assumptions about the size of the target. The target's size is stored in a structure, depending on the type of target, and the camera observing it. Targets are generally considered have an area from 100 to 20000 pixels, eliminating noise and camera errors, where the entire image is saturated.

The perimeter and compactness are used to determine the shape of the object. General knowledge of shape is important in determining the target's orientation. A more thorough explanation of target orientation is given in the next section. The compactness is used to determine if the target is a circle. As the target becomes more circular, the compactness goes to 1. Any target with a compactness greater than 0.8 is considered a circle. Because a circle has no orientation, the angle is always set to zero. To determine if the target is a square the area is compared with the perimeter. For a square, the square root of the area is equal to one quarter the perimeter. The ratio of these two values determines the squareness of the current blob.

The position in *x*, *y*, *z* and θ are determined using the center of mass, vertical distance and inertial angle, respectively. For targets with high degrees of symmetry, the inertial angles are not unique, and must be resolved using a different angular measurement.

3.2.1 The Notion of Angle in Moment Measurements

The angular coordinate of the target (its rotation about *z*) is one of the more crucial measurements for servoing to the target. Normally the rotation in the image plane is defined as the orientation of the major inertial axis. The major inertial axis passes through the center of area and has the same number of pixels on either side. In essence, the inertial angle bisects the area of the target. However, for symmetrical objects, this axis is not unique. In fact for every degree of symmetry, there is an inertial axis. A secondary measure of orientation is used for symmetric objects.

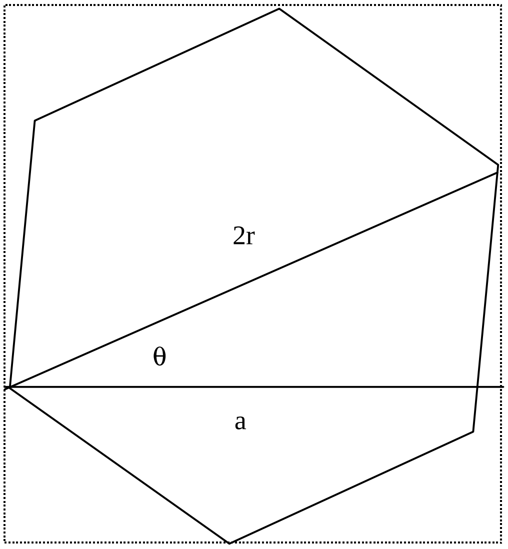


Figure 3: Derivation of Orientation for an n-polygon

The original calculation was based on the ratio of a side to the area using angles, and can be derived from the diagram in Figure 3.

$$A_{npoly} = nr^2 \sin\left(\frac{180}{n}\right)\cos\left(\frac{180}{n}\right) \tag{8}$$

$$\cos(\theta) = \frac{a}{2r} \tag{9}$$

$$r = \sqrt{\frac{A_{npoly}}{2n\sin\left(\frac{360}{n}\right)}} \tag{10}$$

$$\cos(\theta) = \frac{a}{2\sqrt{\frac{A_{npoly}}{2n\sin\left(\frac{360}{n}\right)}}} \tag{11}$$

where *n* is the number of sides in the polygon, *r* is the internal radius of the polygon, *a*, is the width of the bounding box and A_{npoly} is the area of an n-symmetrical polygon. By using the above equation we can derive the angle of any symmetrical object given its area, the area of the bounding box, and the number of degrees of symmetry in the target.

3.3 Boundary Isolation

While area measurements can yield interesting information about the size and position of the target, they do not adequately capture shape information required for grasp planning. A grasp planning algorithm will require a representation of the target's boundary. Most grasp planning algorithms use geometric representations of the boundary [19]-[26]. However, while they are more general, they are more computationally intensive than local approaches because accurate curve fitting is an expensive process. To reduce computational complexity, we have used a purely pixel based grasp planner which uses an efficient representation of the boundary to speed computations. The representation has the following desirable properties:

1. *Representation of interior and exterior of the object.* Besides knowing where the object's boundary is located, we also must know what is inside and outside the target.

2. *Representation of the boundary.* The boundary representation should make angular feature calculation efficient.

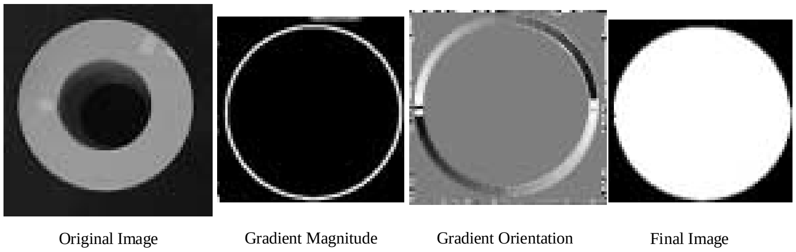


Figure 4: Output of the Vision System for Grasp Planning

The background is represented as black pixels (value 0), the interior is represented by white pixels (value 255) and the boundary is represented by the gradient orientation, which is analogous to the surface normal. The gradient orientation is represented as an angle from 0 to π, discretized on the grayscale interval 1 to 254. The result is a black and white image surrounded by a grayscale shell with a period of two as shown in the "Final Image" portion of Figure 4. This representation allows the grasp-planning algorithm to identify easily the edge, interior and exterior and also to find the surface normal using only a lookup operation.

We use a standard hole elimination procedure to ensure that the grasp planner only generates grasps spanning the exterior boundary of the target. Our planner does not consider interior grasps. The negative of the original image is generated, and the background inverted, so the only remaining elements are holes. The original is then added to the new image to generate a solid target.

The representation in Figure 4 is a combination of the area of the target as determined by a grayscale blob analysis and the boundary of the target as determined by the gradient direction. The gradient direction is analogous to the surface normal, an important measure for the determination of the stability of the grasp. The gradient and gradient direction are determined by nine element Sobel operators where the gradient magnitude is given by [34]:

$$g = \sqrt{\left(g_x^2 + g_y^2\right)} \tag{12}$$

and the gradient orientation is given by:

$$\theta = \tan^{-1}\left(\frac{g_y}{g_x}\right) \tag{13}$$

The gradient measures are based on the derivative of the grayscale; therefore the gradient magnitude and orientation are susceptible to noise. In order to reduce the noise, multiple iterations of a smoothing filter are applied to the holeless image before the gradient orientation is calculated.

The use of the gradient orientation to represent the boundary is critical for the speed performance of the system. Most of the features used to determine the grasp quality in our system are directly based on the surface normal. During the grasp-planning phase of operation, only simple look-ups, additions and subtractions are required to compute all the features.

3.4 Multiple Cameras and the Correspondence Problem

Generating three-dimensional information about the target is perhaps the most difficult problem in vision guided robotics, because computer vision inherently involves projecting a three-dimensional scene onto a two dimensional surface. Sensor based approaches for three-dimensional reconstruction include stereo vision, range imaging, and ultra-sonic sensors which generate range information to complement the two dimensional information from the original camera. Algorithmic solutions generally involve a model of the target, which can be interpreted via the projection equation to determine depth. The model-based approach, while much simpler, is less accurate and requires three-dimensional models of all the possible targets. Sensor based methods are more generic, but much more computationally intensive. The key problem of a multisensor depth extraction is the correspondence problem. The correspondence problem revolves around the question

“Given that the kinematics are not precisely known, how does value *a* from sensor *x* correspond to value *b* from sensor *y*?”

The approach to solving the correspondence problem depends on the configuration of the system in question. For cameras with similar image plains (usually translated in along a line) the stereoscopic transform can be applied. While this transform is very generic, the resolution of the depth strongly depends on the distance between the cameras and their resolutions.

$$\hat{z} = \frac{f_c b}{|x_1 - x_2|} \tag{14}$$

where f_c is the focal length of the camera, b is the interoccular distance, x_1 and x_2 are the correlated points in each image and z is the estimated distance.

For mixed mode three-dimensional measurements such as vision accompanied by a range sensor such as sonar or ladar, the challenge becomes correlating pixel to pixel. While this approach gives much better results than stereo vision, the correlation problem is much more difficult because the targets have very different responses. One returns a bitmap corresponding to color; the other returns a bitmap corresponding to distance. Most approaches require a strict calibration between the range and vision sensor, so correspondence can be determined using the kinematic relationships and sensor models.

Our approach stems from the original constraints we have placed on the system. We use the fact that the target is roughly planar on top, and the robot is only required to servo in four degrees of freedom to use a second camera to determine the depth. Two cameras are placed at orthogonal viewing points as in Figure 1. The camera mounted on the end effector generates *x*, *y*, and θ information, and the external camera generates information on the distance from the manipulator to the target in *z*. In essence we are using the target itself to determine the correspondence between the image frames. By isolating the target in each frame, the system can directly determine the *x*, *y*, *z*, and θ positions of the target.

Our approach has several advantages. First, it allows us to use the same set of algorithms for determining the *x*, *y*, and *z* coordinates of the target. The only value that changes are the intrinsic properties of the different cameras such as focal length, and zoom. Our approach also deals with the correspondence problem at a higher level, reducing the computational complexity. Because our cameras are orthogonal there is no coupling in the image/world transform, dramatically simplifying the mathematics. The major disadvantage of our approach is that it is inappropriate for non-planar targets. Servoing on planar targets in six degrees of freedom using the same approach is possible with the addition of another camera along the *x*-axis.

4 Vision Guided Robotics

Robotic automation systems are an integral part of most modern factories. However, robots are impeded by their requirements for rigid target positioning. By adding sensors such as vision to the system, the constraints on the target can be relaxed. There are two classes of vision guided robotics: look and move (computed kinematics) methods and visual servoing methods. Look and move methods encompass all the calculated kinematics methods. In look and move systems, the pose of the target is calculated from the vision system and the robot is commanded to move to a position in Cartesian space relative to the target. Visual servoing methods encompass all methods that use control theory to regulate the image or robot position to a given offset with respect to the target. We have implemented a hybrid vision guided robotics system, which uses both look and move and visual servoing approaches.

4.1 Look and Move

In the look and move class of vision guided robotic systems, the pose of the target is calculated from the image, and the robot is commanded to move to some point in space with respect to the target. In order to calculate the pose accurately, the kinematics between the camera and the robot must be known with a high degree of precision, and the camera parameters must also be known with a high degree of accuracy. Several adaptive selftuning schemes have been suggested [14] - [18]. Figure 1 shows the relationships between the different kinematic frames in a look and move system.

We calibrated *u*, *v*, and *w* of $\mathbf{x}_{ti}$ with respect to the *x*, *y*, and *z* coordinates of $\mathbf{x}_t$. Because image θ and world θ, are equal, no calibration was required. For a sufficiently distant viewpoint, the relationship between translations of the camera in *x* and *y* and the translation in the image is approximately linear for small yaw and pitch errors. The value for θ was determined by measuring the drift in one variable while holding the other constant. The angle θ was determined as the arctangent between the drift and the magnitude of the move. This calculation assumes that the errors in pitch and yaw are small when compared to θ. Graphs of the raw and compensated values for x are shown in

Figure 5 and Figure 6.

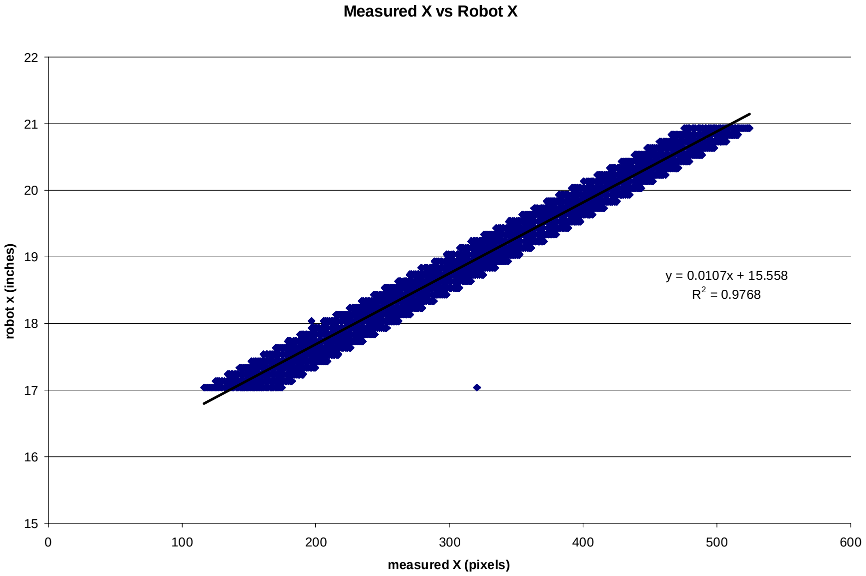


Figure 5: Raw x Calibration Data

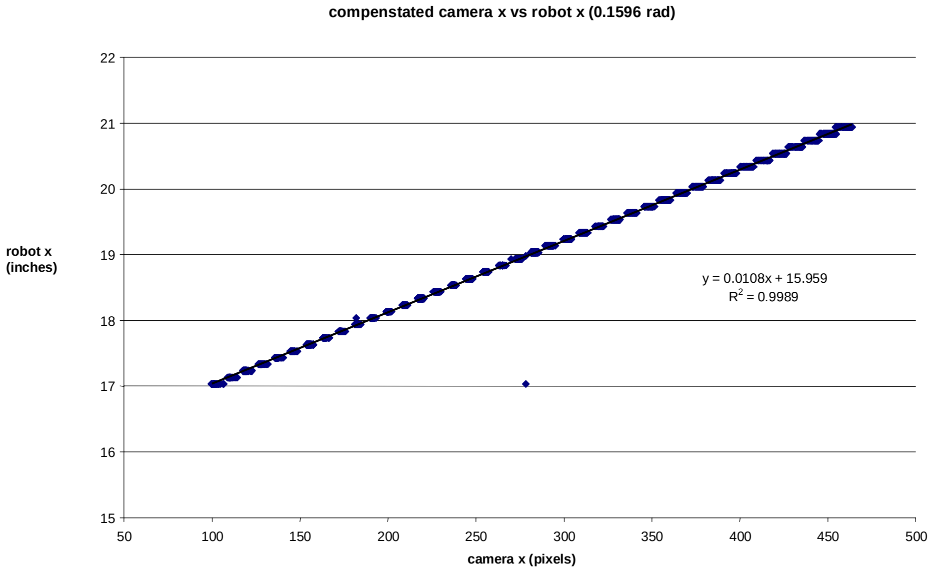


Figure 6: Compensated x Calibration Data

Note the tighter distribution in the compensated figure. The value substituted into the transform is given by the slope of the line. Figure 7 shows the derivation of the angular error *θ*, based on a grid of data in *x* and *y*.

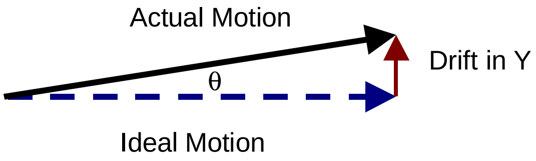


Figure 7: Derivation of Angular Error

For a series of robot motions in *x* where *y* is held constant, the *x* should similarly be constant in the measured coordinates in the camera frame. By measuring the drift, one can estimate *θ*.

While this form of calibration is sufficient for planar motion (generating final position errors of approximately +- 5 pixels) when *z* is added as a variable, the performance begins to degrade. A graph of the behavior of the system using the previously calculated θ correcting with the same range of *x* and *y* and a variable *z* is shown in Figure 8.

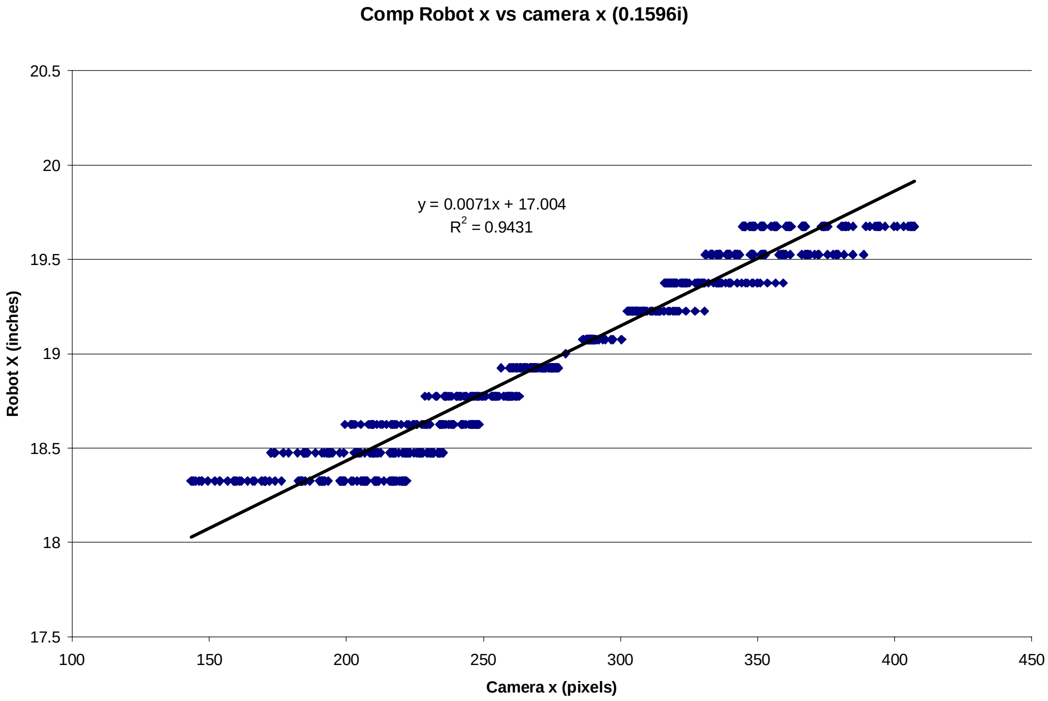


Figure 8: Compensated x Calibration Data with z Motion

It should be apparent from the image that the calibration is no longer accurate enough to generate a reliable position with respect to the target. There are two sources for this error. First, *x* and *y* vary inversely with *z*, so errors increase as z decreases. Second, the sensitivity to pitch and yaw errors increases closer to the target. Instead of a simple linear transform, the system is governed by Eq. (15).

$$\mathbf{x}_{\mathrm{t}} = \mathbf{T}\mathbf{x}_{\mathrm{ti}} \tag{15}$$

Where **T** is the transformation matrix between the image and robot frames. The only method for resolving these issues within a look and move framework is much more accurate calibration. Now all six variables of the calibration matrix must be derived. However, these variables are coupled nonlinearly, so the problem becomes a non-linear programming problem [14].

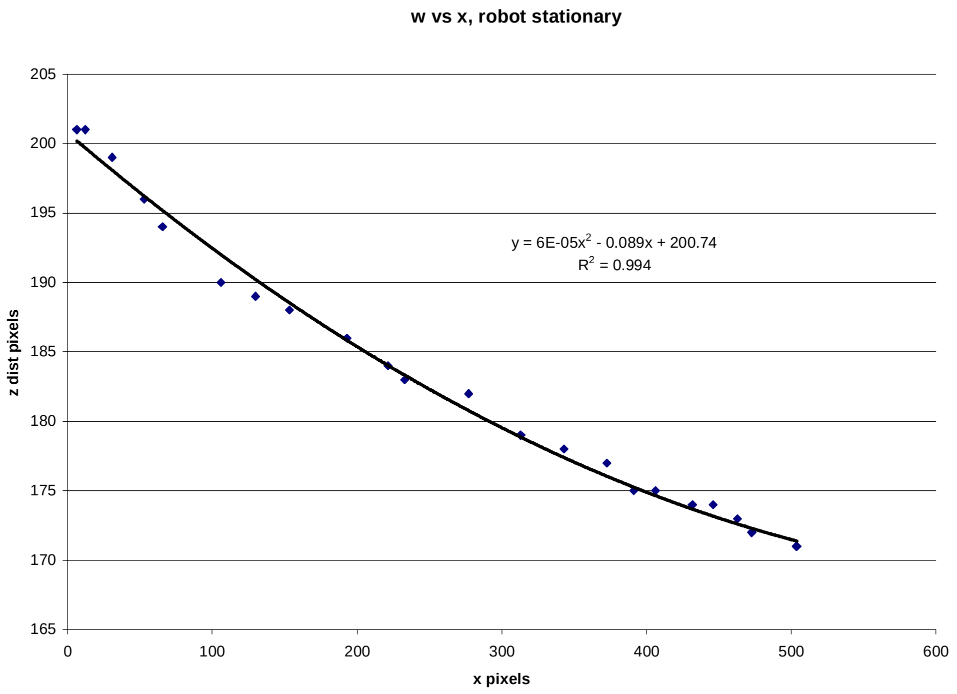


Figure 9: Depth Compensation Measurement

The graph in Figure 9 shows the measured distance from the target in *x* with respect to the measured height of the target. Since we know the height of the target is constant, and we assume the camera is approximately orthogonal to the target, the apparent change in height must be due to the perspective transform. In order to estimate the distance from the external camera in the world, we used the *x* coordinate of the end-effector-mounted camera, because *x* in the end-effector frame is depth or *z* in the external camera frame (see Figure 1). Our attempts to compensate for the target's change in height by directly applying the pinhole perspective model failed. The pinhole model of perspective described by Horaud, *et al.* as [15]:

$$x = c\frac{u}{z} \qquad (16)$$

where c is a constant of proportionality. While the addition of the perspective added some degree of improved performance it did not completely compensate for the change in perspective. This was probably due to several factors including:

1. The measurement of *x* by the end effector camera is not precise.

2. The external camera is not entirely orthogonal to the part.

3. The external camera is not completely parallel to the *yz* world plane.

These factors contribute their own error to the apparent height of the target, causing the perspective transform alone to be insufficient. However, from the graph in Figure 9, we noted that the change in depth could be approximated in the local workspace of the robot by a second-degree polynomial:

$$w = 6.0\times10^{-5}u^{2} - 0.089u + 200.75 \qquad (17)$$

This polynomial approximates the more complex rational sinusoid. The maximum variance of this approximation is approximately +/- 5 pixels, adequate for the current system's needs.

Most computed kinematic systems rely on some form of non-linear programming to perform the approximation of the kinematic mapping. We have taken two approaches to this problem, first, to employ a neural network approximate **T**, and second, have added a visual servoing component. Visual servoing systems are less sensitive to calibration errors, so we have adopted a secondary visual servoing system to perform the final positioning of the robot.

4.2 Visual Servoing

Visual servoing encompasses all the approaches for vision guided robotics where a control law is used to position a robot with respect to a target. Control laws range from simple proportional control laws to very complex adaptive schemes [1] – [13]. The control law can be phrased in terms of the image, or in terms of a position in space. In the first circumstance, image based visual servoing; the control law directly minimizes the error between the current image and the desired image. In the second, Cartesian visual servoing, the pose of the target is calculated and the relative pose between the robot and the target is regulated. The control law can also be directly run on the robot or run with the original robot controller still in the loop. Our visual servoing system is an image based, PD controller run with the robot joint controllers in the loop.

The control system is based on a classic PD controller structure as shown in Figure 10,

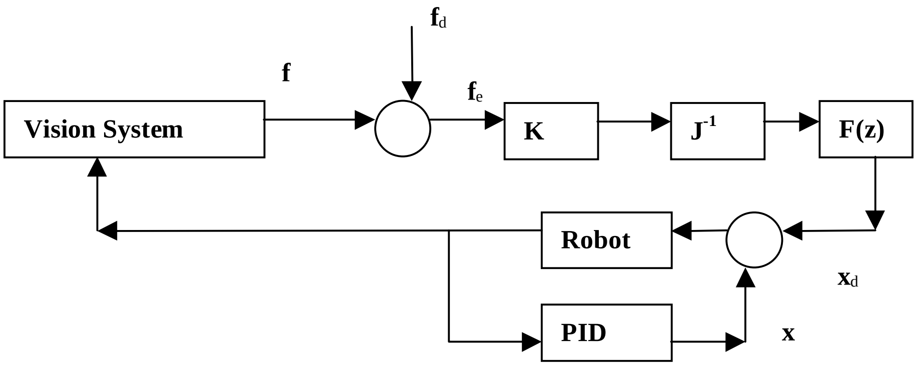


Figure 10: PD Control System

where x_d is the desired position x is the measured position, f is the current feature vector, f_d is the desired feature vector, f_e is the current feature error, **K** is the gain matrix, $\mathbf{J}^{-1}$ is the inverse feature Jacobian and *F(z)* is the discretization function. There are several sources for error in the diagram: first there are possible errors in the features measured by the vision system; second there are potential errors in the model of the feature Jacobian; third there are errors introduced by discretizing the control output; fourth, there are errors in the robot kinematics and PID controllers.

The “plant model” in this case is the inverse feature Jacobian. It is a mapping from the input feature space to the output Cartesian space. It is not a complete plant model because it does not include the inverse kinematics of the robot or the control of the robot joints that are part of existing robot controllers. The Jacobian actually maps changes in input to changes in output, or in its most pure form, image errors to Cartesian velocities. The Jacobian is derived from the geometry of the system.

Given the features to be extracted, and making the assumption that all objects are seen as parallel projections (no perspective distortion), the mapping between each degree of freedom of the target and their projection in the plane of the camera is given by a constant scale factor. In the simplest case, where there is no motion in the *z* plane the Jacobian can be expressed as the derivative of

$$\dot{\mathbf{x}}_{\mathrm{t}} = \begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & c_\theta \end{bmatrix} \dot{\mathbf{x}}_{\mathrm{ti}} \qquad (18)$$

The position of the end effector of the robot is directly related to the position of the projection of the target in the camera frame (note that c_θ is usually 1). If the robot can move in *z* as well as within the plane, then the scale factors become functions of *z*, which makes the system nonlinear. However, it is still relatively simple. Assuming a pinhole camera, the projection can be given as in Eq. (16)

$$u = \frac{c_x}{z} x \qquad (19)$$

$$v = \frac{c_y}{z} y \qquad (20)$$

The result is a system of coupled non-linear equations, which while less straightforward than the strictly linear case are still possible to solve for a control law. In order to create the control law, the image Jacobian and its inverse must be calculated. The most straightforward method for doing so is outlined in Hutchinson, *et al* [1]. Defining the motion of the camera as

$$\dot{\mathbf{C}} = \Omega \times \mathbf{P} + \mathbf{X} \tag{21}$$

where **C** is the speed of the camera, **Ω** is the rotational velocity of the end-effector, **P** is the position of the camera with respect to the robot end-effector and **X** is the translation velocity of the end-effector. Given that we have constrained servoing to only 4 DOF, ω_x and ω_y are zero therefore:

$$\dot{x} = \frac{vz}{c_{proj}}\dot{\theta} + c_x \dot{u} \tag{22}$$

$$\dot{y} = \frac{uz}{c_{proj}}\dot{\theta} + c_y \dot{v} \tag{23}$$

$$\dot{z} = T_z = c_z \dot{w} \tag{24}$$

$$\dot{\theta} = \dot{\theta}_t = \omega_z \tag{25}$$

These equations yield the following Jacobian.

$$\dot{\mathbf{x}}_{\mathrm{ti}} = \mathbf{J}\,\dot{\mathbf{x}}_{\mathrm{t}} \tag{26}$$

$$\begin{bmatrix} \dot{u} \\ \dot{u} \\ \dot{w} \\ \dot{\theta} \end{bmatrix}_{image} = \begin{bmatrix} \frac{c_x}{z} & 0 & -\frac{u}{z} & -v \\ 0 & \frac{c_y}{z} & -\frac{v}{z} & u \\ 0 & 0 & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_z \end{bmatrix} \tag{27}$$

where *u* is the *x* coordinate of the image frame, *v* is the *y* coordinate of the image frame, and *w* is the *z* coordinate of the external image frame. The inverse of this upper triangular matrix gives the inverse Jacobian that is the plant model for this controller. We have assumed that there is a unity relationship between the observed and actual angle, which should hold true for most cameras. The above derivation holds for a camera attached off the center of rotation with a constant zoom and a pinhole model. This is an example of a classic Cartesian feature Jacobian for four degrees of freedom. Solving Eq. (27) for the inverse Jacobian, we obtain:

$$\mathbf{J}^{-1} = \begin{bmatrix} \frac{w}{c_x c_z} & 0 & \frac{u}{c_x c_z} & \frac{wv}{c_x c_z} \\ 0 & \frac{w}{c_y c_z} & \frac{v}{c_y c_z} & -\frac{wu}{c_y c_z} \\ 0 & 0 & \frac{1}{c_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{28}$$

The measurement of *z* in the inverse Jacobian is expressed in terms of *w* because we do not know the distance from the manipulator to the top of the target.

4.3 Gain Scheduling

Control strategies generally must balance the speed of convergence with the number of iterations required to converge. While higher gains tend to converge faster, they can lead to oscillation and even limit cycling. Because of the low bandwidth of our system, it is particularly susceptible to this issue. Because each iteration is computationally expensive, we must reduce the number of iterations to converge to a solution in a timely fashion. However, because we wish to position the robot as accurately as possible, the steady state errors of a limit cycle are highly undesirable. To achieve a rapid response, yet achieve a stable solution, we have employed gain scheduling for our PD controller.

The gain scheduling system uses two gain levels, a high gain approach, and a low gain positioning, to achieve a fast and accurate final position. Like most high gain PD controllers, our PD controller is characterized by an underdamped response. The purpose of the gain scheduling system is to reduce the oscillations when the error is small. The different gains are summarized in Table 1.

Table 1 : Gains

Name	High Gain	Low Gain
P_x	8.0	4.0

P_y	10.0	5.0
P_z	4.0	4.0
P_θ	2.0	2.0
D_x	3.5	1.75
D_y	3.5	1.75
D_z	2.0	2.0
D_θ	1.0	1.0

The lower gain system removed the limit cycling behavior, which primarily occurred in *x* and *y*, and occasionally caused a coupled effect on θ. *Z* and θ posed no limit cycling problems individually. By halving the proportional and derivative gains for the controller in *x* and *y*, we were able to eliminate limit cycling. The ratio between the controller gains in *x* and *y* are approximately the same as the ratio between the *x* and *y* dimensions of the image. The gains were adjusted so a similar percent error would evoke a similar response and *x* and *y* would converge at approximately the same rate, given the same initial conditions.

The high gain system is executed after the initial computed kinematics move. The high gain system continues until the target is within a 15-pixel square of the center of the image. The high gain matrix is then swapped with the low gain matrix. The lower gain matrix has an overdamped response, which drives the robot to its final position smoothly. A good example of the response of the system is shown in Figure 11.

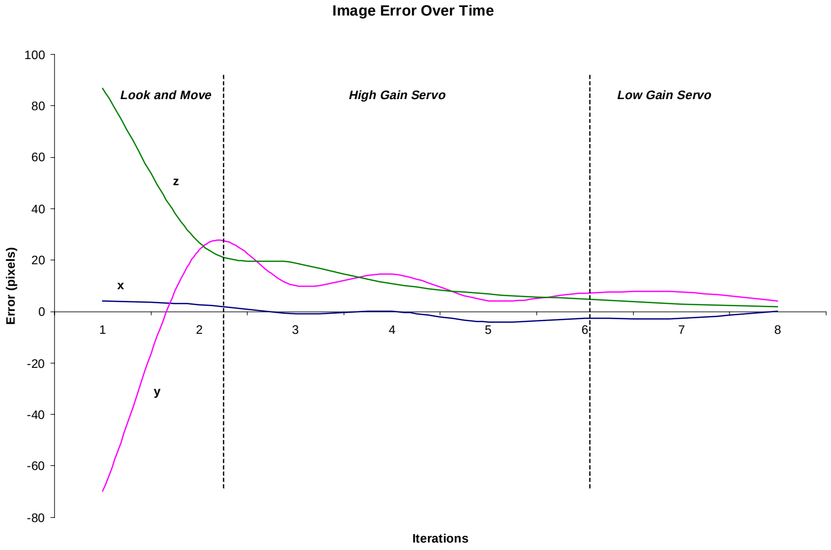


Figure 11: Multi-phase Response for the Hybrid System

As is apparent from Figure 11, there is a marked difference in system response for the three stages of motion. The look and move portion is a linear point to point motion, the high gain servo is characterized by a rapid response and oscillatory underdamped behavior. The low gain servo is more typically underdamped and moves the robot smoothly to its final position. The three stages of motion are very apparent from studying the *y* response.

5 Grasp Planning

Grasp planning has been studied extensively since the mid-1980's. The basic premise of grasp planning is automatically generating a mechanically stable grasp from a model or measurement of an object. The fundamental method for doing so was pioneered by Nguyen [19]. Essentially the grasp directions must lie within the friction cones of the two fingers as shown in Figure 12.

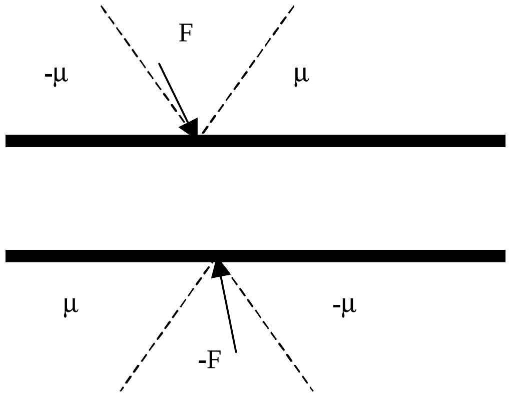


Figure 12: Friction Cones

Nguyen gave an algorithm for determining the largest segment for stable grasps on a polygonal target. His work has further been extended to curved surfaces. Our algorithm is generic and image based. While image based methods have been proposed before, our approach is efficient because we eliminate costly geometric reconstruction and determine the grasps directly from the pixelized image. Previous approaches have used models of the targets, performed geometric primitive fitting, or examined every pixel on the boundary.

We have used a quadtree sampling approach that performs a resolution based breadth first search. Our approach is valid for determining two-fingered grasps with friction of two-dimensional targets.

5.1 General Approach

The algorithm is based on a quadtree resolution expansion of the image. At each level, pixels corresponding to the *x* and *y* midpoints of the current quadtree windows are examined for satisfactory grasp points. If no suitable grasp points are found, the cells of the quadtree are divided, and the procedure is repeated. Because a breadth first search is used, the algorithm has more in common with sampling than searching. The final resolution of the image never alters, just the number and position of the samples taken. If the boundary were stretched along a line, the final result would be a discrete sampling where the frequency quadrupled for every expansion. To determine the suitability of a grasp point, a fitness function is used. The fitness function is a linear combination of features that represent the quality of the current grasp. The fitness function combined with the breadth first search finds the best grasp point at a resolution level. The behavior of the system is tied closely to the fitness function; the fitness function encodes what is considered a good grasp point and what is considered a poor grasp point. Figure 13 illustrates the various components of the grasp planner, including the quadtree expansion, the fitness function features and the definition of a normal.

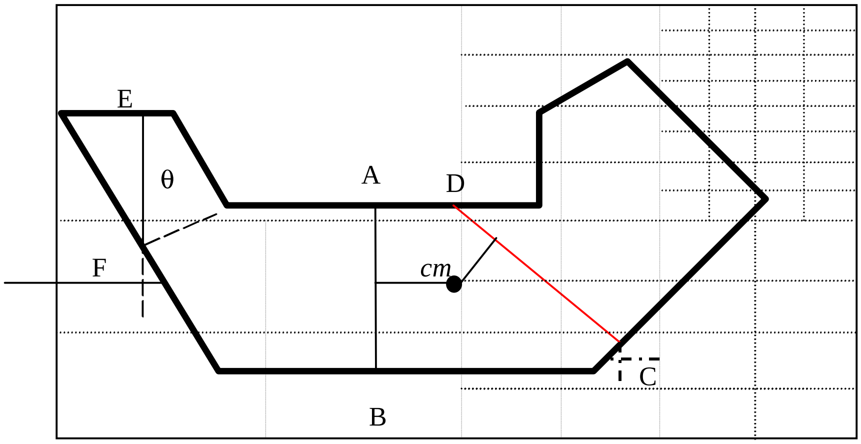


Figure 13: Example Object Expansion and Features

The above figure shows the quadtree expansion and the different features used in calculating the fitness function for the grasp analysis. The dotted rectangles show the quadtree decomposition of the image. At each resolution level the midpoints of the x and y-axis of the window are sampled. Cells not containing edge elements are discarded. The dashed lines above and to the left of C show midpoint sampling. The letter pairs illustrate potential grasp points for analysis.

5.2 Overall Algorithm

The overall algorithm can be described by the following pseudocode.

```
While no suitable grasp point has been found and maximum resolution has not been reached.
    Expand the quadtree (quadruple the sampling rate)
    Execute the fitness function for each node
```

From the pseudocode we observe that there are two critical algorithms. The quadtree expansion algorithm uses the main frame buffer of the composite edge orientation image to create a new set of pointers to the images at the midpoints of each new cell. The fitness function involves several steps:

1. The local features must be calculated,

2. The opposite grasp point must be determined,

3. The composite features must be determined,

4. The fitness function must be calculated.

5. The physical validity of the grasp must be verified by collision checking.

Most of the computational effort of the system goes into evaluating the fitness function.

5.3 Quadtree Expansion or Sampling Algorithm

The quadtree expansion or sampling algorithm is responsible for generating the pixels to be examined at the current sampling frequency or window size. The sampling occurs by generating increasingly smaller windows of the image containing portions of the boundary. The *x* and *y* midpoints of the window are then sampled to generate the current pixels for analysis. By choosing the midpoints of the current window, we ensure that few points are visited twice. The expansion forms a tree of the image, where the nodes on the tree are windows that contain pieces of the boundary. The pseudocode for the algorithm follows:

```
For all the nodes in the lowest level of the tree
```

```
Divide the window for node$_i$ into four sub-windows as shown in Figure 13.
Generate the gray level histogram for each new window
If there is a significant non-white or non-black occupancy
        Part of the boundary is contained in this window
        Add this node to the next level
        Link to the parent node
Else
        Ignore
        Initialize child node reference in parent to null
```

The algorithm expands the current node into four nodes (a quadtree expansion) then stores only those nodes that contain part of the boundary, or mixed nodes. Mixed nodes are easily identified from the gray level histogram because they will have a significant number of non-black or non-white cells. If at least 5% of the image is not a single shade, then the current window contains boundary pixels. While histogram generation can be expensive, it is linear in the number of pixels, and it can be performed quickly on the DSP. The histogram can be reused during the calculation of the curvature feature. It is not necessary to store a complete version of the image at each level because the highest resolution level (complete image) is used for any feature calculations. The result of the expansion algorithm is a list of nodes containing boundary pixels, which can be analyzed using the fitness function to find suitable grasp points.

5.4 Node Evaluation Algorithm

The node evaluation function is the main algorithm of the grasp planning system. The node evaluation algorithm is performed on every node at the lowest level of the tree. The node level function calls the different feature extraction functions as well as the opposite grasp point determination function and the fitness function. This function also records the best grasp point at a given level of resolution. The node evaluation algorithm sequences the function calls so if any of the feature evaluations imply poor grasp points, the rest of the algorithm is skipped. The feature evaluation and fitness function routines are called twice for each node, once for the boundary point corresponding to the *x* midpoint, and again for the algorithm corresponding to the *y* midpoint. The features chosen and the fitness function determine the behavior of the algorithm. The pseudocode for the node evaluation function follows:

```
For every node (window) in the current level
    Get the boundary location at the x midpoint of the window
    If there is a boundary at the x midpoint
        Find the curvature
        If the curvature is small enough
        Find the distance to center of mass
        Find the opposite grasp point
        Find the difference in orientation
            If the grasp length is within the bounds of the gripper
                Evaluate the fitness function
                If the fitness is less than the current min
                    If the grasp is collision free
                        Set the new winner to the current node
    Get the boundary location at the y midpoint of the window
    If there is a boundary at the y midpoint
        Find the curvature
        If the curvature is small enough
        Find the distance to center of mass
        Find the opposite grasp point
        Find the difference in orientation
            If the grasp length is within the bounds of the gripper
                Evaluate the fitness function
                If the fitness is less than the current min
                    If the grasp is collision free
                    Set the new winner to the current node
```

The *if* statements in the algorithm cut down on irrelevant computations when a single feature has already disqualified a given node. The number of disqualified points is high enough that the extra computation effort of the conditionals is justified. The curvature, distance from center of mass and parallelism are all features used in the fitness function. The grasp length is used to determine if the grasp is physically valid for a given gripper. The selected grasp must be larger than the minimum grasp size and smaller than the maximum grasp size, determined by the travel distance of the prismatic fingers. If the point turns out to be a potential best grasp point, the area around the grasp is examined for potential collisions. This step is necessary to ensure that collision free grasps are generated for concave targets. Convex targets do not pose any concern in terms of collisions because the grasp is always normal to the surface.

5.4.1 Fitness Function

The fitness function determines the behavior of the system. We have chosen a fitness function to be a linear weighted sum of three features, the curvature of the boundary in the window, the distance of the grasp line from the center of area, and the angular difference in surface normal between the two grasp points. Each of these features is described in detail in the following sections. The fitness function is described by the following equation:

$$c_{ab} = a\sigma_\theta + b(\theta_1 - \theta_2) + cd_{cm} \tag{29}$$

where σ_θ is the standard deviation of the grayscale histogram, θ_1 - θ_2 is the difference between surface normals of the two grasp points and d_{cm} is the distance from the grasp axis to the center of mass. Examples of these values are shown in Figure 13.

The fitness function determines the behavior of the system by defining the following parameters:

1. What is considered an acceptable grasp

2. Which grasp is considered best

3. How features are combined to yield (1) and (2)

4. The relative and absolute sensitivity of the features

The fitness function is a weighted sum of the three features, but the surface of the fitness function is non-linear because both the curvature and distance to center of area are described by non-linear functions. The fitness function is bounded above and below. The lower threshold is zero because the fitness function is a sum of strictly positive features multiplied by strictly positive scalars. The upper bound is set by the user and determines the level of acceptability. The constants are scaled such that a fitness or cost of unity should lead a marginally stable grasp. Good results with the upper bound set at 0.7 were obtained.

The global sensitivity of the fitness function is uniform because there is no change in the derivative of the fitness function as the magnitude of the individual features change. The relative weightings of the different features determine the local sensitivity. The weightings were determined heuristically and experimentally. The features are weighted from most important to least important in the following order: *curvature, parallelism, distance to center of area.* This ordering was chosen because the object is very unlikely to torque under the influence of gravity because of the high coefficient of static friction for neoprene rubber. The curvature was ranked over the orientation to preferentially treat straight areas where the actual contact point of the grippers would be easier to predict.

Curvature

The curvature of the local boundary segment can be measured in many ways, using calculus, line segment fittings, or any of several other methods. The standard deviation of the surface normal along the boundary segment in the image was chosen as a measure of curvature. This measure of curvature is independent of the size of the current window, is consistent for similar curves, is independent of the curve's position, and is quickly calculated given the grayscale histogram of the boundary orientation.

A line segment is represented by a single spike in the histogram of the gradient orientation. A circle is a constant distribution across all grayscale levels. A number of line segments gives a multi-modal distribution. A curved segment covers some region of the histogram with a non-uniform distribution. Theoretically, the fitness function should never find acceptable grasp points on a circular part; however, as the windows become small enough, there is insufficient resolution in the 8-bit representation of the surface normal to make the segment appear curved. At some given resolution, depending on the grayscale depth, the circular arc is approximated by a line segment. This allows the algorithm to treat both curved and straight surfaces in the same way.

Because the grayscale histogram has already been generated to determine if the window contains a boundary, the histogram can be reused to find the curvature. The curvature can be calculated in a time of *2(N-2)* where *N* is the number of grayscale levels. There are two loops through all the levels but the first and last (which correspond to the interior and the exterior of the object). For the implementation described in the results section, 8 bit grayscales were used, giving a duration of 508 iterations to calculate the curvature, independent of the size of the window. Using the standard deviation of surface normal therefore provides a robust, efficient means of calculating the curvature of a boundary segment.

Parallelism

The parallelism feature determines the grasp stability. Parallelism is measured as the difference between the surface normals of the opposing grasp points. Because the first grasp point is assumed to be along the normal of the surface, it is guaranteed to be within the friction cones. However, the second grasp point may not lie within the opposing friction cone. Therefore, the stability of the grasp is determined by the difference in orientation between the two surface normals. In addition, because we have assumed parallel jaw grippers, the most stable condition occurs when the opposing grasp points are on parallel faces of the target. The absolute value of the difference between the surface normals at the grasp points therefore gives us a measure of the stability of the grasp. The measure is minimum at zero, and approaches π as the angle between grasp points moves toward perpendicular.

Distance to Center of Mass

The distance to center of mass is the simplest feature. The feature is calculated as the perpendicular distance from the center of area of the object to the line connecting the two grasp points. This is a simple measure of the stability of the grasp with respect to rolling about the grasp axis. If the grasp is off center, the object may torque under the influence of gravity exerted at the center of mass. The center of mass is the least important feature and is included to provide some context for evaluating and distinguishing grasps along long straight surfaces such as the edges of a cube.

5.4.1.1 Opposite Point Location

While the opposite point location algorithm is not crucial to the algorithm, it is executed often, and is therefore outlined here. In order to reduce the amount of bookkeeping and memory required to store all nodes in the graph, the search for the opposite point is performed directly on the image, not on the windows or nodes. Because the second grasp point must lay along the line normal to the surface at the first grasp point, the two-dimensional search can be reduced to a single dimensional search along that line. An example of the kind of information along this line is shown below.

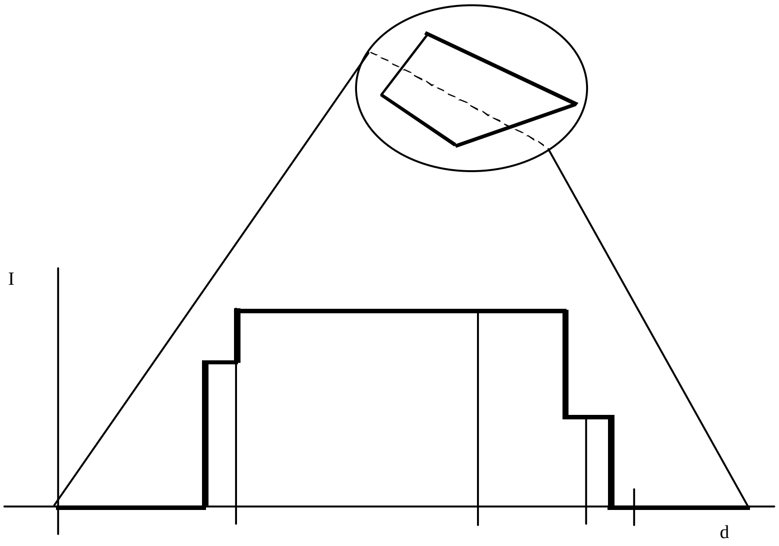


Figure 14: Representation of the Linear Search

Figure 14 represents a linear cross-section of the convex object along the dashed line. The vertical axis represents the intensity of the pixels along the line, and the horizontal axis refers to the position or distance along the line. The zero valued section at either end corresponds to the black background. The high-valued central plateau corresponds to the interior of the object, and the two smaller shelves correspond to the edges of the objects. The value of the edge is proportional to the surface normal as described in the image processing section. The thin vertical lines represent the results of a binary search starting at one edge moving to the opposite edge. It took approximately four iterations to find the opposite edge.

Because the system only has four values, interior, exterior and both edges, steepest decent searches are impossible, because the flat sections are essentially huge local extrema. Examining the data set closely reveals that in fact there are only two values, inside (max) and outside (min). The algorithm terminates when it reaches an edge value, which is neither max nor min. Because there are only two values, the only real information encoded in the image is their positions, a classic binary search problem. A binary search can reach the end in *log(N/n)* time, where *N* is the total number of pixels in the line and *n* is the width in pixels of the opposing boundary.

The opposite point location algorithm also performs a validity check on the selected grasp point. For this implementation, only exterior grasp points are considered (holes are in fact eliminated during the image processing step). Therefore, it is necessary to ensure that the boundary found is an external boundary for concave objects. Figure 15 demonstrates valid and invalid opposing grasp points.

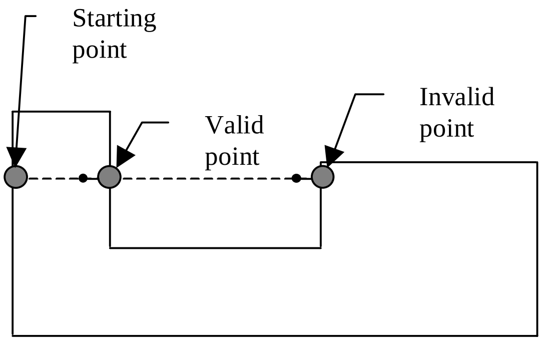


Figure 15: Valid and Invalid Opposing Grasp Points

The far-left point in Figure 15 is the start of the search. The other two points are potential solutions of the binary search. The first point fails because the adjacent area between it and the original point is background. The second point is valid because the area between it and the original point is inside the target. We know that a valid grasp point must be a transition from interior to exterior not exterior to interior. Once a point has been found the algorithm test a small offset on either side of the point along the surface normal. If the transition is from inside to outside, the solution is valid, if not, the current point is perturbed slightly to offer a new starting location and the binary search is continued.

5.5 Collision Detection

The collision detection algorithm is also based entirely on image based methods. The gripper is abstracted as a rectangular projection. The gripper is rotated to be parallel with the current grasp point, then moved so the tip of the gripper would be touching at the point of contact. A window the same dimensions as the gripper window is extracted from the original image centered at the candidate grasp point. The result of the image and the gripper are intersected. Any internal regions occupying the same space as the gripper remain white, all other regions become dark. The histogram of the result is calculated. If the number of interior cells is larger than a small threshold, a collision has occurred. Ideally, a collision should occur if only a single white pixel exists, but quantization errors in the angle allow small slivers of interior pixels to intersect with the target.

5.6 Computational Complexity

It is difficult to expressly estimate the computational complexity of a system, which employs early stopping to reduce computational load. There are essentially two sections to examine: how much it costs to calculate the cost function at each node, and how much it costs to perform the expansion. The cost of node calculations will be presented and, based on those results, the behavior of the system as it expands will be examined.

There are essentially three major steps that follow the expansion of the previous layer, determining which nodes contain boundary nodes, determining the features for each node, and evaluating if any winning nodes generate collisions.

1. *Examine if the current node contains boundary elements.* The complexity at level *i* is given by

$$C_{be} = W_i p_i \qquad (30)$$

where W_i is the number of nodes or windows in the current layer and p_i is the number of pixels in each window in the current layer.

The complexity for each feature calculation and the binary search for the opposite edge follow. The complexity of the curvature calculation is

$$C_{\sigma_\theta} = 2(g_i - 2) \qquad (31)$$

where g_i is the number of grayscales that the histogram is expressed in, this number is usually 256. The upper and lower bounds of g_i are reserved for representing the interior and exterior of the target, and therefore do not have to be counted. The complexity of the distance from center of mass, and relative orientation calculations are small constants C_d and C_o. The complexity of the binary search for the opposite grasp point is given by

$$C_{bs} = \log_2\left(\frac{N}{n}\right) \qquad (32)$$

where N is the maximum length of the line separating the points and n is the width of the edge. Ignoring early stopping for feature evaluation, the overall cost of a feature evaluation must be performed at each grasp point, so the total cost is

$$C_f = 2C_{\sigma_\theta} + C_d + C_o + C_{bs} \tag{33}$$

$$C_f = 2(g_i - 2) + C_d + C_o + \log_2\left(\frac{N}{n}\right) \tag{34}$$

Finally, if the node has the smallest cost, the node must be checked for collisions. The cost of collision detection is given by

$$C_{cd} = 3p_g \tag{35}$$

where p_g is the size of the gripper window. Therefore the total cost for the calculation of node *j* in level *i* is given by

$$C_{ij} = W_i P_i + W_{bi} C_f + W_{wi} C_{cd} \tag{36}$$

where W_{bi} is the number of boundary nodes on level *i* and W_{wi} is the number of winner nodes on level *i*.

It should be apparent from Eqs. (30) - (36) that the node evaluation computational complexity is linear. However, the grasp planner must execute a node calculation for every node in the tree, therefore theoretically, it should increase as

$$C_T = C_{ij}^i \tag{37}$$

where *i* is the number of levels in the current tree. However, because $W_{bi} << W_i$ and $W_{wi} << W_{bi}$, the growth of the system is much slower. By pruning irrelevant nodes from the tree, the computational cost is closer to an order 2 calculation than an exponential calculation, as shown in our results.

6 Neural Networks

Neural networks are a class of non-linear approximators loosely based on the function of the human brain. They use multiple non-linear elements to approximate a global function. There are many network architectures and learning or optimization algorithms, but the most common type is the backpropagation network. We have used the backpropagation network to approximate the transform **T**. In place of more traditional non-linear optimization techniques [14], [16], we use a neural network.

6.1 The Backpropagation Network

Neural networks consist of a group of nodes and weighted connections, organized into layers. For a backpropagation network the input is carried through the network by series of multiplications over the connections and summations at the nodes. The most common architecture is based on the multilayer perceptron shown in Figure 16.

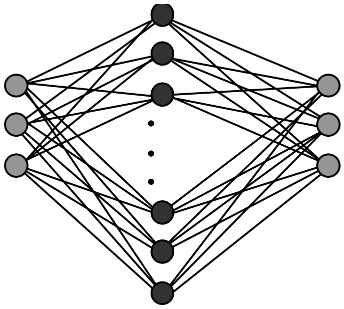


Figure 16: Backpropagation Network

The multi-layer perceptron node is shown below in Figure 17. For each new input a sequence of multiplications and additions is carried out. The result is a large number of non-linear equations acting as a global estimator.

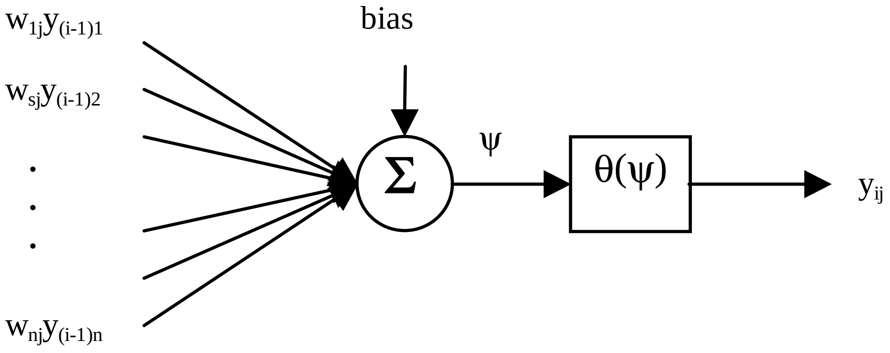


Figure 17: Perceptron Node

The connection weights are held between -1 and 1 and the activation function is usually sigmoidal. The multilayer perceptron node sums the outputs of all connected nodes and a local bias which it feeds into the activation function and then passes to the next layer. Therefore the output can be written as [38]

$$\zeta_i = \sum_i \Theta\left(y_{(i-1)j} w_{ij} + bias\right) \tag{38}$$

$$y_k = \sum_j \zeta_j w_{jk} \tag{39}$$

where w_{ij} is the weight of the *ijth* connection, and Θ(ψ)

$$\Theta(\psi) = \frac{1}{1+e^{-\psi}} \tag{40}$$

Initially Eq. (40) was chosen because it approximates the non-linearity in human neurons. However, from a mathematical perspective, it gives an accurate non-linear estimation.

Backpropagation uses the chain rule of calculus to backpropagate the error at the output nodes through the network, performing a gradient descent over the error surface. Because gradient descent is a local optimization algorithm, steps have to be taken to ensure that the network converges sufficiently quickly, and does not stop in local minima. Two methods were used to aid the gradient descent algorithm, momentum, and delta-bar-delta learning rate modification. The result is the following equation relating the change in weight to the error [38].

$$\Delta w(n)_{ij} = \alpha w_{ij}(n-1) + \eta \delta_i \dot{\Theta}_{(net)} \mathbf{x}_j \qquad (41)$$

where *Θ*' is the derivative of the activation function, α is the momentum coefficient and δ_j is determined as:

$$\delta_j = \begin{cases} y_d - y_{net} & if\ outputlayer \\ \sum_j w_{ij} \delta_{j-1} & otherwise \end{cases} \qquad (42)$$

By iteratively calculating the delta term or node error, the error at the outputs of the network is carried through the network from back to front.

The backpropagation algorithm requires an error signal to be fed back to the output layer; therefore, the net must be trained by a teacher, and the desired output must be known.

We used a backpropagation network with a sigmoidal non-linearity and linear input and output functions because we had achieved good results in previous research using such a network configuration [6].

6.2 Approximating the Camera Kinematic Transform

Typically, non-linear programming has been used to determine the kinematic transform. Non-linear programming and optimization attempt to minimize an objective function with respect to a criterion, and the results should lead to a correct response. Instead of using non-linear programming, we have opted to use backpropagation networks. Neural networks can behave as non-linear functional approximators [38]. Neural networks produce local approximations so results may not be valid over the entire workspace.

In order to approximate the kinematic transform **T**, we captured a training set, by having the robot move through a 3000 point grid in *x*, *y* and *z*, as shown in Figure 18. We decided to only approximate the transform with respect to *x*, *y* and *z* because the addition of θ would add another dimension to the search and therefore the data set. The required data set size would be at least 8000 points, which would take a prohibitively long time to generate. In addition, the mapping between the camera and robot θ is 1 to 1, so the additional computational burden on the neural network is not required. The workspace of the robot with the target at least partially visible is approximately conical with a slope proportional to the viewing angle of the camera. To approximate this conical structure we used two prismatic rectangles. A diagram of the workspace is shown in Figure 18.

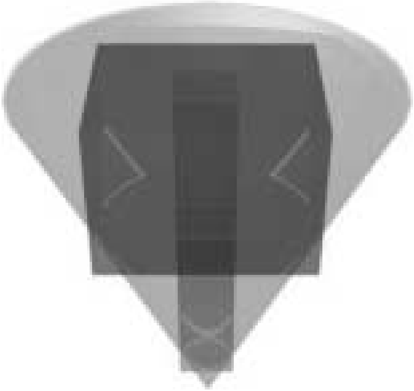


Figure 18: Approximation of Viewable Workspace for Neural Network Training

Figure 18 clearly demonstrates the relationship between the cone that describes the viewable area for all robot positions *x*, *y*, *z*, and the two box training set used for the neural network. The upper rectangle encodes the most likely initial configurations for the target with respect to the robot. The bottom rectangle corresponds to the maximum viewable workspace at the desired height with respect to the target. The top rectangle was 2 inches by 1.5 inches by three inches. The second rectangle was 0.75 inches by 0.75 inches by 5 inches. The top cube was sampled with 2000 points. The vertical rectangle was sampled by 1000 points. This biases the network to the upper portion of the workspace, where the robot is most likely to use the neural network, while still maintaining a context to how the robot will behave lower in the workspace. It also placed an emphasis on targets closer to the center of the target area where the two prisms overlap.

The training set was used to train a backpropagation network with linear input and output layers and a sigmoidal hidden layer. The network was trained to approximate the transform between *u*, *v* and *w*, and *x y* and *z*. The network had three inputs and three outputs, and a twenty-five node hidden layer. The network was completely connected with random weighting on the connections. The network was trained with the standard backpropagation algorithm as previously described. We used a learning rate of 0.15. A graph of the convergence of this network over time is shown in Figure 19.

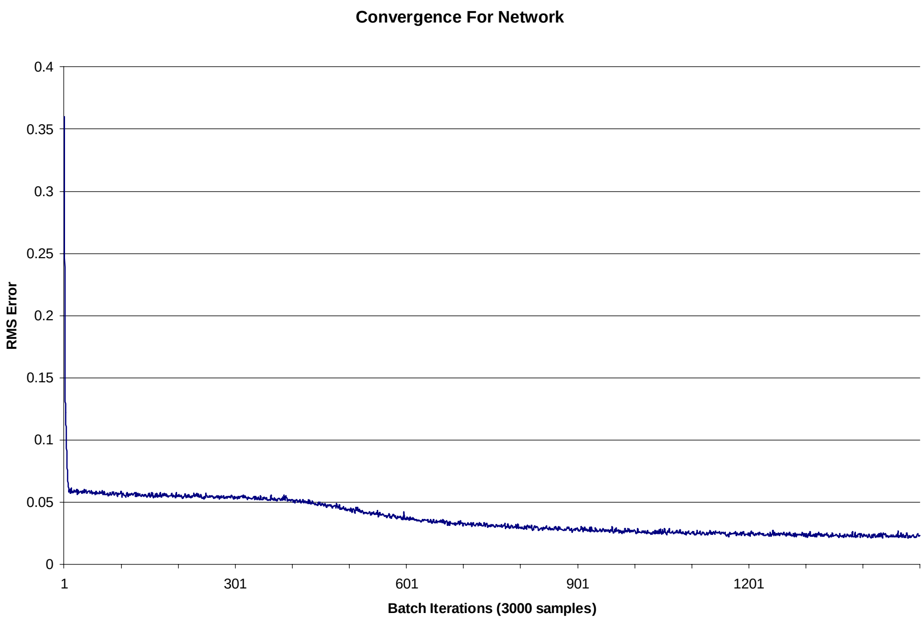


Figure 19: Network Convergence Profile

The convergence profile shows a network that converged to a small error relatively quickly, then took a large number of iterations to converge to its final error of 0.02 or approximately one-hundredth of an inch along each axis. After the network had been trained, the system was tested against the visual servoing and linearly calibrated look and move system. Results of these experiments can be found in section 8.1.2.

7 Overall System Implementation

The system is primarily software driven. Special attention was given to the software structure to make the system easily extensible, portable and robust. The software was based in part on an object-oriented methodology. In addition, the structure made use of the real time message passing capabilities of the QNX operating system. The QNX operating system allows the creation of many processes and simple inter-process communications. The system was also designed to take advantage of the special parallel processing hardware available for image processing and robot control.

7.1 QNX Messaging Architecture

The majority of this information is taken directly from the *QNX Operating System: System Architecture* manual [39]. QNX is a multi-process operating systems with a strong emphasis on inter-process messaging. Processes can be created by the user on the command line, but they are more commonly spawned by other processes. Upon creation, all processes are given process ID that can be used by other processes for inter-process communication. Every process has a message queue that can be used to store pending messages. QNX uses a *Send, Receive, and Reply* architecture for servicing messages. If a message queue is empty and a process attempts to receive a message it will become *Receive* blocked. A calling process *Send*s a message to a receiving process. Until the receiving process checks the message queue, the sending process is *send* blocked. Once a process has checked the message queue, it must *Reply* to the sending message. By changing when the receiving process replies, the operation can be synchronous or asynchronous. Real Time Objects are normally *Receive* or *Reply* blocked, as they wait for a calling process to initiate a function call, or they wait for the results of a function call.

7.2 Real Time Objects

The system is organized into a set of Real Time Objects (RTOs) which encapsulate C++ objects containing the QNX messaging code. Each RTO appears as a single functional object to the other processes, but can be composed of many internal objects, even running on different processors. A diagram of the RTOs in the system, and their component C++ objects is given in Figure 20.

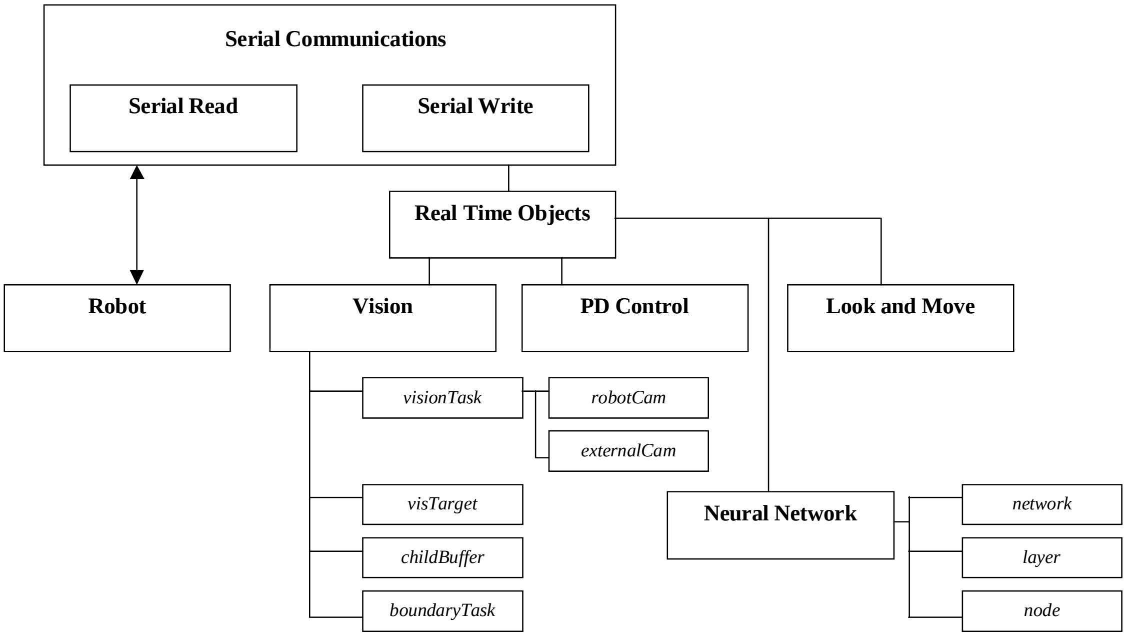


Figure 20: Real Time Object System Architecture

All the bold objects in Figure 20 are RTOs. Each RTO with the exceptions of the vision and neural network classes directly wraps a client C++ object containing the functionality. The vision and classes are exceptions because of the complexity of the algorithms within these classes did not lend themselves to a single object implementation. However, from the point of view of external objects, they appear to be a single object. The Serial Communications RTO is another notable exception. While to client processes (such as the robot RTO) the serial object appears to be a single RTO, it is in fact two RTOs, one to write data to the serial port, and another to read data from the serial port. The launching process only launches the write RTO, and provides its own process ID as an argument, the read RTO is launched by the write RTO then replies to the calling processes directly.

The key component of the RTO structure is the standard message. By defining a standard message class, many of the issues raised by strict type checking can be avoided. RTO's in general do not verify which process is sending, only that the information is correct. The standard message type contains integer message identification and a BYTE array of data. Any input data required for a given function of the RTO must be encoded as ASCII strings in the data vector. The calling and responding RTOs are responsible for encoding and decoding the data in a consistent manner. ASCII strings are used, because, while they are not the most memory efficient, they are extremely flexible. The definition for the message type is given in a global object file that all RTOs must include.

Real time objects are in fact ANSI C main() programs, which use QNX's, send and receive functions to behave like objects. A typical RTO is set up as follows:

```
Main()
    Initialize()
    Read message buffer
    Case message:
            F1:     Parse Arguments for F1
                            Call InternalObject->F1
                    Reply
            F2:     Parse Arguments for F2
                            Reply()
                            Call InternalObject->F2
            F3:     Parse Arguments for F3
                            Reply()
                    Call InternalObject->F3
                    Send Reply
```

```
End Case
End Main
```

The possible messages that the RTO can understand are encoded in a standard header file called a **mes.h* file. The header file contains *#define* commands for all possible commands and comments describing the type and format of input required for each vector. For example, a control law might require the current position and velocity vectors of the target as strings of ASCII numerals as input. The purpose of the RTOs is to parse inter-process messages and to manage inter-process messaging and sequencing. As apparent in the preceding pseudocode, the first command is always *Parse Arguments.* An RTO consists of receiving a message in the system-administered message queue, parsing the message ID, parsing the message data, calling the appropriate function, then replying. Depending on where the reply is place in the execution sequence, the system can be synchronous, asynchronous, or asynchronous with feedback. Function F1 in the example is synchronous because the reply occurs after the worker function has been called. Function F2 is asynchronous because the reply is called before the worker function. Function F3 is asynchronous with feedback, because a reply is sent before the worker function is called, then a follow-up message is sent to the calling function. Having a number of synchronous and asynchronous options available allows the creation of robust real time systems.

8 Experiments and Results

To demonstrate the viability and performance of the system, several tests were run, and are presented in the following sections. Each major component was tested separately, then the entire system was combined and a simple sorting task was used to demonstrate the complete system. The positioning algorithms are tested against a visual servoing control. The grasp planner is tested with several real and synthetic objects. The final system sorts cups based on a quality measure.

The system is composed of a dual 200 MHz Pentium Pro computer with a PCI bus. It contains a Matrox Genesis frame-grabber and digital image processing board. It is connected to the CRS C-500 controller running the RAPL-3 operating system via a serial cable. All image processing is carried out on the Genesis board. The genesis board allows multiple threads of processes to be queued via the PCI bus. By doing the entire image processing on the Genesis board we eliminate the need for transmitting the entire image across the system bus. The serial connection to the robot is kept at a relatively slow 9600 BPS because the serial hardware on the controller side is unreliable at high speed and prone to errors. The robot is driven over the serial port using a master-slave system. A client program written in RAPL-3 resides on the controller, receiving commands and turning them into the appropriate robot motions. Inverse kinematics and control are calculated using the C-500's transputer network. By splitting the computation to multiple specialized processors, we reduce the computational load on the CPU.

8.1 Positioning Accuracy

In order to determine the relative merit of the three different visual guidance algorithms, we tested the positioning accuracy of each system. In addition, we tested the accuracy of the two kinematic based systems in conjunction with the visual servoing system. The final positioning accuracy of each target is compared, and the advantages and disadvantages of each approach are identified.

8.1.1 Experimental Setup

In this experiment, the robot was presented with five targets as shown in Figure 21. The first object, a dixie cup, appears as a circle in the top projection, and was run eight times for each type of robotic positioning system as a baseline for observing the other targets. The other four targets were each targeted once with each algorithm. All robot-positioning algorithms were employed for each target and initial position. The final positioning error was recorded but the target was not manipulated. Because the target remained in the same position for each trial, the behavior of each algorithm could be reliably compared. Particular emphasis was placed on examining the error remaining after the first step, and the number of iterations required to move the robot to a stationary position. Targets were isolated in three degrees of freedom (x, y, and z) initially, then positioned in θ after the grasp point was calculated. Because the mapping between the measured θ and the robot θ is one to one with the camera placed directly over the target, there is no coupling between motion in θ and motion in *x* and *y*. Therefore, it was much more efficient to uncouple the motion in θ initially then move to the desired θ during the grasping phase of motion.

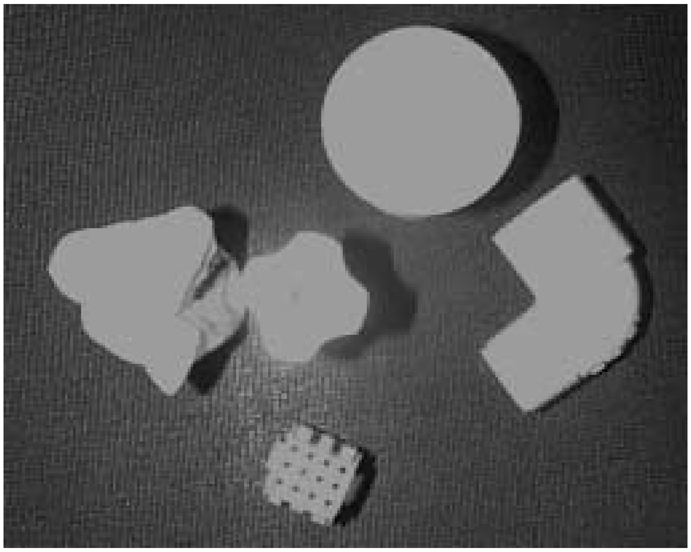


Figure 21: Targets Used in Experiment

8.1.2 Results

The results will be presented in two sections. The first section will compare the results of the two kinematics-based algorithms: linear calibration, and neural network calibration. The second system will compare the responses of the visual servoing system with two hybrid systems, one employing a linear calibration calculation as the first step, another employing a neural network as the first step.

8.1.2.1 Linear Calibration and Neural Calibration

The linear calibration system presented in section 4.1, and the neural network system presented in section 6.2 were employed for a single iteration to determine their relative merit in positioning the manipulator with respect to a target. The neural network was already trained as shown in Figure 19. A new robot position was generated for eight positions of the dixie cup target and once for each of the other targets. The robot was moved to the new position and the resulting error calculated. The average error in *x*, *y*, and *z* and the overall average error are compared to the initial error in Table 2.

Table 2: Comparison of Linear and Neural Computed Kinematics

Trial	Algorithm	Initial Error	Final Error x	Final Error y	Final Error z	Average Final Error
1	Look and Move	102.38	335.46	254.65	167.73	14.33
	Neural Net	102.38	317.64	248.88	148.74	6.42
2	Look and Move	105.0	295.68	303.38	162.27	36.08
	Neural Net	105.0	334.47	236.31	155.27	8.62
3	Look and Move	98.97	322.36	211.84	158.77	16.46
	Neural Net	98.97	331.53	266.80	146.77	17.50
4	Look and Move	79.47	304.27	254.63	160.55	12.81
	Neural Net	79.47	336.19	239.77	157.54	9.46
5	Look and Move	111.6	292.84	296.12	159.11	36.08
	Neural Net	111.6	342.06	234.70	158.11	13.22
6	Look and Move	126.79	292.05	251.75	155.50	17.50
	Neural Net	126.79	342.91	248.66	153.50	14.17
7	Look and Move	67.26	308.11	262.88	161.45	15.35
	Neural Net	67.26	331.84	238.41	153.45	6.96
8	Look and Move	156.26	326.92	284.05	155.82	25.75
	Neural Net	156.26	316.79	357.77	145.80	68.23
mean	Look and Move					22.21
mean	Neural Net					18.07

All values in the table are measured in pixels. The error is measured as the RMS error of the pixels from the current position (x, y, z). The last two rows of the table contain the average output error over all trials. It should be apparent from the table that the neural network is superior to the linearly tuned look and move system. This is expected, because, while the look and move system is tuned with more *a priori* information, the model is incomplete. The neural network can converge to approximate the transform with a high degree of reliability.

The initial position in the third trial begins in the extreme bottom center of the image. The object was partially clipped at the edge of the image frame. This resulted in a violation of one of the core assumptions in the system; the target behaves like a rigid body in the projection. As the target moves towards the center of the image, its center of area changes non-linearly, as more of the target becomes visible, resulting in a deformation of the original target. Examining the third trial yields information about the behavior of the system with respect to a violation of the rigid body assumption. Both the neural network and linear calibration algorithms faired well generating an error of 16.46 for the look and move system and 17.50 for the neural network. Each algorithm also produced similar y errors, where clipping occurred. From these results we can conclude that the rigid body assumption is not strictly limiting to either of these functions, at least in the cases of occlusions or clipping.

The accuracy of the neural network is quite low for the last trial of the test sequence. This trial was anomalous with respect to the rest of the trials. The neural network approximates the transform well in the vicinity of the training data, but has a rapid increase in error for points outside the initial training set. Therefore, initial conditions outside the training set, such as the last trial, can result in a poor first motion. If we remove the worst case trial from each algorithm's results yields the following average errors: look and move system, 20.21, neural network system 10.09. Removing the worst case from each system shows that the neural network system is nearly twice as accurate as the linear look and move system, when the target is completely within in the field of view. However, it also shows that the linear look and move system is more predictable and consistent in its reliability.

To test the generalization of each system with respect to target type, four other targets were chosen. Each target was placed in a random location in the workspace and each positioning algorithm was tested. The final errors for each of the systems were recorded. The results are shown in Table 3.

Table 3: Target Generalization Comparison for Linear and Neural Systems

Target Name	Look and Move Error (pixels)	Neural Network Error (pixels)
Backing	37.48	4.11
Crushed cup	43.81	21.29
Elbow	11.52	16.68
Plug	20.43	19.25
Average	**28.31**	**15.33**

The table shows that the error due to the target type is consistent with the error for the cup. The overall average error is on the order of 28 pixels for the look and move system and 15 pixels for the neural network system, as we expected. These errors translate into position errors of approximately 0.17 and 0.15 inches respectively. For many simple grasping applications, especially in service robots, a tenth of an inch accuracy is more than adequate. However, for industrial applications such as assembly, accuracy must be higher. Higher accuracy can be achieved by the use of much more sophisticated calibration scheme such as in [14] – [18]. We have opted for a visual servoing system instead.

8.1.2.2 Visual Servoing and Hybrid Systems

Both the neural network and linear look and move system do not have the required accuracy for assembly tasks. In order to increase the accuracy of these measurements we have added a visual servoing system as described in section 4.2. This system allows the robot to regulate its position to the desired position. The final position accuracy is therefore limited by the accuracy of the sensor, the accuracy of the robot, the magnitude of the gains, and the maximum number of iterations allotted for convergence.

The same sets of tests described in section 8.1.2.1 were run on a purely visual servoing and the two hybrid systems. The first hybrid system used a linear look and move calculation for its first move, then used the visual servoing system for the subsequent correction. The second hybrid system uses a neural network to compute its first move, then the visual servoing system for the subsequent correction. A final desired error of ± 5 pixels on all controlled axis was used as a termination condition.

The dixie cup target was used in the same manner as above to test the accuracy of the system. Of the eight target positions, each system was run until an accuracy of ± 5 pixels on each joint was achieved. The final error, initial error and number of iterations required are shown in Table 4.

Table 4: Comparison of Visual Servoing, Linear and Neural Hybrid Systems

Algorithm	*Initial Error*	*Final Error*	*Number of Iterations*
Visual Servo	102.38	1.97	9
Look and Move Hybrid	102.38	2.74	10
Neural Net Hybrid	102.38	2.65	8
Visual Servo	105.0	3.19	8
Look and Move Hybrid	105.0	2.85	10
Neural Net Hybrid	105.0	1.72	4
Visual Servo	98.97	1.97	8
Look and Move Hybrid	98.97	2.03	4
Neural Net Hybrid	98.97	2.84	11
Visual Servo	79.47	3.19	8
Look and Move Hybrid	79.47	2.85	7
Neural Net Hybrid	79.47	1.45	4

Visual Servo	111.6	3.61	11
Look and Move Hybrid	111.6	2.59	6
Neural Net Hybrid	111.6	1.22	4
Visual Servo	126.79	1.55	11
Look and Move Hybrid	126.79	3.18	4
Neural Net Hybrid	126.79	3.00	6
Visual Servo	67.26	3.79	7
Look and Move Hybrid	67.26	2.45	4
Neural Net Hybrid	67.26	0.57	4
Visual Servo	156.26	3.86	16
Look and Move Hybrid	156.26	2.12	6
Neural Net Hybrid	156.26	2.95	15
Visual Servo		2.98	9.86
Look and Move Hybrid		2.73	6.5
Neural Net Hybrid		2.05	7

It should be apparent that all the visual servoing systems have approximately the same final error; however, both the neural network and look and move hybrid systems require 33% fewer iterations to converge. This leads to a time saving of over a second for our low bandwidth system. Furthermore, looking at the neural network case especially, the system converges much faster. Eliminating the maximum number of iterations from each hybrid system, we see that they both converge in approximately six iterations.

As in section 8.1.2.1, the algorithms' generality with respect to target type was tested by placing different targets in the workspace, and running the algorithms. The results are shown in Table 5.

Table 5: Target Generalization Comparison

	Visual Servoing	Look and Move Hybrid	Neural Network Hybrid

	Iterations	Error	Iterations	Error	Iterations	Error
backing	11	0.79	7	2.48	4	2.41
crushed	12	2.93	12	2.67	6	4.07
elbow	9	3.36	5	3.06	11	3.04
plug	10	2.95	5	3.79	6	3.80
Average	**10.5**	**2.51**	**7.25**	**3**	**6.75**	**3.31**

As is apparent from Table 5, there is little difference in the accuracy (less than one pixel) of the system to different targets. There is little difference between the average values for the cup, and the measured values for the other targets. However, the performance advantage of the hybrid systems over the visual servoing system is apparent.

The response of the system under the different algorithms was investigated. There are three distinct phases of motion for the computed kinematics systems, and only two phases of motion for the visual servoing system. The following diagrams compare the response of each system for the same target, and starting position.

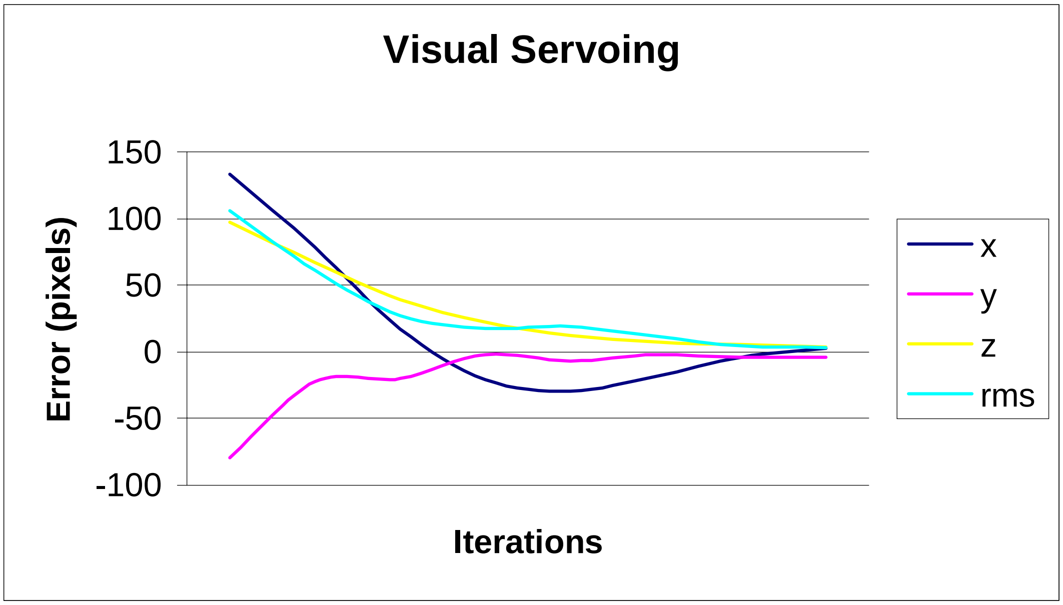


Figure 22: Visual Servoing Response

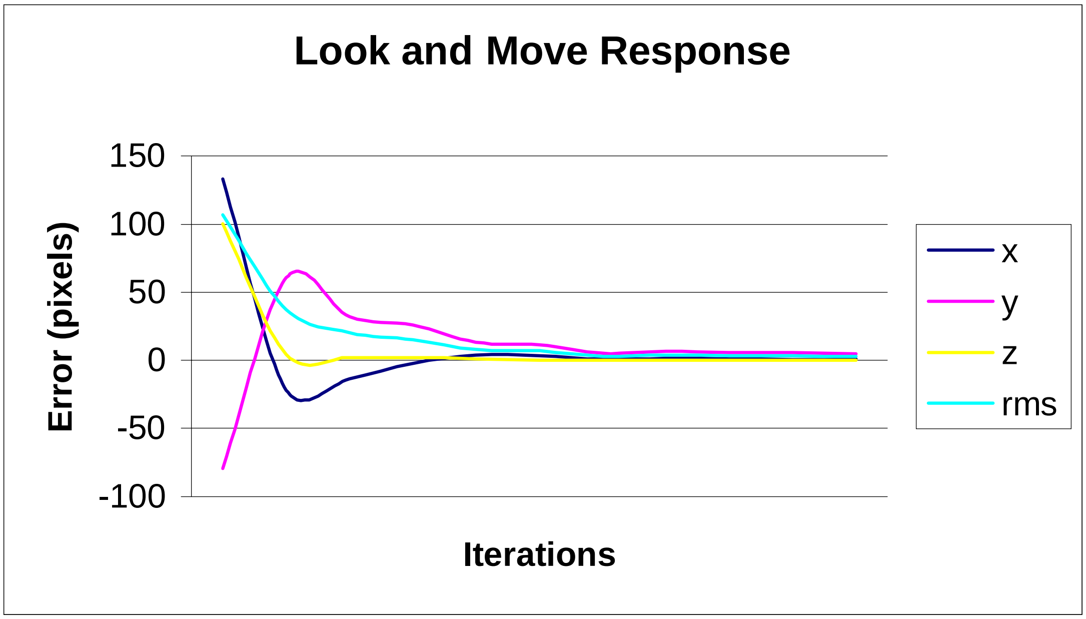


Figure 23: Linear Hybrid System Response

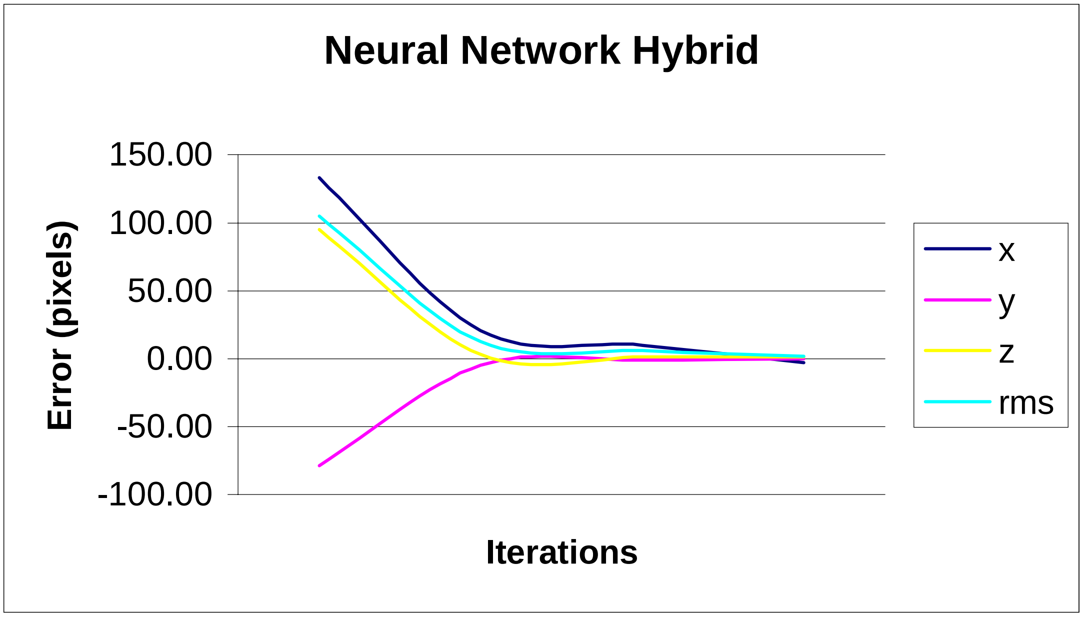


Figure 24: Neural Hybrid System Response

Figure 22 – Figure 27 present a typical response. The visual servoing algorithm displays a standard control response. There is overshoot and oscillation evident in the initial response, then the system enters the lower gain phase and begins an overdamped decent. The RMS error proceeds towards zero, but plateaus during the overdamped phase. The look and move response demonstrates the system behavior for a classic three-phase convergence. The first step moves the RMS error to approximately 30 pixels then the visual servoing system exhibits an underdamped response. After the RMS error is approximately 7 pixels, the overdamped gains servo the system to its final error. The neural network system displays the third type of possible behavior. The initial move places

the target within 10 pixels on all axis of the desired configuration, bypassing the high gain correction and only employing the overdamped gains to bring the target to its desired location. These responses are fairly typical for the system. The visual servoing system is generally characterized by the response shown in Figure 22. The computed kinematic systems are generally characterized by the responses in

Figure 25 – Figure 27, although the look and move can be more accurate than the neural network in certain cases giving the look and move system a response more like Figure 24.

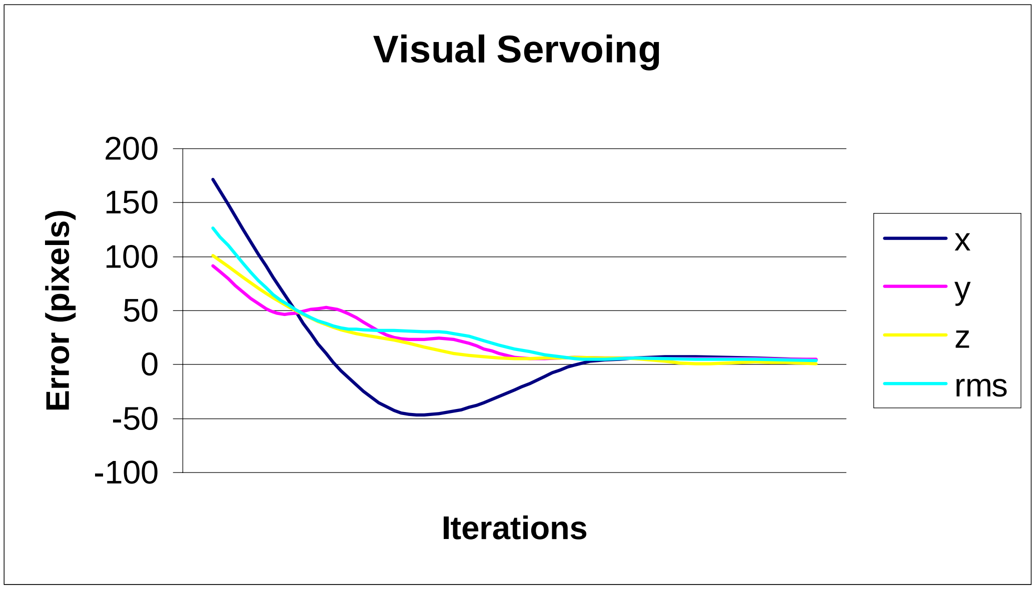


Figure 25: Visual Servoing Response

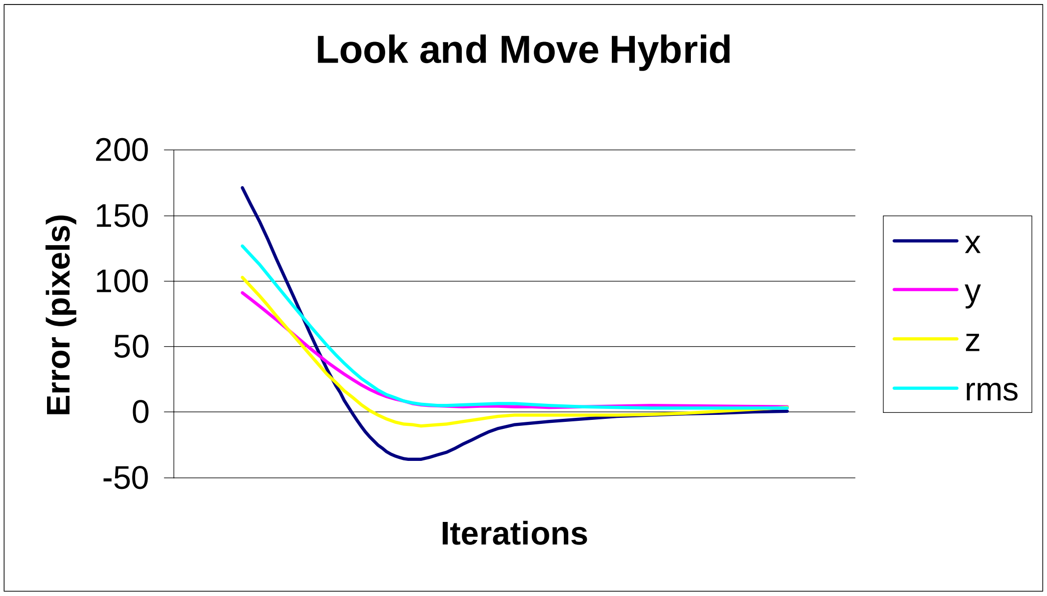


Figure 26: Linear Hybrid System Response

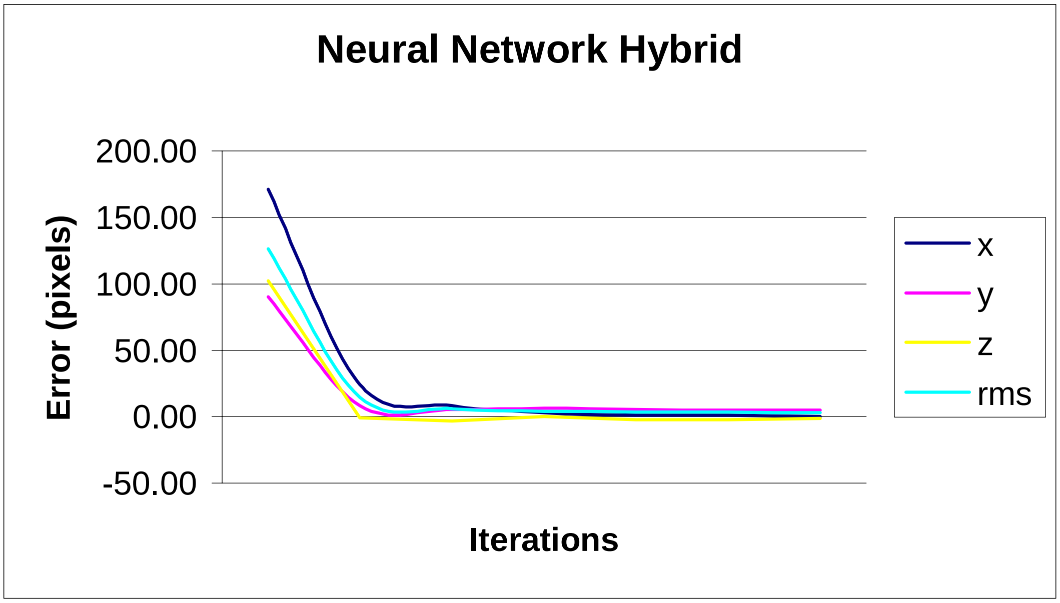


Figure 27: Neural Hybrid System Response

Figure 25 – Figure 27 show the ideal response of the system, where all methods quickly converge. The neural network converges fastest, followed by the look and move system, followed by the visual servoing system. There is very little overshoot in the look and move system, after the initial overshoot due to the look and move motion. We expect the neural network to converge faster because it has a higher accuracy (as shown in section 8.1.2.1), and, presumably, a controller starting with a lower initial error will converge faster. However, the neural network may perform poorer than the look and move system under certain circumstances. There are two basic reasons for this:

1. The neural network is a local approximator. Outside of the workspace where it was trained on, the error increase sharply. Starting points that lay on the boundary of this workspace may lead to incorrect operation. On the other hand the look and move system is based on the geometry of the system. These transforms are valid throughout the workspace; therefore it generalizes much better than the neural network based approach.

2. The control system behavior is not consistent due to the dual gains. The dual gain system can result in odd behavior if the starting point for the controller is arbitrary, as in the neural network and look and move hybrid systems. It may be faster to start fifteen to twenty pixels from the target, and in a single step of the underdamped

controller reach the goal, than to start ten pixels from the target, and use the overdamped controller all the way to the goal. This is really a problem with the controller gains for each system. This problem and its solutions are further discussed in section 8.1.3 and 9.2.

The final sets of images relate the behavior of the system for the PVC elbow joint. This case illustrates point 2 above, where the neural network takes longer to converge, even though it is initially more accurate.

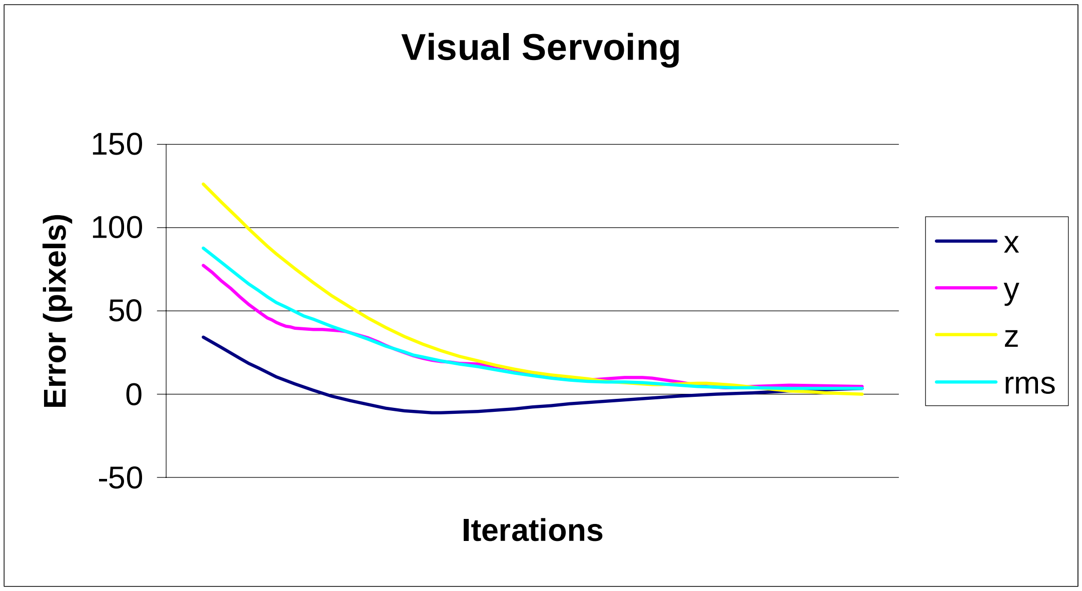


Figure 28: Visual Servoing Response for Elbow Joint

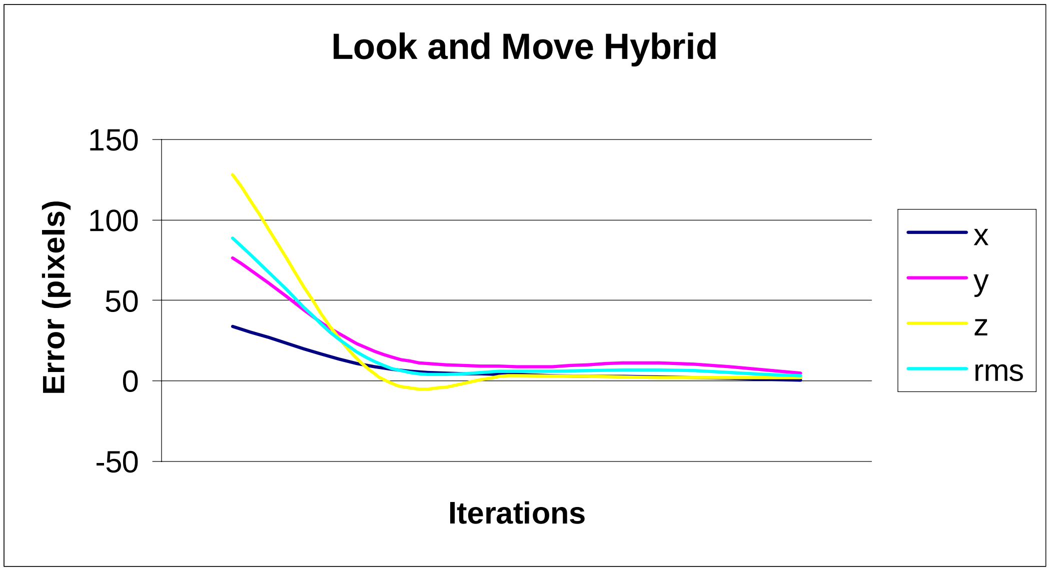


Figure 29: Linear Hybrid System Response for Elbow Joint

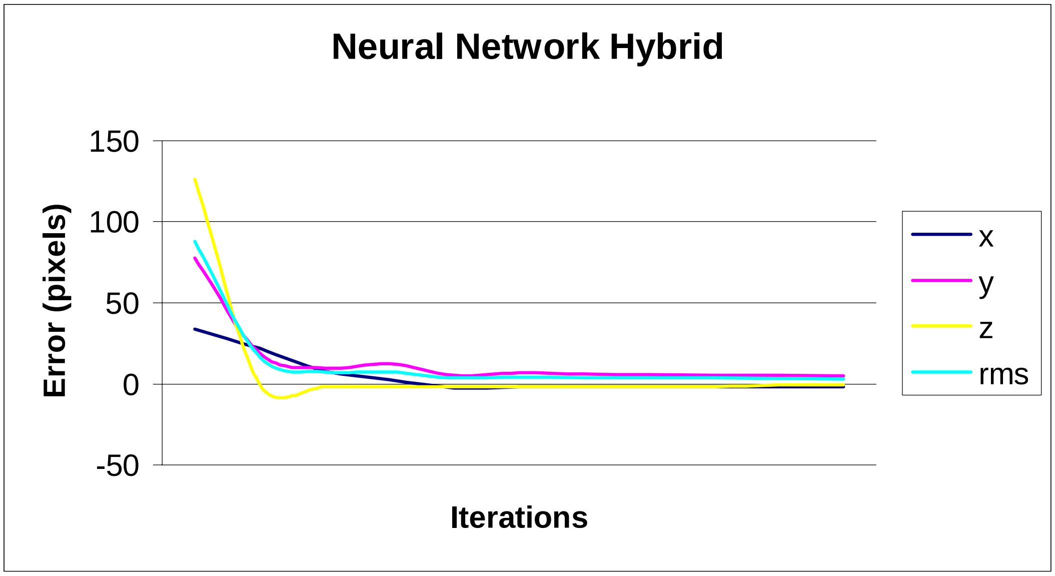


Figure 30: Neural Network Response for Elbow Joint

The long flat section in the neural network RMS error corresponds to a reached goal state. The *y* error also converges to the goal state, but the error plateaus while the overdamped controller converged. The *y* error is within 6 pixels of the target after 4 iterations, but takes another 6 iterations to move within 5 pixels of the target. The look and move system, on the other hand, has a slightly higher initial error, but quickly converges, as the second step moves the system to a better starting point.

To illustrate how the system behaves under the different positioning algorithms, the following three pages contain composite time lapse images of the typical sequence for all three positioning algorithms. The target is the cup described at the beginning of this section. Each algorithm is used to position the robot with respect to the cup. Each frame is given as a pair of images; the top image being the view of the robot mounted camera, and the lower image being the side view of the camera.

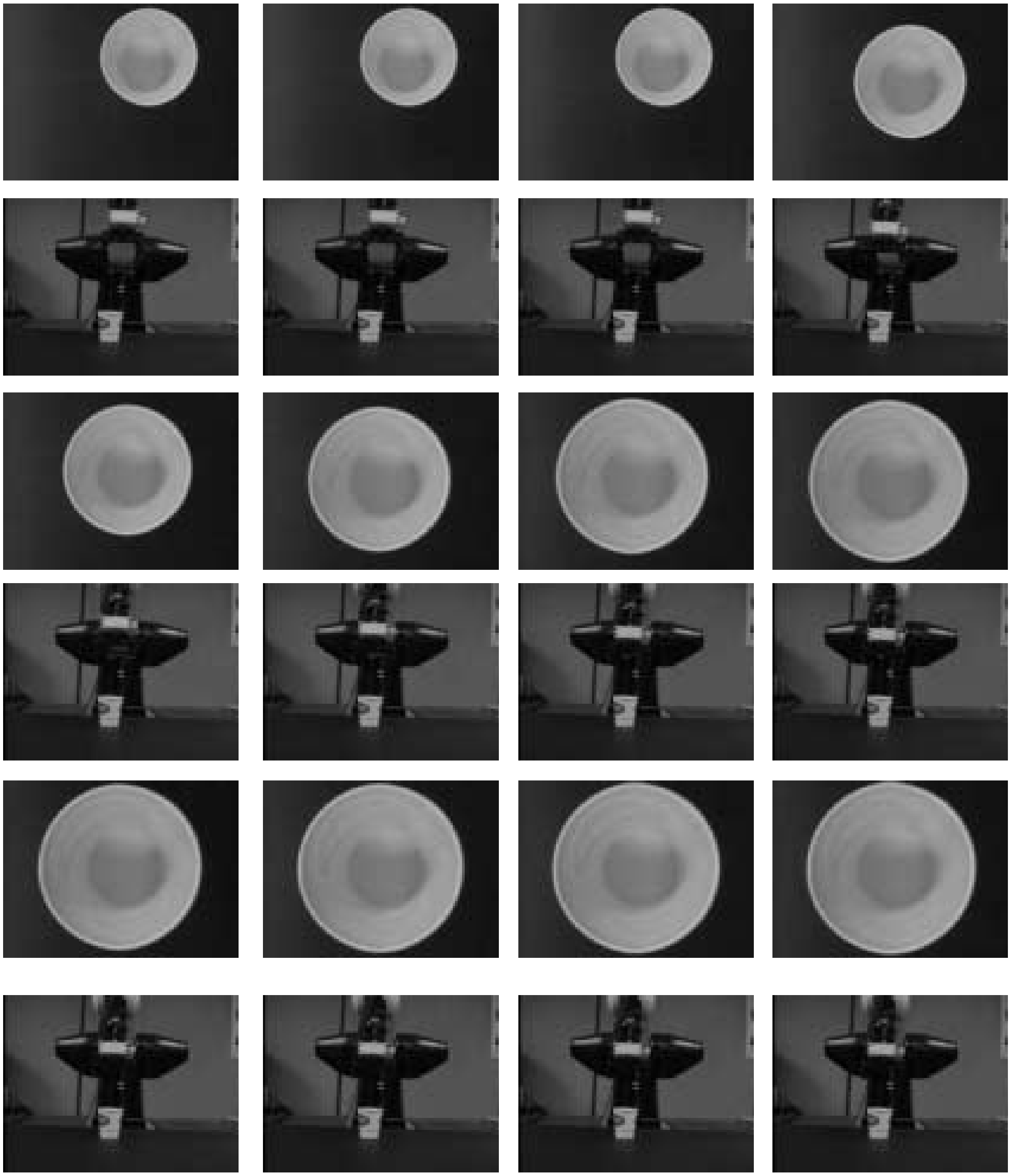


Figure 31: Visual Servoing Frame Sequence

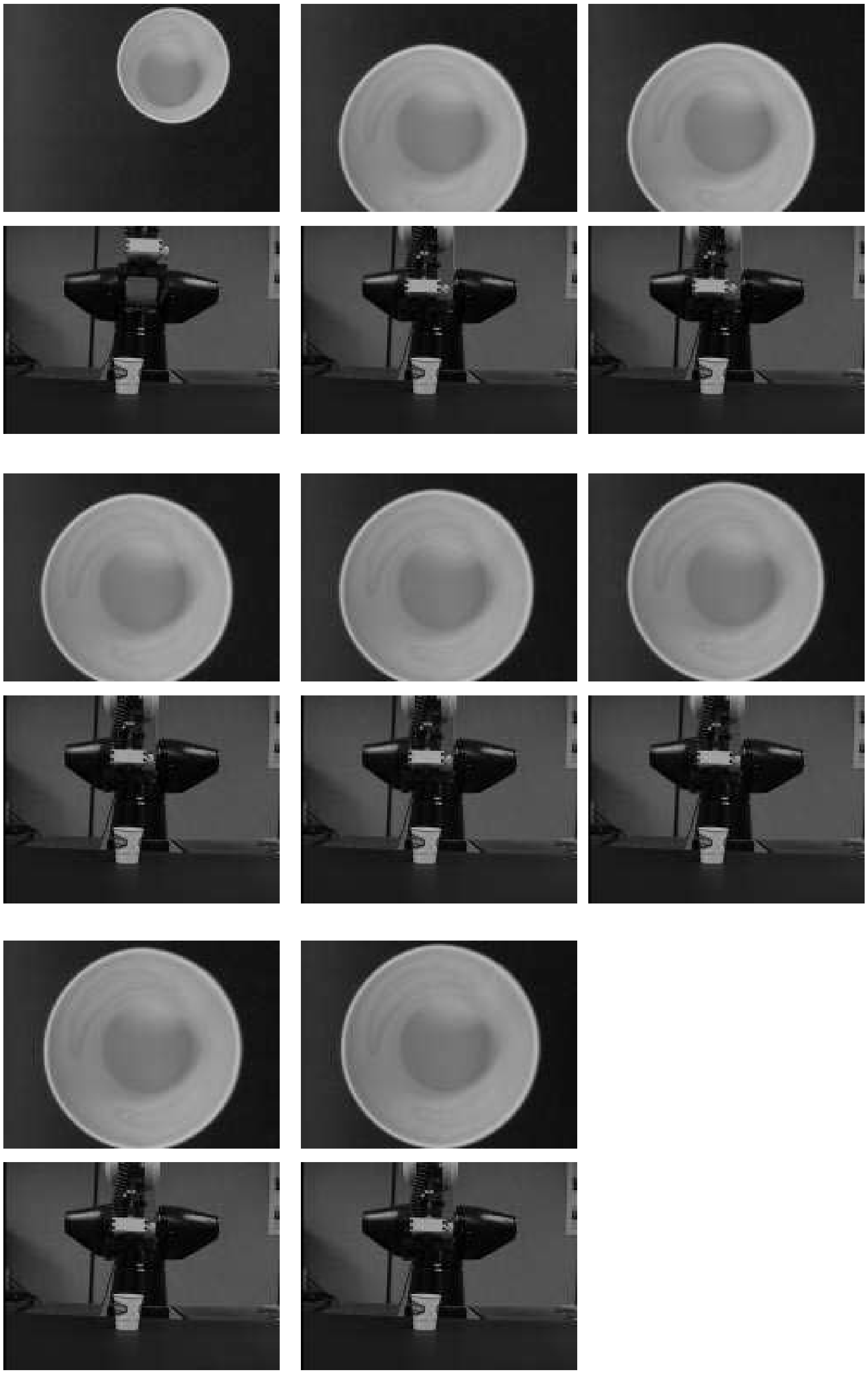


Figure 32: Linear Hybrid System Frame Sequence

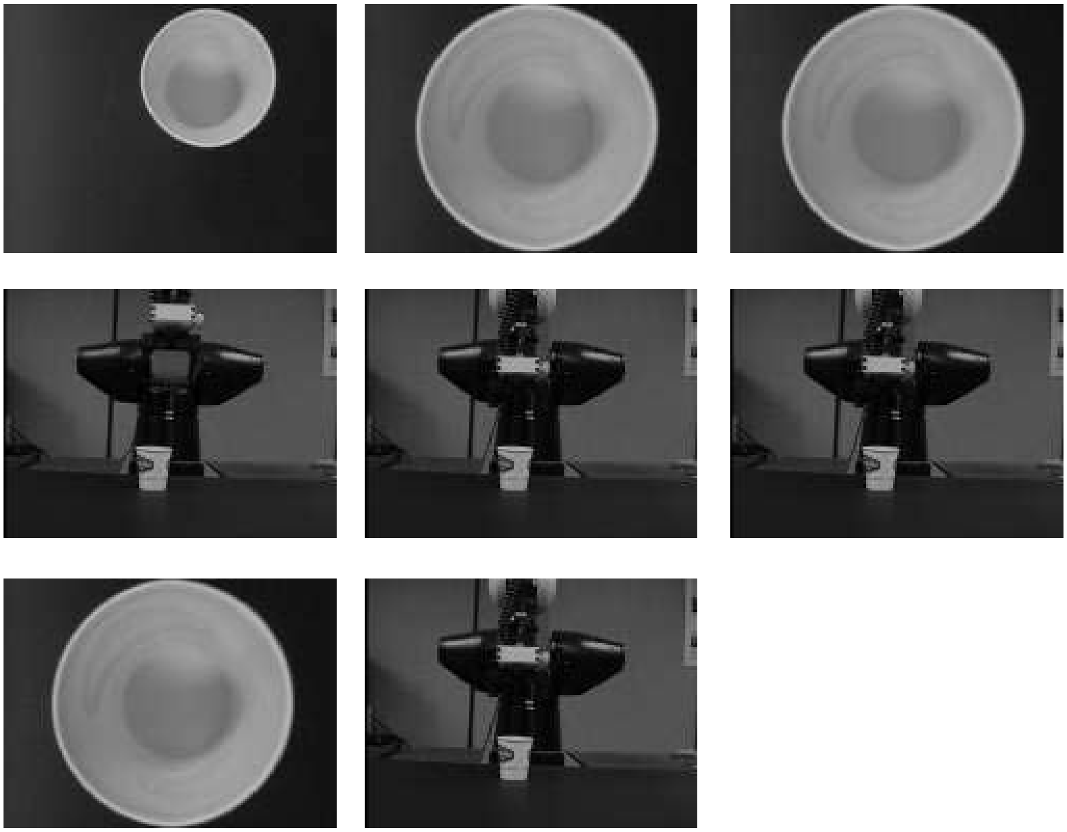


Figure 33: Neural Hybrid System Frame Sequence

The behaviors that were evident in the graphs presented in Figure 22 – Figure 30 can also be observed in the image sequences. The first sequence, corresponding to the visual servoing algorithm, demonstrates the gradual decent of a control algorithm. The final four frames show the small adjustments made by the overdamped control system. The second sequence, corresponding to the look and move system, shows the increased performance from using a calculated kinematic approach first. The second frame of the sequence, corresponding to the system state after the look and move portion, is much closer to the target than the second move of the visual servoing system. The second sequence also visually demonstrates the tolerance for clipping and occlusion exhibited by visual servoing. Even though the look and move portion overshot, clipping the target in the next image, the visual servoing system still converged to the desired position. The final sequence demonstrates the possible performance gains from non-linear optimization. The neural system required only a single iteration of the small gain controller to zero the target.

8.1.3 Discussion

Our results have shown that a vision guided robotic system obeying the constraints in section 2.1 can be designed. We have shown increased performance from combined calculated kinematics and visual servoing techniques, which strike a balance between speed and accuracy. The number of iterations in all cases is low, so small gains in bandwidth can result in very fast response times.

The time response of the system is actually quite slow, and is not suited for tracking tasks. There are some situations where the system can be used to grasp moving objects, which are traveling at low speed along a conveyor, but it is not appropriate for general tracking tasks.

Visual servoing relates directly to one of the assumptions outlined in section 2.1. The object will, in general behave more like a rigid body in projection the closer the body is moved towards the center. Distortions and apparent motion due to the motion of the manipulator will tend to zero the more ideally orthogonal the image projection becomes. As the robot zeros the target, the target behaves more closely obeys the assumptions.

8.2 Grasping and Grasp Planning

To demonstrate the capabilities of the grasp planning system, we evaluate its solution and computational performance for both synthetic and real images. The performance of the grasp planner can be characterized by its ability to find grasps for a wide variety of targets satisfying the constraints described in section 2.1, and the speed at which it arrives at those solutions. The grasp planner will also be demonstrated with a real world application in section 8.3, where the operation of the grasp planner in conjunction with the robot positioning system is demonstrated.

8.2.1 Experimental Setup

For the evaluation of the grasp planning system images of the target are fed directly into the planning system. Some have been artificially created using CorelDraw; others have been captured using the end-effector-mounted camera. All text output has been eliminated, with the exception of the termination message for the purpose of time results. All image output with the exception of the sampled and final point display has also been eliminated. We use the Real Time Clock of the QNX operating system, accurate to within 0.01 ms for all timing results.

8.2.2 Results

The algorithm was tested on several classes of shapes to determine its ability to correctly and efficiently determine acceptable grasp points for all types of objects. Objects were categorized into four broad classes:

1. Straight Convex Objects

2. Curved Convex Objects

3. Straight Concave Objects

4. Curved Concave Objects

The behavior of the algorithm for each of these classes of algorithms is detailed in the following sections.

8.2.2.1 Straight Convex Objects

Objects with straight sides and convex profiles are common in industry. This set includes the common convex polygons. In order for the grasp planner to be generic it must capture the simple as well as complex cases. Not only do we wish the grasp planner to find acceptable grasp points for simple shapes, we wish the grasp planner to find good grasp points. For convex polygons there are many acceptable grasp points and the purpose of the grasp planner is to locate the best grasp point from among the many candidates. In this case the distance metric must distinguish between points on the periphery of the target and grasp points whose contact line passes though the center of mass of the object. In this case the d_{cm} measurement and coefficient must be large enough to distinguish between points of equal parallelism and curvature. To demonstrate the behavior of the system for this case we have used a square block as shown in Figure 34. The first image is the sampled grasp points connected by a line. The second image is the selected grasp point.

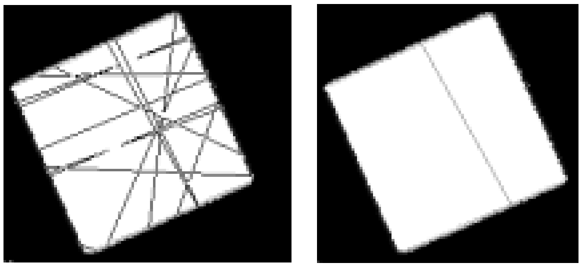


Figure 34: Results for a Square

As is apparent from Figure 34 the system is generates several points at a resolution of 16. The selected grasp points form the line passing near the center of the square. The points corrupted by noise are rejected because of the curvature constraint. The points that satisfy the quality constraint are beaten by the one closest to the center of mass. There may be points slightly close to the center of mass that were not sampled, but it is not the intention of the algorithm to return the optimal grasp point, but the best grasp point at a given level of resolution.

8.2.2.2 Curved Convex Objects

The class of curved convex objects includes completely curved objects such as ellipses as well as mixed curved and straight objects commonly found in industry. The algorithm should behave in a similar fashion for curved convex objects as it does with straight convex objects. The algorithm should select good grasp points over acceptable grasp points based on the position of grasp points relative to the center of mass. To illustrate the behavior of the algorithm for curved convex objects, we a circular sleeve. The results of the algorithm are shown in Figure 35. The first image is the sampled grasp points connected by a line. The second image is the selected grasp point.

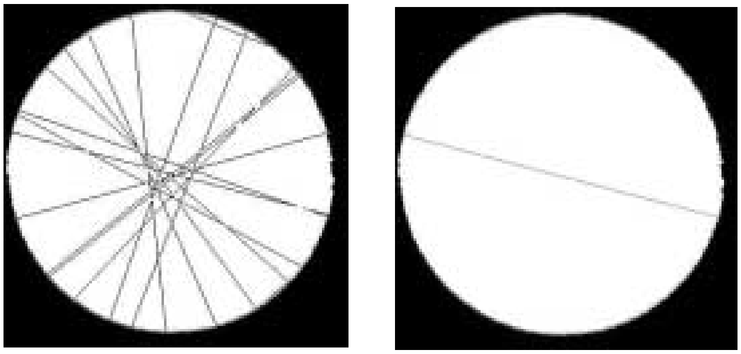


Figure 35: Results or Circle

The grasp planner selected a radial rather than a cord grasp point pair, rejecting non-radial grasp points on the basis of curvature and distance from the center of mass.

8.2.2.3 Straight Concave Objects

The class of straight edged concave objects includes all the non-convex polygons. Non-convex polygons are commonly found in flanges, braces and other assembly pieces. The requirements for grasp planning for non-convex polygons are similar to the requirements for convex polygons. The grasp point must be stable and near the center of mass. An additional constraint needs to be added to ensure that the grasp is physically valid, and that the grippers of the end-effector do not collide with concave sections of the target. To illustrate the behavior of the system with respect to concave polygons, we demonstrate the results with a real and synthetic target.

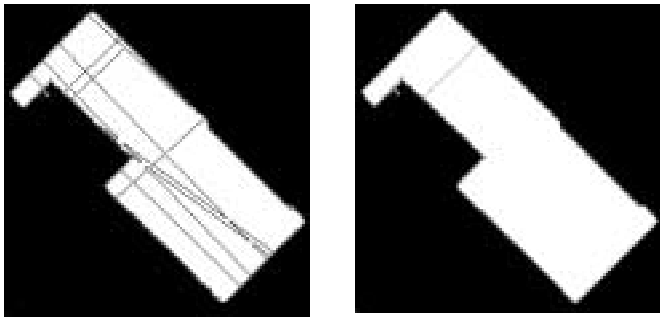


Figure 36: Results for a Brace

The solution is a compromise between the distance from the center of mass and the size of the target. The grasp points along the longer axis of the bracket were rejected because the bracket's length is longer than the distance between the gripper fingers. The solution was achieved after 16 iterations; however it should be apparent from the figure that there are other much better grasp points which were not initially sampled. This is the major drawback of our approach, it finds acceptable grasp points quickly, but may leave good grasp points unexamined.

To further verify our results for this more complex type, and to demonstrate our collision checking capabilities, we have run the grasp planner on a complex shape generated using CorelDraw's elastic band tool. The results are shown in Figure 37.

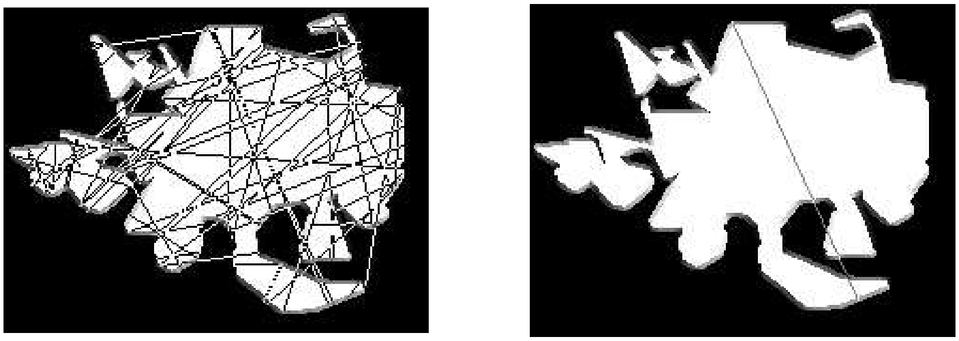


Figure 37: Results for a Complex Polygon

As is apparent from the figure, many more samples were required to generate an acceptable grasp point. The final grasp point comprises a point and a line, a common and logical result for an object this complex.

8.2.2.4 Curved Concave Objects

The class of curved concave objects includes all curved non-convex objects and all objects with mixed straight and curved sides with concavities. This is the most general and complex set. The only assumption we make about this group of objects is that the boundary is piecewise continuous. For this class of shapes, the focus is less on finding the best possible grasp point, and more on finding an acceptable grasp point quickly. In this case, the d_{cm} measure and its coefficient must be small enough not to disqualify points far away from the center of mass if they have good curvature and parallelism values. The algorithm still uses d_{cm} to distinguish between two equally parallel and curved grasp points, but few cases are expected in this class of objects. We expect the class of curved concave objects to require more iterations because the number of potential grasp points is small. The number of invalid grasp points due to grasp length and collisions are also expected to be much higher than the previous cases. To demonstrate the behavior of the system with respect to curved concave objects we used the arbitrary shape shown in Figure 38

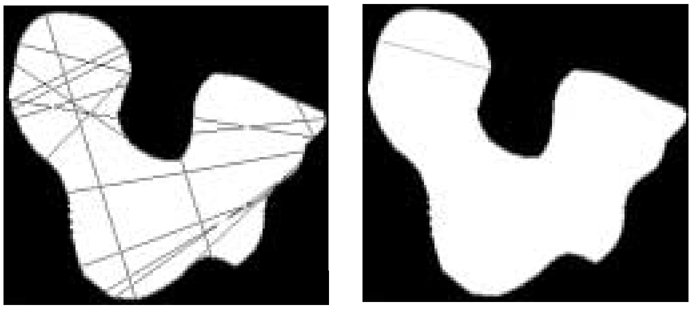


Figure 38: Results for Arbitrary Shape

In this case the algorithm returned a solution much faster than we expected, requiring only two levels of expansion or 32 windows, to generate an acceptable grasp. The grasp satisfies the collision, parallel and curvature constraints quite well, and is a significant distance from the center of mass, demonstrating that the d_{cm} value and coefficient are properly tuned for this class of targets.

To demonstrate the limits of this algorithm, we have tested the grasp planner on a synthetic target generated using the ellipse tool in CorelDraw. Many ellipses were merged to create a curved object with only locally continuous boundary segments. The boundary is concave in many places, and should generate several collisions.

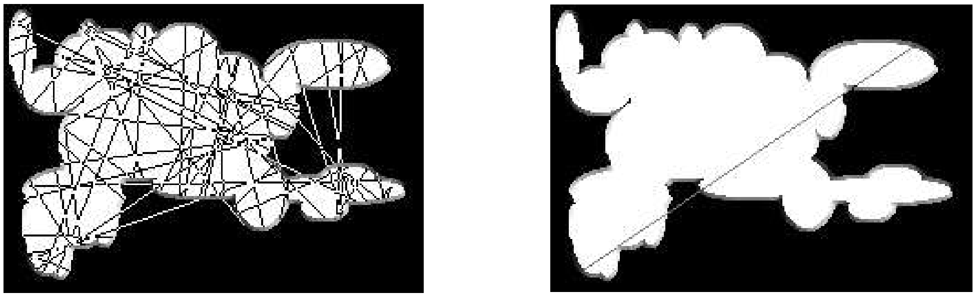


Figure 39: Results for a Very Difficult Target

Again, while the grasp planner was required to generate many more points to find a solution, a solution was found. The final solution is again a point and a line, or in this case, a point and a curve.

8.2.3 Timing Results

The primary benefit of our work is an algorithm that is designed to take an exponential time problem, and reduce it to a smaller complexity by exiting early when an acceptable grasp point is found. In order to measure determine the actual time behavior of the system, A timer was added to the main program loop. At the end of each grasp point determination, the required level of expansion and the amount of time elapsed were recorded. Theoretically, the algorithm should be exponential with the level of expansion;

however, this is not the case. The *if* clauses and the pruning of non-boundary nodes reduces the problem too less than an exponential complexity. A diagram of processing time versus level of expansion is shown in Figure 40.

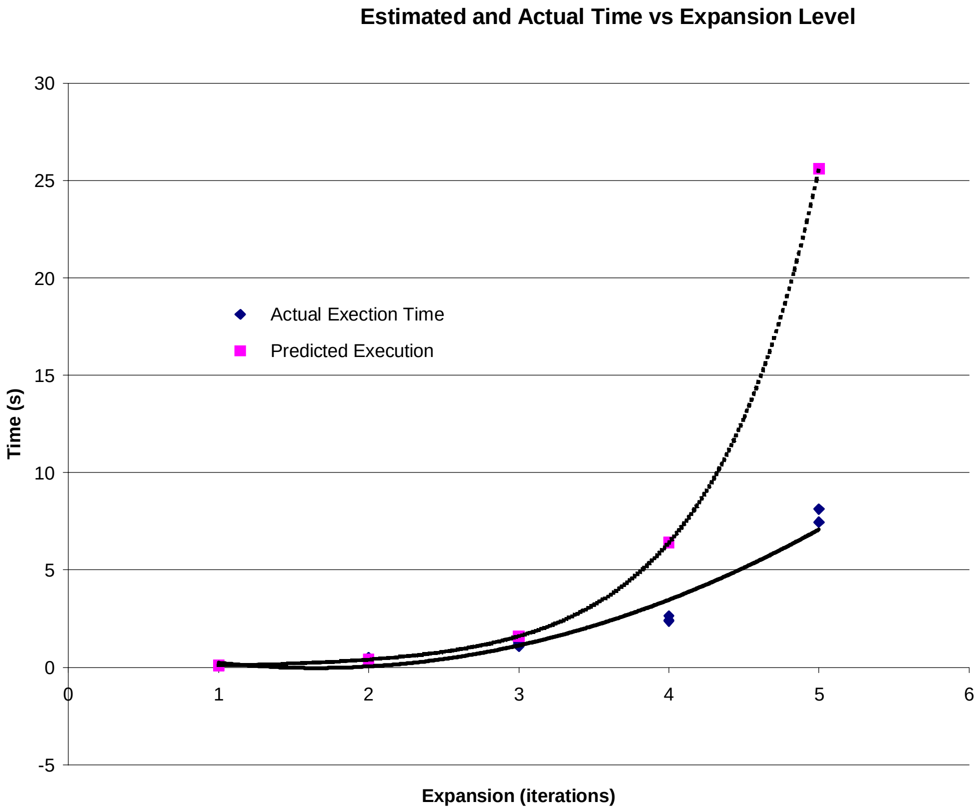
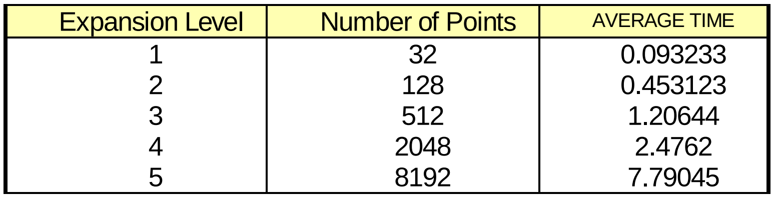


Figure 40: Execution Times

The top curve denotes the ideal exponential. The bottom curve represents the measured response. The measured results are approximated by a polynomial curve of degree 2. The difference in the two curves becomes apparent after 3 iterations. The difference in computation time becomes significant at the fourth level of expansion. This is primarily due to discarding non-edge nodes and early stopping on analysis of unpromising grasp points.

Reading from the graph, the amount of time required to plan a grasp varies from less than a second to more than seven seconds depending on the complexity of the object. The table below shows the average time for each level of expansion as well as the number of points examined in that level.

Table 6: Average Execution Times per level of Expansion



Expansion Level	Number of Points	AVERAGE TIME
1	32	0.093233
2	128	0.453123
3	512	1.20644
4	2048	2.4762
5	8192	7.79045

The algorithm manages to find grasp points very quickly. For most simple applications, a grasp point can be found within one or two levels of expansion. More complex objects require more time but can also be analyzed by the same algorithm. In all cases grasp planning is completed in a matter of seconds.

8.3 A Simulated Sorting Application

In order to demonstrate the possible practical applications of the complete system, we have assembled a sorting cell, which could be used for quality assurance. The cell consists of the manipulator camera system running the neural network hybrid positioning system and using the grasp planner after the target has been zeroed. The sorting application, which is completely fictitious, distinguished between a round, “good” dixie cup and, a crushed “bad” dixie cup. The sorting algorithm is based on a single feature measurement, sometimes called the compactness, given by Eq. (43).

$$f_c = \frac{p^2}{4\pi A} \qquad (43)$$

Compactness is a measure of an object's roundness [36]. A circle ideally should evaluate to one, as shown in Eq. (44).

$$f_c = \frac{(2\pi r)^2}{4\pi(\pi r^2)} = 1 \qquad (44)$$

In practice, there is some variance in the value of f_c because pixel based perimeter measurements can be quite noisy. We define an intact cup as one that retains a certain level of roundness. The robot is places the cup in one of two spots depending on results of the sorting algorithm.

8.3.1 Experimental Setup

Using the simple sorting mechanism described above, cups in various conditions were placed in front of the robot. The robot moves so the target is centered in the field of view and is 155 pixels above the target. The grasp planner then identifies a set of stable grasp points on the periphery of the target, and the compactness of the target is evaluated to determine if the cup was sufficiently intact. The cup is grasped and moved to the appropriate destination. We defined the reject location as to the left of the robot (facing the robot) and the accepted location to the right of the robot (facing the robot). Image sequences showing the results are presented in the following section.

8.3.2 Results

The results of the sorting system are presented as sequences of images corresponding to the operation of the system as a whole for different initial conditions. The numerical results for both the neural hybrid vision guidance system and the grasp planner are presented in preceding sections. The purpose of these results is to demonstrate the functioning of the system as a whole, rather than the exact responses of each component. Each sequence of images contains nine frames. The first six frames represent the initial, middle, and final locations of the robot during the servoing sequence. The following three frames show the sampled and final grasp points, as well as the post grasp move to the appropriate location.

Figure 41 shows the response for a good cup. Figure 42 shows the response for a dented cup. Figure 43 shows the result for a heavily dented cup. Figure 44 demonstrates the response for a marginal cup. Figure 45 illustrates the robustness of the system by interjecting a contaminant.

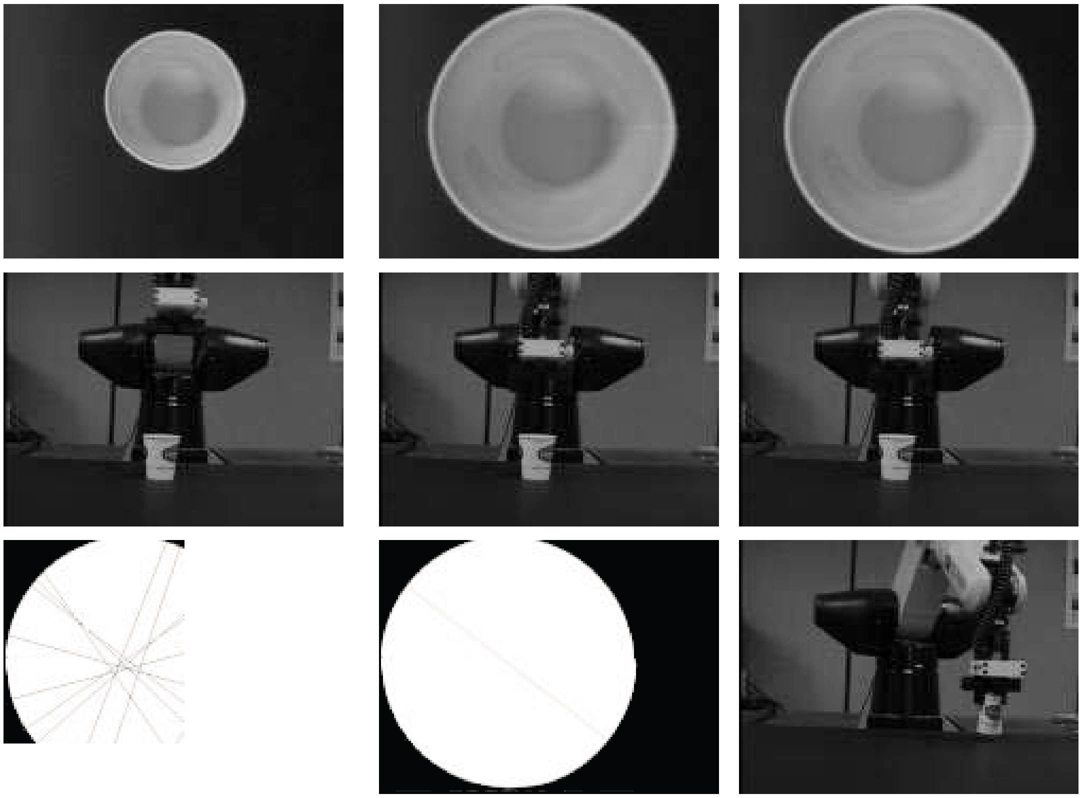


Figure 41: Sorting Frame Sequence for a Good Cup

As is apparent from Figure 41, the system correctly sorted the round cup as “good” by placing it on the right. As expected with the neural hybrid system, the primary move resulted in the cup near the center of view, and the servoing system was only used to perform small corrections.

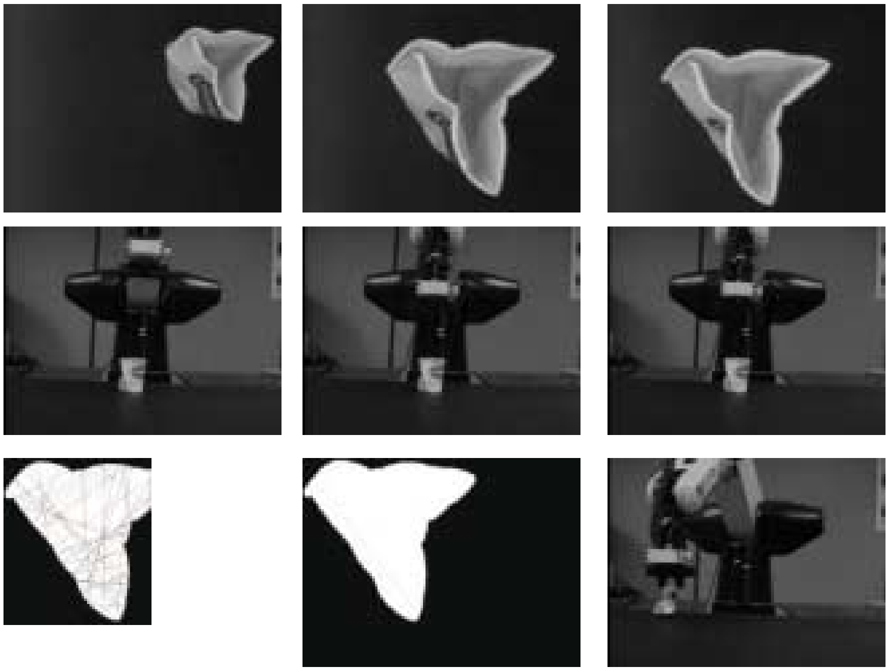


Figure 42: Sorting Frame Sequence for a Crushed Cup

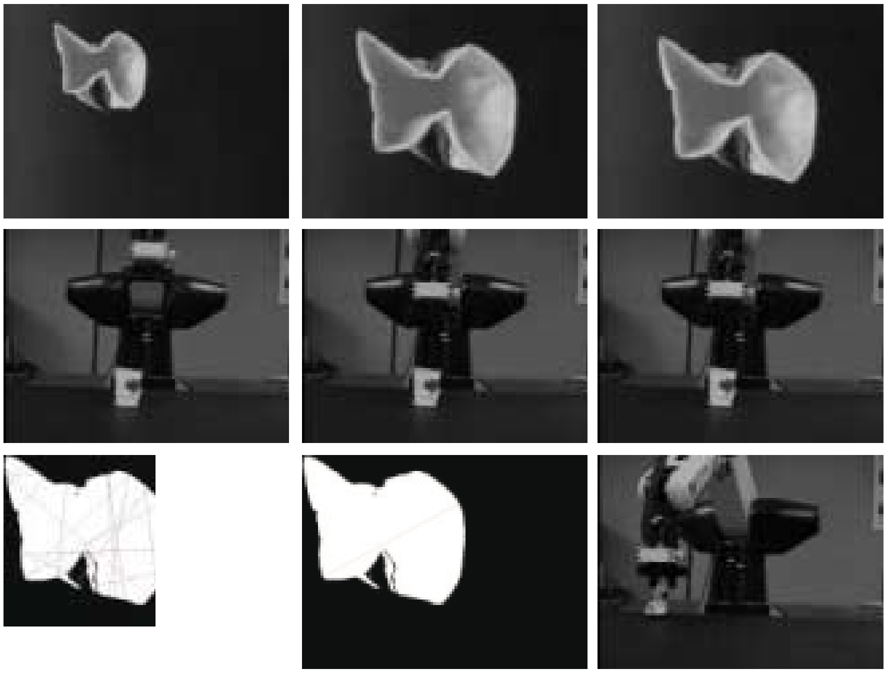


Figure 43: Sorting Frame Sequence for a Badly Crushed Cup

As is apparent from Figure 42 and Figure 43, the system is equally adept at sorting deformed objects. There was little change in the response of the neural network hybrid system due to target type because the input features are impervious to variations in target. Figure 43 shows that the visual servoing system is robust even to the assumption that the targets behave like rigid bodies in projection. In the opening frame the target's projection is heavily weighted to the side of the target, providing a perspective rather than an orthogonal view. After the initial neural network based look and move portion, and several visual servoing steps, the target is nearly centered. Similarly, a strong side projection in the following image is also compensated.

The grasp planner's capabilities are demonstrated for the above situations. While the two objects are strikingly different in shape and complexity from the first target, the grasp planner does quickly generate a suitable grasp point. Note that the grasp points for more complex objects generally involve an edge and a point. These combinations tend to be the most stable collision free configurations on highly concave targets. Even though there is no express evaluation checking for these kinds of solutions, they are generated by the grasp planner based on the fitness function.

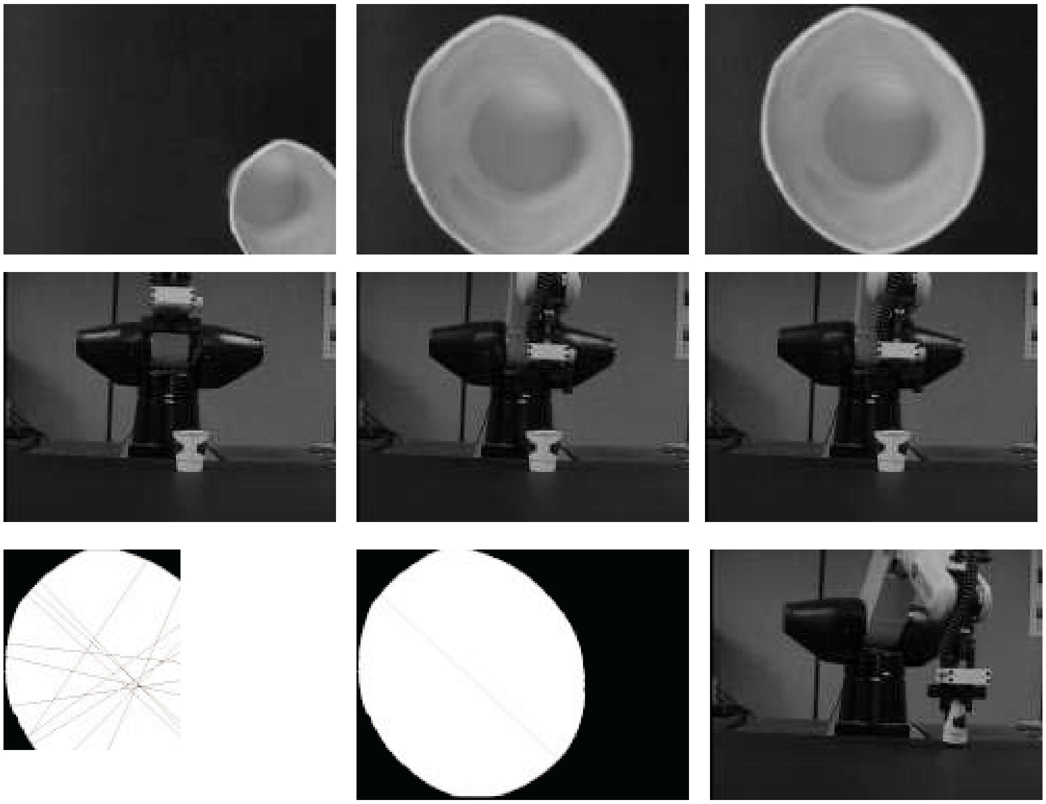


Figure 44: Sorting Frame Sequence for a Marginal Cup

Having demonstrated the successful sorting of a flaw-free and two heavily flawed cups to demonstrate the bounds of operation of the motion and grasp planning systems, we added a marginal cup that had been deformed into an oval. We used this cup to evaluate the quality of sorting, and grasp planning for a slightly non-ideal target. As is apparent in Figure 44, the system behaved quite well, classifying the object as a good cup and quickly generating grasp points.

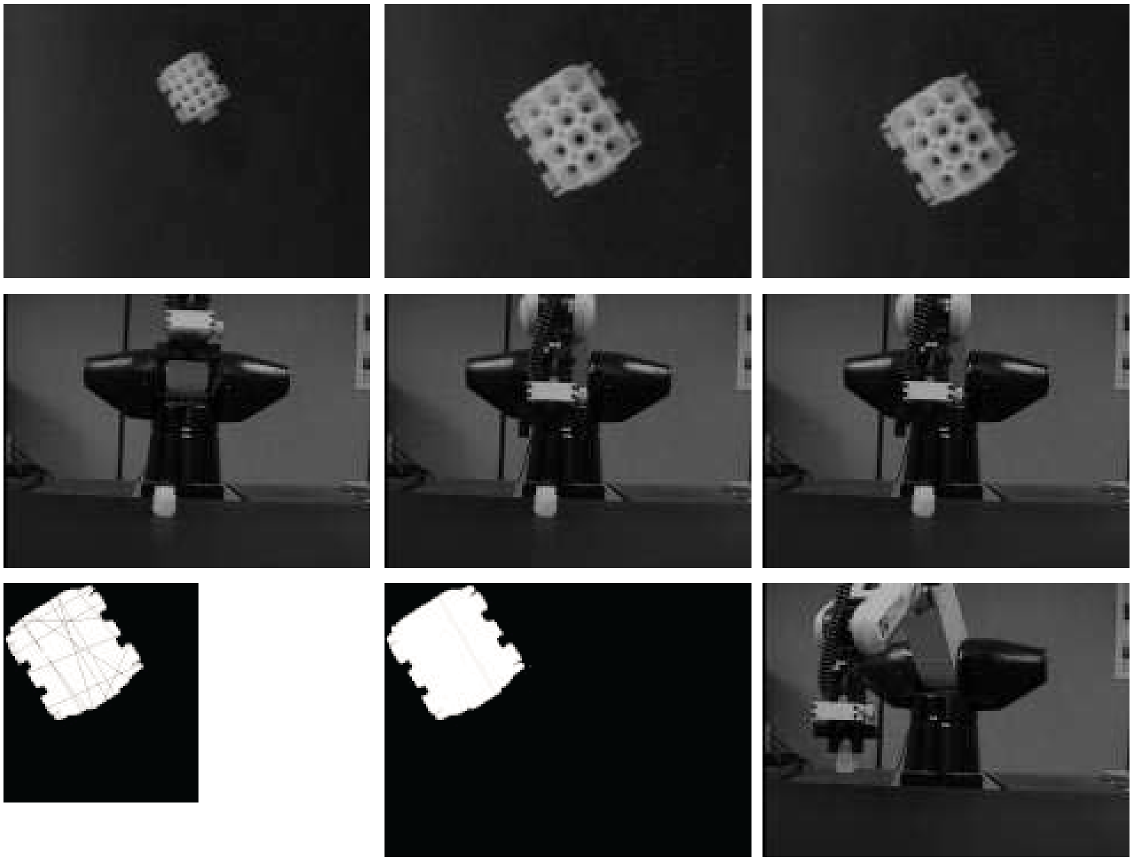


Figure 45: Sorting Frame Sequence for a Contaminant

Finally, as shown in Figure 45, we added a contaminant to the system. The contaminant was an electrical plug header and connection common in electric circuit boards. Even though the target is again very different from the targets in the previous sections, the target was approached, a grasp was planned and the object was correctly classified as a non-ideal cup. This example demonstrates the flexibility of the system.

9 Conclusion

We have shown the effectiveness of each component of the system, and the system itself using both scientific and practical demonstrations. The system is currently under development for target applications in both sorting and tele-operation applications where the targets are undefined, but well behaved.

9.1 Summary

We have developed and demonstrated a four-degree of freedom, generic vision guided robot system. The system was designed to solve a general class of problems, grasping well-contrasted, visible, rigid, stationary objects. We achieved a high level of generality by incorporating general measures at all levels of our design. By ensuring that only our assumptions about the size and color of the target were necessary for proper operation, we were able to create a very useful tool or building block. We chose our input features to be target independent for rigid objects. We used an orthogonal camera setup to limit to changes in target image projections. By ensuring that the input always had the same meaning, and fell within well-established boundaries, it was possible to directly apply the same positioning algorithm to any target.

To position the system, we used a hybrid computed kinematics and visual servoing system. The system would perform a rapid, but relatively inaccurate move based on computed kinematics, then a series of correction moves based on a PD controller operating in a visual servoing loop. The computed kinematics system could be based on a linear approximation around an operating point, or a neural network approximation. The linear approximation was based on an input image partially corrected for rotation and scale. The result was a reasonably accurate approximation of the kinematics. The neural network approximation was based on the training of a backpropagation network. The training data was generated by sampling the robot workspace in two separate operating regions, approximating the cone of vision. In conjunction with the computed kinematics, a PD controller was used to correct small residual errors in motion. The visual servoing system used two gains, an underdamped gain to rapidly approach the solution, and an overdamped gain to slowly approach the global minimum. The overall result was a three-phase system that offered a balance between speed and accuracy.

Manipulation of arbitrary targets required a grasp planner. We designed a grasp planner based on a quadtree expansion of the image. We assumed that the target would have several graspable points, and that finding a sufficient, but not necessarily optimal point was good enough to manipulate most targets in an unknown environment. The grasp planner employed a fitness function to evaluate grasps for stability and quality, and a collision checker to evaluate them for validity. The best grasp point at the current level of the quadtree was examined, and if it was less than a user-specified minimum, the algorithm terminated. Otherwise, the algorithm created another level to the quadtree, and repeated the evaluation algorithm.

The system was tested using several methods. First, we compared both hybrid algorithms with a visual servoing algorithm for a standard object and a set of objects. The neural network hybrid preformed marginally better than the look and move hybrid, and both converged significantly faster than the visual servoing system alone. The grasp planner was tested using both synthetic and real targets, and was found to be efficient for many common targets which fit our assumptions. The whole system was tested using a simulated sorting task where paper cups in various conditions were placed before the robot, and the robot sorted them based on their roundness.

9.2 Future Work

The practical vision guided robotic system is a valuable tool, but several improvements could be made to the system itself, and the system could be applied to a large number of tasks.

In the short term, the gains of the PD controller for the neural hybrid and look and move hybrid systems should be tuned to exclusively work with those systems, instead of using a generic system for both. Additional image pre-processing should be added to ease the contrast requirements for the target, allowing operation in even less structured targets and lighting.

The majority of long term future research should focus on the sensors. The system should be capable of handling a larger range of objects and a larger range of tasks. To handle objects in six degrees of freedom, a stereo vision system should be added to extract the depth of the target. The system should also be able to handle contact tasks that require force/torque and tactile sensors. A sensor fusion engine is needed to generate the same four-vector control input based on much richer sensory input.

The system will also be applied to industry including sorting and tele-operation tasks. The tele-operation of the system will require a user interface, allowing the user to specify targets in the image for the robot to manipulate. This could be especially useful in space or rehabilitation applications, where the targets are not exactly positioned but well known. In order to deploy the system as a sorting system, the bandwidth of the system will have to be increased.

9.3 Conclusions

The system manipulates a wide range of discrete parts in four degrees of freedom without models of the targets. We have demonstrated that an efficient approach to vision guided robotics with limited bandwidth is a hybrid computed kinematics and visual servoing system. Limited bandwidth applications will be the only applications in industry for several years because robotic vendors are unwilling to provide complete access to their controllers, and low cost vision systems will not have the required ability to operate at speeds on the order of 1 kHz. We have also demonstrated that grasp planning can be performed by examining the image without a model reconstruction. The technology and hardware cost now allow vision-guided robotics in practical applications. The proposed system has commercial potential for both tele-operation and sorting applications.

10 References

1. Seth Hutchinson, Gregory Hager, and Peter I. Corke, “A Tutorial on Visual Servo Control”, *IEEE Trans. on Robotics and Automation*, Oct 1996, pp. 651-669.

2. Peter I. Corke, Malcolm C. Good, “Dynamic Effects in Visual Closed Loop Systems,” *IEEE Trans. on Robotics and Automation*, Oct 1996, pp. 671-683.

3. Nikolaos P. Papanikolopoulos, Pradeep K. Khosla, “Adaptive Robotic Visual Tracking: Theory and Experiments,” *IEEE Trans on Automatic Control*, March 1993, pp. 429-445.

4. T.W. Miller III, “Neural Networks for Sensor Based Control of Robots with Vision,” *IEEE Transactions on Systems Man and Cybernetics, Vol. 19, No.4*, 1989, pp. 826-831.

5. H. Hashimoto, T. Kubota, M. Kudon, and F. Harashimo, “Self-Organizing Visual Servo System Based on Neural Networks,” *American Control Conference, Boston MA, 1991*, pp. 31-36.

6. J. Wu and K. Stanley, “Modular Neural-Visual Servoing using a Neural-Fuzzy Decision Network,” *IEEE Conference on Robotics and Automation, Albuquerque, 1997*, pp. 3238-3243.

7. Q.M.J. Wu, C.W. de Silva and Kevin Stanley “Neural Control Systems and Applications,” *Intelligent Adaptive Control: Industrial Applications*, CRC Press, 1998.

8. P. van Der Smagt, F. Groen, “Approximation with Neural Networks: Between Local and Global Approximation,” *IEEE International Conf. on Neural Networks, 1995*, pp. 1060-1064.

9. J. T. Feddema, C .S. G Lee, O. R. Mitchell, “Weighted Selection of Image Features for Resolved Rate Visual Feedback Control,” *IEEE Transactions on Robotics and Automation, Vol. 7, No 1,* 1991, pp. 31-47.

10. Hashimoto, Koichi and Noritsugu, Toshiro, “Performance and Sensitivity in Visual Servoing,” *IEEE Conf. on Robotics and Automation, Leuven, Belgium,* May 1998, pp. 2321-2326.

11. S. Nayar, H. Murase, and S. A. Nene, “A General Learning Algorithm for Robotic Vision,” SPIE Vol. 2304, 1994, pp. 10-17.

12. D. Anguita, G. Parodi, and R. Zunino, “Neural Structures for Visual Motion Tracking,” *Machine Vision and Applications,* August 1995.

13. Kevin Stanley, Q.M.J. Wu, Ali Jerbi and William A. Gruver, “Neural Network-Based Vision Guided Robotics,” *Proceedings of the IEEE Conference on Robotics and Automation, 1999,* pp. 281-286.

14. Ching-Cheng Wang, “Extrinsic Calibration of a Vision Sensor Mounted on a Robot,” *IEEE Trans. on Robotics and Automation,* Vol. 8, No. 2, April 1992, pp. 161-175.

15. Radu Horaud, Fadi Dornaika, Bart Lamiroy, and Stephane Christy, "Object Pose: The Link between Weak Perspective Paraperspective and Full Perspective," *International Journal of Computer Vision* 22(2), 1997, pp. 173-189.

16. Guo-Qing Wei, Klaus Arbter, and Gerd Hirzinger, "Active Self-Calibration of Robotic Eyes and Hand-Eye Relationships with Model Identification," *IEEE Trans. on Robotics and Automation,* Vol. 14, No. 1, February, 1998, pp.158-166.

17. Hanqi Zhuang, Kuanchih Wang, Zvi S. Roth, "Simultaneous Calibration of a Robot and a Hand-Mounted Camera," *IEEE Trans. on Robotics and Automation,* Vol. 11, No. 5, 1995, pp. 649-660.

18. S. Remy, M. Dhome, J. M. Lavest, N. Daucher, "Hand-Eye Calibration," *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1997 pp. 1057-1065.

19. Van-Duc. Nguyen, "Constructing Stable Grasps," *Int. J. Robotics Research,* Vol. 8, No. 1, February 1989, pp. 3-16.

20. Jean Ponce, Darrell Stam, and Bernard Faverjon, "On Computing Two-Finger Force-Closure Grasps of Curved 2D Objects," *Int. J. Robotics Research, Vol. 12, No. 3.*, June 1993 pp. 263-273.

21. David. J. Montana, “Contact Stability for Two-Fingered Grasps,” *IEEE Trans. on Robotics and Automation,* Vol. 8, No. 4, August 1992, pp. 421-430.

22. K. B. Shimoga, “Robot Grasp Synthesis Algorithms: A Survey,” *Int. J. Robotics Research,* Vol. 15, No. 3, June 1996, pp. 230-266.

23. Christian Bard, Christian Laughier, Christine Milesi-Bellier, Bill Triggs, and Gianni Vercelli, “Achieving Dexterous Grasping by Integrating Planning and Vision-Based Sensing,” *Int. J. Robotics Research,* Vol. 14, No. 5, October 1995, pp. 445-464.

24. Farrokh Janabi-Sharifi, and William J. Wilson, “Automatic Grasp Planning for Visual Servo Controlled Robotic Manipulators,” *IEEE Trans. on Systems Man and Cybernetics*, Vol. 28, NO. 5, October 1998, pp. 693-711.

25. Colin Davidson and Andrew Blake “Error Tolerant Planning of Planar Grasp,” *Sixth International Conference on Computer Vision*, 1998, pp. 911-916.

26. Michael Taylor, Andrew Blake, Adrian Cox, “Visually Guided Grasping in 3D,” *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994, pp. 761-766.

27. P.J Sanz, A. P. del Pobil, and J. M. Inesta, “Curvature-Symmetry Fusion in Planar Grasping Characterization from 2D Images,” *Proc. of Industrial Engineering Applications*

of Artificial Intelligence and Expert Systems, Amsterdam, The Netherlands, 1997, pp. 45-52.

28. P.J Sanz, A. P. del Pobil, and J. M. Inesta, and G. Recatala, "Vision Guided Grasping of Unknown Objects for Service Robots," *Proceedings of the 1998 IEEE International Conference on Robotics and Automation,* Leuven, Belgium, May 1998, pp. 3018-3025.

29. Kevin Stanley, Q.M.J. Wu, Ali Jerbi and William A. Gruver "A Fast Two-Dimensional Image Based Grasp Planner" *IEEE International Conference on Intelligent Robots and Systems, 1999,* Kyongju, Korea, Oct. 1999.

30. Christopher E. Smith Scott A. Brandt, and Nikolaos P. Papanikolopoulos, "Eye-In-Hand Robotic Tasks In Uncalibrated Environments" *IEEE Trans. On Robotics and Automation,* Vol. 13, No. 6, 1997, pp. 903-913.

31. Farrokh Janabi-Sharifi, and William J. Wilson, "Automatic Selection of Image Features for Visual Servoing," *IEEE Trans. on Robotics and Automation, Vol. 13, No. 6, December* 1997, pp. 890-903.

32. Q.M.J. Wu, Kevin Stanley, F. Deravi, and Dewey Liew, "Computer Vision," *The Encyclopedia of Electrical and Electronics Engineering,* John Wiley and Sons Inc., 1998.

33. Ramesh Jain, Rangachar Kasturi, Brian C. Schunk, *Machine Vision,* McGraw-Hill, New York, 1995.

34. Bernd Jahne *Practical Handbook on Image Processing for Scientific Applications,* CRC Press, New York, 1997.

35. John C. Russ, *The Image Processing Handbook*, CRC Press, Boca Raton, 1994.

36. E. R. Davies, *Machine Vision,* Academic Press, New York, 1997.

37. John J. Craig, *Introduction to Robotics: Mechanics and Control,* Addison-Wesley, Reading, Massachusetts, 1989.

38. Simon Haykin, *Neural Networks: A Comprehensive Foundation,* Macmillan College Publishing Company, New York, 1994.

39. QNX Operating System: System Architecture, QNX Software Systems Ltd., Kanata Ontario, Canada, 1997.