4-6-2010

Shape and Pose Recovery of Novel Objects Using Three Images from a Monocular Camera in an Eye-In-Hand Configuration

Steven C. Colbert
University of South Florida

Shape and Pose Recovery of Novel Objects Using Three Images from a Monocular Camera in an Eye-in-Hand Configuration

by

Steven C. Colbert

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Mechanical Engineering
Department of Mechanical Engineering
College of Engineering
University of South Florida

Major Professor: Rajiv Dubey, Ph.D.
Redwan Alqasemi, Ph.D.
Sudeep Sarkar, Ph.D.

Date of Approval:
April 6, 2010

Keywords: Shape Reconstruction, Shape from Silhouettes, Object Classification, Robot Vision, Machine Vision

Table of Contents

List of Tables

List of Figures

Shape and Pose Recovery of Novel Objects Using Three Images from a Monocular Camera in an Eye-in-Hand Configuration

Steven C. Colbert

ABSTRACT

Knowing the shape and pose of objects of interest is critical information when planning robotic grasping and manipulation maneuvers. The ability to recover this information from objects for which the system has no prior knowledge is a valuable behavior for an autonomous or semi-autonomous robot. This work develops and presents an algorithm for the shape and pose recovery of unknown objects using no *a priori* information. Using a monocular camera in an eye-in-hand configuration, three images of the object of interest are captured from three disparate viewing directions. Machine vision techniques are employed to process these images into silhouettes. The silhouettes are used to generate an approximation of the surface of the object in the form of a three dimensional point cloud. The accuracy of this approximation is improved by fitting an eleven parameter geometric shape to the points such that the fitted shape ignores disturbances from noise and perspective projection effects. The parametrized shape represents the model of the unknown object and can be utilized for planning robot grasping maneuvers or other object classification tasks.

This work is implemented and tested in simulation and hardware. A simulator is developed to test the algorithm for various three dimensional shapes and any possible imaging positions. Several shapes and viewing configurations are tested and the accuracy of the recoveries are reported and analyzed. After thorough testing of the algorithm in simulation, it is implemented on a six axis industrial manipulator and tested on a range of real world objects: both geometric and amorphous. It is shown that the accuracy of the hardware implementation performs exceedingly well and approaches the accuracy of the simulator, despite the additional sources of error and uncertainty present.

Chapter 1
Introduction

1.1 Motivation

Autonomous robotic manipulators are ubiquitous in many areas of industry, and the role these machines play in automated assembly lines and production and packaging facilities around the world is well known. These robots, while great in number, operate in relatively restricted environments; the task to be completed is well defined and the typical robot will execute the same motion repeatedly and tirelessly.

There is another class of robots, however, which seek to operate in unstructured environments; the tasks to be completed are typically not known beforehand, and the environment within which the robot operates is often unknown. These types of robots are typically designed to be service or assistance oriented. In contrast to their industrial counterparts, the unstructured environment places a heavy requirement on these robots to collect and interpret data from their surroundings such that they can intelligently and successfully complete a task. One of the most fundamental of tasks for an assistive robot is to grasp and pick up or otherwise manipulate a particular object of interest. It is this action that this work seeks to facilitate.

In order for a robot to manipulate an object of interest, several pieces of information must be known about that object before the action can commence. Chiefly, the objects' geometry must be fully defined to a sufficient approximation relative to the robot. That is to say, the location, orientation, shape and size of the object must be known in order to plan the appropriate robotic motions. Given an unstructured environment, this predicates the robot having the ability to interrogate its surroundings and recover the necessary information of the object to be manipulated.

There are two approaches to gathering this information: those that use prior knowledge of the objects, and those that do not. Under the first approach, the robot is programmed in advance with detailed knowledge of several objects it is likely to encounter during operation. This knowledge may

take several forms depending on the sensors the robot employs: camera, sonar, laser range finders, and others. In the end, the robot is provided with a database of objects it is capable of manipulating *a priori*. When the robot then encounters a new object in service, it attempts to match the features of this new object to one of its database models. When it finds the best match, it behaves accordingly. While this method of model based operation has proven accurate, it carries with it the fundamental limitation of being unable to handle objects which it can't identify or which do not fit within the description of one its pre-defined models. In a household situtation, where an assistive robot will be tasked with manipulating a plethora of various objects, this limitation becomes a significant hindrance.

The second approach to recovering the required information of the object is to reconstruct the geometry of the object at runtime based on the sensory data available to the robot: usually in the form of images or range data. Stereo imaging is extremely popular for this purpose, however, it has the drawback of yielding inaccurate results when the object of interest has little to no texture; a case that is often founded with household objects. Another popular method reconstructs the objects' geometry from several images of the object captured from various disparate locations. This method, termed 'shape from silhouettes' has the advantage of functioning effectively for objects with or without texture, but with the drawback of requiring an excessively large amount of images of the object, often rendering the procedure impractical for unstructured environments.

This work seeks to make progress in the run time reconstruction of the geometry of unknown objects by developing a practical shape from silhouettes algorithm that can achieve a sufficiently accurate reconstruction with a minimal number of images.

1.2 Objectives

The principle objective of this work is to develop and implement an algorithm that is capable of reconstructing, to a sufficient degree of accuracy, the three dimensional shape, pose, and orientation of a novel object using a monocular camera mounted on the end-effector of a robotic manipulator for the purposes of grasp and manipulation planning. The specific goals of the algorithm are listed as follows:

- Reconstruct the shape, pose, and orientation of a novel object to a sufficient degree of accuracy that permits the planning and execution of grasping and manipulation maneuvers.

- Calculate these parameters using no *a priori* information of the object, with the exception that a given object is identified as the object of interest.
- Require only a minimal number of images for reconstruction; significantly fewer than the status quo.
- Operate efficiently, such that calculation time of the object parameters is negligible compared to the time required for the robot to capture the images.

1.3 Outline

The rest of this work is divided into six chapters. Chapter 2 provides a short literature review concerning the state of the art of object reconstruction and covers the methods that are model based as well as runtime based. Chapter 3 provides the relevant mathematical and scientific background for understanding the algorithm development. Topics such as coordinate transformations, camera models, machine vision techniques, and superquadrics are covered in this chapter. Chapter 4 presents the development of the reconstruction algorithm. Chapter 5 details the software simulation environment as well as the results of testing the developed algorithm in simulation. Chapter 6 concerns the development of the hardware system as well as the results of testing the algorithm on the hardware. Chapter 7 provides a closing discussion as well the plan for future work and development of the algorithm.

Chapter 2
Literature Review

There has been much recent research in applying visual sensory data to the control of a robotic manipulator in unstructured environments for the expressed purpose of grasping or otherwise manipulating objects. Much of the work makes uses of some form of predefined object models, which limits the practicality in terms of broad adoption and adaptability to new or unknown environments. There has also been research and developments in shape and object reconstruction that is not model or prior knowledge based, though the amount of research and progress in this area is comparatively small.

As the research presented in this thesis makes use of developments from both the model and non-model based areas of research, recent developments in both of these areas are presented in this chapter.

2.1 Model Based Recovery

Kim, Lovelett, and Behal [11] have developed a wheelchair mounted robotic arm that can grasp and manipulate textured objects associated with Activities of Daily Living (ADLs). The system uses a narrow baseline stereo camera and a large database of template images. Once the user identifies the object of interest on the video screen, the system utilizes the stereo camera to perform a sparse stereo reconstruction based on SIFT features [9]. This rough approximation is used to converge on the object to a close proximity, at which point new SIFT features are generated and matched to a template in the database. The final grasping is executed via visual servoing to orient the manipulator in the pose dictated by the template. The authors overcome some of the prominent challenges in planning grasping maneuvers in unstructured environments by separating the motion into gross and fine segments. This allows them to utilize the convenience of a stereo camera from a distance, while relegating the precise positioning to a more robust method. However, the reliance on a template to

finalize the robot positioning limits the system to interacting only with those objects defined in its database.

Effindi, Jarvis, and Suter [39] developed a method for table mounted SCARA robot to identify and grasp previously defined objects using a stereo camera. The scene is observed by the robot mounted camera and a disparity map is generated. Based on the disparity map, the objects in the scene are segment from the background. The segmented objects are matched against images in a database using SIFT features. When a match is found, the shape of the object is approximated by calculating the bounding box that encompasses the 3D points that were generated from the disparity map. The object is then grasped using an overhead maneuver. While this work has a minimal reliance on previously defined models, it is relegated to approximating the shape of any object by a simple bounding box. Though a simple box model may prove sufficient for a certain class of objects, it will fail to capture the nature of many common household objects (such as cups and other rounded shapes) and will likely cause limitations for anything other than overhead grasping maneuvers.

Schlemmer, Biegelbauer, and Vincze [31] combine the shape and appearance of objects in a system designed for eventual use by assistive robots in a household setting. During the training phase, the system is shown an object that is later to be found in a scene, and using a laser range finder, the 3D range data is parametrized by a best fit superquadric. In the detection phase, the scene is scanned by the laser range finder, and the range data analysed to detect any shapes that were learned in the training phase. If a match is found, the object can be grasped by visual servoing using SIFT feature points provided by a monocular camera. Though the system was implemented only on a test bench, and the practicality of using a 3D laser scanner in a household setting is debatable, the work is fundamental in that it showed that the shape of most household objects can be effectively modeled by a simple and compact superquadric representation.

Kragic, Björkman, Christensen, and Eklundh [12] developed a system that uses multiple sets of stereo cameras to allow a manipulator to operate in an unstructured environment using a *detect-approach-grasp* paradigm. The stereo cameras are mounted in both eye-in-hand and stationary configurations and serve to segment the scene and isolate foreground objects for the detection phase. Various key point recognition schemes are employed using the monocular cues, and a database of predefined objects is searched for a match. If a match is found, 2.5D visual servoing is used for the approach and grasp phases. If a 3D CAD model is available for the object, then the full pose infor-

mation of the object is estimated and used to increase the grasping accuracy. The authors describe a scenario where their system would be capable of manipulating an unknown object provided that certain assumptions are made about the object: namely, that the depth of the object is not significant. That is, the authors recognize the inability of stereo camera to reconstruct the full 3D geometry of the object from a single position, and that in cases where the object is unknown, images from multiple vantage points would be required.

Liefhebber and Sijs [16] proposed a vision based control system for a popular, commercially available wheelchair mounted robotic arm. In their system, images gathered from a monocular camera are used to control the orientation of the robot end effector with respect to the object of interest. The user is responsible for controlling the position. SIFT features are extracted from the image and matched to an object in a database and a feedback visual servoing algorithm is employed to maintain a predefined orientation. In addition to the drawback of relying on a database of predefined objects, the use of a monocular camera in this case requires that the 3D model of each object in the database be defined in terms of the 3D location of the SIFT features relative to some object coordinate system. This introduces a rather lengthy and complicated training phase for any object that should be included in the database. Furthermore, SIFT features are notoriously computationally expensive to extract from a given image.

2.2 Runtime Reconstruction

In a series of works by Yamazaki et al. [27, 28, 26], the authors developed a mobile robot that is capable of reconstructing the shape and pose of a novel object using almost no *a priori* knowledge of the object. The reconstruction was achieved by having the robot drive completely around the object and capturing a continuous sequence of images as it travelled. These images were registered with each other using Structure From Motion (SFM) techniques and disparity maps were created. From the disparity maps, a dense 3D reconstruction was performed to yield the geometry of the object. Finally, a statistical analysis of the dense range data was performed to determine the ideal grasping pose. An advantage of the technique the authors employed is that the position of the robot need not be known to a high degree of accuracy. That is, odometry information from the robot is not used for the reconstruction, and the camera parameters are determined directly from the sequence of images. This eliminates many issues that can arise from robot inaccuracies. The downsides to this approach

however, are that it requires the robot to have 360 degree access around the periphery of the object and it necessitates an exceedingly large number of images which results in higher computational costs. Further, since the resulting geometry information is simply a collection of 3D points, any structural information must be extracted using statistical methods, introducing more computational requirements. A final drawback, which was exhibited in the later work [26], is that the location of the object needs to be identified to the system in advance.

Yoshikawa, Koeda, and Fujimoto [40] used an omnidirectional camera and a soft fingered gripper on a robotic manipulator to recognize and grasp unknown shapes. The authors take several images from around the periphery of the object and, using a simplified version of the volume intersection technique, calculate the 2D visual hull of the object on a horizontal plane. They also develop a simple grasp criterion to determine the optimal graping position based on the 2D visual hull. The authors chose to simplify the shape reconstruction to a two dimensional problem citing the real time computational consideration of reconstructing the full three dimensional geometry. This, of course, relegates the system to only being able to grasp objects that lie on a predefined horizontal plane. Furthermore, since the reconstruction is limited to a 2D plane, the system is unable to distinguish objects that have variation along the vertical direction.

Nguyen et. al. [20] developed a mobile manipulator, named "El-E", for the explicit purpose of retrieving "unmodeled, everyday objects" from flat surfaces for people with motor disabilities. The robot can retrieve objects that are identified by the user through the means of a laser pointer. An omnidirectional camera detects the laser pointer and in turn, points a stereo camera rig at the point. The 3D location of the point is determined from the stereo images and the robot then drives to a specified threshold distance from the object, using a 2D laser scanner on the bottom of the mobile base for navigation. Once at the object l, the 2D laser scanner is used, in combination with a e, to detect the height of the flat surface. The end effector of the robot is then oriented perpendicular to the flat surface and positioned above the location indicated by the user's laser pointer. Using a monocular eye-in-hand camera, the object is visually d and the end-effector oriented perpendicular to the axis of least inertia. Finally, the end-effector descends upon the object until infrared sensors in the gripper indicate the presence of an object or an imminent collision. The authors showed highly accurate results using this system with a reported overall task accuracy of 86%. It is worth noting that the shape of the object is never actually fully recovered with this system. It instead relies on the

infrared sensors to determine the presence of an object within the gripper. But given the reported accuracy, perhaps the only drawback of this system is the large variety of sensors required for its operation.

Saxena, Driemeyer, and Ng [3] approached the problem of grasping novel objects slightly differently. The system they developed does not require, "nor tries to build", a 3D model of the object. Instead, the system determines locations in a 2D image which likely represent a valid grasping location on the object. It then finds this same location in multiple views and triangulates the location of the grasping position. The system is "trained" to find valid grasping locations by being shown an initial large training set of rendered images of objects with the ideal grasping locations identified. Then, using various computer vision techniques, builds a classifier to later identify such appropriate areas on novel objects. The authors reported high accuracy, in terms of grasp success rate, for objects ranging from a stapler to an odd shaped power-horn. Even though the system requires being trained in advance, the images in the training set are completely unrelated to the novel objects the robot is later capable of grasping. Thus, it is considered that this system requires no *a priori* information of the novel objects it wishes to manipulate. However, training the classifier takes a considerable amount of time (though it is only required once) and the accuracy of the system was said to improve as the number and quality of the training images improve.

Chapter 3

Fundamental Background

3.1 Coordinate Representations

A cornerstone of the underlying math that enables 3D reconstruction from multiple 2D images is the representation and interpretation of points and lines in space and how they are transformed from one coordinate system to another. Consider the point $\mathbf{P}$ as depicted in Figure 1 in the space $\mathbb{R}^3$. In the Cartesian coordinate frame $\{A\}$, the point can be defined by the vector $\mathbf{P} = \mathbf{r_1} = (\mathrm{x_A}, \mathrm{y_A}, \mathrm{z_A})^{\mathrm{T}}$. Likewise, the same point can be represented in the coordinate frame $\{B\}$: $\mathbf{P} = \mathbf{r_2} = (\mathrm{x_B}, \mathrm{y_B}, \mathrm{z_B})^{\mathrm{T}}$. Clearly, both $\mathbf{r_1}$ and $\mathbf{r_2}$ describe the same point $\mathbf{P}$ in space, so there must be an operation or *mapping* that can be applied to the description of $\mathbf{P}$ in frame $\{A\}$ to yield $\mathbf{P}$ as described in frame $\{B\}$ and vice versa.

This section presents the necessary mathematical constructs that facilitate such mappings from one coordinate space to another. In the general case, these mappings are not limited to purely Euclidean systems but rather any two systems that are related by a general projective transformation. However, such general cases are not directly relevant to this work and are not presented. Rather, this section focuses on coordinate systems that differ by a similarity transformation. That is, a Euclidean transformation combined with an isotropic scaling. For a full treatment of projective geometry in the general case, the reader is directed to [34].

3.1.1 Homogeneous Coordinates

A homogeneous coordinate is an alternative representation of a point is space which is obtained by adding an additional element to the coordinate tuple. That is, a 2-element vector becomes 3-elements, a 3-element vector becomes 4-elements, and so on. Any two n-element vectors are said to belong to the same equivalence class, provided they differ by a common multiple [34]. That is, if $(a_1, a_2, ..., a_n) = k(b_1, b_2, ..., b_n)$ then both $\mathbf{a}$ and $\mathbf{b}$ are homogeneous representations of the vector

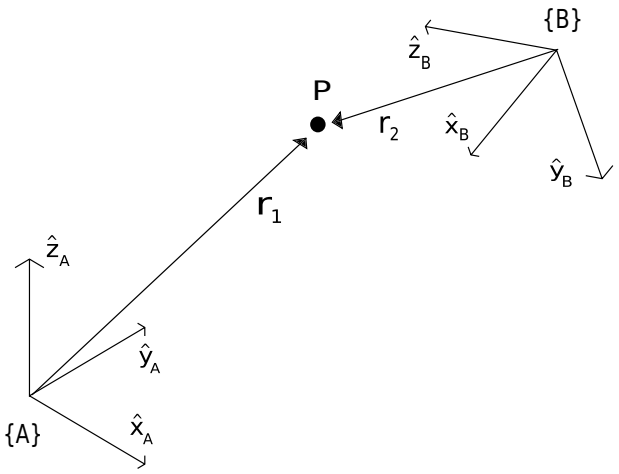

Figure 1.: A point in $\mathbb{R}^3$. The point $\mathbf{P}$ can be defined in two separate coordinate frames $\{A\}$ and $\{B\}$.

$\mathbf{c} = (c_1, c_2, ..., c_{n-1}, 1) = (\frac{a_1}{a_n}, \frac{a_2}{a_n}, ..., \frac{a_n}{a_n}) = (\frac{b_1}{b_n}, \frac{b_2}{b_n}, ..., \frac{b_n}{b_n})$. This representation is extremely useful when mapping coordinates from one system to another and is used extensively in this work. Thus the derivation and proof of the homogeneous representation for two and three dimensions is given in the following sections.

3.1.1.1 Two Dimensions

In a 2-D plane in the space $\mathbb{R}^2$, a point $\mathbf{P}$ can be represented as a two element vector $\mathbf{P} = (\mathrm{x,y})^{\mathrm{T}}$. Furthermore, the implicit equation of a line on that plane is given by

$$ax + by + c = 0 \tag{3.1}$$

where any point $\mathbf{P}$ that satisfies the equation lies on the line. Thus, we can represent a line $\mathbf{L}$ on the two dimensional plane as a three element vector $\mathbf{L} = (\mathrm{a, b, c})^{\mathrm{T}}$. Now, we can rewrite Equation 3.1 as an inner product of two vectors $(x, y, 1)(a, b, c)^T = 0$ and thus $(x, y, 1)\mathbf{L} = 0$. But, since the point $\mathbf{P}$ lies on the line $\mathbf{L}$ if and only if it satisfies Equation 3.1, then $(x, y, 1)^T$ must also be an equivalent and valid representation for the point $\mathbf{P}$.

If a non-zero constant k is introduced into Equation 3.1, then we see that the equivalent equation $k(x, y, 1)\mathbf{L} = 0$ will still hold provided that point $\mathbf{P}$ lies on the line $\mathbf{L}$. Thus in general, the three

element vector $(kx, ky, k)^T$ is a valid representation for the point $\mathbf{P} = (x, y)^T$ for any non-zero value of k. This three-element form is the homogeneous representation of the 2-D point, and is a many-to-one representation. Given a homogeneous representation of a 2-D point $\mathbf{X} = (x, y, w)^T$, one determines the coordinates of that point in $\mathbb{R}^2$ as $\mathbf{P} = (\frac{x}{w}, \frac{y}{w})^T$. Conversely, given a point $\mathbf{P} = (x, y)^T$ in $\mathbb{R}^2$, one easily constructs a homogeneous representation as $\mathbf{X} = (x, y, 1)^T$.

3.1.1.2 Three Dimensions

Homogeneous coordinates in three dimensions are developed in much the same fashion as two dimensions. In the space $\mathbb{R}^3$, a two dimensional plane is defined by the implicit equation

$$\pi_1 x + \pi_2 y + \pi_3 z + \pi_4 = 0 \tag{3.2}$$

and thus a plane can be represented as a four-element vector $\pi = (\pi_1, \pi_2, \pi_3, \pi_4)^T$, and any point $\mathbf{P} = (x, y, z)^T$ that satisfies Equation 3.2 lies on the plane π. Like the 2-D case, we can represent Equation 3.2 as the inner product of two vectors $(x, y, z, 1)(\pi_1, \pi_2, \pi_3, \pi_4)^T = (x, y, z, 1)\pi = 0$. It is readily seen that $(x, y, z, 1)^T$ must also be a valid representation for $\mathbf{P}$. Introducing a non-zero constant k into Equation 3.2 yields the equivalent equation $k(x, y, z, 1)\pi = 0$, which holds provided the point lies on the plane. Thus in general, $(kx, ky, kz, k)^T$ is a valid representation of the point $\mathbf{P}$ and is the homogeneous representation of a point in $\mathbb{R}^3$.

Given the homogeneous coordinates of a point $\mathbf{X} = (x, y, z, w)^T$, one determines the coordinates of the point in $\mathbb{R}^3$ as $\mathbf{P} = (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^T$. Given a point $\mathbf{P} = (x, y, z)^T$ in $\mathbb{R}^3$, a homogeneous representation can be created as $\mathbf{X} = (x, y, z, 1)^T$.

3.1.2 Coordinate Transformations

A note on conventions: In what follows, we adopt the notation convention presented in [24] to represent the various entities involved in coordinate transformation representation. Briefly:

- Braces $\{\}$ indicated a Cartesian frame of frame of reference. $\{A\}$ represents Cartesian frame (see Figure 1).
- A left superscript indicates the frame in which the quantity is defined. ${}^A\mathbf{r}$ is the vector $\mathbf{r}$ defined in frame $\{A\}$.

- The hat ^ indicates a unit vector. $\hat{x}$ is a unit vector in the x direction.
- A right subscript indicates that the entity belongs to a certain frame. $\hat{x}_A$ is the unit vector in the x direction of the frame $\{A\}$.
- A left subscript and superscript indicates the quantity is defined relative between two frames. ${}^{A}_{B}\mathbf{R}$ is the quantity $\mathbf{R}$ and is described in frame $\{B\}$ relative to frame $\{A\}$.
- Vectors, with the exception of unit vectors, are bold faced and considered as column vectors. Thus, the following are all equivalent

$$\mathbf{P} = (P_1, P_2, P_3)^T = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

- Matrices are represented as bold faced.

Given a point $\mathbf{P}$ in space $\mathbb{R}^3$ defined in some coordinate system $\{B\}$, we wish to be able to express this point in terms of any other arbitrary coordinate system $\{A\}$ (see Figure 1). For now, we assume that coordinate frames $\{A\}$ and $\{B\}$ are related by a Euclidean transformation $\mathbf{T}$. That is, they are related by an arbitrary translation and rotation, but the scale remains constant in all directions. Then, the point $\mathbf{P}$ expressed in the coordinate frame $\{B\}$ can be defined as ${}^{B}\mathbf{P} = {}^{B}_{A}\mathbf{T}{}^{\mathrm{A}}\mathbf{P}$. In order to find a valid transformation for ${}^{\mathrm{B}}_{\mathrm{A}}\mathbf{T}$, we break down the transformation into its individual translation and rotation components and then show how the two components can be combined into the single transformation.

Figure 2 shows two coordinate frames $\{A\}$ and $\{B\}$ that are separated by a pure translation. If we imagine that both coordinate frames exist in the same encompassing global frame, then the origins of each coordinate can be represented by the vectors $\mathbf{P}_{A_{ORG}}$ and $\mathbf{P}_{B_{ORG}}$ and the translation of $\{B\}$ with respect to $\{A\}$ is the vector ${}^{A}_{B}\mathbf{t} = \mathbf{P}_{\mathrm{B_{ORG}}} - \mathbf{P}_{A_{ORG}}$. The point ${}^{B}\mathbf{p}$ is the point $\mathbf{p}$ expressed in the coordinates of frame $\{B\}$, and from vector addition we see that ${}^{A}\mathbf{p} = \mathbf{P}_{B_{ORG}} - \mathbf{P}_{A_{ORG}} + {}^{B}\mathbf{p}$. So, we can represent the translation portion of the Euclidean transformation as

$${}^{\mathrm{A}}\mathbf{p} = {}^{\mathrm{A}}_{\mathrm{B}}\mathbf{t} + {}^{\mathrm{B}}\mathbf{p} \tag{3.3}$$

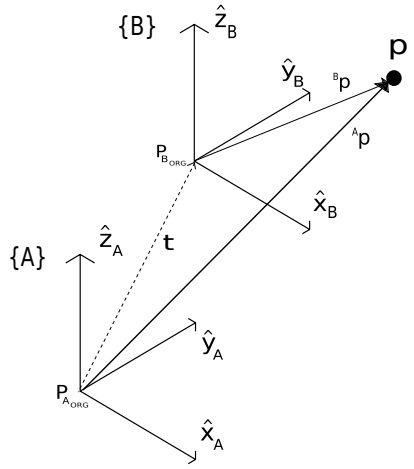

Figure 2.: Two coordinate frames separated by pure translation.

Now consider the case where the origin of two coordinate frames are coincident, but there exists an arbitrary rotation about some axis between the frames. Figure 3 shows this scenario for the two coordinate frames $\{A\}$ and $\{B\}$. Now, the projection of one vector onto another vector is accomplished through dot product of the two vectors. So we can represent the unit vectors of the frame $\{B\}$ expressed in terms of the unit vectors of frame $\{A\}$ according to the equation

$$\begin{bmatrix} {}^A\hat{x}_B & {}^A\hat{y}_B & {}^A\hat{z}_B \end{bmatrix} = \begin{bmatrix} \hat{x}_B \cdot \hat{x}_A & \hat{y}_B \cdot \hat{x}_A & \hat{z}_B \cdot \hat{x}_A \\ \hat{x}_B \cdot \hat{y}_A & \hat{y}_B \cdot \hat{y}_A & \hat{z}_B \cdot \hat{y}_A \\ \hat{x}_B \cdot \hat{z}_A & \hat{y}_B \cdot \hat{z}_A & \hat{z}_B \cdot \hat{z}_A \end{bmatrix} = \begin{bmatrix} {}^B\hat{x}_A^T \\ {}^B\hat{y}_A^T \\ {}^B\hat{z}_A^T \end{bmatrix} \tag{3.4}$$

Now, if given a point ${}^B\mathbf{p}$ and one wants to find the representation of the point ${}^A\mathbf{p}$, we just project the point ${}^B\mathbf{p}$ using a dot product in the same manner as Equation 3.4,

$$\begin{aligned} {}^Ap_x &= {}^B\hat{x}_A \cdot {}^B\mathbf{p} \\ {}^Ap_y &= {}^B\hat{y}_A \cdot {}^B\mathbf{p} \\ {}^Ap_z &= {}^B\hat{z}_A \cdot {}^B\mathbf{p} \end{aligned} \tag{3.5}$$

If we let the 3x3 matrix be termed a rotation matrix $\mathbf{R}$, then it can be seen that transforming ${}^B\mathbf{p}$ into ${}^A\mathbf{p}$ can be easily accomplished with the matrix product

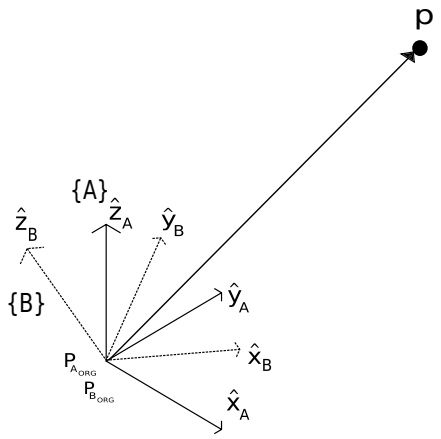

Figure 3.: Two coordinate frames separated by pure rotation.

$$^{A}\mathbf{p} = {}^{A}_{B}\mathbf{R}\,{}^{B}\mathbf{p} \tag{3.6}$$

where ${}^{A}_{B}\mathbf{R}$ is defined using Equation 3.4

$${}^{A}_{B}\mathbf{R} = \begin{bmatrix} {}^{A}\hat{x}_B & {}^{A}\hat{y}_B & {}^{A}\hat{z}_B \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{3.7}$$

and thus Equation 3.6 represents the transformation equation for two frames separated by a pure rotation.

We now consider the general case where two frames $\{A\}$ and $\{B\}$ are separated by both a general translation and orientation. Figure 4 illustrates the concept. Since the two components of the transformation are separable, they can be combined through the principle of superposition. Think of this as creating a new temporary coordinate frame with the origin at $\mathbf{P}_{B_{ORG}}$ but with an orientation that is aligned with $\{A\}$. Now, in order to transform ${}^{B}\mathbf{p}$ into ${}^{A}\mathbf{p}$, we first rotate ${}^{B}\mathbf{p}$ into this new temporary frame. But since the temporary frame has the same orientation as $\{A\}$ this can be accomplished with the rotation matrix ${}^{A}_{B}\mathbf{R}$. The last step is to translate the result into $\{A\}$ via the translation vector ${}^{A}_{B}\mathbf{t}$ and so the full equation becomes

$$^{A}\mathbf{p} = {}^{A}_{B}\mathbf{R}^{\mathrm{B}}\mathbf{p} + {}^{A}_{B}\mathbf{t} \tag{3.8}$$

We wish to express Equation 3.8 in the form ${}^{A}\mathbf{p}={}^{\mathrm{A}}_{\mathrm{B}}\mathbf{T}{}^{\mathrm{B}}\mathbf{p}$ and this can accomplished by rearranging the equation as a matrix product as

$$ {}^{A}\mathbf{p} = \left[\begin{array}{c|c} {}^{A}_{B}\mathbf{R} & {}^{A}_{B}\mathbf{t} \\ \hline 0\,0\,0 & 1 \end{array}\right] {}^{\mathrm{B}}\mathbf{p} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{3.9} $$

where

$$ {}^{A}_{B}\mathbf{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10} $$

In the course of rearranging Equation 3.8 into Equation 3.9, we have converted the points ${}^{A}\mathbf{p}$ and ${}^{B}\mathbf{p}$ into their homogeneous representation. By using this representation, we are able to represent the entire transformation (rotation and translation) as a single matrix multiplication. For this reason, ${}^{A}_{B}\mathbf{T}$ is called a *homogeneous transformation matrix*. Further, by defining the fourth row of ${}^{A}_{B}\mathbf{T}$ as we have, ${}^{A}_{B}\mathbf{T}$ is now a 4x4 matrix which can be inverted in a simple manner. The inverse of a homogeneous transformation matrix can be viewed as an inverse mapping. That is,

$$ {}^{B}\mathbf{p} = {}^{B}_{A}\mathbf{T}{}^{\mathrm{A}}\mathbf{p} = \left({}^{A}_{B}\mathbf{T}\right)^{-1}{}^{\mathrm{A}}\mathbf{p} \tag{3.11} $$

3.1.3 3D to 2D Projection

In order to more clearly represent 3D to 2D projection, we start with the simpler case of 2D to 1D projection. Figure 5 shows a 2D plane with origin $(0, 0)$ and point (x, y) on that plane. Further, there is line $\mathbf{L}$ in the plane located a distance d from the origin and oriented parallel to y axis. We wish to find the projection of the point (x, y) onto the line $\mathbf{L}$ which we indicated as point $\mathbf{P}$. From the figure, it can be seen that point $\mathbf{P}$ is equal to x' and can be calculated via similar triangles

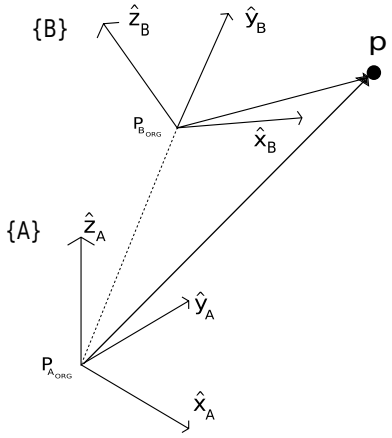

Figure 4.: Two coordinate frames separated by a general translation and rotation.

$$\begin{aligned} \frac{y}{x} &= \frac{d}{x'} \\ x' &= \frac{dx}{y} \end{aligned} \tag{3.12}$$

Now, we can express Equation 3.12 in the homogeneous form by converting point $\mathbf{P}$ into a general homogeneous representation and allowing the right hand side of the equation to become a matrix multiplication.

$$\mathbf{P} = \begin{bmatrix} kp \\ k \end{bmatrix} = \begin{bmatrix} d & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.13}$$

Recall from Section 3.1.1 that in order to retrieve the canonical representation of a point in a space $\mathbb{R}^n$, simply divide through by the last element in the vector. Thus we have

$$\begin{aligned} \mathbf{P} &= \begin{bmatrix} kp \\ k \end{bmatrix} = \begin{bmatrix} dx \\ y \end{bmatrix} \\ \mathbf{P} &= \begin{bmatrix} p \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{dx}{y} \\ 1 \end{bmatrix} \\ \mathbf{P} &= \frac{dx}{y} = x' \end{aligned} \tag{3.14}$$

which is in agreement with Equation 3.12.

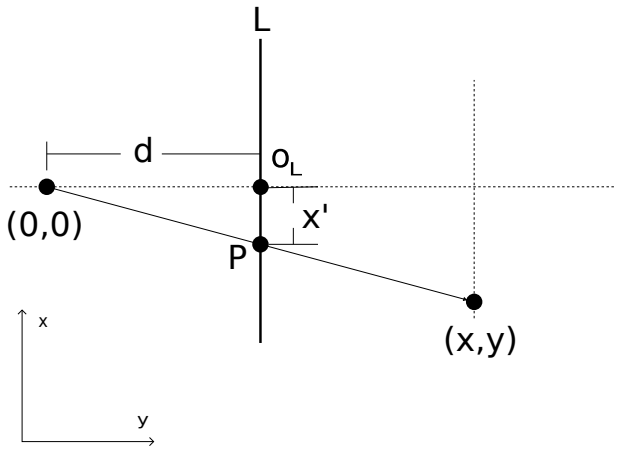

Figure 5.: 2D to 1D projection.

The case of 3D to 2D projection is a natural extension of the 2D to 1D case just discussed and the result is now presented without derivation. Given a point $(x, y, z)^T$ in the space $\mathbb{R}^3$ and a 2D plane located a distance d from the origin as measured along the z axis, then, provided that the plane is perpendicular to the z axis and the origin of the plane is the point $(0, 0, d)^T$ the projection of $(x, y, z)^T$ into the plane at point $\mathbf{P}$ is given as

$$\mathbf{P} = \begin{bmatrix} wP_x \\ wP_y \\ w \end{bmatrix} = \begin{bmatrix} d & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.15}$$

Equation 3.15 is the homogeneous projection of a 3D coordinate onto a 2D plane. If we multiply the point $(x, y, z)^T$ by a non-zero constant k, we see that the equation will still be valid as the constant k will be absorbed into w. Thus, Equation 3.15 is a many-to-one mapping in that any point lying along the line defined by the origin and the point $(x, y, z)^T$ will project to the same point on the plane.

3.1.4 2D to 3D Projection

As it was shown that a 3D to 2D projection is a many-to-one mapping, there is no general way (from a single 2D projection) to recover the particular 3D point that caused the projection. In fact, the number of possible points that can generate a unique 2D projection is infinite. However, there is

something all of these infinite points have in common: they all lie along the same line. This can be shown by inverting equation 3.15 as follows

$$\begin{bmatrix} \frac{1}{d} & 0 & 0 \\ 0 & \frac{1}{d} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} wP_x \\ wP_y \\ w \end{bmatrix} = \begin{bmatrix} \frac{1}{d} & 0 & 0 \\ 0 & \frac{1}{d} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$
$$\begin{bmatrix} \frac{wP_x}{d} \\ \frac{wP_y}{d} \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.16}$$

but since w can take on any value and still be a valid homogeneous representation of $\mathbf{P}$, the left hand side of the equation does not represent the original point $(x, y, z)^T$, but rather a vector from the origin that passes through that point , i.e. a line in the space $\mathbb{R}^3$. Hence we need to augment the right hand side of Equation 3.16 with a non-zero constant k. The appropriate form is

$$\begin{bmatrix} \frac{wP_x}{d} \\ \frac{wP_y}{d} \\ w \end{bmatrix} = k \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.17}$$

Thus, given the location of the plane in space, and a point on that plane, we are able to determine the line that passes through that point in the plane and the origin of the space, an operation called backprojection. Backprojection is a fundamental tool for the reconstruction of 3D objects from multiple 2D images and is used extensively in Section 4.3.

3.2 Machine Vision

According to [36], the goal of machine vision is to extract useful information about a scene based on its two dimensional projections. An image is a 2D projection of a scene, and as such does not contain any directly available three dimensional information of the scene. The field of machine vision is concerned with the methods and algorithms that enable the recovery of information from 2D projections. This information can be combined with the theory of projective geometry (covered in the previous sections) to enable the 3D reconstruction of scenes.

Broadly, the entire goal of this thesis is to extract three dimensional information about objects in a scene from multiple 2D images of that scene. In that respect, the entire work falls under the guise of machine vision. The aim of this section however, is not to provide an exhaustive background on machine vision. Rather, it presents only a few of the key concepts and methods from machine vision that are used as part of the overall algorithm. These concepts are described in a broad fashion with pictorial examples; no algorithms for their implementation are given. The interested reader is directed to one of many reference texts which derive the algorithms in detail: [36, 30, 35]. All of the machine vision algorithms described in the following sections are freely available in many Open Source computer vision software libraries such as the OpenCV library [18].

3.2.1 Camera Models

The first step in machine vision is understanding the mathematics by which a camera converts a three dimensional scene into a two dimensional image. Though there are a wide variety of camera models, this work focuses on the models which are related to common CCD type digital cameras. The first camera model described is the ideal camera model. The second model is an extension of the first and allows for the non-ideal distortions and other effects that arise with CCD cameras.

3.2.1.1 Ideal Camera Model

The ideal camera model is illustrated in Figure 6. In this figure, the center of projection or *focal point* of the camera is represented by the point O. The coordinate system of the camera is oriented so that the camera is looking down the Z axis; an axis also referred to as the *principle camera ray*. The principle camera ray is perpendicular to and intersects the imaging plane at a point O'. The imaging plane is located a distance f from the focal point; a distance termed the *focal length* of the camera.

An image of a point $\mathbf{P}$ in space is imaged to the point $\mathbf{P}'$ at the location of intersection of image plane at the line connecting points $\mathbf{P}$ and $\mathbf{O}$. This position can be calculated analytically through the use of similar triangles. By inspection of the figure it is readily seen that

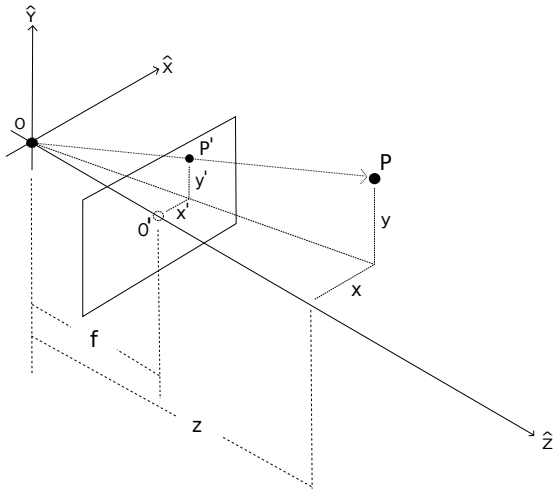

Figure 6.: The ideal camera model. The position of the point p' in the image plane can be determined from the position of the point p and the focal length of the camera f by means of similar triangles.

$$\mathbf{P'} = \left(x', y'\right)^T \tag{3.18}$$

$$x' = f\left(\frac{x}{z}\right) \tag{3.19}$$

$$y' = f\left(\frac{y}{z}\right) \tag{3.20}$$

or using homogeneous matrix notation

$$\mathbf{P'} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{MP} \tag{3.21}$$

The 3x3 matrix $\mathbf{M}$ is the camera *intrinsics matrix* and for this ideal camera is defined by the single parameter f. For this type of camera, all objects in a scene project onto the image plane in focus and the size of the imaged object is relative only to its distance from the camera and camera's focal length. The astute reader will notice that Equation 3.21 is equivalent to Equation 3.15.

3.2.1.2 CCD Camera Model

When considering images that are formed by the action of a CCD camera, the ideal camera model of the previous section must be extended to account for some specific deviations and distortions.

The first deviation from the ideal model comes from the representation of the origin of the image plane. In the ideal model, the origin of image plane is the intersection of the principle camera ray with the image plane, but in a CCD image the origin is taken as the upper left corner of the imaging sensor. Thus, the model must account for an offset in the x and y direction. This is accomplished with the constants c_x and c_y which become part of the camera intrinsics matrix

$$\mathbf{M} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

The second deviation of a CCD camera from the ideal model is that of different focal lengths in the x and y directions. This condition can arise, for example, with cheap or poor quality CCD sensors where the pixel elements are rectangular rather than square. Allowing for this modification, the final form of the camera intrinsics matrix for a typical CCD camera becomes

$$\mathbf{M} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.23}$$

where the units of f_x, f_y are pixels per unit distance and the units of c_x, c_y are pixels.

While the intrinsics matrix of Equation 3.23 takes into account non-square pixels and the offset of the image, it is assuming the lens of the camera is perfect. In reality, most lenses are not perfect and introduce some amount of distortion. The primary type of distortion is called radial or barrel distortion, and the effect can be seen in Figure 7. From the figure, it is clear that lines that should be straight are warped into curves. The amount of warping increases with distance from the image center. The other main type of distortion that can be introduced is called tangential distortion and arises when the camera's imaging sensor is not perfectly parallel to the camera lens [17]. Though there are many other forms of distortion possible in a camera, the radial and tangential are the most common and have the most significant effect [17, 34].

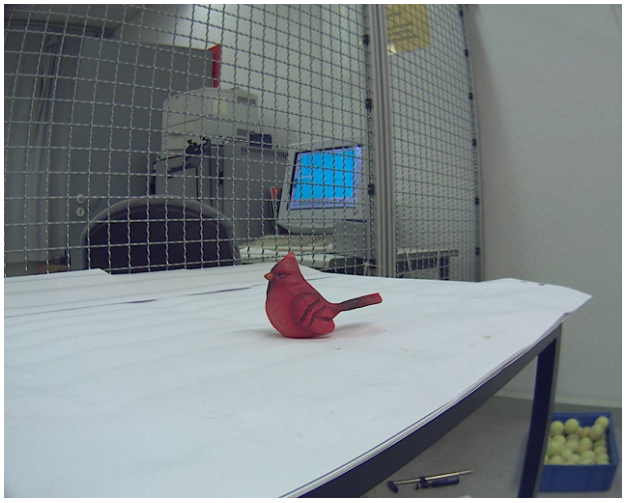

Figure 7.: An example of radial distortion. Notice how lines that should be straight become more distorted the farther away they are from the image center.

It is beyond the scope of this work to derive the parameters for the radial and tangential distortion, but they are presented here, without derivation, as given in [17]. Let the point $(x_d, y_d)^T$ be the image location of the point $(x_w, y_w, z_w)^T$ after projection by a camera with radial and tangential distortion. Then, the undistorted position of the point $(x_p, y_p)^T$ is given as

$$x_p = \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) x_d + \left(2 p_1 x_d y_d + p_2 (r^2 + 2 x_d^2)\right) \tag{3.24}$$

$$y_p = \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) y_d + \left(2 p_2 x_d y_d + p_1 (r^2 + 2 y_d^2)\right) \tag{3.25}$$

The parameters k_1, k_2, k_3 represent the radial distortion and the parameters p_1, p_2 represent the tangential distortion. The value r is the distance of the point $(x_d, y_d)^T$ from the image center. The point $(x_p, y_p)^T$ is the undistorted image point as if the point $(x_w, y_w, z_w)^T$ were projected by a distortion-free camera using the intrinsics matrix $\mathbf{M}$ of Equation 3.23.

3.2.2 Digital Image Representation

Before discussing any of the algorithms of machine vision, it is critical to understand how an image is represented in a computer. When humans *see* an image, they immediately recognize features such as: color, shapes, foreground, background, faces, etc. They do not see a piece of fibrous paper with multicolored ink tediously and precisely applied, nor do they see an extremely dense array of

liquid crystal elements that display only one of three colors. For humans, the process of converting this raw sensory data into what is perceived as an image is so effortless, we often overlook the enormous complexity of the visual process.

Machines have a much more difficult time recognizing features in an image. To a machine, an image (like all other machine data) is purely a sequence of numbers. At the lowest level, the sequence is composed of purely 1's and 0's, otherwise known as binary data. At one level of abstraction higher, an image can be represented by sequence of integer values that represent the intensity of the image at a certain locations. Other representations that are not integer valued are also used to represent intensity values, however, the integer representation is the most commonly used and unless otherwise noted, is what is used throughout this work.

Since the machine representation of an image is a sequence of discrete numbers, or *digits*, the machine representation is often termed a *digital image*. Consider the small representative digital image shown in Figure 8. This image has seven rows and seven columns for a total of 49 pixels. It is also seen that the image is composed of five different intensity levels ranging from black to white. In a typical digital image, each pixel is stored in the computer's memory as an unsigned integer represented by 8 bits. The minimum value this integer can have is 0 (binary code 00000000) and the maximum is 255 (binary code 11111111) for a total of 256 possible intensity levels. In general, a value of 0 corresponds to pure black and a value of 255 corresponds to pure white; other values are various shades of gray. In the case of Figure 8, the array of intensity levels may look something like

$$Image = \begin{bmatrix} 255 & 255 & 200 & 200 & 200 & 255 & 255 \\ 255 & 200 & 150 & 150 & 150 & 200 & 255 \\ 200 & 150 & 80 & 80 & 80 & 150 & 200 \\ 200 & 150 & 80 & 0 & 80 & 150 & 200 \\ 200 & 150 & 80 & 80 & 80 & 150 & 200 \\ 255 & 200 & 150 & 150 & 150 & 200 & 255 \\ 255 & 255 & 200 & 200 & 200 & 255 & 255 \end{bmatrix} \tag{3.26}$$

The discrete nature of the digital image representation presents some immediate shortcomings. In particular, two quantities suffer losses during the discretization process: intensity and shape. As previously mentioned, the typical 8-bit unsigned integer representation of a pixel is only capable of

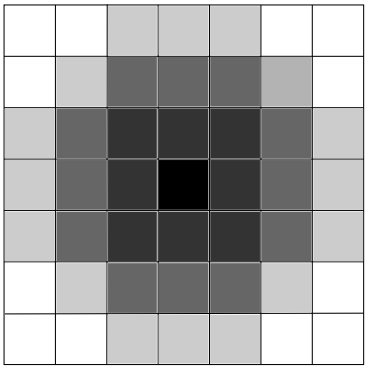

Figure 8.: An representative example of a digital image.

representing 256 discrete levels of intensity. Therefore, if the difference in light intensity detected from an object in scene by neighboring pixels is below some threshold, then the two pixels will be digitized to the same value, even though there may have been an intensity difference present. In the case of shape representation, the square nature of pixels renders the digital image incapable of perfectly representing a curved shape. This is easily seen in Figure 8, which represents the best possible approximation of a circle within a 7x7 grid. With such a coarse resolution, it is clearly not an accurate representation. This limitation can be somewhat overcome by increasing the resolution of the digital image. A 1280x1280 pixel image, for example, will be better able to approximate a circle than a 7x7 image. In practice, such high resolution images are often used in machine vision applications.

Digital images are not limited to representing only grayscale values. Color image representation is achieved in many different ways, the most popular of which may be the RGB color model. RGB stands for Red, Green, and Blue and indicates that a color image represented with this model is composed of varying amounts of the colors red, green, and blue. Typically, each pixel in an RGB image is represented by 24 bits. The first 8 bits are the amount of red in the pixel, the next 8 bits the amount of green, and the next 8 bits the amount of blue. Conceptually, one can think of a color RGB image as three grayscale images stacked one on top of the other, with a special interpretation assigned to each layer. The RGB model for color representation is one of the most popular representation for digital images because this is the native format used by most display

Figure 9.: A color RGB image. Left to right: the full color image, the red plane, the green plane, the blue plane. Full image from: http://beautifulwork.files.wordpress.com/2009/06/crayons.jpg

devices. That is, an LCD/LED screen is made up of many pixels, each of which has a red, green, and blue component. The intensity at which each of these components is operated determines the perceived color of the pixel. Figure 9 shows a color 24-bit RGB image and each of the three color planes that comprise it.

Throughout this work, grayscale images are represented as 2D arrays of 8-bit integers and color images are represented as 3D arrays of 8-bit integers where the 3rd dimension indicates the color plane i.e. R, G, and B respectively. With the basic overview of how an image is represented in the computer, we are now in position to discuss some of the important machine vision algorithms from a high level perspective.

3.2.3 Neighborhoods

We are seldom interested in the value of a single pixel in an image in and of itself. Indeed, such a singular quantity yields little information of value. Instead, we are concerned with the values of the pixels contained within a defined area. It is only when analysing the values of pixels across an area that any useful information can be obtained. In machine vision, this area has a special name called a *neighborhood*. Neighborhoods can be any shape or size and are used to specify the area of interest surrounding a specified anchor pixel. In many algorithms the pixels defined by a neighborhood are used to compute a value that is representative of some feature: averages, differences, etc. And more often than not, this value is calculated at every pixel in the image. That is, the neighborhood is slid across the image allowing each pixel to become the anchor pixel. The result of this operation is a new image where each pixel in this image is the result of the neighborhood operation about the

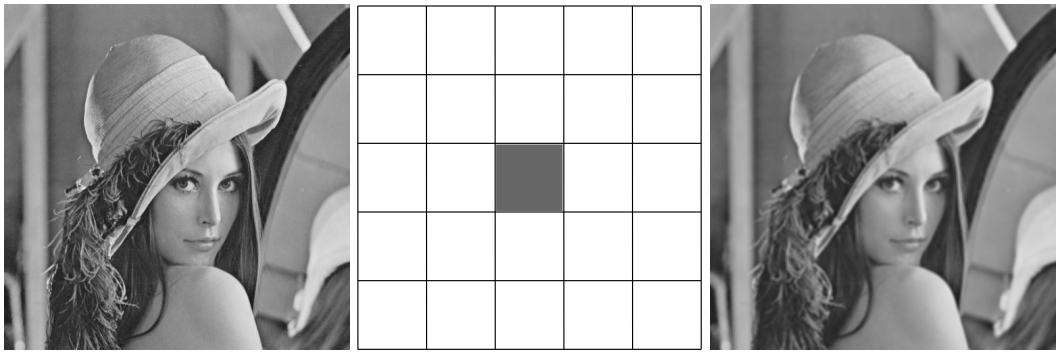

Figure 10.: A smoothing operation using a neighborhood. From left to right: the original sharp image, the neighborhood mask defined for this operation with the anchor pixel darkened, the result of the averaging operation over the defined neighborhood.

respective pixel in the original image.

Consider a simple image smoothing operation. A initial image is given, and we wish to reduce the sharpness of the image through some form of smoothing operation. I naive implementation consists of using a 5x5 neighborhood and averaging all the values in the neighborhood at each anchor point. Figure 10 gives an example of this operation. It shows the original image, the 5x5 neighborhood with the anchor pixel shown darkened, and the result of using that neighborhood along with an averaging operation. In short, the neighborhood was slid across the image to every pixel, for each pixel in the original image that was aligned with the anchor pixel, the corresponding pixel in the results image was set to the average value of all pixels contained in the neighborhood. The effect is to produce a smoothed image.

Perhaps the two most popular neighborhoods in machine vision algorithms are the 4-neighborhood and 8-neighborhood, which as their name implies, contain the 4 or 8 closest pixels to the anchor pixel. The 4-neighborhood and 8-neighborhood are shown in Figure 11. In general, a neighborhood can be any size. For example, a 24-neighborhood would contain the 24 nearest pixels to the anchor pixel. This is exactly the neighborhood used in Figure 10.

3.2.4 Connected Components

In a binary image, it is often necessary to identify independent regions in an image. This process is called *connected component labeling* [30, 36] and its uses include counting the number of items in

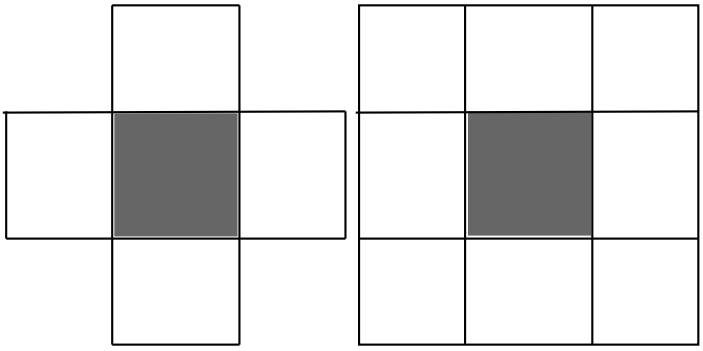

Figure 11.: A 4-neighborhood and 8-neighborhood.

an image and finding holes within an item. An algorithm for the implementation of such a procedure is beyond the scope of this text and the interested reader is referred to nearly any text on Machine Vision. In short, a connected components algorithm scans the image and assigns a unique number to each fully separated region. Every foreground pixel that is an 8-neighbor with another pixel will have the same number as that pixel. This process is explained pictorially in Figure 12. In the figure, image (a) is a binary image showing three separate regions. Image (b) is the labeled image which shows the unique number assigned to each pixel of the separate component. Both images have had their range expanded to show detail.

After labeling the connected components in an image, counting the number of items is as simple as finding the value of the largest label. One can find holes in an item by first inverting the image and then applying the connected components algorithm. If there are holes in any items, they will show up as separate components and typically have much smaller area than the largest component which is the actual background.

3.2.5 Edge Detection

Edges are one set of features in an image that "are relevant to estimating the structure and properties of an object in a scene" [36]. Edges are characterized as significant local intensity changes in an image and often represent the boundary between objects or regions. If we consider the left image in Figure 10, we see that the scene is composed of many different objects: a woman, a hat, a mirror, etc. Further, each one of these components can be thought as being made of up sub-regions: the woman has a face and arm, the hat has decorations, and so on. Many of these independent

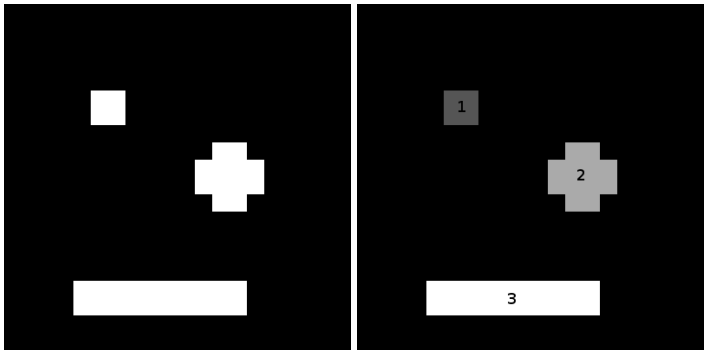

Figure 12.: Connected component labeling. Left: a binary image consisting of three separate components. Right: the labeled image which identifies a unique number for each component.

regions have an intensity level different than their neighboring regions. Therefore, an operation that can detect these changes in intensity can detect the boundaries of the regions. This is exactly the function of an edge detector.

The simplest edge detector can simply identify pixels in an image whose neighborhood pixels differ by a certain threshold value. While this approach works, it will often lead to many spurious or unimportant edges. In practice, edge detection algorithms usually employ a variety of criteria when determining whether or not a pixel belongs to an edge. Some of these criteria might be edge strength, zero crossing of the second derivative, edge value of neighboring pixels, and so on. A practical edge detection algorithm that has seen widespread adoption is the Canny edge detector. It uses a variety of operations above and beyond basic intensity differences, but yields strikingly good results. The details of the implementation are left to one of the previously cited computer vision or digital image processing texts, or even the original Canny paper [23]. The results of Canny edge detection applied to the left image in Figure 10 is shown in Figure 13.

3.2.6 Flood Fill

A flood fill algorithm is, conceptually, the opposite of an edge detector. Instead of representing regions in an image by their boundaries (identified by local intensity changes), regions are represented by local intensity uniformity (or near uniformity). The algorithm is fed a seed point in the image then checks the neighbors of that point, if the neighbors do not differ by a certain threshold, then they are assumed to be equivalent and assigned the same value. The algorithm then shifts

Figure 13.: Canny edge detection. The results of applying the Canny edge detector to the left image in Figure 10. Shown inverted for clarity.

Figure 14.: An example of a flood fill operation. Left: original image. Right: a flooded region.

over to the new neighbor and repeats the process. The process continues until there are no more neighboring pixels that meet criteria. The result is a single continuous area of uniform value that represents some region in the image. The paint bucket tool in many image manipulation programs is an example of a flood fill algorithm. Figure 14 gives an example of a flood fill algorithm. A point on the yellow peppers was given as a seed point along with an appropriate threshold, the resulting filled area is indicated in blue in the second image. Notice that even with an advanced flood fill algorithm, various holes and imperfections remain in the filled area. In practice, several machine vision algorithms are combined in a sequence which yields the desired results for a particular application.

Figure 15.: A binary image example. The binary representation of the flooded region in Figure 14(b). Dynamic range expanded to show detail.

3.2.7 Binary Image Features

The ultimate goal of identifying independent regions in an image is so that the regions can later be processed for characteristic information. Often times, this information is in the form of location, shape, and orientation information. Such features can be calculated directly from what is called the binary image representation. That is, once the region has been identified through some process (edge detection, flood fill, etc.) then it is separated from the rest of the image and every pixel in the region is given a value of 1 and all other pixels in the image are given a value of zero. If we take Figure 14(b) as an example and consider the flooded blue region as the target, then the binary image of that region is shown in Figure 15, which has had the dynamic range expanded to show detail.

Once the binary image of a region has been created, calculating the region's area, centroid, and orientation are relatively trivial matters. If we let a binary image of size $n \times m$ be denoted by $B[i, j]$, where n is the number of rows and m the number of columns, then the area A and the $(\bar{x}, \bar{y})$ location of the centroid can be given as

$$A = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} B[i, j] \tag{3.27}$$

$$\bar{x} = \frac{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} jB[i, j]}{A}$$

$$\bar{y} = \frac{-\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} iB[i, j]}{A} \tag{3.28}$$

Note that the above equations assume the origin of the image to be in the upper left corner and that the positive y direction is up. The orientation of a binary image is more involved, and the derivation

is not presented here. Rather, the result from [36] is presented and the interested reader should consult a machine vision text for the derivation. The orientation θ of a binary image is given by

$$\begin{aligned} x' &= x - \bar{x} \\ y' &= y - \bar{y} \\ a &= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(x'_{ij})^2 B[i,j] \\ b &= 2\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} x'_{ij}y'_{ij} B[i,j] \\ c &= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(y'_{ij})^2 B[i,j] \\ \tan(2\theta) &= \frac{b}{a-c} \end{aligned} \tag{3.29}$$

There are clearly two valid solutions to Equation 3.29 (by nature of the periodicity of the $\tan$ function). The two possible results represent the axis greatest and least inertia, and they are always separated by 90 degrees. In the case where $a = c$ and $b \neq 0$, the orientation is undefined. This will happen when the shape has no clearly defined major or minor axis, such as a circle.

3.2.8 A Note on Image Segmentation

The act of partitioning an image into pieces that represent independent regions is called *segmentation*. Segmentation is currently still a very active field of research and is considered by some to be one of the more difficult problems in the field of machine vision. In general, there is not a single segmentation algorithm that work for every image and successful algorithms are usually tuned to their specific application.

It is not the purpose of this work to solve the segmentation problem. As will be shown in a later section, some assumptions must be made about the nature of the objects of interest in order to make the task of segmentation tractable. With the aid of these assumptions, the machine vision techniques just described can be employed to yield reasonably reliable segmentations.

3.3 Superquadrics

The purpose of this section is to give a brief introduction to geometric modelling tool used in this work: the superquadric. A superficial introduction to the nature of superquadric shapes is followed by their formal derivation and mathematical representation as well as some useful properties that can be directly derived from the superquadric parameters.

3.3.1 Overview

Superquadrics are a family "of three dimensional geometric shapes that includes superellipsoids, supertoroids, and superhyperboloids of one and two sheets" [1]. In common parlance, the term superquadric is often used to describe a superellipsoid and, for the remainder of this thesis, this convention is adopted and the two terms are used interchangeably.

As a parametric shape, a superquadric is a very convenient modeling tool that is capable of representing a large variety of regular shapes in a simple, closed form expression. Figure 16 shows several examples of superquadrics of various parameter values. The figure is intended to give a broad perspective of the shapes capable of being modeled by superquadrics. From the figure, it can be seen that most regular convex shapes can be faithfully represented by a superquadric.

All superquadrics share a fundamental feature; they all exhibit three mutually orthogonal planes of symmetry. Therefore, they can not perfectly model a general convex or otherwise amorphous shape without extending the model through local deformations; a facet we do not entertain in this work. However, many household objects are well represented quite well by superquadrics, a fact shown in [31]. And to the extent that the goal of this work is to reconstruct the shape of such household objects, superquadrics are well suited to the task.

3.3.2 Derivation and Representation

It was shown in [5] that the spherical product of a pair of two dimensional curves results in a three dimensional surface. Mathematically, if we have the two dimensional curves

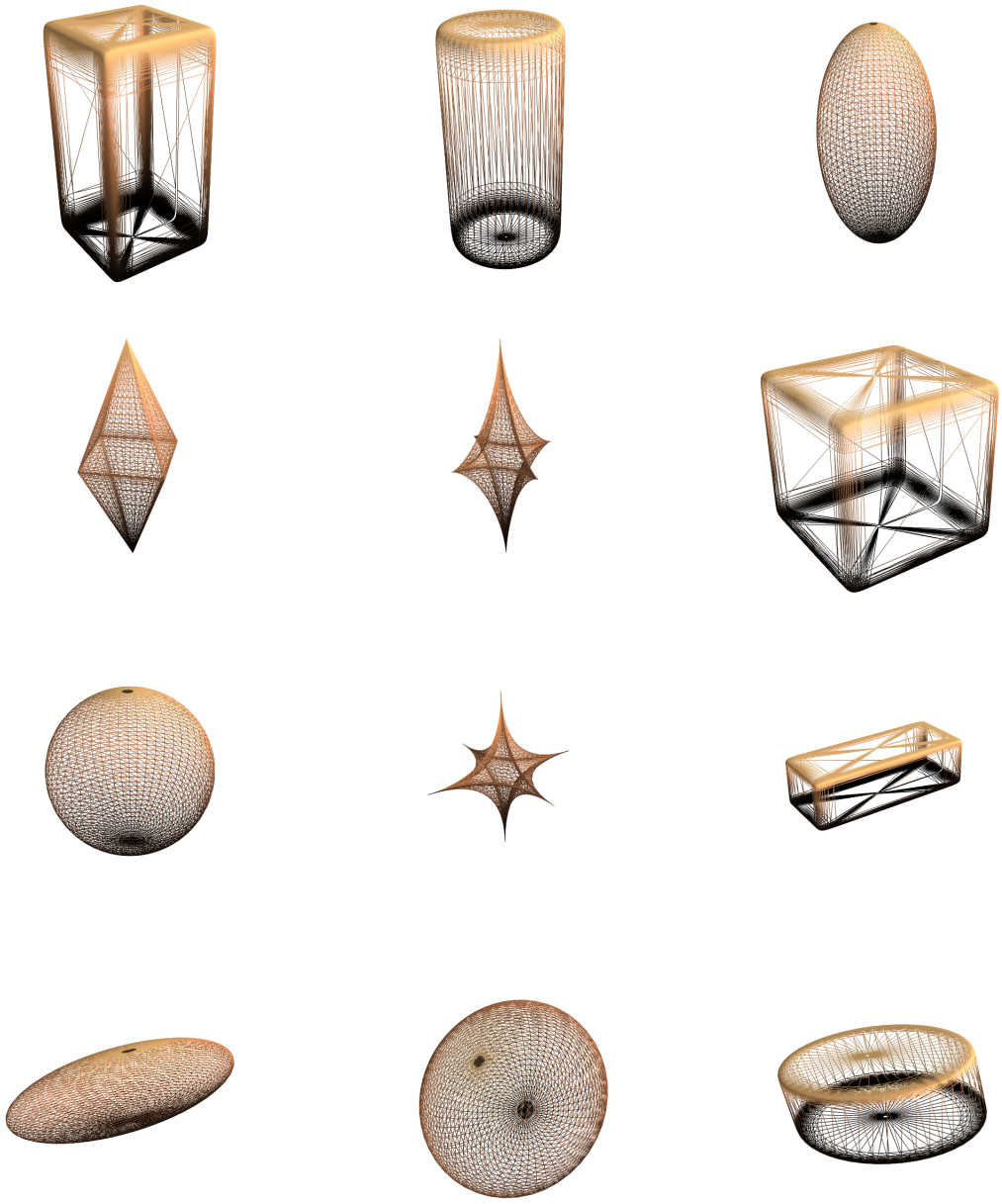

Figure 16.: A sample of superquadrics shapes. The shapes in this figure were created with various parametrizations. Notice the wide and continuous variation of possible shapes.

$$\mathbf{h}(\omega) = \begin{bmatrix} h_1(\omega) \\ h_2(\omega) \end{bmatrix}, \quad \omega_0 \leq \omega \leq \omega_1$$
$$\mathbf{m}(\eta) = \begin{bmatrix} m_1(\eta) \\ m_2(\eta) \end{bmatrix}, \quad \eta_0 \leq \eta \leq \eta_1$$

then the spherical product of the two curves $\mathbf{h}(\omega) \otimes \mathbf{m}(\eta)$ is the three dimensional surface $\mathbf{x}(\omega, \eta)$

$$\mathbf{x}(\omega, \eta) = \begin{bmatrix} m_1(\eta)h_1(\omega) \\ m_1(\eta)h_2(\omega) \\ m_2(\eta) \end{bmatrix}, \quad \begin{matrix} \omega_0 \leq \omega \leq \omega_1 \\ \eta_0 \leq \eta \leq \eta_1 \end{matrix} \tag{3.30}$$

A superquadric is created by the spherical product of two superellipses. The implicit equation of a superellipse in (x, y) has the form

$$\left(\frac{x}{a}\right)^{\frac{2}{\epsilon}} + \left(\frac{y}{b}\right)^{\frac{2}{\epsilon}} = 1$$

which can be parametrized as follows

$$\mathbf{s}(\theta) = \begin{bmatrix} a\cos^{\epsilon}\theta \\ b\sin^{\epsilon}\theta \end{bmatrix}, \quad -\pi \leq \theta \leq \pi \tag{3.31}$$

It was noted in [1] that exponentiation with ϵ represents a *signed power function* and should be interpreted as $\cos^{\epsilon}\theta = sign(\cos\theta)|\cos\theta|^{\epsilon}$. The authors also point out that failure to treat the function as a signed function is a common oversight in many texts, but is a crucial fact when rendering superquadrics in software. The following derivation was presented in [1].

Given two superellipses $\mathbf{s}_1(\eta), \mathbf{s}_2(\omega)$, the superellipsoid $\mathbf{r}(\eta, \omega)$ can be obtained using the spherical product from Equation 3.30

$$\mathbf{r}(\eta, \omega) = \mathbf{s}_1(\eta) \otimes \mathbf{s}_2(\omega) = \begin{bmatrix} \cos^{\epsilon_1}\eta \\ a_3\sin^{\epsilon_1}\eta \end{bmatrix} \otimes \begin{bmatrix} a_1\cos^{\epsilon_2}\omega \\ a_3\sin^{\epsilon_2}\omega \end{bmatrix} =$$
$$= \begin{bmatrix} a_1\cos^{\epsilon_1}\eta\cos^{\epsilon_2}\omega \\ a_2\cos^{\epsilon_1}\eta\sin^{\epsilon_2}\omega \\ a_3\sin^{\epsilon_1}\eta \end{bmatrix}, \quad \begin{matrix} -\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2} \\ -\pi \leq \omega \leq \pi \end{matrix} \tag{3.32}$$

Equation 3.32 is the explicit equation of a superellipsoid. An implicit equation can be derived from the explicit form using the identity $\cos^2\alpha + \sin^2\alpha = 1$. Equation 3.32 can be rewritten in the form

$$\left(\frac{x}{a_1}\right)^2 = \cos^{2\epsilon_1}\eta\cos^{2\epsilon_2}\omega \tag{3.33}$$

$$\left(\frac{y}{a_2}\right)^2 = \cos^{2\epsilon_1}\eta\sin^{2\epsilon_2}\omega \tag{3.34}$$

$$\left(\frac{z}{a_3}\right)^2 = \sin^{2\epsilon_1}\eta \tag{3.35}$$

Raising Equations 3.33 and 3.34 by $\frac{1}{\epsilon_2}$ and adding them together results in

$$\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}} = \cos^{\frac{2\epsilon_1}{\epsilon_2}}\eta \tag{3.36}$$

Now, we raise Equation 3.35 by $\frac{1}{\epsilon_1}$ and Equation 3.36 by $\frac{\epsilon_2}{\epsilon_1}$ and add the two results together to yield the implicit equation for a superellipsoid

$$\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}} = 1 \tag{3.37}$$

Thus, any point (x, y, z) that satisfies Equation 3.37 lies on the surface of the superellipsoid. If we consider the left hand side of Equation 3.37 as a function

$$F(x,y,z) = \left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}} \tag{3.38}$$

then for a point $(x, y, z)^T$, if Equation 3.38 evaluates to less than 1, the point lies inside the superquadric. If it evaluates to greater than 1, the point lies outside the superqaudric. For this reason, Equation 3.38 is termed the *inside-outside* function.

Points that satisfy Equation 3.37 will be expressed in the superquadric centered coordinate system. It is desirable to be able to represent a superquadric in general position and orientation. This can be accomplished through the use of a homogeneous transformation matrix (see Section 3.1.2). We define a homogeneous transformation matrix

$$ {}^{W}_{Q}\mathbf{T} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.39} $$

which is the transformation of the superquadric with respect to the world. However, what need is to be able to express world coordinates in the superquadric centered coordinate frame. To do that, we invert ${}^{W}_{Q}\mathbf{T}$ to yield the transformation of the world with respect to the superquadric

$$ {}^{Q}_{W}\mathbf{T} = \left({}^{W}_{Q}\mathbf{T}\right)^{-1} = \begin{bmatrix} n_x & n_y & n_z & -(p_x n_x + p_y n_y + p_z n_z) \\ o_x & o_y & o_z & -(p_x o_x + p_y o_y + p_z o_z) \\ a_x & a_y & a_z & -(p_x a_x + p_y a_y + p_z a_z) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.40} $$

So now, given a set of homogeneous world coordinates $(x_w, y_w, z_w, 1)^T$, we can calculate the superquadric coordinates according to the relation

$$ \begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix} = {}^{Q}_{W}\mathbf{T} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.41} $$

Applying Equation 3.41 to Equation 3.37 yields the inside-outside function for a superquadric in general position and orientation

$$ \begin{aligned} F(x_w, y_w, z_w) = \Bigg(& \left(\frac{n_x x_w + n_y y_w + n_z z_w - p_x n_x - p_y n_y - p_z n_z}{a_1} \right)^{\frac{2}{\epsilon_2}} + \\ & + \left(\frac{o_x x_w + o_y y_w + o_z z_w - p_x o_x - p_y o_y - p_z o_z}{a_2} \right)^{\frac{2}{\epsilon_2}} \Bigg)^{\frac{\epsilon_2}{\epsilon_1}} + \\ & + \left(\frac{a_x x_w + a_y y_w + a_z z_w - p_x a_x - p_y a_y - p_z a_z}{a_3} \right)^{\frac{1}{\epsilon_1}} \end{aligned} \tag{3.42} $$

As will be shown in later sections, Equation 3.42 forms the basis for the minimization function that will be used in the recovery process.

At first glance, it may appear that Equation 3.42 contains 17 independent parameters. This is not actually the case as 9 of the parameters are related. If we look at the portion of Equation 3.40 that represents the rotation matrix

$$ {}^{Q}_{W}\mathbf{R} = \begin{bmatrix} n_x & n_y & n_z \\ o_x & o_y & o_z \\ a_z & a_y & a_z \end{bmatrix} \tag{3.43} $$

and recall from Section 3.1.2 that the columns of this matrix represent the unit vectors of frame $\{Q\}$ expressed in the coordinates of frame $\{W\}$, we see that ${}^{W}_{Q}\mathbf{R}$ is orthonormal and therefore has a total of six constraints. We can represent this rotation matrix as the combination of three sequential rotations about the coordinate axes by three independent amounts. This is the Euler Angle representation of a rotation. For the purposes of superquadric representation, we use the ZYZ Euler Angle convention which rotates frame $\{W\}$ first about $\hat{Z}_W$ by angle ϕ, then about the new $\hat{Y}_w$ by angle θ, and finally about the new $\hat{Z}_w$ by angle ψ. This operation is captured in the following equation

$$ \begin{aligned} {}^{Q}_{W}\mathbf{R} &= \mathbf{R}_{\hat{Z}}(\phi)\mathbf{R}_{\hat{Y}}(\theta)\mathbf{R}_{\hat{Z}}(\psi) = \\ &= \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} \cos\phi\cos\theta\cos\psi - \sin\phi\sin\psi & -\cos\phi\cos\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\theta \\ \sin\phi\cos\cos + \cos\phi\sin\psi & -\sin\phi\cos\sin + \cos\phi\cos\psi & \sin\phi\sin\theta \\ -\sin\theta\cos\psi & \sin\theta\sin\psi & \cos\theta \end{bmatrix} \end{aligned} \tag{3.44} $$

Equation 3.44 can be substituted directly into Equation 3.40. Further, we see that the parameters $(n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y, a_z)$ can be defined in terms of (ϕ, θ, ψ) and therefore Equation 3.42 actually only has 11 independent parameters $(a_1, a_2, a_3, \epsilon_1, \epsilon_2, \phi, \theta, \psi, p_x, p_y, p_z)$. The physical meanings of these parameters are listed below.

- (a_1, a_2, a_3) are the dimensions of the superquadric in the x, y, and z directions, respectively, as measured from the centroid of the superquadric.

- (ϵ_1, ϵ_2) are the shape exponentials which determine the shape of the superquadric independent of its size. See Figure 16.
- (ϕ, θ, ψ) are the ZYZ Euler Angles which determine the orientation of the superquadric with respect to the world frame.
- (p_x, p_y, p_z) are the (x, y, z) world coordinates of the centroid of the superquadric. i.e. the origin of the superquadric-centered coordinate system.

3.3.3 Useful Properties

The 11 parameters of the superquadric can be used to directly compute many geometric properties of the shape above and beyond what is already plainly defined by the parameters. Some of the properties that are easily calculated include

- The radial distance between a point and the surface
- Surface area
- Volume
- Moments of inertia

The derivation for each one these properties is presented in [1]. In this work, we use the volume of the superquadric as a term in an accuracy metric and thus the expression for volume is given below without derivation.

The volume of a superquadric in terms of its 11 parameters given as

$$V = 2a_1a_2a_3\epsilon_1\epsilon_2 B(\frac{\epsilon_1}{2}+1, \epsilon_1)B(\frac{\epsilon_2}{2}, \frac{\epsilon_2}{2}) \tag{3.45}$$

where B is the beta function and is defined in terms of gamma functions

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} \tag{3.46}$$

$$\Gamma(z) = 2\int_0^\infty t^{z-1}e^{-t}dt \tag{3.47}$$

Chapter 4
Reconstruction Algorithm

The reconstruction algorithm can be logically presented as a sequence of macroscopic operations, each of which carries their own set of challenges. By treating each of the operations as its own independent task, the challenges of each operation can be dealt with efficiently and independent of the rest of the algorithm. The result is an algorithm that is conceptually easier to understand and easier to maintain.

The algorithm is broken down into the following high level tasks: image capture, segmentation and silhouette generation, surface approximation, and shape fitting. Each of these tasks depends on the results of the previous task in order to properly function. This chapter presents the implementation of each of these tasks in the order in which they are executed.

4.1 Image Capture

The algorithm proposed by this work requires three images of the object of interest. These images must be captured from three disparate viewing locations. In theory, the images could be captured from anywhere around the object. However, most objects are not floating in space, and access to the entire periphery of the object will not always be available. Therefore, the images are captured from three mutually orthogonal positions: two from in front of the object, and one from overhead. For the simulation environment (to be presented in a later chapter) such a viewing condition poses no problems. However, a robotic manipulator will often have a singular configuration or kinematic limitation when trying to achieve three orthogonal positions. In such a case the viewing positions are only approximately orthogonal. As will be shown in the results chapters, this relaxation does not have a significant impact on accuracy. An illustration of the viewing positions is shown in Figure 17.

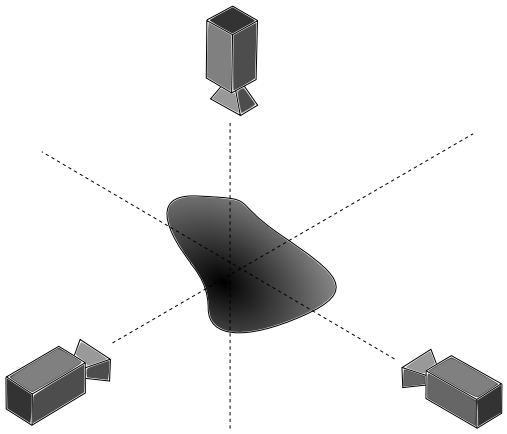

Figure 17.: Three orthogonal viewing directions. This constraint is relaxed in the presence of manipulator kinematic limitations.

After capturing the images, they must be corrected for the distortions introduced by camera as described in Section 3.2.1.2. The topics of camera calibration and image undistortion are presented in the next sections.

4.1.1 Camera Calibration

As detailed in Section 3.2.1.2, there are 9 internal parameters of the camera that need to be found: the focal lengths f_x, f_y the image sensor offsets c_x, c_x, and the radial and tangential distortion coefficients k_1, k_2, k_3, p_1, p_2. It is beyond the scope of this work to derive or implement such a calibration routine. Rather, we use the camera calibration function available in the OpenCV library [18]. This function uses a sequence of images of a known pattern (a chessboard) and employs various regression techniques to calculate the 9 internal camera parameters. A full mathematical derivation of this calibration function is given in [17].

In addition to the 9 internal camera parameters, the external or *extrinsic* camera parameters are also required. These are the 6 parameters (3 rotation, 3 translation) that make up the transformation of the camera with respect to the world base frame. This transformation is required in order to project points in world space into the image. If we let the extrinsic camera parameters be denoted by the transformation matrix ${}^{C}_{W}\mathbf{T}$, the intrinsic camera matrix denoted by $\mathbf{M}$, and a point in homogeneous world coordinates be ${}^{W}\mathbf{P}$, then the projection of ${}^{W}\mathbf{P}$ into the image is given as

$$\mathbf{P}' = \mathbf{M}_W^C \mathbf{T}^W \mathbf{P} \tag{4.1}$$

where the division of the homogeneous coordinate by its last element is implicit. The OpenCV library provides a function for calculating the matrix ${}_W^C\mathbf{T}$. It uses a set of known point correspondences (3D points and their corresponding 2D image locations) along with the camera intrinsics and distortion parameters, and employs least squares minimization to compute the solution.

In general, computing ${}_W^C\mathbf{T}$ is impractical since the value of this matrix changes as the camera moves. Instead we compute the relative transformation of the camera with respect to the robot end effector. Since this transformation is constant no matter where the camera is located, ${}_W^C\mathbf{T}$ can then be computed at any point by knowledge of the robot end effector position and orientation alone. If we let ${}_R^W\mathbf{T}$ represent the transformation of the robot end effector with respect to the world, then the transformation of the camera with respect to the robot end effector can be computed as

$${}_C^R\mathbf{T} = \left({}_W^C\mathbf{T}\,{}_R^W\mathbf{T}\right)^{-1} \tag{4.2}$$

Once ${}_C^R\mathbf{T}$ is known, ${}_W^C\mathbf{T}$ can be computed at any location according to the formula

$${}_W^C\mathbf{T} = \left({}_R^W\mathbf{T}\,{}_C^R\mathbf{T}\right)^{-1} \tag{4.3}$$

Since the value ${}_R^W\mathbf{T}$ is known to high degree of accuracy for a well calibrated robot, Equation 4.3 is used to accurately determine the location and orientation of the camera when each image is captured.

4.1.2 Image Undistortion

Equation 4.1 is valid for cameras with no distortions. But as previously mentioned, most cameras will have distortions and calculating those distortions is part of the camera calibration process. So, in general, the act of a camera of a set of world points will result in a distorted image which must be corrected if Equation 4.1 is to hold. The validity of Equation 4.1 is crucial for image based computations such as measuring distances in the image space or backprojecting rays into three dimensional space. Equations 3.24 and 3.25 provide a formula for correcting a distorted image into its undistorted form. Further, the OpenCV library [18] provides a function for automating

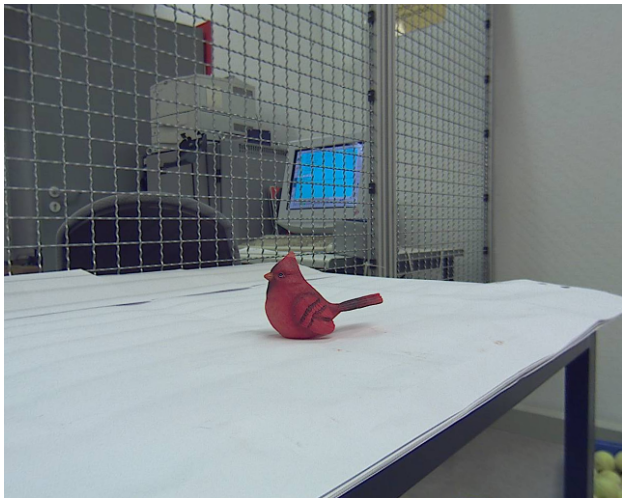

Figure 18.: An undistorted image. The image in Figure 7 after correcting for radial and tangential distortions.

this process. After calibrating the camera to determine its intrinsic and distortion parameters, this function is applied to the image in Figure 7 to yield the undistorted form in Figure 18. Notice in the figure that the lines that should be straight are now straight as opposed to highly curved as in the distorted image.

4.2 Segmentation and Silhouette Generation

Once the images have been captured and, if necessary, pre-processed into an undistorted form, the next step of the algorithm is to convert these images into silhouette images of the object of interest. A silhouette image is nothing but a binary image (see Section 3.2.7) with the object of interest represented as the foreground, and everything else as background. In order to accomplish this, the object of interest must be segmented from the rest of the image. As was mentioned in Section 3.2.8, the methods used to accomplish segmentation are highly dependent on the nature of the image and the problem, and there is not a single technique that works for all situation.

Since the aim of this work is not to solve general problem of segmentation, the objects of interest used in evaluation of the algorithm were chosen such that they facilitated the task of segmentation. That is, all objects used in this work are uniform in color and also a color that is dissimilar from the background. In the case of simulated testing, where the nature of the background can also be

```
def generate_silhouette(image):
    # image is an NxMx3 array
    # representing an RGB image
    rows = image.shape[0]
    cols = image.shape[1]
    silhouette = zeros((rows, cols))
    for i in range(rows):
        for j in range(cols):
            sum = (image[i,j,0] +
                   image[i,j,1] +
                   image[i,j,2])
            if sum != 0:
                silhouette[i,j] = 1
    return silhouette
```

Figure 19.: A silhouette generation algorithm. This algorithm, written in Python, will calculate the silhoutte from the image of a simulated object.

controlled, the color of the object is irrelevant (provided it still differs from the background). For testing conducted in the real world, the test objects are all red in color, and thus the task becomes one of segmenting a red object from the background. The segmentation algorithm was implemented differently for both cases, with the latter being much more complex. Each of the segmentation algorithms is now presented in turn.

4.2.1 Segmenting Simulated Shapes

For testing the algorithm in simulation, the background of the simulator is set to black. That is, every pixel in the background is represented by the RGB pixel tuple $(0, 0, 0)$. Since the object of interest is the only object present in the simulated scene, and given the fact that the object is not black in color, any non-zero pixel represents an object pixel. Thus, the silhouette image for the object of interest can be calculated with the very simple algorithm presented in Figure 19. Some example images of simulated objects, and their corresponding silhouettes are shown in Figure 20.

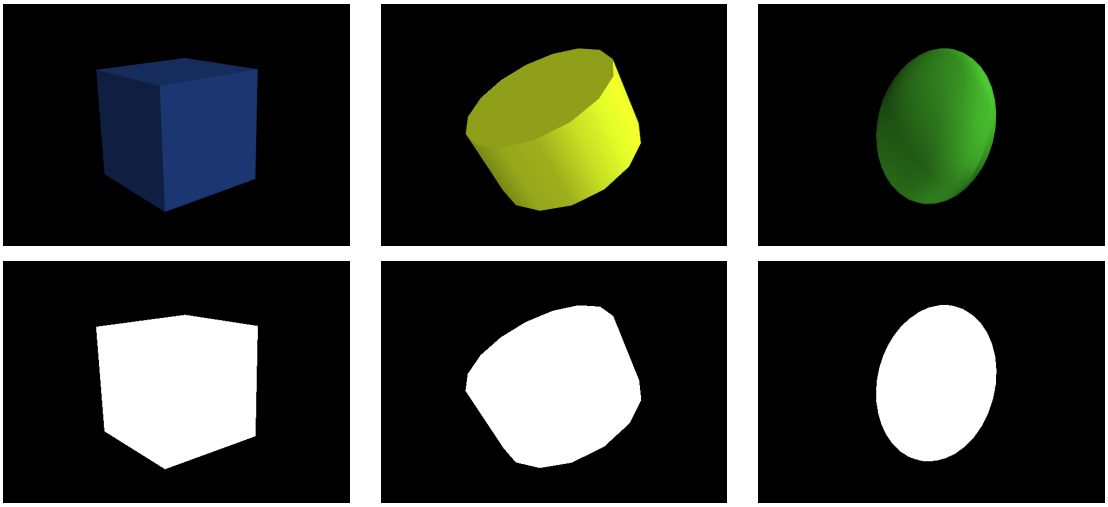

Figure 20.: Images of simulated objects and their corresponding silhouettes. The silhouettes have had their dynamic range expanded to show detail.

4.2.2 Color Based Segmentation of Real Objects

In contrast to the ease with which the silhouette can be calculated from the image of a simulated object, the process to do so with a real image is much more involved. Rather than attempt to explain the process with pseudo-code, we walk through the process step-by-step. No claim is made that the following segmentation algorithm is the optimal approach to segmenting our particular test objects from the background. Again, the aim is not to solve the problem of segmentation, but rather have functional segmentation to the extent necessary to test and validate the reconstruction algorithm. To that end, the segmentation method presented below suited our needs.

Consider the image in Figure 21, which is a scaled-down version of an image that was captured by the robot during the reconstruction process. The red box near the center of the image is the object of interest, and is the object we wish to segment from the background. The background however, is relatively busy and contains other sources of red color. Even though other sources of red are small, it is not enough to simply assume that all red belongs to the object. Instead, we want to locate and isolate the largest patch of red in the image.

In order to locate the largest area of red in the image, we must have a method to determine which pixels in the image represent red pixels. It would be convenient if the color of a pixel could be expressed as a singular value rather than the RGB triplet that has been previously described. It turns

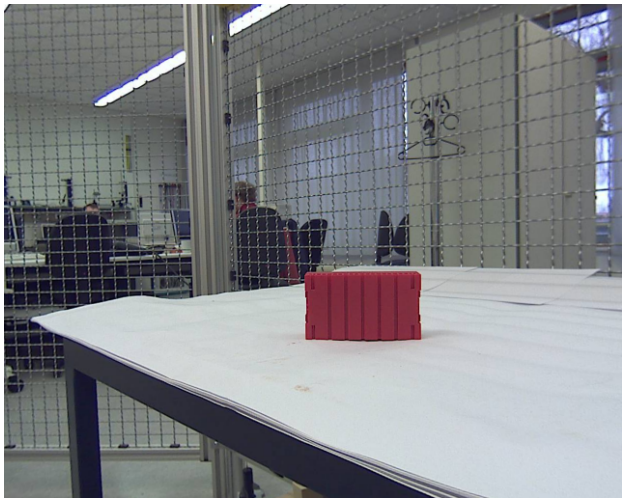

Figure 21.: An image of an object of interest as captured by the robot.

out that this is in fact possible by converting the image into a different color-space. The HSV color space represents the components of a pixel as Hue, Saturation, and Value, rather than components of red, green, and blue. In the HSV color model, the hue plane contains the color information, the saturation plane contains the amount of color present, and the value plane contains the overall intensity level of the pixel. For the math involved in RGB to HSV color space conversion, the reader is directed to a text on image processing such as [35]. The hue, saturation, and value planes of the image in Figure 21 are shown in Figure 22

The range of possible values for the Hue plane is 0-360 which represents the number of degrees in a circle. That is, Hue is a circular quantity with 0 and 360 representing the same color: pure red. Since 360 is too large a value to represent with 8 bits, the range is usually scaled to 0-180 at the cost of a small amount of resolution. Now that the color is represented by a singular value, we can construct a threshold that bounds the color we want to keep, and use this threshold to determine which pixels are red pixels. Figure 23 shows the results of thresholding the hue plane so the red pixels are defined as those pixels with a hue less than 20 or greater than 160 (remember that hue is circular and wraps at 180). It is clearly seen in the figure that the thresholding correctly identifies all the pixels belonging to the red box, however, it also identifies other pixels in the image that are red in color, but are otherwise unsaturated. That is, they are not a ‘strong‘ red.

The shortcomings of thresholding the hue plane can be overcome by adding the additional criteria that not only must the pixel be red in color, but it must also be ‘strong‘ in the color. This can be

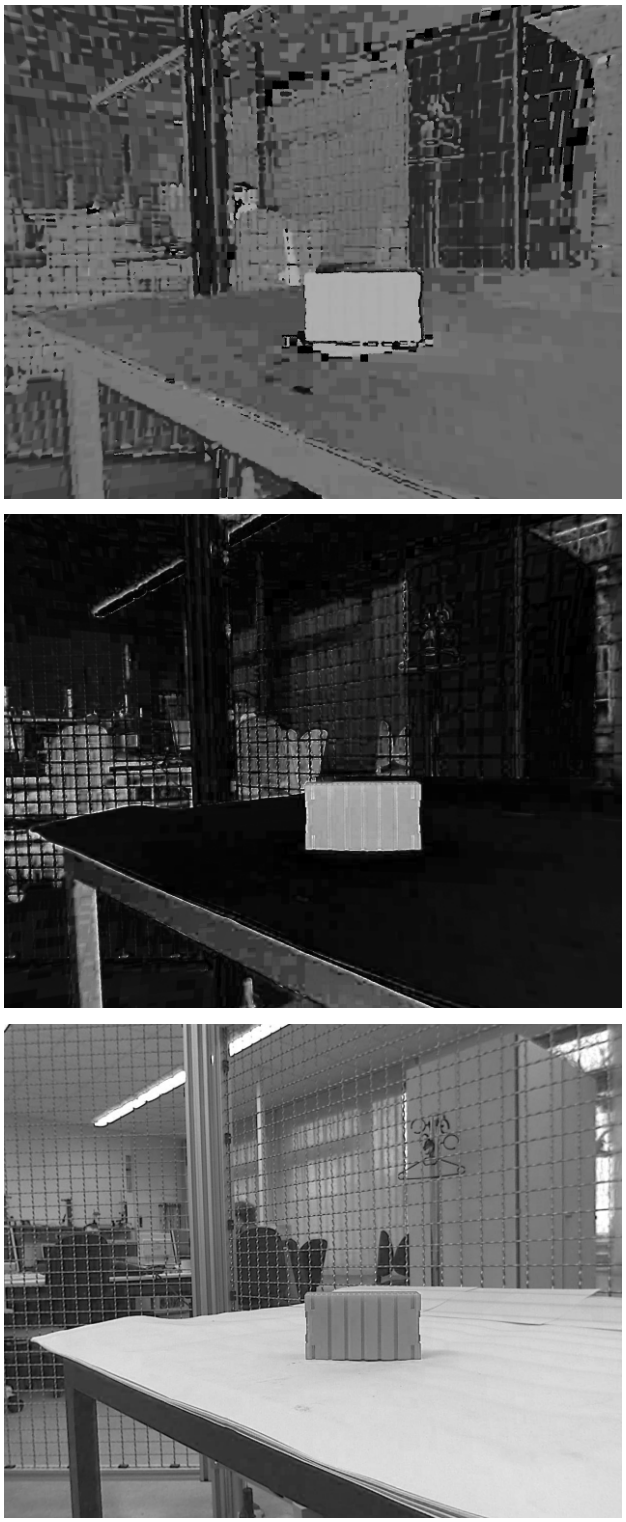

Figure 22.: HSV color space. Top to bottom: the hue, saturation, and value planes of the image in Figure 21.

Figure 23.: Hue plane thresholding. The pixels identified as red pixel by thresholding the Hue plane.

accomplished by thresholding the saturation plane from Figure 22 and combining it using an AND operation with the results of the hue thresholding. The result of this operation is shown in Figure 24 using a threshold value of 150. Thus, any red pixel (as so previously defined) with a saturation value greater than 150 is considered a pixel of interest.

Figure 24 is very close to identifying only those pixels which belong to the red box. However, it is still not perfect, and there are many extraneous pixels that are small sources of red elsewhere in the image. However, the bulk of the pixels belong to the object of interest, and now we can use the binary image properties discussed in Section 3.2.7 to calculate the centroid of this image. Since the vast majority of the pixel belong to the object of interest, the centroid of the image will be somewhere on that object. This location can then be used as a seed location for a flood fill algorithm. The calculated centroid for the image in Figure 24 is identified in Figure 25 by the red circle.

Up to this point, the entire goal was to find a pixel that is assured to belong to the object of interest. Even though the image in Figure 24 is close to being the silhouette of the object (and indeed it could be cleaned up quite nicely), it will not always be the case that the segmentation will be so accurate at this stage. Instead, once a pixel is known to belong to the object of interest, we use a flood fill algorithm (see Section 3.2.6) that is designed to find areas of similar color. The flood fill algorithm chosen is the cvFloodFill algorithm available in the OpenCV library. We use the computed centroid as the seed value and set the threshold so that a neighboring pixel is considered

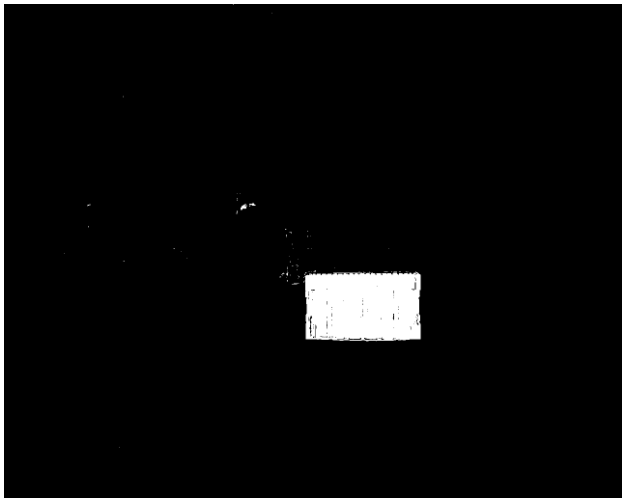

Figure 24.: Hue and saturation thresholding. The results of combining hue and saturation thresholding with an AND operation.

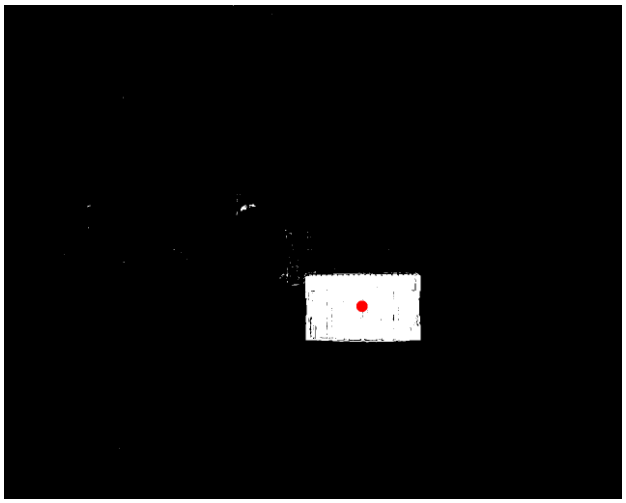

Figure 25.: Seed point determination. The calculated centroid of the image in Figure 24 is identified by the red circle. This is the seed point.

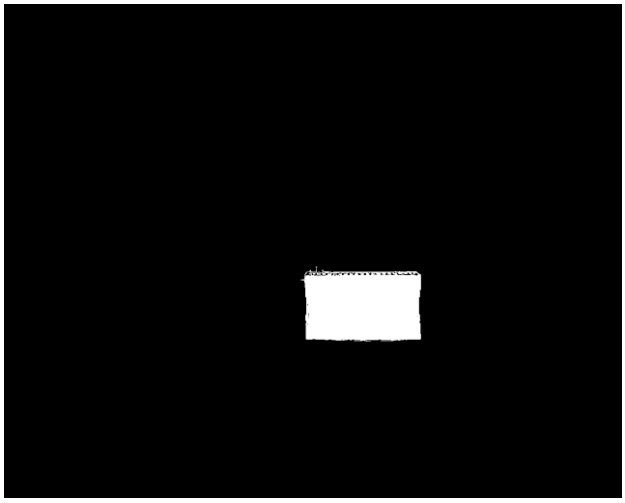

Figure 26.: Flood filling red pixels. The results of using a flood fill operation on the red pixels of image in Figure 21 using the seed point from Figure 25.

part of the same object provided its intensity does not change by more than ± 45 in any of the RGB channels. The result of this operation is shown in Figure 26. It is clearly seen that the flood fill operation has completely isolated the object of interest from the rest of the scene. However, there are still some interior pixels that have not been correctly identified. This is addressed in the final step of segmentation.

The goal now, is to ensure that all the pixels within the body of the object are completely filled. The first reaction might be to use morphological operations to fill the small holes. However, there is no knowing in advance how small the holes will be and therefore selection of a proper morphological kernel is difficult, and morphological hole filling does not guarantee to preserve the image size. A better alternative is to use connected component labeling as discussed in Section 3.2.4. Once the image has been labeled, any component that is not the background can be assumed to be a hole in the silhouette and thus merged. This final step in the segmentation operation yields a useful silhouette and is shown in Figure 27.

This segmentation algorithm is not fool-proof. For example, if the object is hollow or has a hole at the location of the computed centroid, the flood fill algorithm will fail because the seed point will be invalid. Thus, this segmentation algorithm assumes all objects are without any holes. See the note on segmentation in Section 3.2.8.

Figure 27.: Hole filling. The final generated silhouette after using connected components to fill the holes in the image in Figure 26.

4.3 Surface Approximation

The silhouettes generated from the 2D images captured by the robot are used to calculate a rough approximation of the 3D surface of the object. This portion of the reconstruction algorithm falls within a class of algorithms called, appropriately enough, *shape from silhouettes*. This section explains in detail the portion of the reconstruction algorithm that gives an initial approximation to the surface of the object.

4.3.1 Shape from Silhouettes Overview

Shape from silhouettes is the process of reconstructing the 3D shape of an objection of interest by segmenting the object from the background of each 2D image and reprojecting the visual cones. The union of these cones gives an approximation of the 3D object surface. As the of the number of images approaches infinity, the reconstructed 3D surface is termed "the visual hull" [2]. The visual hull is the theoretically best possible 3D reconstruction of an object using only two dimensional silhouettes. It can be shown that, due to the nature of silhouettes, any shape from silhouettes algorithm is incapable of reconstructing surface concavities. This does not represent a large problem for the reconstruction algorithm presented in this work however, because the chosen modeling tool (the superquadric) also cannot represent surface concavities in the form to which we limit it.

Shape from silhouettes has been well studied [8, 2, 21, 25, 37]. In these implementations, calculating the intersection of the visual cones was accomplished through voxel coloring using an octree representation. With these methods, the shape of the object is converged upon in an iterative process. Initially, the object is assumed to be contained within a bounding volume, and this volume is successively split and refined until the desired accuracy is achieved. This requires multiple passes of the algorithm across each image. Further, the nature of the algorithms are such that they must spend time testing voxels that reside in the interior of the object, rather than only dealing with the surface of the object.

Recently, Lippiello et al. [41] introduced a new shape from silhouettes algorithm that reconstructs only the surface of the object. This algorithm functions by constructing a sphere of points around the object and then shrinking this sphere until the points intersect the object. This approach has several advantages over the voxel coloring methods and their octree representations. The octree data structure is rather complex and implementing one efficiently is not a trivial task. Indeed, there has been much research in the efficient implementation of this data structure [37, 21]. In contrast, the data structure in the Lippiello algorithm is a simple array of points; much more tractable and efficient. Furthermore, since this approach reconstructs only the surface of the object, it is computationally more efficient than the voxel coloring methods.

While the algorithm presented in [41] is attractive compared to the historical implementations, we found that there was room for improvement. This algorithm, just like its historical counterparts, has an iteration requirement. In the authors' original implementation, each point is shrunk along its radius iteratively, by a dynamic amount, until that point intersects all the silhouette images. We found this iteration step to be unnecessary. Using the properties of projective geometry, the amount the point must be shrunk can be determined analytically in a single step. By making this modification, the resulting shape from silhouettes algorithm can be applied in a single pass, without iteration, resulting in a drastic performance improvement over other approaches. Each step in the shape from silhouettes algorithm is now presented in detail.

4.3.2 Construction of the Bounding Sphere

The first step in constructing a sphere of points that bounds the object is determining where to place the centroid of the sphere. Logically, the best location to place the centroid of the sphere is

at the centroid of the object. This can be calculated by reprojecting a line from the camera center of each image through the centroid of the silhouette and determine the point which minimizes the sum of squared distances to each line. The geometric interpretation of this is shown in Figure 28. In this figure the three camera centers are shown at points C_1, C_2, C_3 along with the associated image plane. Each image plane shows the imaged silhouette of the unknown object in the center. A line is projected through each camera center that passes through the respective location of the imaged silhouette centroid. It is seen that these lines have an approximate intersection that corresponds roughly to the 3D location of the centroid of the unknown object. In general, the three lines will not intersect at a common point, and as previously mentioned, the point P is found through minimization.

A line in three dimensional space can be specified by two points, since we know the location of the camera center, the task becomes one of finding another point on the line that passes through the camera center and the imaged silhouette centroid. This can be accomplished by inverting the camera matrix and representing the centroid pixel as a homogeneous coordinate. Given a camera matrix that projects points in the camera coordinates onto the imaging plane ${}^{I}_{C}\mathbf{T}$ (see Section 3.2.1.2) the inverse of that matrix ${}^{C}_{I}\mathbf{T}$ will project homogeneous pixel coordinates to camera centered coordinates. Note, that the resulting camera centered coordinate is only guaranteed to lie on the line defined by the camera center and the pixel on the image plane. This is sufficient information however, for defining the line. Thus, the second point on the line, as defined in general world coordinates can be found via the transformation

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = {}^{W}_{C}\mathbf{T}\,{}^{C}_{I}\mathbf{T} \begin{bmatrix} p_x \\ p_y \\ 1 \\ 1 \end{bmatrix} \tag{4.4}$$

where the matrix ${}^{C}_{I}\mathbf{T}$ has been implicitly augmented by an identity row and column so the resulting size is 4x4.

We now have two points defining a line, we will label them $\mathbf{x}_1$ and $\mathbf{x}_2$, and a point we wish to find, defined as $\mathbf{x}_0$, which is the approximate centroid of the object. The perpendicular squared distance d^2 between the point and the line is given as [42]

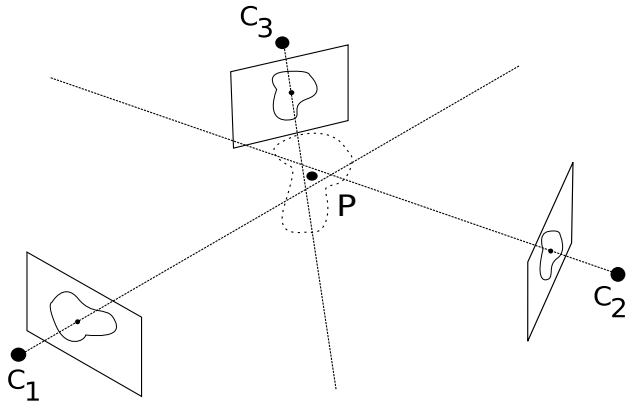

Figure 28.: Centroid localization. The geometric interpretation of finding the 3D object centroid from multiple silhouettes.

$$d^2 = \frac{|(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_1 - \mathbf{x}_0)|^2}{|(\mathbf{x}_2 - \mathbf{x}_1)|^2} \tag{4.5}$$

Then for three lines, the function we wish to minimize is

$$\mathbf{S}(x_0, y_0, z_0) = \sum_{i=1}^{3} d_i^2 \tag{4.6}$$

The value (x_0, y_0, z_0) that minimizes the function $\mathbf{S}$ can be found with one of the many freely available numerical minimization software routines. In this work, we use an implementation of Powell's method [32], available in [15], to find the vector that minimizes $\mathbf{S}$.

Once the 3D centroid of the object has been calculated, the only remaining piece of information needed in order to constructed the sphere is the radius. That is, we need to know how large to make the sphere in order that it completely bounds the object, and thus we are interested in finding the point on the object that is farthest away from its centroid. Provided that the radius of the sphere is greater than the distance from this point to the centroid, then the sphere will bound the object.

The first step in calculating the sphere radius is determining which silhouette image has the largest silhouette radius. The silhouette radius is defined as the farthest distance from the silhouette centroid to the edge of the silhouette. We call this point r_{max}. Figure 29 gives the geometric interpretation

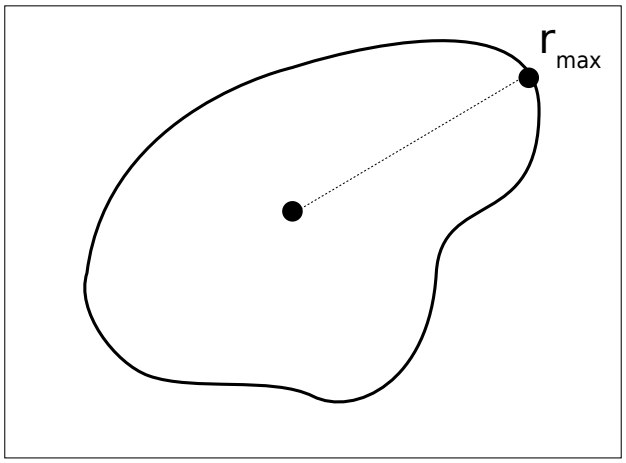

Figure 29.: The geometric interpretation of the silhouette radius r_{max}.

of the quantity. The silhouette radius is calculated for each silhouette, and silhouette with largest r_{max} is selected for further processing.

Once the largest r_{max} is found, the radius of the sphere can be determined through simple geometry, which is drawn out in Figure 30. In the figure, $\mathbf{p}_1$ is the camera center of the image containing the largest r_{max}, $\mathbf{p}_2$ is the centroid location found in the previous step, $\mathbf{p}_3$ is a point on the line that is the backprojection of the point r_{max} through the camera center $\mathbf{p}_1$ and can be calculated using Equation 4.4, and $\mathbf{p}_4$ is the point we wish to determine. It is seen in the figure that the point $\mathbf{p}_4$ will always yield the radius of a sphere that encompass the viewable portion of the object. However, there is no guarantee that the sphere will completely bound the object in actual 3D space. Rather, the only guarantee is that the sphere will completely bound the object *as viewed from the images*. But given that nothing about the object's shape is known except for what is provided in the images, this is sufficient information. The point $\mathbf{p}_4$ is calculated in the following manner:

Let a point $\mathbf{p}$ that lies on the line defined by the points $\mathbf{p}_3$ and $\mathbf{p}_1$ be be parametrized as

$$\mathbf{p} = \mathbf{p}_3 + (\mathbf{p}_3 - \mathbf{p}_1)t \tag{4.7}$$

and let the plane that contains the point $\mathbf{p}_2$ and is normal to the vector $\mathbf{p_1} - \mathbf{p_2}$ be defined as

$$(\mathbf{p}_1 - \mathbf{p}_2) \bullet (\mathbf{p} - \mathbf{p}_2) = 0 \tag{4.8}$$

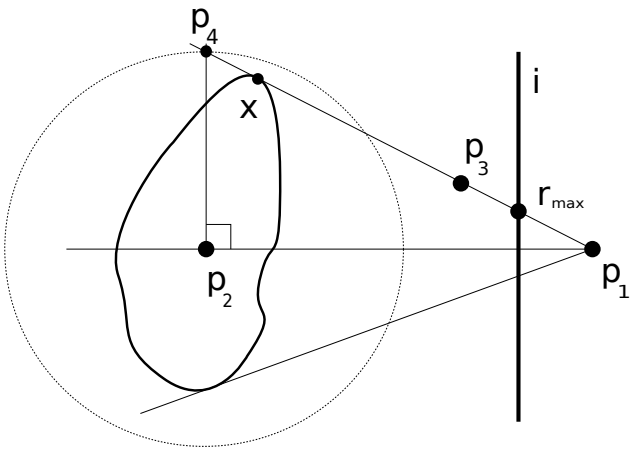

Figure 30.: The geometry for finding the radius of the bounding sphere.

The point that satisfies both Equations 4.7 and 4.8 is the point of intersection between the line and the plane. This point can be found by combining both equations and solving for the parameter t

$$t = \frac{-(\mathbf{p}_1 - \mathbf{p}_2) \bullet (\mathbf{p}_3 - \mathbf{p}_2)}{(\mathbf{p}_1 - \mathbf{p}_2) \bullet (\mathbf{p}_3 - \mathbf{p}_1)} \tag{4.9}$$

Equation 4.9 can be substituted into Equation 4.7 to yield the point $\mathbf{p}$. Inspection of Figure 30 shows that the point $\mathbf{p}_4$ is equal to $\mathbf{p}$. Thus the scalar radius of the sphere r is calculated as

$$r = |\mathbf{p} - \mathbf{p}_2| \tag{4.10}$$

In practice the radius r will be increased by a constant factor (a factor of 1.5 is used in this work) to ensure that the sphere absolutely encompasses the object. This is an 'insurance policy' against error causing corner cases such as point $\mathbf{x}$ being coincident with point $\mathbf{p}_4$. In such a case, any quantization error could cause the radius to be too small, leading to reconstruction error. Since no accuracy is sacrificed by increasing the radius of the sphere by this factor, there is no argument against such an operation.

It is noted that in general, one is unable to solve for the point $\mathbf{x}$ as shown in Figure 30. Since that particular point can be located anywhere along the line $\mathbf{p}_3 + (\mathbf{p}_3 - \mathbf{p}_1)t$ (provided the shape has the appropriate geometry), solving for $\mathbf{x}$ would require an additional constraint which is not available.

Once the centroid and radius of the sphere have been determined, all that remains is to generate a set of points that are evenly distributed across the surface of the sphere. As it turns out, a perfectly even distribution of points on the surface of a sphere is only possible for a certain restricted number of cases (the platonic solids) [13]. Therefore, we must settle for an approximately even distribution. The search for the best approximation is a puzzling mathematical problem and is still actively researched. For the purposes of this work, a distribution that is visually 'even' is sufficient. We use an algorithm based on the Golden Ratio [4] to place the points on the surface of the sphere. The algorithm divides the sphere horizontally into n equal segments. Then for each segement increment the longitude by the golden ratio ϕ and place a point at that location on the surface. This algorithm is detailed in Figure 31. Figure 32 shows a simulated object and the sphere of points that were generated to encompass it using the procedure detailed in this section.

4.3.3 Approximating the 3D Surface

Once the sphere of points is generated, the next step is to modify the position of each point such that the resulting set of points approximates the 3D surface of the object. This is accomplished through the following procedure:

1. Let the center of the camera be $\mathbf{c}_0$.
2. Let the center of the sphere be $\mathbf{x}_0$.
3. Let $\mathbf{x}_i$ be any point in the sphere other than $\mathbf{x}_0$.
4. Let $\mathbf{x}_{i_{new}}$ be the updated position of point $\mathbf{x}_i$.
5. Let the projection of the center of the sphere into the image be $\mathbf{x}'_0$.
6. Then, for each point $\mathbf{x}_i$:
   (a) Project $\mathbf{x}_i$ into the silhouette image to get $\mathbf{x}'_i$.
   (b) If $\mathbf{x}'_i$ does not intersect the silhouette:
      i. Find the pixel point $\mathbf{p}'$ that lies on the edge of the silhouette along the line segment $\mathbf{x}'_i\mathbf{x}'_0$.
      ii. Reproject $\mathbf{p}'$ into $\mathbb{R}^3$ to get the point $\mathbf{p}$.

```
def sphere_points(n=1000):
    # Constructs n evenly distributed
    # points on the unitsphere.

    x = array(n)
    y = array(n)
    z = array(n)

    # the golden ratio
    phi = (1 + sqrt(5)) / 2

    # a unit sphere has diameter 2
    dz = 2.0 / float(n)

    # azimuth increment
    az_incr = 2 * pi / phi

    for i in range(n):
        z[i] = dz * i - 1 + (dz/2)
        azimuth = i * az_incr

        # the radius of the circular cross
        # section of the sphere at point z
        r = sqrt(1 - z*z)
        x[i] = r * cos(azimuth)
        y[i] = r * sin(azimuth)

    points = (x, y, z)
    return points
```

Figure 31.: An algorithm for evenly spaced points on a sphere. This algorithm is written in Python.

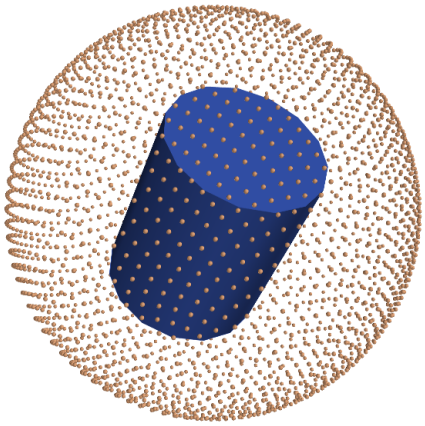

Figure 32.: A sphere of points generated around a simulated object.

iii. Let the line $\mathbf{c}_0\mathbf{p}$ be $\mathbf{L}_1$.

iv. Let the line $\mathbf{x}_0\mathbf{x}_i$ be $\mathbf{L}_2$.

v. Let $\mathbf{x}_{i_{new}}$ be the point of intersection of lines $\mathbf{L}_1$ and $\mathbf{L}_2$

7. Repeat steps 2-6 for each silhouette image.

The geometric interpretation of step 6b is shown in Figure 33. The result of applying this operation is a set of points that approximates the surface of the object and is shown in Figure 34, where the procedure was applied to the scene in Figure 32.

Step 6b in the procedure is worth discussion as it represents the improvement this work has made to this algorithm over the original version presented by the authors in [41]. This portion of the procedure analytically determines where to place the point $\mathbf{x}_i$ along the line connecting $\mathbf{x}_i$ and $\mathbf{x}_0$ such that $\mathbf{x}'_i$ lies on the edge of the silhouette. In [41], rather than treat each point individually, the authors shrink the entire radius of the sphere at once, for all $\mathbf{x}_i$, by an amount that is dynamically determined based on a point $\mathbf{x}'_j$ that lies closest to, but does not intersect, at least one of the silhouettes. This process is repeated until $\mathbf{x}'_j$ intersects every silhouette. When this happens, point $\mathbf{x}_j$ is removed from computation and the process is repeated for all remaining $\mathbf{x}_i$. The authors state "the step is variable because, when a point is back projected near the silhouette contours, the step is reduced to reach a better approximation of the object model". Step 6b shows that such an approximation is unnecessary because the position of the point can be calculated exactly, in a single step.

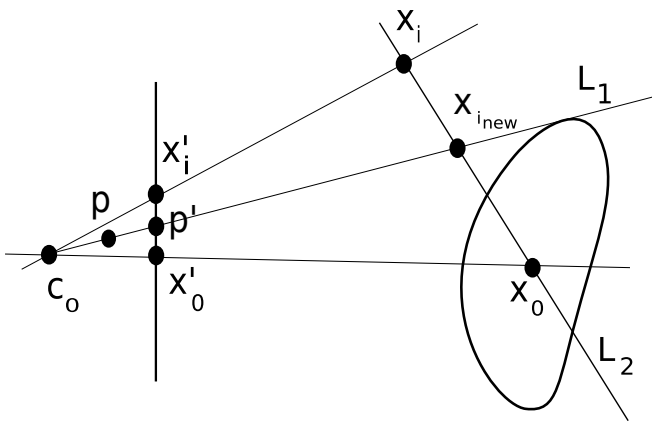

Figure 33.: The geometry of point $\mathbf{x}_{i_{new}}$. The point is the intersection of lines $\mathbf{L}_1$ and $\mathbf{L}_2$. The line $\mathbf{L}_2$ is defined by known points $\mathbf{x}_i$ and $\mathbf{x}_0$. The line $\mathbf{L}_1$ is defined by point $\mathbf{c}_0$, which is the camera center, and point $\mathbf{p}$, which is the reprojection of the image point $\mathbf{p}'$ into $\mathbb{R}^3$.

It is noted that in the perfectly theoretical case, the lines $\mathbf{L}_1$ and $\mathbf{L}_2$ will have an intersection. However, since the point $\mathbf{p}'$ is not sub-pixel accurate, the lines will typically not intersect. Instead, one finds the point of nearest intersection of the two lines. This turns out to be the midpoint of the line segment that is the perpendicular distance between the two lines, and therefore has a closed form solution.

4.3.4 Perspective Projection Error

The results of the procedure in Section 4.3.3 will, in general, only yield a rough approximation of the objects surface. This is due in large part to the error introduced by perspective projection. Consider Figure 35, which illustrates the concept in two dimensions. In the figure, the ray that passes through the camera center C represents the boundary of the object's silhouette. It is this boundary to which the points in the bounding sphere (or circle for the 2D case) are shrunk. The black points on the circle represent the original location of the points on the sphere, and the red points are the locations of the points after they have been shrunk to the silhouette boundary. As seen in the figure, not all of the points will actually shrink to the object's surface. Rather, they shrink to the *perspective projection* of the outermost surface. Now, as the number of images of the object approaches infinity, the approximation will converge to the convex visual hull [2]. This concept is

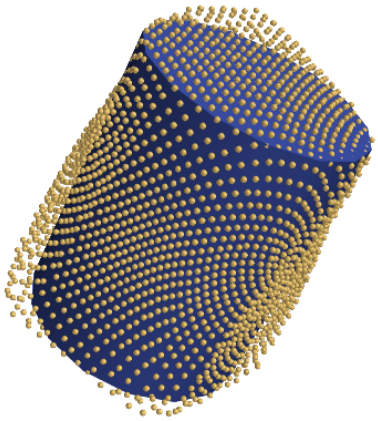

Figure 34.: The results of surface approximation. This image shows the results of applying the procedure of Section 4.3.3 to this image in Figure 32.

shown in Figure 36. However, when the number of images is small, like the three images used in this thesis, the approximation of the surface remains fairly rough, as seen in Figure 34. An extreme example is shown in Figure 37. Since capturing a large number of images in not feasible from both a kinematic and computational stand point, a method to refine the approximation generated from a small number a views is needed. The next step in the algorithm accomplishes this.

4.4 Geometric Shape Fitting

As mentioned at the end of Section 4.3.4, it is typically not possible or otherwise impractical to capture a large enough number of images such that the surface approximation of Section 4.3.3 is sufficiently accurate. This is due to both the kinematic limitations of the manipulator and computational considerations. A manipulator will not, in general, have complete access around the entire periphery of the object, and even if it did, processing a large number of images introduces a significant computational burden; the 134 images used for the reconstruction in [26] took in excess of 100 seconds to process. Therefore, a means of improving the approximation of surface from a small number of images was developed. As will be shown in the results, this method provides a sufficiently accurate approximation to the shape of object to allow for grasping and manipulation planning.

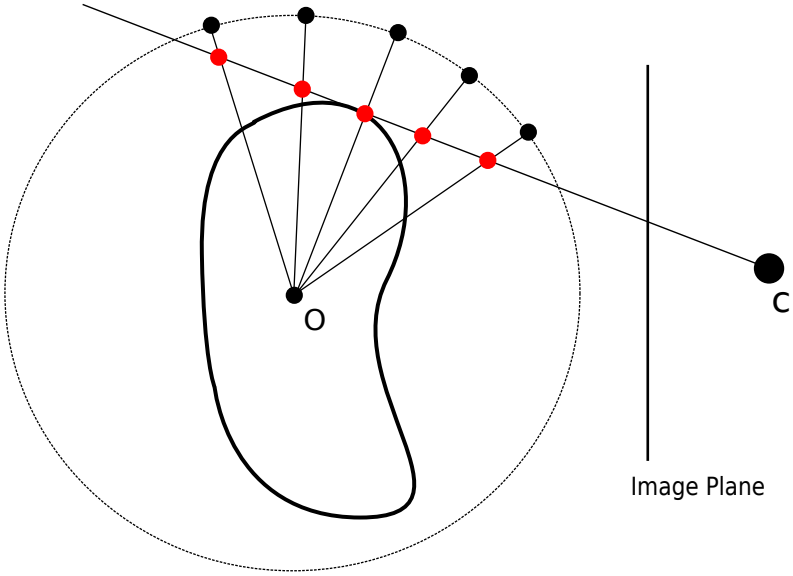

Figure 35.: A 2D illustration of perspective projection error. Given the ray through point C that is the boundary of the imaged silhouette of the object, the black points on the encompassing circle will only be shrunk to the locations indicated by the red points. This results in an approximation of the objects surface. The accuracy of the approximation will increase as the number of images increases.

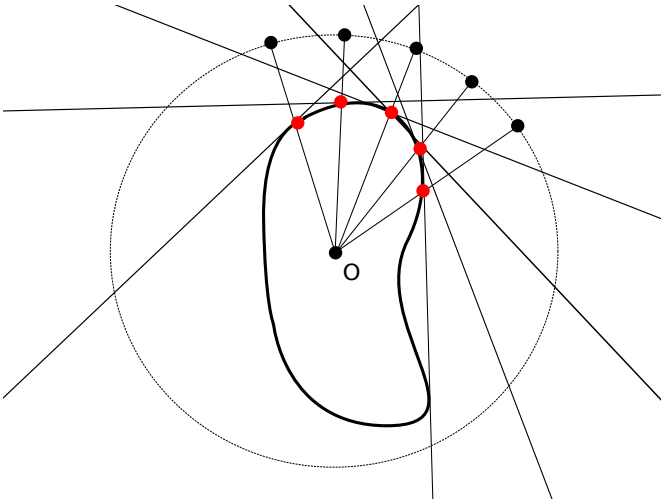

Figure 36.: An illustration of the visual hull concept. As the number of images of the object in Figure 35 becomes large, the approximation converges to the convex boundary of the object.

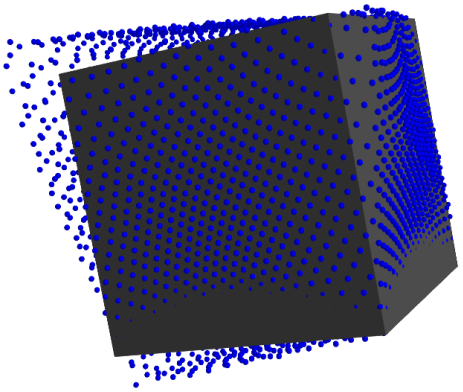

Figure 37.: An example of extreme perspective projection error.

The approximation of the surface of the object is improved by fitting a superquadric to the set points that are the result of the procedure in Section 4.3.3. The motivation for using superquadrics are three fold:

1. The parametrized nature of a superqaudric, as detailed in Section 3.3, naturally lends itself to grasp planning. Indeed, the 11 parameters of the superquadric can be directly used to determine the position, shape, size, and orientation of the object.

2. Superquadrics, as shown in [31], are capable of accurately modeling many everyday household object.

3. The structured nature of the superquadric has the effect of ignoring localized noise due to quantization and segmentation error.

The following sections detail the procedure by which a superquadric is fit to the points. First, the classical superquadric cost function is presented followed by a modified version which provides the added benefit of forcing the superquadric to effectively ignore perspective projection errors like those seen in Figure 37.

4.4.1 Classical Superquadric Cost Function

The goal is to find the superquadric that best fits the set of points which is the output of the point surface reconstruction detailed in Section 4.3.3. The standard method of solving such an optimization problem is to formulate a cost function of the parameters of interest and then use numerical methods to find the values of the parameters that minimizes the value of the function. For superquadrics, the inside-outside function, as developed in Section 3.3, provides an ideal basis for a cost function. If we let the inside-outside function, Equation 3.42, be denoted by F, then a naive cost function could be

$$\min_{\Lambda} \sum_{i=1}^{n} (F-1)^2 \tag{4.11}$$

where

$$\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_{11}\} = \{a_1, a_2, a_3, \epsilon_1, \epsilon_2, \phi, \theta, \psi, p_x, p_y, p_z\} \tag{4.12}$$

Since the inside-outside function evaluates to 1 if a point lies on the surface of the superquadric, it would appear that Equation 4.11 will be at a minimum for a superquadric that is best fit to the points. However in practice, Equation 4.11 will not yield a unique solution in certain side cases. An example is when the points do not form a closed boundary. In such a case, the superquadric can extend in the direction of the boundary opening without changing the value of the minimization function. Two modifications to Equation 4.11 are suggested in [1] to yield a robust and efficient cost function. The first modification is to multiply the error term by $\sqrt{\lambda_1\lambda_2\lambda_3}$ which has the effect of recovering the smallest superquadric to fit the set of points, thus alleviating the problem with the aforementioned corner cases. The second modification is to raise the inside-outside function F to the power of ϵ_1, which according to [1], "makes the error metric independent of the shape of the superquadric that is regulated by ϵ_1" and thereby promotes faster convergence, while not affecting the shape of the superquadric. Thus, the function to minimize, according to [1] is

$$\min_{\Lambda} \sum_{i=1}^{n} (\sqrt{\lambda_1\lambda_2\lambda_3}(F^{\epsilon_1}-1)^2) \tag{4.13}$$

Equation 4.13 is a non-linear equation with 11 independent variables and must be solved via iterative numerical methods. There are several algorithms suitable for the solution of such an equation.

The authors of [1] use, for example, a Levenberg-Marquardt method, however this work makes use of the L-BFGS-B non-linear constrained optimization routine [7] which is available as a FORTRAN implementation in the Scipy library [15]. This routine allows for specifying limits on any number of the independent variables which allows for controlling the allowable shapes of the superquadric and promoting robust convergence. The following constraints are imposed on the independent variables

- $a_1, a_2, a_3 > 0.0$ - The superquadric must have positive size in all directions.
- $0.1 < \epsilon_1, \epsilon_2 < 2.0$ - The superquadric is constrained to everyday convex shapes while also avoiding singular conditions. This constraint promotes robust convergence.
- $-\infty \leq \phi, \theta, \psi, p_x, p_y, p_z \leq \infty$ - No constraints. The superquadric can have any position and any orientation.

4.4.2 Error Rejecting Cost Function

The cost function given in Equation 4.13 is suitable when the points to which the superquadric is fit are a good approximation to the object. When perspective projection error is present however, as in Figure 37, the cost function will cause the superquadric to overestimate the size of the object as it tries to adjust for the outlying points. This effect is shown Figure 38. In that figure, a superquadric is fit to the points in Figure 37 and is shown as the opaque yellow surface. While at first glance it may appear that the superquadric accurately models the cube, in fact the superqaudric has 25% more volume than the cube.

To increase the accuracy in such cases, the error metric is modified with an additional term to force the superquadric to best fit the points without extending beyond the boundary defined by the points. That is, since the absolute best approximation the points can provide (when infinity images are available) is the convex visual hull, it is known for certain that no part of the object lies outside the bounds of the points. We therefore restrict the superquadric to be contained within the set of points by heavily penalizing points that evaluate to less than one in the inside-outside function. The modified cost function is given as

$$\min_{\Lambda}\Bigg[w\sum_{i=1}^{n}(\sqrt{\lambda_1\lambda_2\lambda_3}(F^{\epsilon_1}-1))^2+ \Bigg(\left((1-w)\sum_{i=1}^{n}(\sqrt{\lambda_1\lambda_2\lambda_3}(F^{\epsilon_1}-1))^2 \in F^{\epsilon_1}<1\right)\Bigg] \quad (4.14)$$

This work uses a value of $w = 0.2$, effectively placing an eighty percent weight on the error for points that lie within the recovered superquadric. This modification significantly reduces the effect of perspective projection error and allows extremely accurate recovery of regular objects from three orthogonal views. This value was determined empirically, via trial and error, to provide a good balance between stability and accuracy improvement. The result of applying this constraint is shown in Figure 39. In this figure, the opaque yellow surface is the recovered superquadric for the points in Figure 37 this time calculated with Equation 4.14. The original object is shown as a wireframe due to the fact that boundaries of the superquadric lie directly on the boundary of the cube and they cannot both be visualized simultaneously. As opposed to the reconstruction using the classical cost function (Figure 38), this reconstruction overestimates the volume by only 7%; nearly a 20% improvement at hardly any additional computational cost.

It is noted that in both Figures 38 and 39, the corners of the superquadric are not sharp. This is a direct consequence of limiting the lower bounds of the parameters ϵ_1, ϵ_2 to 0.1 during the optimization process. Should those parameters be allowed to reach 0.0, then the superquadric would have sharp corners as well, but the solution would be unstable and perhaps not converge due to the singularity. Instead, we accept the trade off of a small loss in accuracy for increased robustness. In practice, knowing that the values of ϵ_1, ϵ_2 to be 0.1 allows one to assume the object is prismatic, and the important parameters become the sizes a_1, a_2, a_3. Of the two reconstructions, the one using Equation 4.14 provided the most accurate results for these parameters.

4.4.3 Initial Parameter Estimation

The numerical optimization method used to solve Equation 4.14 requires an initial guess for the model parameters in order to begin the minimization process. This initial guess should be fairly accurate as a completely erroneous guess will lead to non-convergence and false results. This section describes how to compute the parameters for initial guess. The methods presented here follow those which are presented in [1].

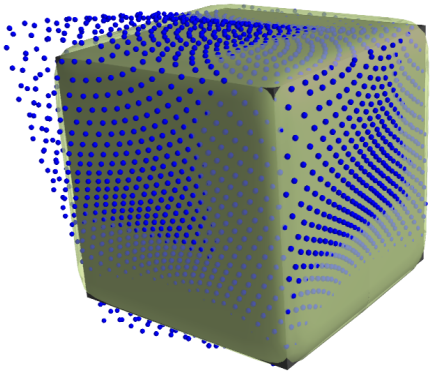

Figure 38.: Overestimation in the presence of perspective projection error.

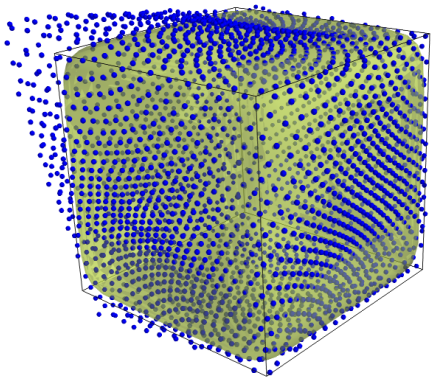

Figure 39.: The effects of the modified cost function. The original object is shown as a wireframe. Compare the accuracy of the recovered superquadric to that in Figure 38.

The initial values of ϵ_1, ϵ_2 are always 1, which means the superquadric always starts the iteration process as an ellipsoid. The initial position of the superquadric is estimated as the centroid of the cloud of points. That is,

$$p_{x_0} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_{w_i} \tag{4.15}$$

$$p_{y_0} = \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_{w_i} \tag{4.16}$$

$$p_{z_0} = \bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_{w_i} \tag{4.17}$$

The initial orientation of the superquadric is determined via the matrix of moments about the centroid. This matrix is given as

$$\mathbf{M}_C = \frac{1}{n}\sum_{i=1}^{n}\begin{bmatrix} (y_i-\bar{y})^2+(z_i-\bar{z})^2 & -(y_i-\bar{y})(x_i-\bar{x}) & -(z_i-\bar{z})(x_i-\bar{x}) \\ -(x_i-\bar{x})(y_i-\bar{y}) & (x_i-\bar{x})^2+(z_i-\bar{z})^2 & -(z_i-\bar{z})(y_i-\bar{y}) \\ -(x_i-\bar{x})(z_i-\bar{z}) & -(y_i-\bar{y})(z_i-\bar{z}) & (x_i-\bar{x})^2+(y_i-y)^2 \end{bmatrix} \tag{4.18}$$

It is desired to find the rotation matrix that diagonalizes the matrix $\mathbf{M}_C$; this is the rotation matrix that will align with the superquadric centered coordinate system. Thus we seek a rotation matrix $\mathbf{T}_R$ such that $\mathbf{D} = \mathbf{T}_R^{-1}\mathbf{M}_C\mathbf{T}_R$. It is well known that the eigenvectors of a matrix, $\mathbf{Q}$, have this diagonalizing property [10]: $\mathbf{D} = \mathbf{Q}^{-1}\mathbf{M}_C\mathbf{Q}$, and thus $\mathbf{T}_R = \mathbf{Q}$. That is, the rotation matrix is composed of the eigenvectors of matrix $\mathbf{M}_C$

$$\mathbf{T}_R = eig(\mathbf{M}_C) \tag{4.19}$$

The order of the eigen vectors is important. It is desired to insure that the Z-axis is aligned with the axis of least inertia for elongated objects, and the axis of greatest inertia for flat objects; a conceptually natural assignment. This can be accomplished by investigation of the magnitudes of the eigenvalues. For the ordered set of eigenvalues $k_1 < k_2 < k_3$ corresponding to the eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ then the $\mathbf{z}$ axis in the matrix $\mathbf{T}_R$ is assigned as follows

if $|k_1 - k_2| < |k_2 - k_3|$ **then** $\mathbf{z} = \mathbf{e}_3$ **else** $\mathbf{z} = \mathbf{e}_1$

The rotation matrix $\mathbf{T}_R$ represents Equation 3.44 and the parameters ϕ, θ, ψ can be directly extracted from that matrix.

$$\theta = \arctan(\frac{\mathbf{T}_R[2,3]}{\sin\phi\mathbf{T}_R[1,3]}) \tag{4.20}$$

$$\phi = \arctan\left(\frac{\mathbf{T}_R[2,3]}{\mathbf{T}_R[1,3]}\right) \tag{4.21}$$

$$\psi = \arctan\left(-\frac{\mathbf{T}_R[3,2]}{\mathbf{T}_R[3,1]}\right) \tag{4.22}$$

It is noted that in software application, these equations should be calculated via the atan2 function.

The final parameters a_1, a_2, a_3 are initially determined by the size of the bounding box that is aligned with the object centered coordinate system. Programmatically, this is most easily determined by using Equation 3.43 to rotate the points so that they are viewed from the superquadric centered system via

$$\mathbf{x}_q = {}^{Q}_{W}\mathbf{R}\mathbf{x}_w \tag{4.23}$$

The limits of the bounding box are then

$$a_1 = \max(|x_{q_i} - \bar{x}|),\ i = 1..n \tag{4.24}$$

$$a_2 = \max(|y_{q_i} - \bar{y}|),\ i = 1..n \tag{4.25}$$

$$a_3 = \max(|z_{q_i} - \bar{z}|),\ i = 1..n \tag{4.26}$$

4.5 Example Reconstruction

The preceding sections of this chapter have given detailed descriptions and derivations on every step of the proposed reconstruction algorithm. This section gives an illustrated example of each step of the algorithm starting with the image capture and ending with the final reconstruction.

Figure 40 shows the step by step reconstruction of a simulated prismatic object. Image (a) shows the original object. Images (b - d) show the three silhouettes generated from the three captured images following the procedure in Section 4.2.1. In this case, the imaging locations were all mutually orthogonal and aligned with the object. Image (e) shows the sphere of points that was constructed

around the original object using the information derived from the silhouettes as explained in Section 4.3.2. Images (f - h) show the point cloud after all the points have been shrunk so that their reprojections intersect or lie on the boundary of the silhouettes by using the procedure in Section 4.3.3. Images (i - j) show the superquadric that minimizes Equation 4.14 for this particular set of points. This final step was described in Section 4.4.

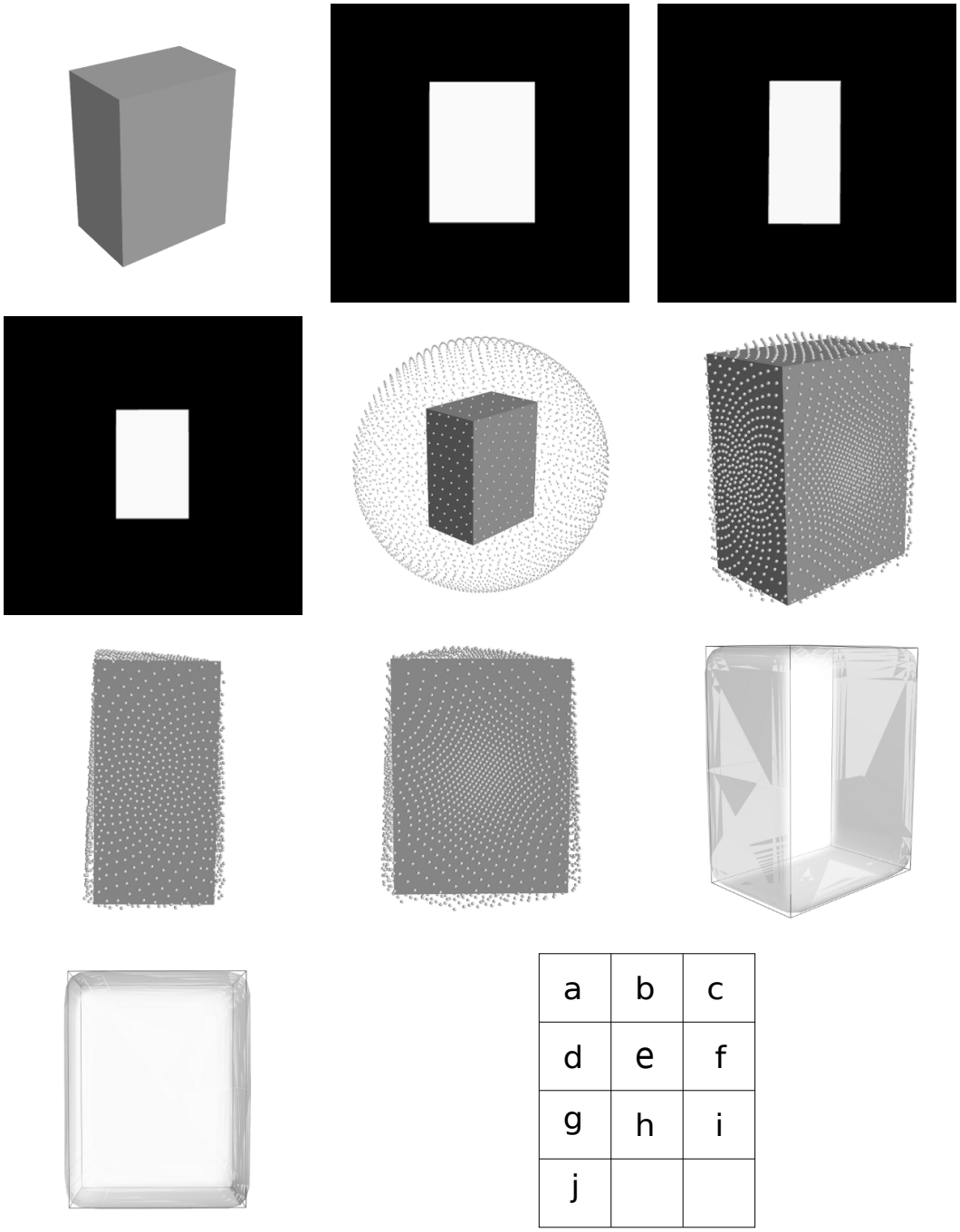

Figure 40.: The reconstruction process as a step by step simulation. (a) The original shape. (b - d) The generated silhouettes. (e) The encompassing sphere of points. (f - h) The point cloud after the points have been shrunk to the silhouette boundaries. Error due to perspective projection is clearly seen. (i - j) The superquadric that was fit to the point cloud. Original shape shown as a wire frame. Notice the ability of the superquadric to ignore the perspective projection error.

Chapter 5
Simulation

The reconstruction algorithm developed and presented in Chapter 4 was implemented and tested in simulation before being implemented and tested in hardware. This chapter describes the software simulation environment that was developed for the purposes of testing the algorithm as well as the results of testing the algorithm on several different simulated shapes.

5.1 Core Algorithm

The algorithm of Chapter 4 was broken apart into two encapsulated concepts so as to facilitate software reuse: 1) image capture and processing and 2) object reconstruction. The image capture and processing is highly dependent on the scene being observed and the cameras/simulation environments involved and therefore must be implemented on a case by case basis. The object reconstruction portion, however, is hardware agnostic. Provided that the silhouette images and camera information (intrinsics and extrinsics) are available, object reconstruction can proceed. By separating the procedures into separate entities, the object reconstruction routines can be used by both the simulation environment and the hardware implementation.

The object reconstruction routines consist of the following:

- Calculation of the bounding sphere of points - Section 4.3.2
- The surface approximation of the object - Section 4.3.3
- The fitting of the superquadric to the point cloud - Section 4.4

These routines are implemented in Python [19], making use of the NumPy and Scipy libraries [15] for numerical algorithms and array type data structures. Portions that are particularly computationally intensive are written in Cython [38] for enhanced performance.

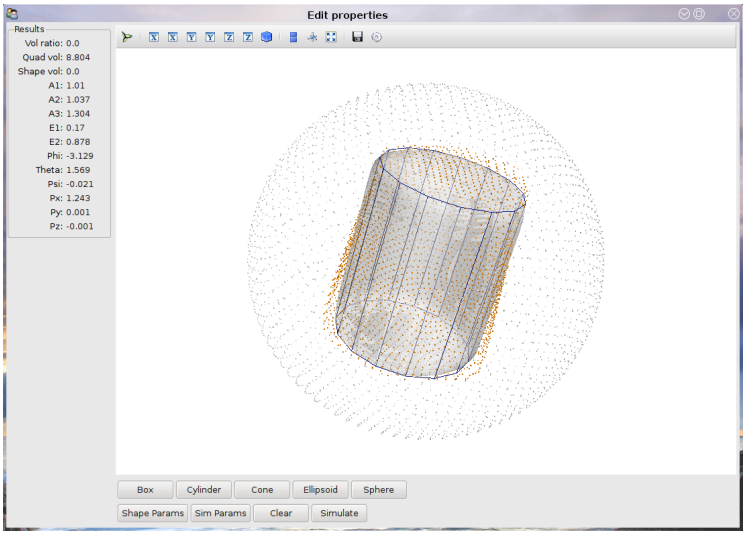

Figure 41.: A screenshot of the graphical simulator.

5.2 Graphical Simulator

A graphical simulator was built on top of the core algorithm for the purposes of testing the algorithm on a wide variety of shapes and camera viewing positions. The simulator was also written in Python using the Traits graphical toolkit [14]. The 3D rendering was achieved with the help of the Mayavi 3D Visualization library [33]. A screenshot of the simulator in action is shown in Figure 41. In the figure, the simulator is showing the original object as a wireframe, the original sphere of points, the points after being shrunken to the surface, and the recovered superquadric. They are shown simultaneously for illustration purposes only.

5.2.1 Capabilities

The simulator contains two main entities: the simulated object, and the simulated camera. Both of these objects are highly configurable. The simulated object can take on a wide variety of regular shapes from prisms to ellipsoids, and its geometry can be modified in height, width, and depth, and its centroid can be placed in any location in space. The camera can be moved to any location in space by specifying a point of focus (the point the camera is looking at), the distance the camera is from this point, and the azimuth and elevation angles which orient the camera with respect to the point of focus.

In order to simulate a reconstruction, the user creates an object of interest and then specifies the three locations of the camera to capture the images. The user then clicks the 'simulate' button and the reconstruction process begins. When the reconstruction has completed, the recovered superquadric is rendered into the simulation environment and the original object converted to a wire-frame. This allows for quick visual inspection of the accuracy of the reconstruction. Additionally, the eleven parameters of the recovered superquadric are displayed in a data box on the left side of the simulator window. This allows for direct comparison of the recovered parameters versus the ground truth.

5.2.2 Limitations

The simulator currently has two main limitations: 1) It cannot simulate arbitrary objects. It is limited to prismatic and ellipsoidal objects. 2) It cannot simulate multiple object simultaneously. Though these limitations exist, they were not a limitation in terms of testing the algorithm developed in this work. The algorithm, in its current form, is unable to deal with either of these cases. That is, it is not designed to reconstruct non-prismatic or non-ellipsoidal objects (though it will be shown later that it still gives useful results for these cases), and it cannot handle multiple objects of interest in the same image.

5.3 Simulation Trials and Results

The algorithm was tested using the simulator for a variety of objects that represent the range of shapes likely to be encountered in a domestic setting. Namely, the algorithm was tested on prisms, cylinders, cubes, and spheres, all of various sizes. Further, each of the shapes was tested with a variety of camera viewing configurations. That is, the orientation of the three orthogonal views with respect to the object was varied, as well as the distance from the camera to the object. It was found that the algorithm was robust to these variations, and consistently yielded accurate results when the camera locations were kept within sound reason.

Rather than present the results of every tested variation, a representative subset of the results is presented in Table 1. The table gives the numerical results for the reconstruction of a simulated prism, cylinder, cube, and sphere. For each case, the values of the eleven reconstructed superquadric parameters are given along with the ground truth for the shape. In addition, a twelfth value is given,

v_f, which stands for the volume fraction. It is defined as the volume of the recovered superquadric divided by the volume of the ground truth shape. The volume fraction was used in [25] as a convenient singular value measure of the accuracy of the reconstruction. Each of these reconstructions is depicted in Figure 42 with the original shape shown as a wireframe and the reconstruction overlayed as an opaque surface.

The results in the table clearly indicate that the algorithm is capable of recovering the shape of the original object to high degree of accuracy. Most parameters deviate from ground truth by only a few percent. Care should be taken when interpreting the values for $(a_1, a_2, \phi, \theta, \psi)$ in the table. Since the only orientation enforced during recovery is the orientation of the object's Z-axis, the values of these five parameters can be interchanged for orientation values that differ by multiples of $\pi/2$. Another way to think of this is given a cube with the Y-axis pointing up, swapping the X and Z axes still results in a cube. These orientation ambiguities become even more pronounced with the cylinder and sphere, where the orientation of the latter is wholly irrelevant. Finally, we give a reminder the the values of (ϵ_1, ϵ_2) are bounded within the range $[0.1, 2.0]$ for the purposes of stability of the minimization routine. Therefore, even though the ground truth of an object may dictate that $\epsilon_1 = 0.0$, the best that the minimization routine is allowed to report is $\epsilon_1 = 0.1$. This situation occurs frequently for shapes with sharp corners such as the prism and cube.

Table 1: Simulation Results

Shape	Prism		Cylinder		Cube		Sphere	
	Truth	Reco	Truth	Reco	Truth	Reco	Truth	Reco
a_1	0.4	0.422	1.0	0.987	0.5	0.515	1.0	0.96
a_2	0.5	0.518	1.0	0.993	0.5	0.515	1.0	0.96
a_3	1.25	1.265	1.5	1.544	0.5	0.513	1.0	0.968
ϵ_1	0.0	0.1	0.0	0.186	0.0	0.1	1.0	0.793
ϵ_2	0.0	0.172	1.0	0.724	0.0	0.1	1.0	0.781
ϕ	1.571	-1.571	1.571	-1.575	0.0	-0.558	0.0	0.49
θ	1.571	1.571	1.571	1.573	0.0	3.13	0.0	-0.005
ψ	-1.571	-1.571	0.0	0.013	0.0	-0.558	0.0	-0.188
p_x	0.0	-0.009	0.0	-0.01	0.0	-0.008	0.0	-0.012
p_y	0.0	0.007	0.0	0.0	0.0	0.008	0.0	0.005
p_z	0.0	0.003	0.0	0.007	0.0	-0.001	0.0	0.004
v_f		1.087		1.088		1.077		1.092

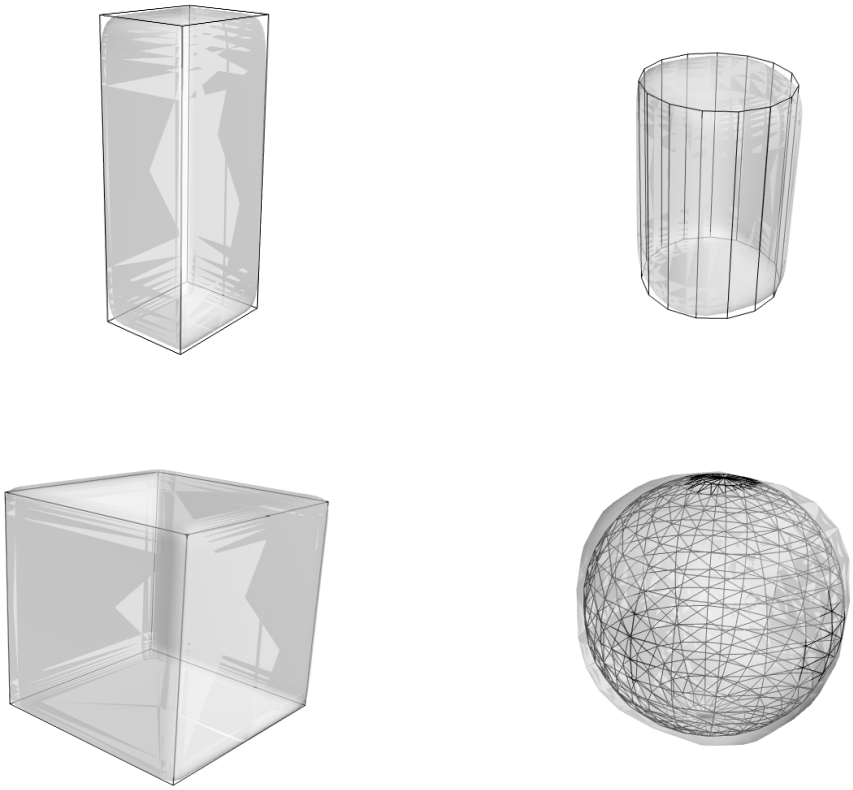

Figure 42.: The reconstruction of simulated objects. Clockwise from upper left: a prism, a cylinder, a sphere, and a cube. The numerical results for these reconstructions are given in Table 1.

Chapter 6

Hardware Implementation

6.1 Hardware Components and Setup

The implementation hardware consists of three main entities: the robotic manipulator which performs the required motions, the camera to capture the images, and the network which consists of the various computers responsible for controlling the robot, the camera, and performing the actual object reconstruction computations.

It is desired to have these various systems interconnected in the most decoupled and hardware/-operating system agnostic manner in order to facilitate software reuse on and with other platforms, robots, and cameras. Thus, portability was a chief goal behind the system design. The following sections describe each subsystem component in detail.

6.1.1 Robot

The robotic arm used in this work is a KUKA KR6/2, manufactured by KUKA Robotics GmbH [29]. It is a six axis, low payload, industrial manipulator with high accuracy and a repeatability of < 0.1 mm. Its smaller size (though still too large for use on a mobile platform) and large workspace makes it well suited for laboratory use and a wide range of experiments. The robot setup, including the camera described in Section 6.1.2 is shown in Figure 43.

The KUKA control software provides a proprietary user interface environment developed in Windows XP Embedded, which in turn runs atop the real time VxWorks operating system. The user interface provides a programming interface to the robot utilizing the proprietary KUKA Robot Language (KRL) as well as an OPC (OLE for Process Control) server that allows for connections from outside computers and the reading and writing of OLE system variables. As KRL does not provide facilities for communicating with outside processes or computers, the OPC server connection was

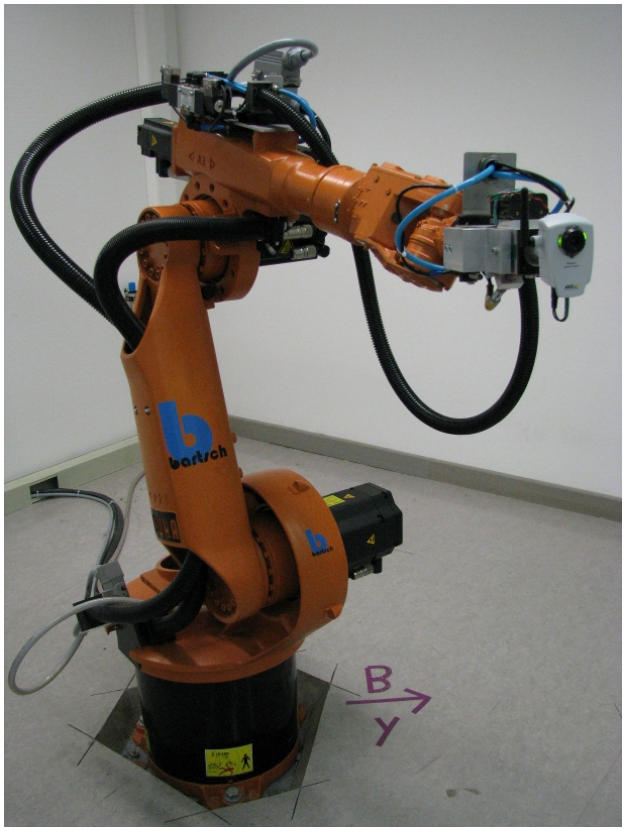

Figure 43.: Robot setup with camera.

used in conjunction with a simple KRL program to export control to an outside machine. The details of this are delayed until Section 6.1.3.

The end effector mounted on the KUKA robot is a generic, two-fingered, pneumatically actuated gripper. It is designed in such a fashion that many different tools can be used with the robot and interchanged on-line. The gripper accepts any tool with a mount that consists of a 5 cm x 5 cm block, with hemispheres cut into either side. These hemispheres mate with their counterpart in the gripper and thus the entire assembly is self-centering. With this design, the robot can autonomously pick up a tool, perform an operation, and then change tools, all with no interference by a human operator. The gripper can be seen in Figure 44.

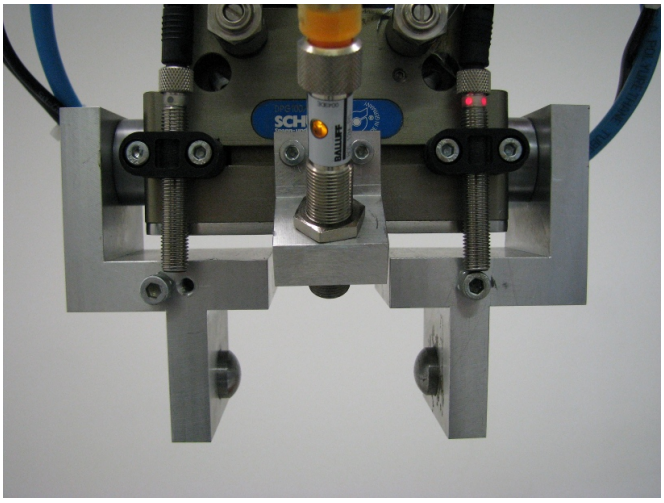

Figure 44.: Self centering robot end effector.

6.1.2 Camera

The camera used for image acquisition is an Axis 207MW wireless network camera. It is relatively inexpensive and has megapixel resolution. The main beneficial feature of the camera is that it contains a built in HTTP web server with support for acquiring images via CGI requests. This means that the camera can be used by any programming language with libraries supporting HTTP and CGI connections. Needless to say, the list of qualifying languages is extensive.

In order to transform the camera into a completely wireless component, a wireless power supply was developed. Namely, a custom voltage regulator was designed and fabricated to regulate the voltage of a battery pack down to the required 5V for the camera. The regulator will operate with any DC voltage from 7 - 25V, allowing interoperation with a wide variety of battery packs. A schematic of the developed voltage regulator is shown in Figure 45.

A camera mount was designed for use with the gripper. It provides for the mounting of the camera, battery pack and voltage regulator, and routes the cables internally to avoid possible entanglement with the robot. Given the self centering design of the gripper, this type of mount has the advantage that the extrinsic parameters of the camera do not change when the camera is mounted and dismounted. Thus, the robot is free to pick up and put down the camera any number of times without requiring recalibration of the camera extrinsics. The camera, mount, and voltage regulator can be seen grasped by the robot in Figure 46.

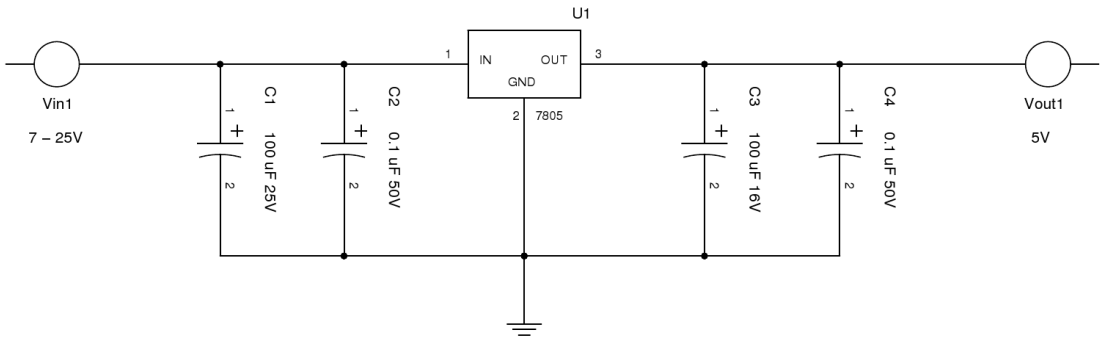

Figure 45.: The camera voltage regulator schematic.

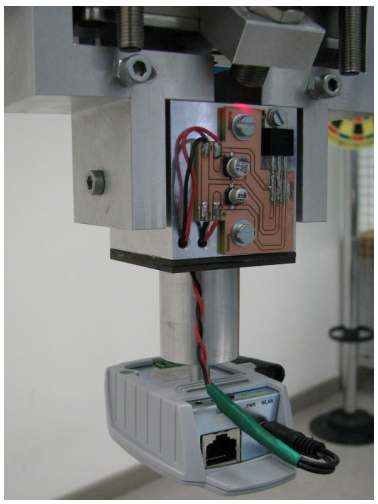

Figure 46.: The camera mounted in the gripper.

6.1.3 Network

In order to achieve our goal of portability, the network was designed around distributed components that use free and open source standards for interprocess communication. Each component in the network is capable of operating independently on its own machine from anywhere that has access to the central switch. In the case of our experiments, the central switch is a local 802.11 Wi-Fi router providing Wi-LAN access to the local computers in the laboratory. In our network setup, there are four distributed components that share information across the LAN:

1. The KUKA robot computer running KRL programs and the OPC server
2. The Axis 207MW wireless network camera
3. The object reconstruction software
4. The external KUKA control software

The logical arrangement of these components, their interconnection, and the communication protocols used are illustrated in Figure 47 and are explained in detail in the following sections.

6.1.3.1 External KUKA Controller and the OPC Server

As previously mentioned, the KUKA robot software provides an OPC server that can be used to read and write system variables at run time. While OPC itself is an open standard, using it remotely requires extensive DCOM configuration which is both tedious and error prone, as well as limiting in that it requires the client machine to run a Microsoft Windows operating system. The OpenOPC project [6] provides a solution to this problem. Built on Python [19], OpenOPC provides a platform agnostic method of making remote OPC requests. It runs a service on the host machine (in this case Windows XP embedded) which responds to requests from the client machine. The host service then proxies the OPC request to the (now local) OPC server, thus bypassing all DCOM related issues. The network communication uses TCP/IP to transmit serialized Python objects [22].

A simple program was written in the KRL language and runs on the KUKA robot computer in parallel with the OPC server. This program sits in an idle loop monitoring the system variables until a command variable changes to True. At this point, the program breaks out of the loop and moves

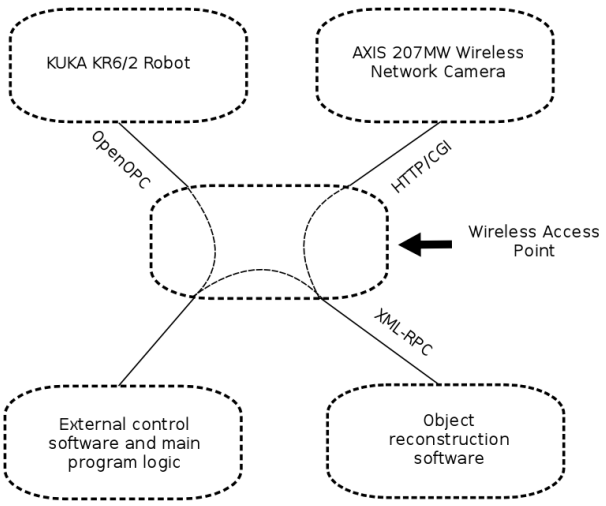

Figure 47.: Network and communication layout.

the robot to a position dictated by other system variables which are also set by the client machine. At the completion of the motion, the program re-enters the idle loop and the process repeats.

The external KUKA controller (the client) runs on a separate machine under the Ubuntu Linux operating system. This machine makes a connection to the OpenOPC service running on the KUKA computer and makes the appropriate requests to read and write the system variables. In this manner, this external machine is able to specify a desired robot position, either absolute or relative, and then, by setting the command variable to True, forces the robot to execute the motion. This machine also acts as the main control logic, synchronizing the robot motion with the image capturing and object reconstruction.

6.1.3.2 Wireless Camera and Object Reconstruction

The wireless camera presents itself on the network as an HTTP server where images can be obtained by making CGI requests. The image is received in the form of raw JPEG data which must be converted to RGB raster format for the purposes of image processing. So that the data need not traverse the network twice, the connection to the camera is made from the object reconstruction program and images are captured and converted upon request by the main control program.

The connection between the main controller and object reconstruction programs utilizes the XML-RPC protocol. A form of remote procedure call, XML-RPC completely abstracts the network connection from the programming interface. This allows calling methods on the host machine as if

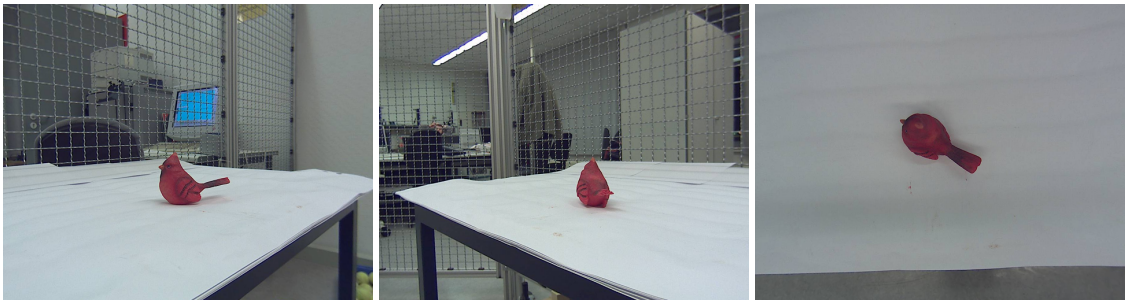

Figure 48.: Three images captured by the robot during a test run. The nature of the disparate viewing locations can be inferred from these images.

they were local to the client software, drastically simplifying the interchange of complex data and operations. XML-RPC is an open standard and available on all major platforms and for many different programming languages. Our implementation utilizes the XML-RPC implementation available in the Python standard library.

6.2 Viewing Positions

The robot is programmed to observe the scene from three locations. Due to kinematic constraints of the robot, these locations are not mutually orthogonal, but they approach such a condition. The three images captured by the robot during one of the test runs are shown in Figure 48. From these images, one can see the nature of the disparate viewing locations; the frontal views are not perfectly horizontal nor is the overhead view perfectly vertical. We note that during reconstruction, the robot is not informed of the location of the object on the table. Rather, it is merely assumed that the object is visible in all three images of the scene; the location of the object in the scene is determined as part of the reconstruction (the p_x, p_y, p_z parameters of the superquadric).

6.3 Test Objects

The algorithm was tested on four different objects: a prismatic battery box, an elongated cylinder composed of two stacked cups, a ball of yarn, and a small cardinal statue. These four objects are shown in Figure 49. The first three objects represent the range of geometric shapes frequently encountered in domestic settings: prismatic, cylindrical, and ellipsoidal. It was expected that the algorithm would achieve accurate reconstructions for these shapes. The last object is amorphous

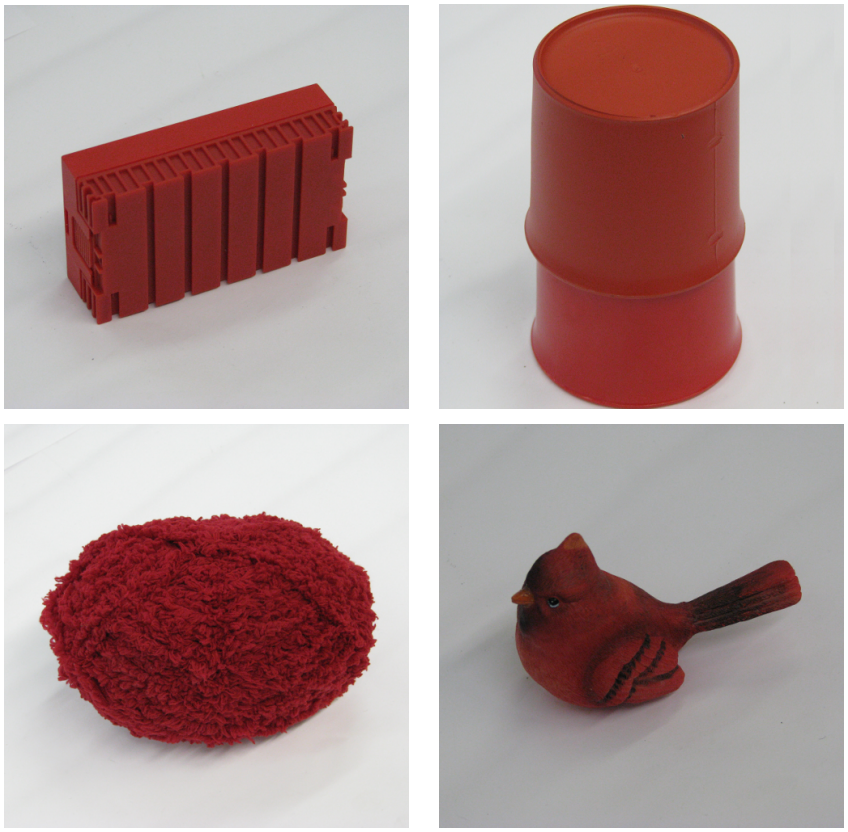

Figure 49.: The four real-world test objects. Clockwise from upper left: a prismatic battery box, a stack of cups, a cardinal statue, and a ball of yarn.

and was included to test the robustness of the algorithm when presented with data that is incapable of being accurately described by the model. In all cases, the test objects are red in color to ease the task of segmentation and subsequently reliable silhouette generation. Again, it is not the aim of this work to solve the broader machine vision problem of segmentation.

6.4 Experimental Trials and Results

This sections discusses the reconstruction results of each of the test objects mentioned in Section 6.3. Each of the cases (with the exception of the cardinal) is accompanied by a rendered figure which shows the ground truth overlayed by the calculated reconstruction. The ground truth is shown as wire frame and the reconstruction as an opaque surface. The accuracy is discussed from a qualitative perspective in the frame of whether or not the reconstructed shape could be used to plan a grasping

maneuver. The numerical results, presented in the same fashion and with the same meaning as the simulated reconstructions in Table 1, are given in Table 2.

When interpreting the accuracy of the results, it must be kept in mind that there are several sources of error that are compounded into these results which are not present in the simulation:

- Imprecise camera calibration: intrinsics and extrinsics
- Robot kinematic uncertainty
- Imperfect segmentation
- Ground truth measurement uncertainty

The last bullet is noteworthy. Since the object is placed randomly in the robot's workspace the only practical way of measuring the ground truth position and orientation is to use a measuring device attached to the end effector of the robot. Though more accurate than attempting to manually measure from the robot base, the error is compounded by both machine inaccuracy and human error.

It must be pointed out, that despite all of these sources of error, the accuracy of most reconstructions is within a couple millimeters of ground truth. Compare this with the results in [27], where a reconstruction with over 200 images resulted in an error of 10 millimeters. Furthermore, the algorithm requires only 0.3 seconds on average to perform the complete reconstruction. This is a significant time savings compared to the 100 seconds required for the reconstruction in [26].

6.4.1 Battery Box

The reconstruction of the battery box, shown in Figure 50, was overall the most accurate of all the reconstructions. It is clearly seen that the model correctly captures the height, width, depth, and shape of the battery box with only a slight deviation in position and orientation. The numerical values of the results in Table 2 confirm this. Though this reconstruction has the largest deviation from unity for the volume fraction, there is no question that the resultant model can be used as a model for grasp planning. Furthermore, the accuracy of the shape representation opens the door for other possibilities such as task inference based on shape and/or appearance.

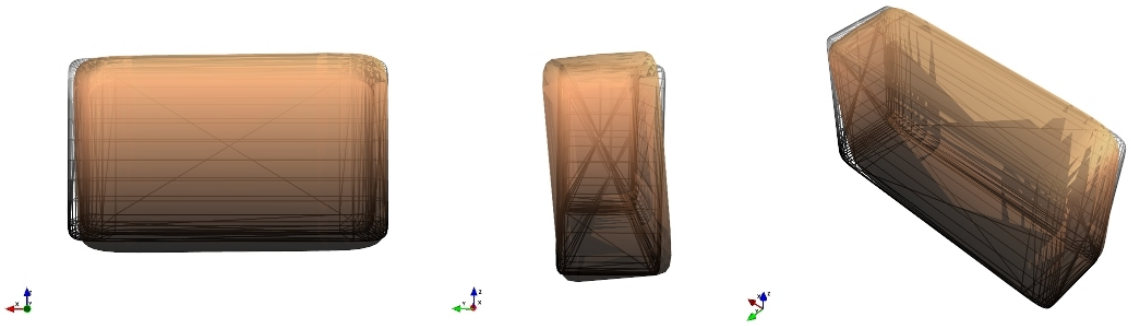

Figure 50.: The reconstruction of the battery box. Ground truth is shown as a wire frame.

6.4.2 Cup Stack

The reconstruction of the stack of cups, which would be accurately approximated as a cylinder, did not achieve high accuracy in all parameters. Namely, the shape parameters ϵ_1, ϵ_2 were inaccurate with respect to ground truth. Shown if Figure 51, it is seen that the reconstructed shape is bordering on prismatic rather than cylindrical. This is a byproduct that stems from the nature of perspective projection shadows and can be eliminated by either more views, or a view perfectly in line with the major axis. The rest of the reconstruction parameters (height, width, depth, position) however, are all accurate, with only the orientation deviating slightly. Since the ground truth shape is cylindrical and vertical, the only orientation parameter of significance is θ, and in this case the values of 0 and π are equivalent. Thus, we see that the orientation has only a slight amount of error. It is noted again that this error stems from a combination of the many compounded error sources mentioned in the beginning of this section.

Despite the non-cylindrical shape of the object, it is believed that the overall size and position are still accurate enough to attempt a grasping maneuver based on the model parameters. A robot designed to operate in a domestic setting should have no problem with the margin of error present in this reconstruction.

6.4.3 Yarn Ball

The yarn barn reconstruction, Figure 52, is nearly as accurate as the battery box. There is slight deviation in the orientation similar to the two previous cases. For this case, the orientation parameter ψ is insignificant (since the shape has symetry about that axis), and 0 and π are equivalent values

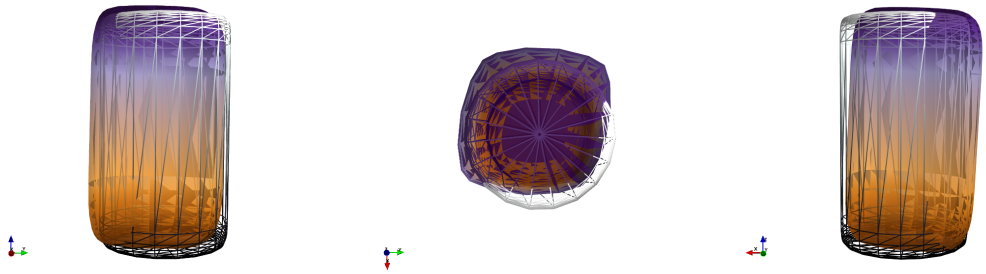

Figure 51.: The reconstruction of the stack of two cups. Ground truth is shown as a wire frame.

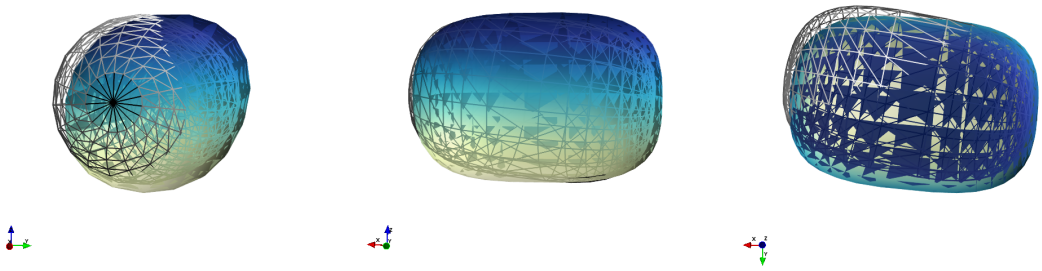

Figure 52.: The reconstruction of the yarn ball. Ground truth is shown as a wire frame.

for ϕ. The yarn ball was the largest of all objects tested at 150mm in length and 100mm in diameter. And though such an object is likely too large to be grasped by most domestic sized manipulators, the accuracy is sufficient to plan the maneuver provided the manipulator has sufficient capacity. It is noted that the size parameters a_1, a_2, a_3 can be directly used as a criteria to determine if an object is within capability limits of the manipulator.

6.4.4 Cardinal Statue

The figurine of the cardinal was included to test how the algorithm performs when provided with data that does not fit well with the reconstruction model and assumptions. This test case is shown in Figure 53. Since it would be difficult to model the ground truth as a wire frame, the results of the surface approximation phase of the algorithm are used instead. From the figure, it is clear that there would be no way to infer from the box shape which is the final reconstruction that the original object was a bird. However, it is interesting to note that the reconstruction is very close to what a

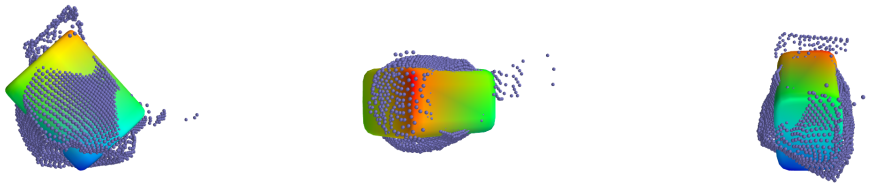

Figure 53.: The reconstruction of the cardinal statue. (a) Side view. (b) Top view. (c) Rear view. The points are the results of the surface approximation phase. The opaque surface is the fitted superquadric. A perspective projection shadow is clearly evident in the bottom right corner of the point cloud in (c).

human would likely provide if asked to select a bounding box that best describes the object. That is, the reconstructed shape does an excellent job of capturing the bulk form of the statue despite the fact that the data is ill formed with respect to the modeling assumptions. It is not a stretch of the imagination to think that a grasp could be accurately planned for this object using the reconstructed shape.

This example shows that, even when the object does not take a form that can be accurately modeled by a single superquadric, the algorithm still generates useful results.

6.5 Limitations

This section details the main limitations encountered during the implementation and testing of the algorithm on the robotic hardware.

The simulation environment developed in this work demonstrated excellent reconstructions when three orthogonal views of the object were available with the object centered in each view. In actual implementation, it was found that it is difficult for the robot to reach three mutually orthogonal positions without exceeding its joint limits or reaching singular positions, thus the positions reached were only approximately orthogonal and they were not, in general, centered on the object.

There is however, no hard requirement from the algorithm that the object be centered in the frame. Indeed, it is designed to function provided only that the object is completely visible in the frame, wherever that may be. However, when the object is located at an oblique location from the camera,

Table 2: Experimental Results

Shape	Battery Box		Cup Stack		Yarn Ball		Cardinal Statue	
	Truth	Reco	Truth	Reco	Truth	Reco	Truth	Reco
a_1	30	32.9	34	41.0	50	57.1	25[1]	24.0
a_2	15	16.9	34	37.8	50	51.5	25[1]	18.4
a_3	52.5	51.6	60	61.2	75	74.4	30[1]	29.5
ϵ_1	0.1	0.2	0.1	0.3	0.7	0.6	*	0.1
ϵ_2	0.1	0.2	1.0	1.4	1.0	1.1	*	0.4
ϕ	0.0	3.12	0.0	-0.3	-0.17	3.07	*	-9.35
θ	1.57	1.56	0.0	3.10	1.53	1.52	*	-0.82
ψ	0.0	0.10	0.0	-2.60	0.0	0.86	*	6.01
p_x	880	878.4	898	893.8	898	893.9	898	892.0
p_y	-924	-924.6	-915	-917.5	-915	-912.7	-915	-908.9
p_z	865	864.9	892	894.8	855	854.1	862	867.3
v_f		1.18		1.13		1.14		*

[1]Approximation based on the bounding box that would encompass the bulk of mass.

*The value has no meaning in the context of this shape.

large perspective projection shadows are introduced, and this tends to stretch the reconstructed shape in the direction of the shadow. This error can be handled in one of two ways. First, if the centroid of the silhouette is determined to be too far from the image center, the robot can be repositioned and a new image captured. Second, the direction and amount of skew can be determined by analysis of the set of points and the effect compensated for by placing constraints on the appropriate variables before executing the minimization routine.

Image segmentation is widely regarded as one of the more difficult problems in computer vision. Up to this point, we have developed our system without regard to the process of segmenting the object of interest from the background. i.e. we have assumed a perfect silhouette to be available. In a simulated environment, perfect segmentation is trivial to achieve, as full control over the environment is available. The situation become exceedingly dire, however, in a real environment with an unstructured background, fluorescent lighting, and other confounding effects. Our testing environment consisted solely of red objects with very little to no reflectance. This drastically simplified the task of segmentation, though the results were still not perfect, and the thresholds had to be manually tuned for various lighting conditions or variations in color shade. These simplifications and manual threshold tuning are obviously unacceptable in the context of achieving our original goal of requiring no prior information about the object.

Chapter 7

Conclusion and Future Work

This work has developed an algorithm capable of recovering the three dimension geometry of a novel object using only three camera views. It was shown that by capturing the three images of the novel object taken from disparate locations, the algorithm is able to calculate a parametrized model of that object with sufficient accuracy to allow for the planning of robotic grasping and manipulation maneuvers. In contrast to other efforts in the literature, the proposed algorithm requires fewer images, significantly less computation time, and yields an overall higher reconstruction accuracy. Furthermore, the parameters of the reconstructed model can be directly used for grasp and manipulation planning. No further analysis of the shape or time consuming statistical methods are necessary.

The algorithm was implemented in both simulation and hardware. In both environments, the algorithm yielding exceedingly accurate reconstructions. Despite additional sources of error and uncertainty in the hardware environment, the accuracy of the algorithm decreased only marginally. Furthermore, when presented with data that were unable to be modeled accurately according the modeling paradigm, the algorithm still generated useful and logical results. With respect to the results presented here, the algorithm has met the objectives of Section 1.2 and it is believed that there is merit to further investigation and research into this proposed method of novel object recognition.

7.1 Future Work

Future plans include integrating a grasping algorithm based on the reconstructed superquadric parameters and testing the accuracy of the grasp on a variety of household objects; specifically those involved with ADLs. The algorithm will also be tested to observe the behavior when the viewing locations become less and less disparate. It is planned to investigate what can be done to increase the accuracy to an acceptable level when such a condition arises, such as incorporating the appear-

ance data that is discarded by using only silhouettes. That is, an attempt will be made to incorporate structure that can be inferred from the raster images with the structure of the superquadric to improve the accuracy and overall robustness of the reconstruction. It is also planned to investigate incorporating other sensory information to augment the abilities of the optical reconstruction by providing depth information that cannot be recovered due to projection shadows. Work in this area has already begun with the use of a single point laser range finder. We also plan to investigate robust regression techniques that will fit multiple superquadrics to the point cloud, thereby allowing us to reconstruct complex objects as a sequence of superquadrics. Finally, since our algorithm depends on high quality silhouettes in order to achieve accurate results, having facilities for robust segmentation is a high priority. We will investigate the adoption of segmentation algorithms that can, to a large extent, automatically adapt to various lighting conditions and object façades.

References

[1] A. Jaklic, A. Leonardis, and F. Solina. *Segmentation and Recovery of Superquadrics*, volume 20 of *Computational Imaging and Vision*. Kluwer Academic Publishers, 2000.

[2] A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *Transactions of Pattern Analysis and Machine Intelligence*, 16(2), February 1994.

[3] Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng. Robotic Grasping of Novel Objects using Vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[4] Author Unknown. Evenly Distributed Points on Sphere. http://www.cgafaq.info/wiki/Evenly_distributed_points_on_sphere, 3 April 2010.

[5] A. H. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

[6] Barry Barnreiter. The OpenOPC Project. http://openopc.sourceforge.net/, 16 December 2009.

[7] C. Zhu, R.H. Byrd, and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN Routines for Large Scale Bound Constrained Optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.

[8] Charles R. Dyer. *Volumetric Scene Reconstruction From Multiple Views*, pages 469–489. Foundations of Image Understanding. Kluwer, Boston, 2001.

[9] D. G. Lowe. Object Recognition from Local Scale-Invariant Features. *Proceedings of the International Conference on Computer Vision*, 2:1150–1157, 1999.

[10] D. G. Zill and M. R. Cullen. *Advanced Engineering Mathematics, Third Edition*. Jones and Bartlett Publishers, 2006.

[11] D. Kim, R. Lovelett, and A. Behal. Eye-in-Hand Stereo Visual Servoing of an Assistive Robot Arm in Unstructured Environments. *International Conference on Robotics and Automation*, pages 2326–2331, May 2009.

[12] Danica Kragic, Marten Bjorkman, Henrik I. Christensen, and Jan-Olof Eklundh. Vision for robotic object manipulation in domestic settings. *Robotics and Autonomous Systems*, 52:85–100, 2005.

[13] Dave Rusin. Topics on Sphere Distributions. http://www.math.niu.edu/ rusin/known-math/95/sphere.faq, 3 April 2010.

[14] Enthought Inc. The Traits Framework for Validation and Event-Driven Programming in Python. http://code.enthought.com/projects/traits/, 2001–.

[15] Eric Jones, Travis Oliphant, Pearu Peterson, et al. Scipy: Open source scientific tools for Python, 2001.

[16] Freek Liefhebber and Joris Sijs. Vision-based control of the Manus using SIFT. *International Conference on Rehabilitation Robotics*, June 2007.

[17] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[18] Gary Bradski, Trevor Darrell, Intel Corporation, et al. OpenCV Computer Vision Library. http://opencv.willowgarage.com/wiki/.

[19] Guido van Rossum et al. The Python Programming Language. http://www.python.org, 2009.

[20] Hai Nguyen, Cressel Anderson, Alexander Trevor, et al. El-E: An Assistive Robot that Fetches Objects from Flat Surfaces. *HRI Workshop on Robotic Helpers: User Interaction Interfaces and Companions in Assistive and Therapy Robots*, 2008.

[21] Hiroshi Noborio, Shozo Fukuda, and Suguru Arimoto. Construction of the Octree Approximating Three-Dimensional Objects by Using Multiple Views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6), November 1988.

[22] I. de Jong. Python Remote Objects. http://pyro.sourceforge.net/, 2010.

[23] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–714, 1986.

[24] John J. Craig. *Introduction to Robotics: Mechanics and Control - Third Edition*. Pearson Prentice Hall, 2005.

[25] K. Shanmukh and Arun Pujari. Volume Intersection with Optimal Set of Direction. *Pattern Recognition Letters*, 12:165–170, 1991.

[26] K. Yamazaki, M. Tomono, and T. Tsubouchi. *Picking up an Unkown Object through Autonomous Modeling and Grasp Planning by a Mobile Manipulator*, volume 42/2008 of *STAR*. Springer Berlin / Heidelberg, 2008.

[27] Kimitoshi Yamazaki, Masahiro Tomono, Takashi Tsubouchi, and Shin'ichi Yuta. 3-D Object Modelling by a Camera Equipped on a Mobile Robot. *International Conference on Robotics and Automation*, April 2004.

[28] Kimitoshi Yamazaki, Masahiro Tomono, Takashi Tsubouchi, and Shin'ichi Yuta. A Grasp Planning for Picking up an Unknown Object for a Mobile Manipulator. *International Conference on Robotics and Automation*, 2006.

[29] KUKA Robotics GmbH. KR6/2. http://www.kuka-robotics.com/en/products/industrial_robots/low/kr6_2/.

[30] Linda Shapiro and George Stockman. *Computer Vision*. Prentice Hall, February 2001.

[31] M.J. Schlemmer, G. Biegelbauer, and M. Vincze. Rethinking Robot Vision - Combining Shape and Appearance. *International Journal of Advanced Robotic Systems*, 4(3):259–270, 2007.

[32] Powell MJD. An efficient method for finding the method of a function of several variables without calculating derivatives. *Computer Journal*, 7:152–162, 1964.

[33] Prabhu Ramachandran and Gaël Varoquaux. The Mayavi data visualizer. http://code.enthought.com/projects/mayavi, 2005.

[34] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2003.

[35] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing - Third Edition*. Pearson Prentice Hall, 2008.

[36] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunck. *Machine Vision*. McGraw-Hill, Inc., 1995.

[37] Richard Szeliski. Rapid Octree Construction from Image Sequences. *CVGIP: Image Understanding*, 58(1), July 1993.

[38] Stefan Behnel, Robert Bradshaw, Dag Sverre Seljebotn, Greg Ewing, et al. Cython: C-Extensions for Python. http://www.cython.org, 2009.

[39] Sutono Effendi, Ray Jarvis, and David Suter. Robot Manipulation Grasping of Recognized Objects for Assistive Technology Support Using Stereo Vision. *Australasian Conference on Robotics and Automation*, 2008.

[40] Tsuneo Yoshikawa, Masanao Koeda, and Hiroshi Fujimoto. *Experimental Robotics*, volume 54/2009 of *Springer Tracts in Advanced Robotics*, chapter Shape Recognition and Optimal Grasping of Unkown Object by Soft-Fingered Robotic Hands with Camera, pages 537–546. Springer Berlin/Heidelberg, 2009.

[41] V. Lippiello and F. Ruggiero. Surface Model Reconstruction of 3D Objects From Multiple Views. *International Conference on Robotics and Automation*, pages 2400–2405, May 2009.

[42] Eric W. Weinstein. Point-Line Distance–3-Dimensional. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html, 2010.