MONOCULAR DEPTH PERCEPTION AND ROBOTIC GRASPING OF NOVEL OBJECTS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Ashutosh Saxena

June 2009

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

(Andrew Y. Ng)   Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

(Stephen Boyd)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

(Sebastian Thrun)

Approved for the University Committee on Graduate Studies.

Abstract

The ability to perceive the 3D shape of the environment is a basic ability for a robot. We present an algorithm to convert standard digital pictures into 3D models.

This is a challenging problem, since an image is formed by a projection of the 3D scene onto two dimensions, thus losing the depth information. We take a supervised learning approach to this problem, and use a Markov Random Field (MRF) to model the scene depth as a function of the image features. We show that, even on unstructured scenes of a large variety of environments, our algorithm is frequently able to recover accurate 3D models.

We then apply our methods to robotics applications: (a) obstacle avoidance for autonomously driving a small electric car, and (b) robot manipulation, where we develop vision-based learning algorithms for grasping novel objects. This enables our robot to perform tasks such as open new doors, clear up cluttered tables, and unload items from a dishwasher.

Acknowledgements

First and foremost, I would like to thank my advisor Andrew Y Ng for his continuous support and the stream of ideas that kept me occupied. In addition to being instrumental in suggesting amazing research directions to pursue, he has been very helpful in providing the resources and the motivation to execute them. Finally, he has been a very patient mentor who also provided a strong moral support.

I also thank Sebastian Thrun, Daphne Koller, Oussama Khatib, Kenneth Salisbury and Stephen Boyd for their ideas. The discussions with Sebastian Thrun and Oussama Khatib have been very motivational.

I would like to thank Isaac Asimov, and a number of other science fiction writers for getting me excited about the field of robotics and artificial intelligence.

I owe much to the students I worked with, specifically Sung Chung, Justin Driemeyer, Brad Gulko, Arvind Sujeeth, Justin Kearns, Chioma Osondu, Sai Soundararaj, Min Sun, and Lawson Wong. I really enjoyed working with Min Sun, Justin Driemeyer and Lawson Wong over a number of projects; their ideas and tenacity have been instrumental in many of the researches presented in this dissertation.

I also thank Stephen Gould, Geremy Heitz, Kaijen Hsiao, Ellen Klingbeil, Quoc Le, Andrew Lookingbill, Jeff Michels, Paul Nangeroni, and Jamie Schulte. I give warm thanks to Jamie Schulte and Morgan Quigley for help with the robotic hardware and the communication software.

I would like to thank Pieter Abeel, Adam Coates, Chuong Do, Zico Kolter, Honglak Lee, Rajat Raina, Olga Russakovsky, and Suchi Saria for useful discussions. I am indebited to Olga for her help in expressing my ideas into the papers I wrote.

Parts of the work in this thesis were supported by the DARPA LAGR program

Contents

List of Tables

List of Figures

xix

xxii

Chapter 1

Introduction

Modern-day robots can be carefully hand-programmed to perform many amazing tasks, such as assembling cars in a factory. However, it remains a challenging problem for robots to operate autonomously in unconstrained and unknown environments, such as the home or office. For example, consider the task of getting a robot to unload novel items from a dishwasher. While it is possible today to manually joystick (tele-operate) a robot to do this task, it is well beyond the state of the art to do this autonomously. For example, even having a robot recognize that a mug should be picked up by its handle is difficult, particularly if the mug has not been seen before by the robot. A key challenge in robotics is the ability to perceive novel environments, in order to take an action.

One of the most basic perception abilities for a robot is perceiving the 3D shape of an environment. This is useful for both navigation and manipulation. In this dissertation, we will first describe learning algorithms for depth perception using a single still image.

1.1 Depth perception from a single image

Cameras are inexpensive and widely available. Most work on depth perception has focused on methods that use multiple images and triangulation, such as stereo vision. In this work however, we were interested in the problem of estimating depths from
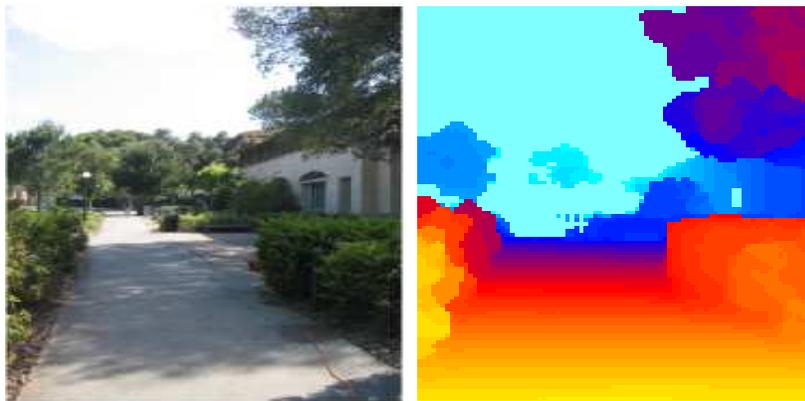
Figure 1.1: (a) A single still image, and (b) the corresponding depths. Our goal is to estimate the depths given a single image.

a single image (see Fig. 1.1). This has traditionally been considered an impossible problem in computer vision; and while this problem is indeed impossible in a narrow mathematical sense, it is nonetheless a problem that humans solve fairly well. We would like our robots to have a similar ability to perceive depth from one image. In addition to being a problem that is interesting and important in its own right, single-image depth perception also has significant advantages over stereo in terms of cost and of scaling to significantly greater distances.

In order to estimate depths from an image, humans use a number of monocular (single image) cues such as texture variations, texture gradients, interposition, occlusion, known object sizes, light and shading, haze, defocus, etc. [9] For example, many objects' texture will look different at different distances from the viewer. In another example, a tiled floor with parallel lines will appear to have tilted lines in an image. The distant patches will have larger variations in the line orientations, and nearby patches with almost parallel lines will have smaller variations in line orientations. Similarly, a grass field when viewed at different distances will have different texture gradient distributions. There are a number of other patterns that we have learned to estimate depth from image features. For example, a large blue patch in the upper part of scene is more likely to be sky. In the past, researchers have designed algorithms to estimate 3D structure from a single image for a few very specific settings.

Nagai et al. [98] used Hidden Markov Models to performing surface reconstruction from single images for known, fixed objects such as hands and faces. Criminsi et al. [21] compute aspects of geometry of the scene using vanishing points and parallel lines in the image. However, this method works only for images with clear parallel lines such as manhattan-type environments and fails on the general scenes that we consider.

In order to learn some of these patterns, we have developed a machine learning algorithm. Our method uses a supervised learning algorithm, which learns to estimate depths directly as a function of visual image features. These features are obtained from a number of low-level features computed at each point in the image. We used a 3D scanner to collect training data, which comprised a large set of images and their corresponding ground-truth depths. Using this training set, we model the conditional distribution of the depth at each point in the image given the monocular image features.

However, local image cues alone are usually insufficient to infer the 3D structure. For example, both blue sky and a blue object would give similar local features; hence it is difficult to estimate depths from local features alone.

Our ability to "integrate information" over space, i.e. understand the relation between different parts of the image, is crucial to understanding the scene's 3D structure. For example, even if part of an image is a homogeneous, featureless, gray patch, one is often able to infer its depth by looking at nearby portions of the image, so as to recognize whether this patch is part of a sidewalk, a wall, etc. Therefore, in our model we will also capture relations between different parts of the image. For this, we use a machine learning technique called Markov Random Field that models the statistical correlations between the labels of neighboring regions.

Our algorithm was able to accurately estimate depths from images of a wide variety of environments. We also studied the performance of our algorithm through a live deployment in the form of a web service called Make3D. Tens of thousands of users have successfully used it to automatically convert their own pictures into 3D models.

A 3D model built from a single image will almost invariably be an incomplete

model of the scene, because many portions of the scene will be missing or occluded. We then consider 3D reconstruction from multiple views. Such views are available in many scenarios. For example, robot(s) may be able to take images from different viewpoints while performing a task. In another example, consider the few pictures taken by a tourist of a scene from a few different places.

Given such sparse set of images of a scene, it is sometimes possible to construct a 3D model using techniques such as structure from motion (SFM) [33, 117], which start by taking two or more photographs, then find correspondences between the images, and finally use triangulation to obtain 3D locations of the points. If the images are taken from nearby cameras (i.e., if the baseline distance is small), then these methods often suffer from large triangulation errors for points far-away from the camera. (I.e., the depth estimates will tend to be inaccurate for objects at large distances, because even small errors in triangulation will result in large errors in depth.) If, conversely, one chooses images taken far apart, then often the change of viewpoint causes the images to become very different, so that finding correspondences becomes difficult, sometimes leading to spurious or missed correspondences. (Worse, the large baseline also means that there may be little overlap between the images, so that few correspondences may even exist.) These difficulties make purely geometric 3D reconstruction algorithms fail in many cases, specifically when given only a small set of images.

However, when tens of thousands of pictures are available—for example, for frequently-photographed tourist attractions such as national monuments—one can use the information present in *many* views to reliably discard images that have only few correspondence matches. Doing so, one can use only a small subset of the images available, and still obtain a "3D point cloud" for points that were matched using SFM. This approach has been very successfully applied to famous buildings such as the Notre Dame; the computational cost of this algorithm was significant, and required about a week on a cluster of computers [164].

In the specific case of images taken from a pair of stereo cameras, the depths are estimated by triangulation using the two images. Over the past few decades, researchers have developed very good stereo vision systems (see [155] for a review).

Although these systems work well in many environments, stereo vision is fundamentally limited by the baseline distance between the two cameras. Specifically, their depth estimates tend to be inaccurate when the distances considered are large (because even very small triangulation/angle estimation errors translate to very large errors in distances). Further, stereo vision also tends to fail for textureless regions of images where correspondences cannot be reliably found. One of the reasons that such methods (based on purely geometric triangulation) often fail is that they do not take the monocular cues (i.e., information present in the single image) into account.

On the other hand, humans perceive depth by seamlessly combining monocular cues with stereo cues. We believe that monocular cues and (purely geometric) stereo cues give largely orthogonal, and therefore complementary, types of information about depth. Stereo cues are based on the difference between two images and do not depend on the content of the image. Even if the images are entirely random, it would still generate a pattern of disparities (e.g., random dot stereograms [9]). On the other hand, depth estimates from monocular cues are entirely based on the evidence about the environment presented in a single image.

In this dissertation, we investigate how monocular cues can be integrated with any reasonable stereo system, to obtain better depth estimates than the stereo system alone. Our learning algorithm models the depths as a function of single image features as well as the depth obtained from stereo triangulation if available, and combines them to obtain better depth estimates. We then also consider creating 3D models from multiple views (which are not necessarily from a calibrated stereo pair) to create a larger 3D model.

1.2 Robot applications using vision

Consider the problem of a small robot trying to navigate through a cluttered environment, for example, a small electric car through a forest (see Figure 1.2). For small-sized robots, the size places a frugal payload restriction on weight and power supply. In these robots, only light-weight small sensors that take less power could be used. Cameras are an attractive option because they are small, passive sensors with

low power requirements.

In the problem of high-speed navigation and obstacle avoidance for a small car, we have applied our ideas of single image depth perception to obstacle avoidance. We had two other motivations for this work: First, we believe that single-image monocular cues have not been effectively used in most obstacle detection systems; thus we consider it an important open problem to develop methods for obstacle detection that exploit these monocular cues. Second, the dynamics and inertia of high speed driving (5m/s on a small remote control car) means that obstacles must be perceived at a distance if we are to avoid them, and the distance at which standard binocular vision algorithms can perceive them is fundamentally limited by the "baseline" distance between the two cameras and the noise in the observations, and thus difficult to apply to this problem.

Figure 1.2: The remote-controlled car driven autonomously in various cluttered unconstrained environments, using our algorithm.

For obstacle avoidance for a car, the problem simplifies to estimating the distances to the nearest obstacles in each driving direction (i.e., each column in the image). We use supervised learning to train such a system, and feed the output of this vision system into a higher level controller trained using reinforcement learning. Our method was able to avoid obstacles and drive autonomously in novel environments in various locations.

1.2.1 Robot grasping

We then consider the problem of grasping novel objects. Grasping is an important ability for robots and is necessary in many tasks, such as for household robots cleaning up a room or unloading items from a dishwasher.

If we are trying to grasp a previously known object, and we are given a full and accurate 3D model of the object, then for grasping one can build physics-based models, which model the contact forces using known geometry of the hand and the object in order to compute stable grasps, i.e., the grasps with complete restraint. A grasp with complete restraint prevents loss of contact and thus is very secure. A form-closure of a solid object is a finite set of wrenches (force-moment combinations) applied on the object, with the property that any other wrench acting on the object can be balanced by a positive combination of the original ones [85]. Intuitively, form closure is a way of defining the notion of a "firm grip" of the object [82]. It gurantees maintenance of contact as long as the links of the hand and the object are well approximated as rigid and as long as the joint actuators are sufficiently strong [123]. [75] proposed an efficient algorithm for computing all n-finger form-closure grasps on a polygonal object based on a set of sufficient and necessary conditions for form-closure.

The primary difference between form-closure and force-closure grasps is that force-closure relies on contact friction. This translates into requiring fewer contacts to achieve force-closure than form-closure [73, 47, 90]. A number of researches the conditions necessary (and sufficient) for form- and force-closure, e.g., [131, 102, 119, 120, 6]. Some of these methods also analyze the dynamics using precise friction modeling at the contact point (called friction cones, [85]).

However, in practical scenarios it is often very difficult to obtain a full and accurate 3D reconstruction of an object, specially the ones seen for the first time through vision. For stereo systems, 3D reconstruction is difficult for objects without texture, and even when stereopsis works well, it would typically reconstruct only the visible portions of the object. Even if more specialized sensors such as laser scanners (or active stereo) are used to estimate the object's shape, we would still only have a 3D reconstruction of the front face of the object.

In contrast to these approaches, we propose a learning algorithm that neither
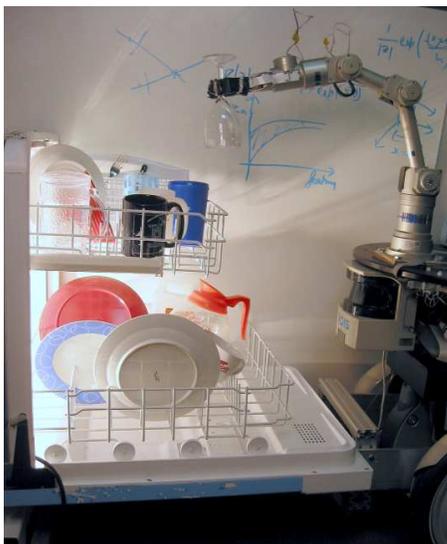
(a) Our robot unloading items from a (b) Grasping in cluttered environment with a
dishwasher.   multi-fingered hand

requires, nor tries to build, a 3D model of the object. Instead it predicts, directly
as a function of the images, a point at which to grasp the object. Informally, the
algorithm takes two or more pictures of the object, and then tries to identify a point
within each 2-d image that corresponds to a good point at which to grasp the object.
(For example, if trying to grasp a coffee mug, it might try to identify the mid-point
of the handle.) Given these 2-d points in each image, we use triangulation to obtain
a 3D position at which to actually attempt the grasp. Thus, rather than trying to
triangulate every single point within each image in order to estimate depths—as in
dense stereo—we only attempt to triangulate one (or at most a small number of)
points corresponding to the 3D point where we will grasp the object. This allows
us to grasp an object without ever needing to obtain its full 3D shape, and applies
even to textureless, translucent or reflective objects on which standard stereo 3D
reconstruction fares poorly.

   We test our method on grasping a number of common office and household objects
such as toothbrushes, pens, books, cellphones, mugs, martini glasses, jugs, keys, knife-
cutters, duct-tape rolls, screwdrivers, staplers and markers. We also test our method
on the task of unloading novel items from dishwashers (see Figure 1.2.1a).

We then consider grasping with more general robot hands where one has to not only infer the "point" at which to grasp the object, but also address the high dof problem of choosing the positions of all the fingers in a multi-fingered hand. This work also focused on the problem of grasping novel objects, in the presence of significant amounts of *clutter*. (See Figure 1.2.1b.)

Our approach begins by computing a number of features of grasp quality, using both 2-d image and the 3D point cloud features. For example, the 3D data (obtained from 3D sensors such as stereo vision) is used to compute a number of grasp quality metrics, such as the degree to which the fingers are exerting forces normal to the surfaces of the object, and the degree to which they enclose the object. Using such features, we then apply a supervised learning algorithm to estimate the degree to which different configurations of the full arm and fingers reflect good grasps.

Our grasp planning algorithm that relied on long range vision sensors (such as cameras, Swiss Ranger), errors and uncertainty ranging from small deviations in the object's location to occluded surfaces significantly limited the reliability of these open-loop grasping strategies. Therefore, we also present new optical proximity sensors mounted on robot hand's fingertips and learning algorithms to enable the robot to reactively adjust the grasp. These sensors are a very useful complement to long-range sensors (such as vision and 3D cameras), and provide a sense of "pre-touch" to the robot, thus helping in the grasping performance.

We also present an application of our algorithms to opening new doors. Most prior work in door opening (e.g., [122, 110]) assumes that a detailed 3D model of the door (and door handle) is available, and focuses on developing the control actions required to open one specific door. In practice, a robot must rely on only its sensors to perform manipulation in a new environment. In our work, we present a learning algorithm to enable our robot to autonomously navigate anywhere in a building by opening doors, even those it has never seen before.

1.3   First published appearances of the described contributions

Most contributions described in this thesis have first appeared as various publications:

- Chapter 2: Saxena, Chung, and Ng [134, 135], Saxena, Sun, and Ng [149, 152, 151], Heitz, Gould, Saxena, and Koller [43]

- Chapter 3: Saxena, Schulte, and Ng [146], Saxena, Sun, and Ng [150], Saxena, Chung, and Ng [135], Saxena, Sun, and Ng [152]

- Chapter 4: Michels, Saxena, and Ng [88], Saxena, Chung, and Ng [135]

- Chapter 5: Saxena, Driemeyer, Kearns, Osondu, and Ng [138], Saxena, Driemeyer, Kearns, and Ng [136], Saxena, Wong, Quigley, and Ng [154], Saxena, Driemeyer, and Ng [141, 140],

- Chapter 6: Saxena, Wong, and Ng [153], Klingbeil, Saxena, and Ng [64], Hsiao, Nangeroni, Huber, Saxena, and Ng [51]

The following contributions have appeared as various publications: Saxena, Driemeyer, and Ng [141], Saxena and Ng [144], Saxena, Gupta, and Mukerjee [142], Saxena, Gupta, Gerasimov, and Ourselin [143], Saxena and Singh [147], Saxena, Srivatsan, Saxena, and Verma [148], Saxena, Ray, and Varma [145], Soundararaj, Sujeeth, and Saxena [165]. However, they are beyond the scope of this dissertation, and therefore are not discussed here.

Chapter 2

Single Image Depth Reconstruction

2.1 Introduction

Upon seeing an image such as Fig. 2.1a, a human has no difficulty in estimating the 3D depths in the scene (Fig. 2.1b). However, inferring such 3D depth remains extremely challenging for current computer vision systems. Indeed, in a narrow mathematical sense, it is impossible to recover 3D depth from a single image, since we can never know if it is a picture of a painting (in which case the depth is flat) or if it is a picture of an actual 3D environment. Yet in practice people perceive depth remarkably well given just one image; we would like our computers to have a similar sense of depths in a scene.

Recovering 3D depth from images has important applications in robotics, scene understanding and 3D reconstruction. Most work on visual 3D reconstruction has focused on binocular vision (stereopsis, i.e., methods using two images) [155] and on other algorithms that require multiple images, such as structure from motion [33] and depth from defocus [23]. These algorithms consider only the geometric (triangulation) differences. Beyond stereo/triangulation cues, there are also numerous *monocular* cues—such as texture variations and gradients, defocus, color/haze, etc.— that contain useful and important depth information. Even though humans perceive depth by seamlessly combining many of these stereo and monocular cues, most work on depth estimation has focused on stereo vision.
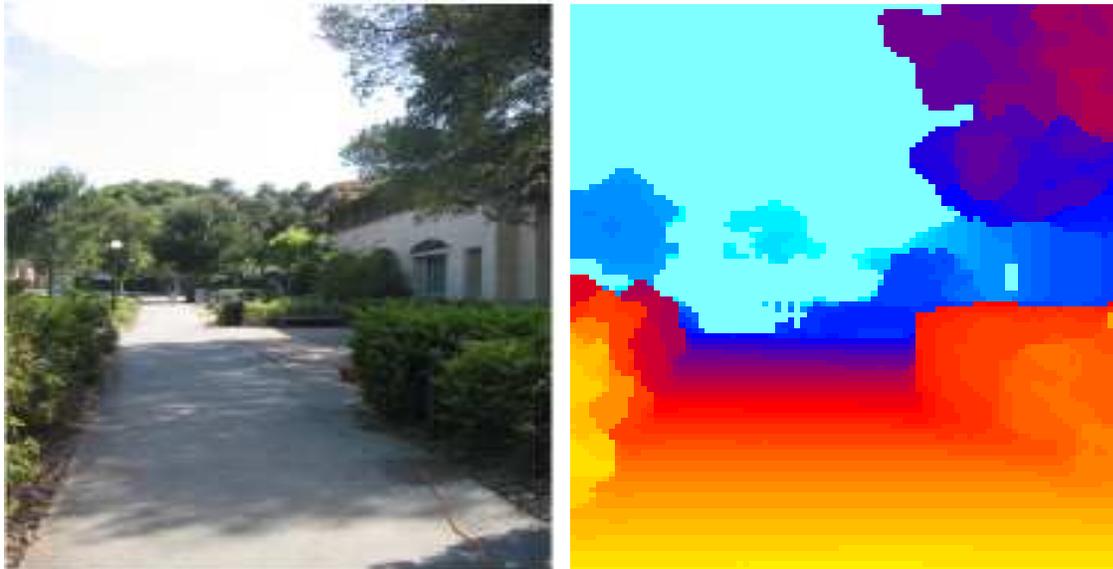
Figure 2.1: (a) A single still image, and (b) the corresponding (ground-truth) depthmap. Colors in the depthmap indicate estimated distances from the camera.

Depth estimation from a *single* still image is a difficult task, since depth typically remains ambiguous given only local image features. Thus, our algorithms must take into account the global structure of the image, as well as use prior knowledge about the scene. We also view depth estimation as a small but crucial step towards the larger goal of image understanding, in that it will help in tasks such as understanding the spatial layout of a scene, finding walkable areas in a scene, detecting objects, etc. We apply supervised learning to the problem of estimating depths (Fig. 2.1b) from a single still image (Fig. 2.1a) of a variety of unstructured environments, both indoor and outdoor, containing forests, sidewalks, buildings, people, bushes, etc.

Our approach is based on modeling depths and relationships between depths at multiple spatial scales using a hierarchical, multiscale Markov Random Field (MRF). Taking a supervised learning approach to the problem of depth estimation, we used a 3D scanner to collect training data, which comprised a large set of images and their corresponding ground-truth depths. (This data has been made publically available on the internet at: *http://make3d.stanford.edu*) Using this training set, we model the conditional distribution of the depth at each point in the image given the monocular

image features.

Next, we present another approach in which we use the insight that most 3D scenes can be segmented into many small, approximately planar surfaces.[1] Our algorithm begins by taking an image, and attempting to segment it into many such small planar surfaces. Using a segmentation algorithm, [32] we find an over-segmentation of the image that divides it into many small regions (superpixels). An example of such a segmentation is shown in Fig. 2.2b. Because we use an over-segmentation, planar surfaces in the world may be broken up into many superpixels; however, each superpixel is likely to (at least approximately) lie entirely on only one planar surface.

For each superpixel, our algorithm then tries to infer the 3D position and orientation of the 3D surface that it came from. This 3D surface is not restricted to just vertical and horizontal directions, but can be oriented in any direction. Inferring 3D position from a single image is non-trivial, and humans do it using many different visual depth cues, such as texture (e.g., grass has a very different texture when viewed close up than when viewed far away); color (e.g., green patches are more likely to be grass on the ground; blue patches are more likely to be sky). Our algorithm uses supervised learning to learn how different visual cues like these are associated with different depths. Our learning algorithm uses a Markov random field model, which is also able to take into account constraints on the relative depths of nearby superpixels. For example, it recognizes that two adjacent image patches are more likely to be at the same depth, or to be even co-planar, than being very far apart. Though learning in our MRF model is approximate, MAP inference is tractable via linear programming.

Having inferred the 3D position of each superpixel, we can now build a 3D mesh model of a scene (Fig. 2.2c). We then texture-map the original image onto it to build a textured 3D model (Fig. 2.2d) that we can fly through and view at different angles.

This chapter is organized as follows. Section 2.2 gives an overview of various methods used for 3D depth reconstruction. Section 2.3 describes some of the visual cues used by humans for depth perception, and Section 2.4 describes the image features

---

[1]Indeed, modern computer graphics using OpenGL or DirectX models extremely complex scenes this way, using triangular facets to model even very complex shapes.
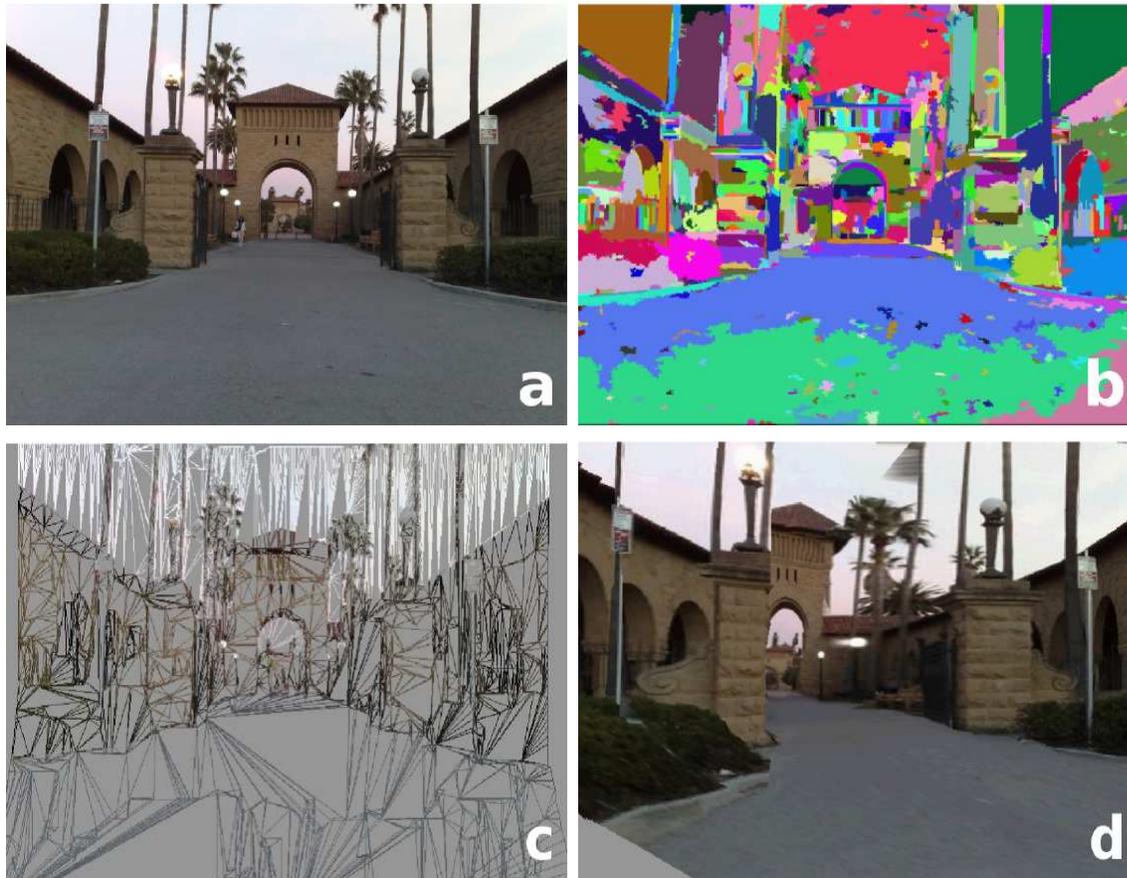
Figure 2.2: (a) An original image. (b) Oversegmentation of the image to obtain "superpixels". (c) The 3D model predicted by the algorithm. (d) A screenshot of the textured 3D model.

used to capture monocular cues. We describe our first probabilistic model, which models depths at each point in the image, in Section 2.5. We then present a second representation, based on planes, in Section 2.8, followed by its probabilistic model in Section 2.10. In Section 2.11, we describe our setup for collecting aligned image and laser data. The results of depth prediction on single images are presented in Section 2.12, 2.13 and 2.14. We then also describe how to incorporate object information in Section 2.15.

2.2 Related work

Although our work mainly focuses on depth estimation from a single still image, there are many other 3D reconstruction techniques, such as: explicit measurements with laser or radar sensors [124], using two (or more than two) images [155], and using video sequences [20]. Among the vision-based approaches, most work has focused on stereo vision (see [155] for a review), and on other algorithms that require multiple images, such as optical flow [3], structure from motion [33] and depth from defocus [23]. Frueh and Zakhor [35] constructed 3d city models by merging ground-based and airborne views. A large class of algorithms reconstruct the 3D shape of known objects, such as human bodies, from images and laser data [175, 2]. Structured lighting [156] offers another method for depth reconstruction

For a few specific settings, several authors have developed methods for depth estimation from a single image. Examples include shape-from-shading [186, 79] and shape-from-texture [81, 74]; however, these methods are difficult to apply to surfaces that do not have fairly uniform color and texture. Nagai et al. [98] used Hidden Markov Models to performing surface reconstruction from single images for known, fixed objects such as hands and faces. Hassner and Basri [41] used an example-based approach to estimate depth of an object from a known object class. Han and Zhu [40] performed 3D reconstruction for known specific classes of objects placed in untextured areas.

In work contemporary to ours, Delage, Lee and Ng (DLN) [25, 26] and Hoiem, Efros and Hebert (HEH) [46] assumed that the environment is made of a flat ground

with vertical walls. DLN considered indoor images, while HEH considered outdoor scenes. They classified the image into horizontal/ground and vertical regions (also possibly sky) to produce a simple "pop-up" type fly-through from an image. Their method, which assumes a simple "ground-vertical" structure of the world, fails on many environments that do not satisfy this assumption and also does not give accurate metric depthmaps.

There are some algorithms that can perform depth reconstruction from single images in very specific settings. Methods such as shape from shading [186, 79] and shape from texture [74, 81, 80] generally assume uniform color and/or texture,[2] and hence would perform very poorly on the complex, unconstrained, highly textured images that we consider. Hertzmann et al. [44] reconstructed high quality 3D models from several images, but they required that the images also contain "assistant" objects of known shapes next to the target object. Torresani et al. [178] worked on reconstructing non-rigid surface shapes from video sequences. Torralba and Oliva [177] studied the Fourier spectrum of the images to compute the mean depth of a scene.

Single view metrology [21] describes techniques that can be used to compute aspects of 3D geometry of a scene from a single perspective image. It assumes that vanishing lines and points are known in a scene, and calculates angles between parallel lines to infer 3D structure from images of Manhattan-type environments. (See Fig. 2.3.) On the other hand, our method is not restricted to Manhattan-type environments and considers images of a large variety of environments.

Building on these concepts of single image 3D reconstruction, Hoiem, Efros and Hebert [45] and Sudderth et al. [168] integrated learning-based object recognition with 3D scene representations.

Our approach draws on a large number of ideas from computer vision such as feature computation and multiscale representation of images. A variety of image features and representations have been used by other authors, such as Gabor filters [100], wavelets [167], SIFT features [96], etc. Many of these image features are

---

[2]Also, most of these algorithms assume Lambertian surfaces, which means the appearance of the surface does not change with viewpoint.
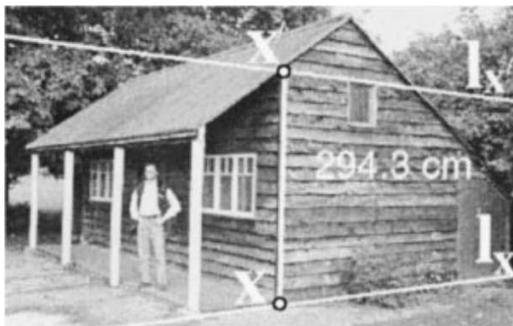
Figure 2.3: In Single view metrology [21], heights are measured using parallel lines. (Image reproduced with permission from [21].)

used for purposes such as recognizing objects [97, 160], faces [187], facial expressions [133], grasps [138]; image segmentation [68], computing the visual gist of a scene [103] and computing sparse representations of natural images [104]. Stereo and monocular image features have been used together for object recognition and image segmentation [67].

Beyond vision, machine learning has been successfully used for a number of other ill-posed problems. In [144], we considered the problem of estimating the incident angle of a sound, using a single microphone.

Our approach is based on learning a Markov Random Field (MRF) model. MRFs are a workhorse of machine learning, and have been successfully applied to numerous problems in which local features were insufficient and more contextual information had to be used. Examples include image denoising [92], stereo vision [155], image segmentation [32], range sensing [27], text segmentation [71], object classification [97], and image labeling [42]. For the application of identifying man-made structures in natural images, Kumar and Hebert used a discriminative random fields algorithm [70]. Since MRF learning is intractable in general, most of these models are trained using pseudo-likelihood; sometimes the models' parameters are also hand-tuned.

There is also ample prior work in 3D reconstruction from multiple images, as in stereo vision and structure from motion. It is impossible for us to do this literature justice here, but recent surveys include [155] and [78], and we discuss this work further in Section 3.3.

2.3   Monocular cues for depth perception

Images are formed by a projection of the 3D scene onto two dimensions. Thus, given only a single image, the true depths are ambiguous, in that an image might represent an infinite number of 3D structures. However, not all of these possible depths are equally likely. The environment we live in is reasonably structured, and thus humans are usually able to infer a (nearly) correct 3D structure, using prior experience.

Humans use numerous visual cues to perceive depth. Such cues are typically grouped into four distinct categories: monocular, stereo, motion parallax, and focus cues [76, 158]. Humans combine these cues to understand the 3D structure of the world [181, 121, 185, 76]. Below, we describe these cues in more detail. Our probabilistic model will attempt to capture a number of monocular cues (Section 2.5), as well as stereo triangulation cues (Section 3.2).

There are a number of monocular cues such as texture variations, texture gradients, interposition, occlusion, known object sizes, light and shading, haze, defocus, etc. [9] For example, many objects' texture will look different at different distances from the viewer. Texture gradients, which capture the distribution of the direction of edges, also help to indicate depth [80]. For example, a tiled floor with parallel lines will appear to have tilted lines in an image [21]. The distant patches will have larger variations in the line orientations, and nearby patches with almost parallel lines will have smaller variations in line orientations. Similarly, a grass field when viewed at different distances will have different texture gradient distributions. Haze is another depth cue, and is caused by atmospheric light scattering [99].

However, we note that local image cues alone are usually insufficient to infer the 3D structure. For example, both blue sky and a blue object would give similar local features; hence it is difficult to estimate depths from local features alone.

The ability of humans to "integrate information" over space, i.e. understand the relation between different parts of the image, is crucial to understanding the scene's 3D structure [180, chap. 11]. For example, even if part of an image is a homogeneous, featureless, gray patch, one is often able to infer its depth by looking at nearby portions of the image, so as to recognize whether this patch is part of a sidewalk,

Figure 2.4: The convolutional filters used for texture energies and gradients. The first nine are 3x3 Laws' masks. The last six are the oriented edge detectors spaced at 30^0 intervals. The nine Laws' masks are used to perform local averaging, edge detection and spot detection.

a wall, etc. Therefore, in our model we will also capture relations between different parts of the image.

Humans recognize many visual cues, such that a particular shape may be a building, that the sky is blue, that grass is green, that trees grow above the ground and have leaves on top of them, and so on. In our model, both the relation of monocular cues to the 3D structure, as well as relations between various parts of the image, will be learned using supervised learning. Specifically, our model will be trained to estimate depths using a training set in which the ground-truth depths were collected using a laser scanner.

2.4 Point-wise feature vector

Inspired in part by humans, we design features to capture some of the monocular cues discussed in Section 2.3.

In our first approach, we divide the image into small rectangular patches (arranged in a uniform grid), and estimate a single depth value for each patch.[3] Note that dividing an image into rectangular patches violates natural depth discontinuities, therefore in our second method we will use the idea of over-segmented patches (see Section 2.7). We use two types of features: *absolute* depth features—used to estimate the absolute depth at a particular patch—and *relative* features, which we use to estimate relative depths (magnitude of the difference in depth between two patches). These features try to capture two processes in the human visual system: local feature

---

[3]The resolution of the grid could be made arbitarily high and be made equal to the resolution of the image, thus we can depth value for each point in the image.

processing (absolute features), such as that the sky is far away; and continuity features (relative features), a process by which humans understand whether two adjacent patches are physically connected in 3D and thus have similar depths.[4]

We chose features that capture three types of local cues: texture variations, texture gradients, and color. Texture information is mostly contained within the image intensity channel [180],[5] so we apply Laws' masks [24, 88] to this channel to compute the texture energy (Fig. 2.4). Haze is reflected in the low frequency information in the color channels, and we capture this by applying a local averaging filter (the first Laws' mask) to the color channels.

Studies on monocular vision in humans strongly indicate that texture gradients are an important cue in depth estimation. [185] When well-defined edges exist (e.g., scenes having regular structure like buildings and roads, or indoor scenes), vanishing points can be calculated with a Hough transform to get a sense of distance. However, outdoor scenes are highly irregular. To compute an estimate of texture gradient that is robust to noise, we convolve the intensity channel with six oriented edge filters (shown in Fig. 2.4). We thus rely on a large number of different types of features to make our algorithm more robust and to make it generalize even to images that are very different from the training set.

One can envision including more features to capture other cues. For example, to model atmospheric effects such as fog and haze, features computed from the physics of light scattering [99] could also be included. Similarly, one can also include features based on surface-shading [79].

2.4.1   Features for absolute depth

We first compute summary statistics of a patch i in the image $I(x, y)$ as follows. We use the output of each of the 17 (9 Laws' masks, 2 color channels and 6 texture gradients) filters F_n, $n = 1, ..., 17$ as: $E_i(n) = \sum_{(x,y) \in \text{patch}(i)} |I * F_n|^k$, where $k \in$

---

[4]If two neighboring patches of an image display similar features, humans would often perceive them to be parts of the same object, and therefore to have similar depth values.

[5]We represent each image in YCbCr color space, where Y is the intensity channel, and Cb and Cr are the color channels.
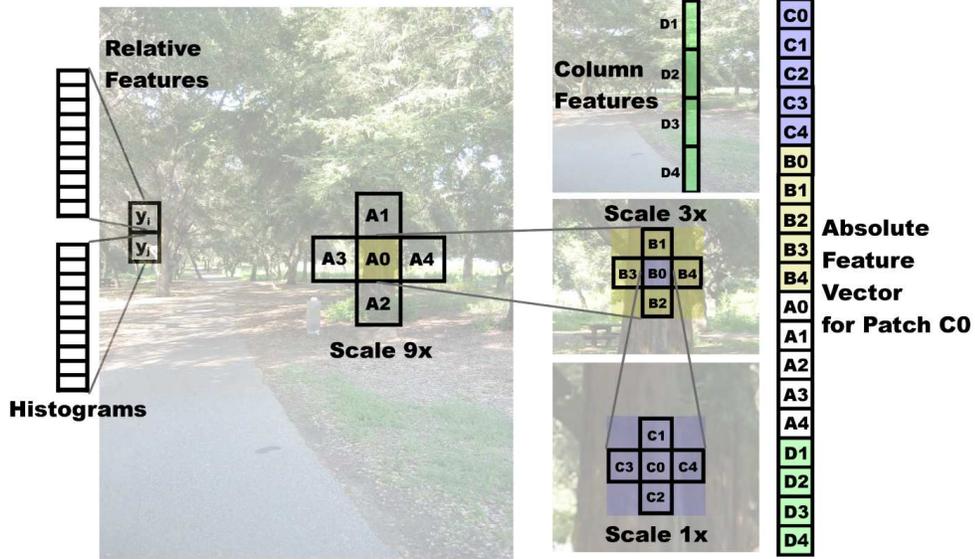
Figure 2.5: The absolute depth feature vector for a patch, which includes features from its immediate neighbors and its more distant neighbors (at larger scales). The relative depth features for each patch use histograms of the filter outputs.

$\{1, 2, 4\}$ give the sum absolute energy, sum squared energy and kurtosis respectively.[6] These are commonly used filters that capture the texture of a 3x3 patch and the edges at various orientations. This gives us an initial feature vector of dimension 34.

To estimate the absolute depth at a patch, local image features centered on the patch are insufficient, and one has to use more global properties of the image. We attempt to capture this information by using image features extracted at multiple spatial scales (image resolutions).[7] (See Fig. 2.5.) Objects at different depths exhibit very different behaviors at different resolutions, and using multiscale features allows us to capture these variations [182]. For example, blue sky may appear similar at different scales, but textured grass would not. In addition to capturing more global information, computing features at multiple spatial scales also helps to account for different relative sizes of objects. A closer object appears larger in the image, and

---

[6]We used $k \in \{1, 2\}$ for point-wise features, but used $k \in \{2, 4\}$ for segment-based features in Section 2.9.1 because they gave better results.

[7]The patches at each spatial scale are arranged in a grid of equally sized non-overlapping regions that cover the entire image. We use 3 scales in our experiments.

hence will be captured in the larger scale features. The same object when far away will be small and hence be captured in the small scale features. Features capturing the scale at which an object appears may therefore give strong indicators of depth.

To capture additional global features (e.g. occlusion relationships), the features used to predict the depth of a particular patch are computed from that patch as well as the four neighboring patches. This is repeated at each of the three scales, so that the feature vector at a patch includes features of its immediate neighbors, its neighbors at a larger spatial scale (thus capturing image features that are slightly further away in the image plane), and again its neighbors at an even larger spatial scale; this is illustrated in Fig. 2.5. Lastly, many structures (such as trees and buildings) found in outdoor scenes show vertical structure, in the sense that they are vertically connected to themselves (things cannot hang in empty air). Thus, we also add to the features of a patch additional summary features of the column it lies in.

For each patch, after including features from itself and its 4 neighbors at 3 scales, and summary features for its 4 column patches, our absolute depth feature vector x is $19 * 34 = 646$ dimensional.

2.4.2   Features for boundaries

We use a different feature vector to learn the dependencies between two neighboring patches. Specifically, we compute a 10-bin histogram of each of the 17 filter outputs $|I * F_n|$, giving us a total of 170 features y_{is} for each patch i at scale s. These features are used to estimate how the depths at two different locations are related. We believe that learning these estimates requires less global information than predicting absolute depth, but more detail from the individual patches. For example, given two adjacent patches of a distinctive, unique, color and texture, we may be able to safely conclude that they are part of the same object, and thus that their depths are close, even without more global features. Hence, our relative depth features y_{ijs} for two neighboring patches i and j at scale s will be the differences between their histograms, i.e., $y_{ijs} = y_{is} - y_{js}$.
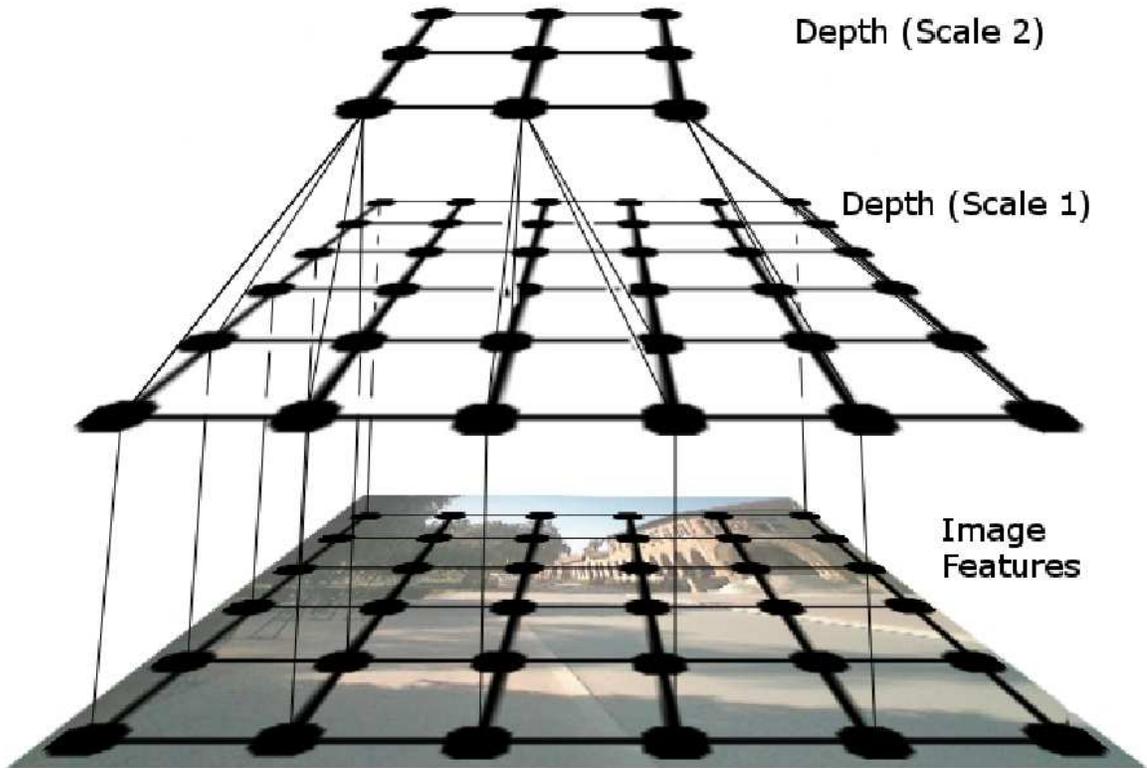
Figure 2.6: The multiscale MRF model for modeling relation between features and depths, relation between depths at same scale, and relation between depths at different scales. (Only 2 out of 3 scales, and a subset of the edges, are shown.)

2.5 Point-wise probabilistic model

In this model, we use points in the image as basic unit, and infer only their 3D location. The nodes in this MRF are a dense grid of points in the image, where the value of each node represents its depth. We convert the depths to log scale for emphasizing fractional (relative) errors in depth.

Since local images features are by themselves usually insufficient for estimating depth, the model needs to reason more globally about the spatial structure of the scene. We capture the spatial structure of the image by modeling the relationships between depths in different parts of the image. Although the depth of a particular patch depends on the features of the patch, it is also related to the depths of other

parts of the image. For example, the depths of two adjacent patches lying in the same building will be highly correlated. We will use a hierarchical multiscale Markov Random Field (MRF) to model the relationship between the depth of a patch and the depths of its neighboring patches (Fig. 2.6). In addition to the interactions with the immediately neighboring patches, there are sometimes also strong interactions between the depths of patches which are not immediate neighbors. For example, consider the depths of patches that lie on a large building. All of these patches will be at similar depths, even if there are small discontinuities (such as a window on the wall of a building). However, when viewed at the smallest scale, some adjacent patches are difficult to recognize as parts of the same object. Thus, we will also model interactions between depths at multiple spatial scales.

2.5.1   Gaussian model

Our first model will be a jointly Gaussian Markov Random Field (MRF) as shown in Eq. 2.1. To capture the multiscale depth relations, we will model the depths $d_i(s)$ for multiple scales $s = 1, 2, 3$. In our experiments, we enforce a hard constraint that depths at a higher scale are the average of the depths at the lower scale.[8] More formally, we define $d_i(s+1) = (1/5) \sum_{j \in N_s(i) \cup \{i\}} d_j(s)$. Here, $N_s(i)$ are the 4 neighbors of patch i at scale s.[9]

In Eq. 2.1, M is the total number of patches in the image (at the lowest scale); Z is the normalization constant for the model; x_i is the absolute depth feature vector for patch i; and θ and $\sigma = \{\sigma_1, \sigma_2\}$ are parameters of the model.[10] In detail, we use different parameters $(\theta_r, \sigma_{1r}, \sigma_{2r})$ for each row r in the image, because the images we consider are taken from a horizontally mounted camera, and thus different rows of the image have different statistical properties. For example, a blue patch might

---

[8]One can instead have soft constraints relating the depths at higher scale to depths at lower scale. One can also envision putting more constraints in the MRF, such as that points lying on a long straight edge in an image should lie on a straight line in the 3D model, etc.

[9]Our experiments using 8-connected neighbors instead of 4-connected neighbors yielded minor improvements in accuracy at the cost of a much longer inference time.

[10]Here each σ^2 represents the (empirical) variance of each term, and θ are the weights parameterizing the space of linear functions.

represent sky if it is in upper part of image, and might be more likely to be water if in the lower part of the image.

$$P_G(d|X;\theta,\sigma) = \frac{1}{Z_G} \exp\left(-\sum_{i=1}^{M} \frac{(d_i(1) - x_i^T\theta_r)^2}{2\sigma_{1r}^2} - \sum_{s=1}^{3}\sum_{i=1}^{M}\sum_{j\in N_s(i)} \frac{(d_i(s) - d_j(s))^2}{2\sigma_{2rs}^2}\right)$$
(2.1)

$$P_L(d|X;\theta,\lambda) = \frac{1}{Z_L} \exp\left(-\sum_{i=1}^{M} \frac{|d_i(1) - x_i^T\theta_r|}{\lambda_{1r}} - \sum_{s=1}^{3}\sum_{i=1}^{M}\sum_{j\in N_s(i)} \frac{|d_i(s) - d_j(s)|}{\lambda_{2rs}}\right)$$
(2.2)

Our model is a conditionally trained MRF, in that its model of the depths d is always conditioned on the image features X; i.e., it models only $P(d|X)$. We first estimate the parameters θ_r in Eq. 2.1 by maximizing the conditional log likelihood $\ell(d) = \log P(d|X;\theta_r)$ of the training data. Since the model is a multivariate Gaussian, the maximum likelihood estimate of parameters θ_r is obtained by solving a linear least squares problem.

The first term in the exponent above models depth as a function of multiscale features of a single patch i. The second term in the exponent places a soft "constraint" on the depths to be smooth. If the variance term σ_{2rs}^2 is a fixed constant, the effect of this term is that it tends to smooth depth estimates across nearby patches. However, in practice the dependencies between patches are not the same everywhere, and our expected value for $(d_i - d_j)^2$ may depend on the features of the local patches.

Therefore, to improve accuracy we extend the model to capture the "variance" term σ_{2rs}^2 in the denominator of the second term as a linear function of the patches i and j's relative depth features y_{ijs} (discussed in Section 2.4.2). We model the variance as $\sigma_{2rs}^2 = u_{rs}^T|y_{ijs}|$. This helps determine which neighboring patches are likely to have similar depths; for example, the "smoothing" effect is much stronger if neighboring patches are similar. This idea is applied at multiple scales, so that we learn different σ_{2rs}^2 for the different scales s (and rows r of the image). The parameters u_{rs} are learned to fit σ_{2rs}^2 to the expected value of $(d_i(s) - d_j(s))^2$, with a constraint that

$u_{rs} \geq 0$ (to keep the estimated σ^2_{2rs} non-negative), using a quadratic program (QP).

Similar to our discussion on σ^2_{2rs}, we also learn the variance parameter $\sigma^2_{1r} = v^T_r x_i$ as a linear function of the features. Since the absolute depth features x_i are non-negative, the estimated σ^2_{1r} is also non-negative. The parameters v_r are chosen to fit σ^2_{1r} to the expected value of $(d_i(r) - \theta^T_r x_i)^2$, subject to $v_r \geq 0$. This σ^2_{1r} term gives a measure of the uncertainty in the first term, and depends on the features. This is motivated by the observation that in some cases, depth cannot be reliably estimated from the local features. In this case, one has to rely more on neighboring patches' depths, as modeled by the second term in the exponent.

After learning the parameters, given a new test-set image we can find the MAP estimate of the depths by maximizing Eq. 2.1 in terms of d. Since Eq. 2.1 is Gaussian, $\log P(d|X; \theta, \sigma)$ is quadratic in d, and thus its maximum is easily found in closed form (taking at most 1-2 seconds per image). More details are given in Appendix 2.6.1.

2.5.2   Laplacian model

We now present a second model (Eq. 2.2) that uses Laplacians instead of Gaussians to model the posterior distribution of the depths. Our motivation for doing so is three-fold. First, a histogram of the relative depths $(d_i - d_j)$ is empirically closer to Laplacian than Gaussian (Fig. 2.7, see [52] for more details on depth statistics), which strongly suggests that it is better modeled as one.[11] Second, the Laplacian distribution has heavier tails, and is therefore more robust to outliers in the image features and to errors in the training-set depthmaps (collected with a laser scanner; see Section 2.11). Third, the Gaussian model was generally unable to give depthmaps with sharp edges; in contrast, Laplacians tend to model sharp transitions/outliers better [129].

This model is parametrized by θ_r (similar to Eq. 2.1) and by λ_{1r} and λ_{2rs}, the *Laplacian spread* parameters. Maximum-likelihood parameter estimation for the

---

[11]Although the Laplacian distribution fits the log-histogram of multiscale relative depths reasonably well, there is an unmodeled peak near zero. This arises due to the fact that the neighboring depths at the finest scale frequently lie on the same object.
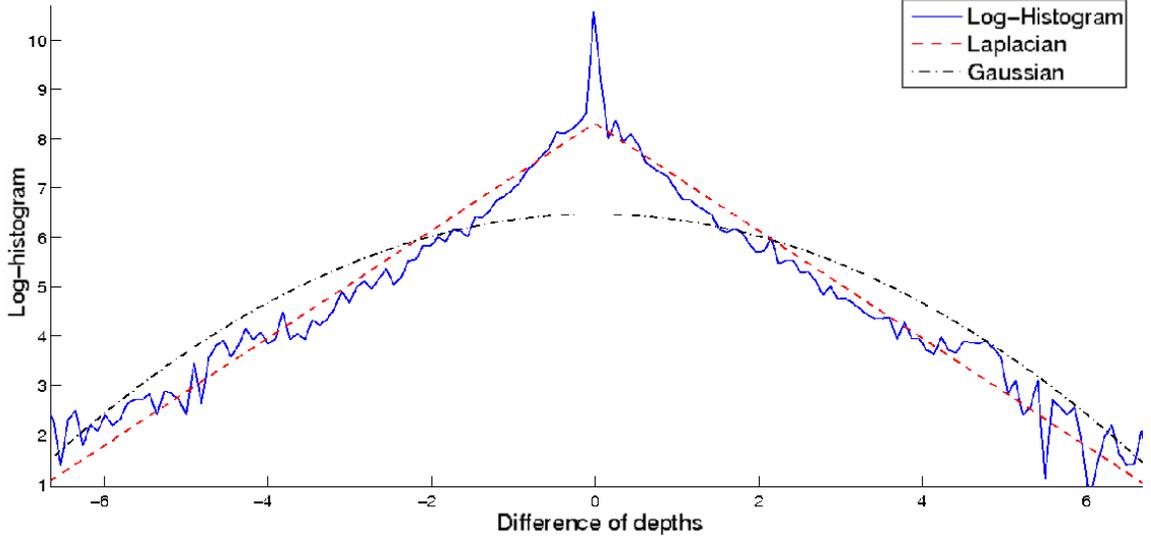
Figure 2.7: The log-histogram of relative depths. Empirically, the distribution of relative depths is closer to Laplacian than Gaussian.

Laplacian model is not tractable (since the partition function depends on θ_r). However, by analogy to the Gaussian case, we approximate this by solving a linear system of equations $X_r\theta_r \approx d_r$ to minimize L_1 (instead of L_2) error, i.e., $\min_{\theta_r} ||d_r - X_r\theta_r||_1$. Here X_r is the matrix of absolute depth features. Following the Gaussian model, we also learn the Laplacian spread parameters in the denominator in the same way, except that the instead of estimating the expected values of $(d_i - d_j)^2$ and $(d_i(r) - \theta_r^T x_i)^2$, we estimate the expected values of $|d_i - d_j|$ and $|d_i(r) - \theta_r^T x_i|$, as a linear function of u_{rs} and v_r respectively. This is done using a Linear Program (LP), with $u_{rs} \geq 0$ and $v_r \geq 0$.

Even though maximum likelihood (ML) parameter estimation for θ_r is intractable in the Laplacian model, given a new test-set image, MAP inference for the depths d is tractable and convex. Details on solving the inference problem as a Linear Program (LP) are given in Appendix 2.6.2.

Remark. We can also extend these models to combine Gaussian and Laplacian terms in the exponent, for example by using a L_2 norm term for absolute depth, and a L_1 norm term for the interaction terms. MAP inference remains tractable in this setting,

and can be solved using convex optimization as a QP (quadratic program).

2.6 MAP inference

2.6.1 Gaussian model

We can rewrite Eq. 2.1 as a standard multivariate Gaussian,

$$P_G(d|X;\theta,\sigma) = \frac{1}{Z_G} \exp\left(-\frac{1}{2}(d - X_a\theta_r)^T \Sigma_a^{-1}(d - X_a\theta_r)\right)$$

where $X_a = (\Sigma_1^{-1} + Q^T\Sigma_2^{-1}Q)^{-1}\Sigma_1^{-1}X$, with Σ_1 and Σ_2 representing the matrices of the variances $\sigma_{1,i}^2$ and $\sigma_{2,i}^2$ in the first and second terms in the exponent of Eq. 2.1 respectively.[12] Q is a matrix such that rows of Qd give the differences of the depths in the neighboring patches at multiple scales (as in the second term in the exponent of Eq. 2.1). Our MAP estimate of the depths is, therefore, $d^* = X_a\theta_r$.

During learning, we iterate between learning θ and estimating σ. Empirically, $\sigma_1 \ll \sigma_2$, and X_a is very close to X; therefore, the algorithm converges after 2-3 iterations.

2.6.2 Laplacian model

Exact MAP inference of the depths $d \in \mathbb{R}^M$ can be obtained by maximizing $\log P(d|X;\theta,\lambda)$ in terms of d (Eq. 2.2). More formally,

$$d^* = \arg\max_d \log P(d|X;\theta,\lambda)$$
$$= \arg\min_d c_1^T|d - X\theta_r| + c_2^T|Qd|$$

where, $c_1 \in \mathbb{R}^M$ with $c_{1,i} = 1/\lambda_{1,i}$, and $c_2 \in \mathbb{R}^{6M}$ with $c_{2,i} = 1/\lambda_{2,i}$. Our features are given by $X \in \mathbb{R}^{M\times k}$ and the learned parameters are $\theta_r \in \mathbb{R}^k$, which give a naive

---

[12]Note that if the variances at each point in the image are constant, then $X_a = (I + \sigma_1^2/\sigma_2^2 Q^T Q)^{-1}X$. I.e., X_a is essentially a smoothed version of X.

estimate $\tilde{d} = X\theta_r \in \mathbb{R}^M$ of the depths. Q is a matrix such that rows of Qd give the differences of the depths in the neighboring patches at multiple scales (as in the second term in the exponent of Eq. 2.2).

We add auxiliary variables ξ_1 and ξ_2 to pose the problem as a Linear Program (LP):

$$d^* = \arg\min_{d,\xi_1,\xi_2} \qquad\qquad c_1^T\xi_1 + c_2^T\xi_2$$

$$\text{s.t.} \qquad\qquad -\xi_1 \le d - \tilde{d} \le \xi_1$$

$$-\xi_2 \le Qd \le \xi_2$$

In our experiments, MAP inference takes about 7-8 seconds for an image.

2.7 Plane parametrization

In our first method, our patches were uniform in size and were arranged in a uniform grid. However, often the structures in the image can be grouped into meaningful regions (or segments). For example, all the patches on a large uniformly colored wall would lie on the same plane and sometimes some regions consists of many planes at different angles (such as trees). Therefore, we use the insight that most 3D scenes can be segmented into many small, approximately planar surfaces (patches). Our method starts by taking an image, and attempting to segment it into many such small planar surfaces (see 2.8 for details). Note that in this case the patches are variable in size and are *not* arranged into a uniform grid. (See Fig. 2.8.) Each of these small patches will be our basic unit of representation (instead of using a uniform rectangular patch earlier). Such representation follows natural depth discontinuities better, and hence models the image better for the task of depth estimation.

In this method, other than assuming that the 3D structure is made up of a number of small planes, we make no explicit assumptions about the structure of the scene. This allows our approach to even apply to scenes with significantly richer structure than only vertical surfaces standing on a horizontal ground, such as mountains, trees,
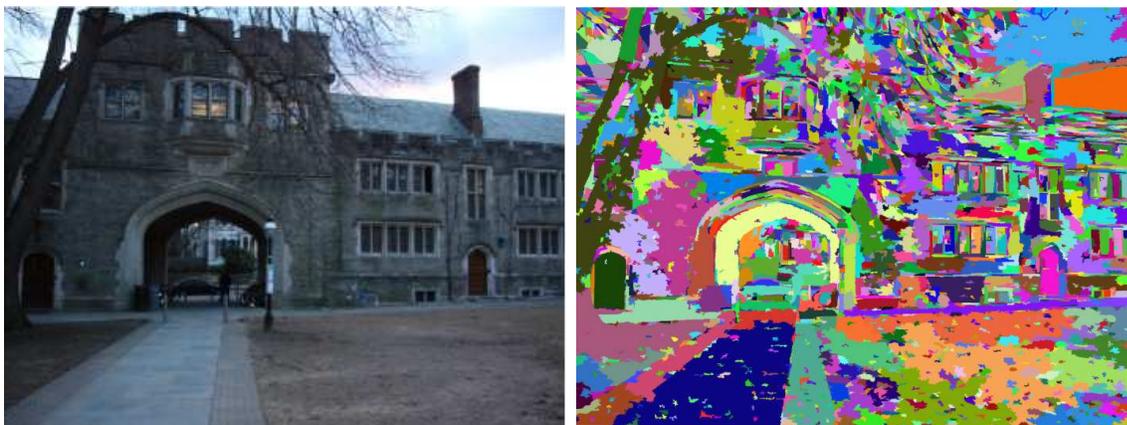
Figure 2.8: (Left) An image of a scene. (Right) Oversegmented image. Each small segment (superpixel) lies on a plane in the 3d world. (*Best viewed in color.*)

etc.

2.8   Representation

Our goal is to create a full photo-realistic 3D model from an image. Following most work on 3D models in computer graphics and other related fields, we will use a polygonal mesh representation of the 3D model, in which we assume the world is made of a set of small planes.[13] In detail, given an image of the scene, we first find small homogeneous regions in the image, called "Superpixels" [32]. Each such region represents a coherent region in the scene with all the pixels having similar properties. (See Fig. 2.8.) Our basic unit of representation will be these small planes in the world, and our goal is to infer the location and orientation of each one.

More formally, we parametrize both the 3D location and orientation of the infinite plane on which a superpixel lies by using a set of plane parameters $\alpha \in \mathbb{R}^3$. (Fig. 2.9) (Any point $q \in \mathbb{R}^3$ lying on the plane with parameters α satisfies $\alpha^T q = 1$.) The value $1/|\alpha|$ is the distance from the camera center to the closest point on the plane, and

---

[13]This assumption is reasonably accurate for most artificial structures, such as buildings. Some natural structures such as trees could perhaps be better represented by a cylinder. However, since our models are quite detailed, e.g., about 2000 planes for a small scene, the planar assumption works quite well in practice.
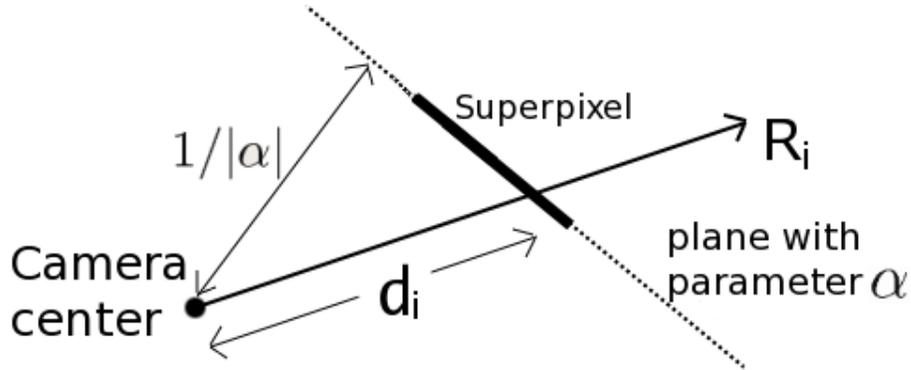
Figure 2.9: A 2-d illustration to explain the plane parameter α and rays R from the camera.

the normal vector $\hat{\alpha} = \frac{\alpha}{|\alpha|}$ gives the orientation of the plane. If R_i is the unit vector (also called the ray R_i) from the camera center to a point i lying on a plane with parameters α, then $d_i = 1/R_i^T\alpha$ is the distance of point i from the camera center.

2.9 Segment-wise features

In plane-parameter representation, our basic unit of representation is a superpixel (a planar segment). Therefore, we compute features for a irregular segment S_i (compared to a rectangular patch in Section 2.4).

For each superpixel, we compute a battery of features to capture some of the monocular cues discussed in Section 2.3. We also compute features to predict meaningful boundaries in the images, such as occlusion and folds.

2.9.1 Monocular image features

To compute the segment-based features, we sum the filter outputs over the pixels in superpixel S_i as $E_i(n) = \sum_{(x,y)\in S_i} |I * F_n|^k$. (Note in point-wise representation we did the summation over a uniform rectangular patch instead of a irregular superpixel.)

The shape of the segment gives information about depth as well. For example, a big superpixel is more likely to be part of a wall vs a short irregular superpixel

that is more likely to be a irregular object, such as part of a tree. Our superpixel shape and location based features (14, computed only for the superpixel) included the shape and location based features in Section 2.2 of [46], and also the eccentricity of the superpixel. (See Fig. 2.15.)

We attempt to capture more "contextual" information by also including features from neighboring superpixels (we pick the largest four in our experiments), and at multiple spatial scales (three in our experiments). The features, therefore, contain information from a larger portion of the image, and thus are more expressive than just local features. This makes the feature vector x_i of a superpixel $34*(4+1)*3+14 = 524$ dimensional.

2.9.2   Features for boundaries

Another strong cue for 3D structure perception is boundary information. If two neighboring superpixels of an image display different features, humans would often perceive them to be parts of different objects; therefore an edge between two superpixels with distinctly different features, is a candidate for a occlusion boundary or a fold. To compute the features ϵ_{ij} between superpixels i and j, we first generate 14 different segmentations for each image for 2 different scales for 7 different properties based on textures, color, and edges. We found that features based on multiple segmentations are better for indicating boundaries. The groupings (segments) produced by over-segmentation algorithm were empirically found to be better than comparing histograms in Section 2.4.2. We modified [32] to create segmentations based on these properties. Each element of our 14 dimensional feature vector ϵ_{ij} is then an indicator if two superpixels i and j lie in the same segmentation. For example, if two superpixels belong to the same segments in all the 14 segmentations then it is more likely that they are coplanar or connected. Relying on multiple segmentation hypotheses instead of one makes the detection of boundaries more robust. The features ϵ_{ij} are the input to the classifier for the occlusion boundaries and folds.

Figure 2.10: (Left) Original image. (Right) Superpixels overlaid with an illustration of the Markov Random Field (MRF). The MRF models the relations (shown by the edges) between neighboring superpixels. (Only a subset of nodes and edges shown.)

2.10 Probabilistic model

It is difficult to infer 3D information of a region from local cues alone (see Section 2.3), and one needs to infer the 3D information of a region in relation to the 3D information of other regions.

In our MRF model, we try to capture the following properties of the images:

- **Image Features and depth**: The image features of a superpixel bear some relation to the depth (and orientation) of the superpixel.
- **Connected structure**: Except in case of occlusion, neighboring superpixels are more likely to be connected to each other.
- **Co-planar structure**: Neighboring superpixels are more likely to belong to the same plane, if they have similar features and if there are no edges between them.
- **Co-linearity**: Long straight lines in the image plane are more likely to be

straight lines in the 3D model. For example, edges of buildings, sidewalk, windows.

Note that no single one of these four properties is enough, by itself, to predict the 3D structure. For example, in some cases, local image features are not strong indicators of the depth (and orientation) (e.g., a patch on a blank feature-less wall). Thus, our approach will combine these properties in an MRF, in a way that depends on our "confidence" in each of these properties. Here, the "confidence" is itself estimated from local image cues, and will vary from region to region in the image.

Our MRF is composed of five types of nodes. The input to the MRF occurs through two variables, labeled x and ϵ. These variables correspond to features computed from the image pixels (see Section 2.9 for details.) and are always observed; thus the MRF is conditioned on these variables. The variables ν indicate our degree of confidence in a depth estimate obtained only from local image features. The variables y indicate the presence or absence of occlusion boundaries and folds in the image. These variables are used to selectively enforce coplanarity and connectivity between superpixels. Finally, the variables α are the plane parameters that are inferred using the MRF, which we call "Plane Parameter MRF."[14]

Occlusion Boundaries and Folds: We use the variables $y_{ij} \in \{0, 1\}$ to indicate whether an "edgel" (the edge between two neighboring superpixels) is an occlusion boundary/fold or not. The inference of these boundaries is typically not completely accurate; therefore we will infer *soft* values for y_{ij}. (See Fig. 2.11.) More formally, for an edgel between two superpixels i and j, $y_{ij} = 0$ indicates an occlusion boundary/fold, and $y_{ij} = 1$ indicates none (i.e., a planar surface).

In many cases, strong image gradients do not correspond to the occlusion boundary/fold, e.g., a shadow of a building falling on a ground surface may create an edge between the part with a shadow and the one without. An edge detector that relies just on these local image gradients would mistakenly produce an edge. However, there are other visual cues beyond local image gradients that better indicate whether two planes are connected/coplanar or not. Using learning to combine a number of such

---

[14]For comparison, we also present an MRF that only models the 3D location of the points in the image ("Point-wise MRF," see Appendix).
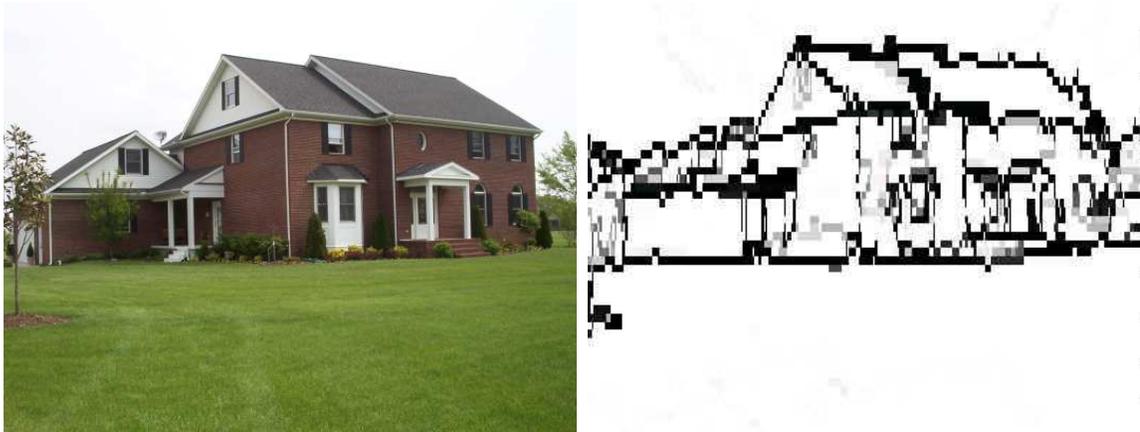
Figure 2.11: (Left) An image of a scene. (Right) Inferred "soft" values of $y_{ij} \in [0, 1]$. ($y_{ij} = 0$ indicates an occlusion boundary/fold, and is shown in black.) Note that even with the inferred y_{ij} being not completely accurate, the plane parameter MRF will be able to infer "correct" 3D models.

visual features makes the inference more accurate. In [84], Martin, Fowlkes and Malik used local brightness, color and texture for learning segmentation boundaries. Here, our goal is to learn occlusion boundaries and folds. In detail, we model y_{ij} using a logistic response as $P(y_{ij} = 1|\epsilon_{ij}; \psi) = 1/(1 + \exp(-\psi^T \epsilon_{ij}))$. where, ϵ_{ij} are features of the superpixels i and j (Section 2.9.2), and ψ are the parameters of the model. During inference, we will use a mean field-like approximation, where we replace y_{ij} with its mean value under the logistic model.

Now, we will describe how we model the distribution of the plane parameters α, conditioned on y.

Fractional depth error: For 3D reconstruction, the fractional (or relative) error in depths is most meaningful; it is used in structure for motion, stereo reconstruction, etc. [65, 155] For ground-truth depth d, and estimated depth $\hat{d}$, fractional error is defined as $(\hat{d} - d)/d = \hat{d}/d - 1$. Therefore, we will be penalizing fractional errors in our MRF.

MRF Model: To capture the relation between the plane parameters and the image features, and other properties such as co-planarity, connectedness and co-linearity, we

formulate our MRF as

$$P(\alpha|X, \nu, y, R; \theta) = \frac{1}{Z} \prod_i f_1(\alpha_i|X_i, \nu_i, R_i; \theta)$$

$$\prod_{i,j} f_2(\alpha_i, \alpha_j|y_{ij}, R_i, R_j) \qquad (2.3)$$

where, α_i is the plane parameter of the superpixel i. For a total of S_i points in the superpixel i, we use x_{i,s_i} to denote the features for point s_i in the superpixel i. $X_i = \{x_{i,s_i} \in \mathbb{R}^{524} : s_i = 1, ..., S_i\}$ are the features for the superpixel i. (Section 2.9.1) Similarly, $R_i = \{R_{i,s_i} : s_i = 1, ..., S_i\}$ is the set of rays for superpixel i.[15] ν is the "confidence" in how good the (local) image features are in predicting depth (more details later).
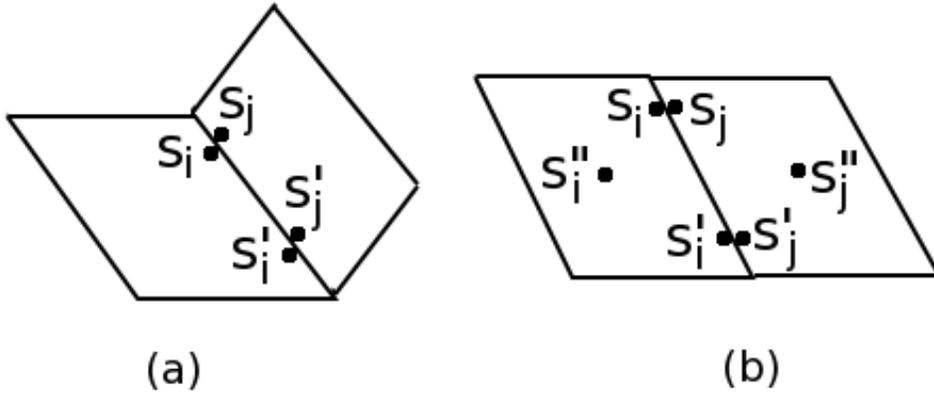
Figure 2.12: Illustration explaining effect of the choice of s_i and s_j on enforcing (a) Connected structure and (b) Co-planarity.

The first term $f_1(\cdot)$ models the plane parameters as a function of the image features x_{i,s_i}. We have $R_{i,s_i}^T \alpha_i = 1/d_{i,s_i}$ (where R_{i,s_i} is the ray that connects the camera to the 3D location of point s_i), and if the estimated depth $\hat{d}_{i,s_i} = x_{i,s_i}^T \theta_r$, then the fractional

---

[15]The rays are obtained by making a reasonable guess on the camera intrinsic parameters—that the image center is the origin and the pixel-aspect-ratio is one—unless known otherwise from the image headers.

error would be

$$\frac{\hat{d}_{i,s_i} - d_{i,s_i}}{d_{i,s_i}} = \frac{1}{d_{i,s_i}}(\hat{d}_{i,s_i}) - 1 = R_{i,s_i}^T \alpha_i(x_{i,s_i}^T \theta_r) - 1$$

Therefore, to minimize the aggregate fractional error over all the points in the superpixel, we model the relation between the plane parameters and the image features as

$$f_1(\alpha_i|X_i, \nu_i, R_i; \theta) = \exp\left(-\sum_{s_i=1}^{S_i} \nu_{i,s_i} \left| R_{i,s_i}^T \alpha_i(x_{i,s_i}^T \theta_r) - 1 \right|\right) \qquad (2.4)$$

The parameters of this model are $\theta_r \in \mathbb{R}^{524}$. We use different parameters (θ_r) for rows $r = 1, ..., 11$ in the image, because the images we consider are roughly aligned upwards (i.e., the direction of gravity is roughly downwards in the image), and thus it allows our algorithm to learn some regularities in the images—that different rows of the image have different statistical properties. E.g., a blue superpixel might be more likely to be sky if it is in the upper part of image, or water if it is in the lower part of the image, or that in the images of environments available on the internet, the horizon is more likely to be in the middle one-third of the image. (In our experiments, we obtained very similar results using a number of rows ranging from 5 to 55.) Here, $\nu_i = \{\nu_{i,s_i} : s_i = 1, ..., S_i\}$ indicates the confidence of the features in predicting the depth $\hat{d}_{i,s_i}$ at point s_i.[16] If the local image features were not strong enough to predict depth for point s_i, then $\nu_{i,s_i} = 0$ turns off the effect of the term $\left| R_{i,s_i}^T \alpha_i(x_{i,s_i}^T \theta_r) - 1 \right|$.

The second term $f_2(\cdot)$ models the relation between the plane parameters of two superpixels i and j. It uses pairs of points s_i and s_j to do so:

$$f_2(\cdot) = \prod_{\{s_i, s_j\} \in N} h_{s_i, s_j}(\cdot) \qquad (2.5)$$

We will capture co-planarity, connectedness and co-linearity, by different choices of $h(\cdot)$ and $\{s_i, s_j\}$.

---

[16]The variable ν_{i,s_i} is an indicator of how good the image features are in predicting depth for point s_i in superpixel i. We learn ν_{i,s_i} from the monocular image features, by estimating the expected value of $|d_i - x_i^T \theta_r|/d_i$ as $\phi_r^T x_i$ with logistic response, with ϕ_r as the parameters of the model, features x_i and d_i as ground-truth depths.
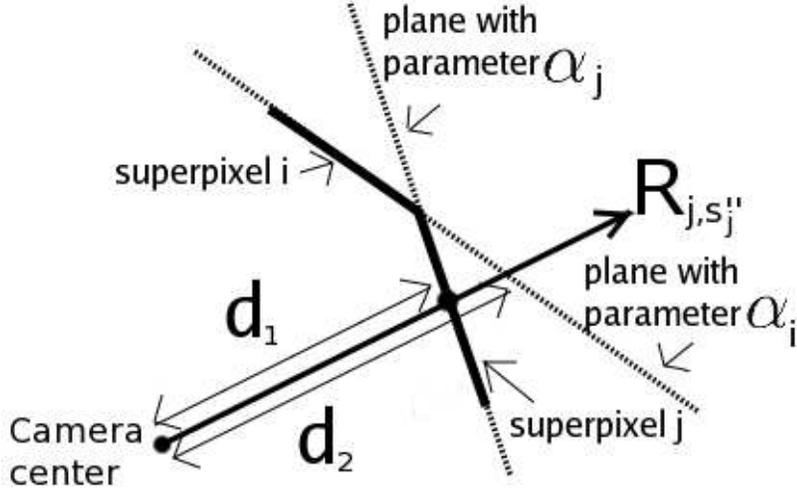
Figure 2.13: A 2-d illustration to explain the co-planarity term. The distance of the point s_j on superpixel j to the plane on which superpixel i lies along the ray $R_{j,s_{j''}}$ is given by $d_1 - d_2$.

Connected structure: We enforce this constraint by choosing s_i and s_j to be on the boundary of the superpixels i and j. As shown in Fig. 2.12a, penalizing the distance between two such points ensures that they remain fully connected. The relative (fractional) distance between points s_i and s_j is penalized by

$$h_{s_i,s_j}(\alpha_i, \alpha_j, y_{ij}, R_i, R_j) = \exp\left(-y_{ij}|(R_{i,s_i}^T \alpha_i - R_{j,s_j}^T \alpha_j)\hat{d}|\right) \qquad (2.6)$$

In detail, $R_{i,s_i}^T \alpha_i = 1/d_{i,s_i}$ and $R_{j,s_j}^T \alpha_j = 1/d_{j,s_j}$; therefore, the term $(R_{i,s_i}^T \alpha_i - R_{j,s_j}^T \alpha_j)\hat{d}$ gives the fractional distance $|(d_{i,s_i} - d_{j,s_j})/\sqrt{d_{i,s_i}d_{j,s_j}}|$ for $\hat{d} = \sqrt{\hat{d}_{s_i}\hat{d}_{s_j}}$. Note that in case of occlusion, the variables $y_{ij} = 0$, and hence the two superpixels will not be forced to be connected.

Co-planarity: We enforce the co-planar structure by choosing a third pair of points s_i'' and s_j'' in the center of each superpixel along with ones on the boundary. (Fig. 2.12b) To enforce co-planarity, we penalize the relative (fractional) distance of point s_j'' from

(a) 2-d image
(b) 3D world, top view
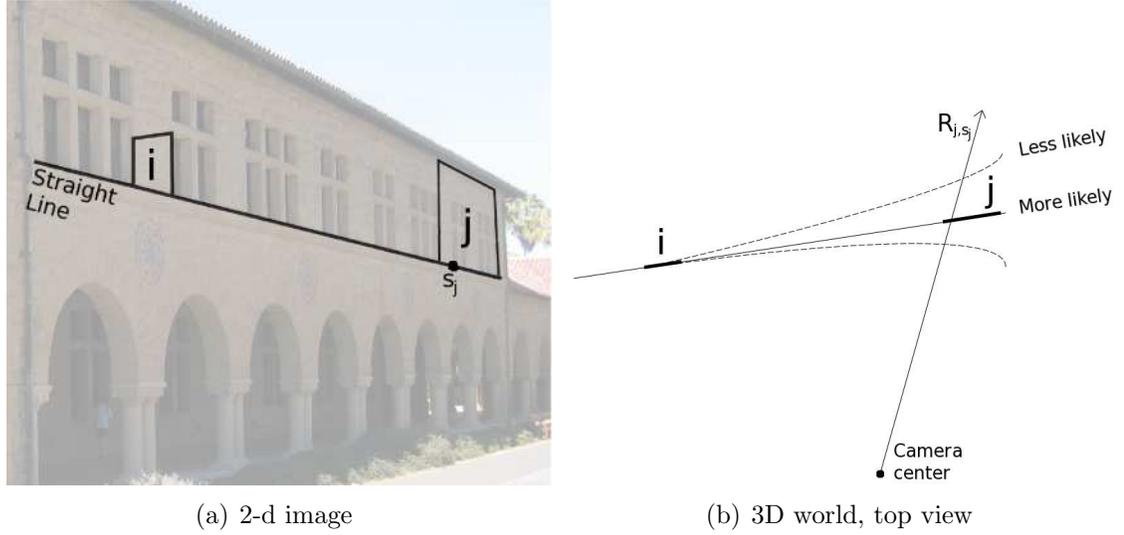
Figure 2.14: Co-linearity. (a) Two superpixels i and j lying on a straight line in the 2-d image, (b) An illustration showing that a long straight line in the image plane is more likely to be a straight line in 3D.

the plane in which superpixel i lies, along the ray $R_{j,s_j''}$ (See Fig. 2.13).

$$h_{s_j''}(\alpha_i, \alpha_j, y_{ij}, R_{j,s_j''}) = \exp\left(-y_{ij}|(R_{j,s_j''}^T \alpha_i - R_{j,s_j''}^T \alpha_j)\hat{d}_{s_j''}|\right) \tag{2.7}$$

with $h_{s_i'',s_j''}(\cdot) = h_{s_i''}(\cdot)h_{s_j''}(\cdot)$. Note that if the two superpixels are coplanar, then $h_{s_i'',s_j''} = 1$. To enforce co-planarity between two distant planes that are not connected, we can choose three such points and use the above penalty.

Co-linearity: Consider two superpixels i and j lying on a long straight line in a 2-d image (Fig. 2.14a). There are an infinite number of curves that would project to a straight line in the image plane; however, a straight line in the image plane is more likely to be a straight one in 3D as well (Fig. 2.14b). In our model, therefore, we will penalize the relative (fractional) distance of a point (such as s_j) from the ideal straight line.

In detail, consider two superpixels i and j that lie on planes parameterized by α_i and α_j respectively in 3D, and that lie on a straight line in the 2-d image. For a point s_j lying on superpixel j, we will penalize its (fractional) distance along the ray R_{j,s_j}

from the 3D straight line passing through superpixel i. I.e.,

$$h_{s_j}(\alpha_i, \alpha_j, y_{ij}, R_{j,s_j}) = \exp\left(-y_{ij}|(R_{j,s_j}^T \alpha_i - R_{j,s_j}^T \alpha_j)\hat{d}|\right) \qquad (2.8)$$

with $h_{s_i,s_j}(\cdot) = h_{s_i}(\cdot)h_{s_j}(\cdot)$. In detail, $R_{j,s_j}^T \alpha_j = 1/d_{j,s_j}$ and $R_{j,s_j}^T \alpha_i = 1/d'_{j,s_j}$; therefore, the term $(R_{j,s_j}^T \alpha_i - R_{j,s_j}^T \alpha_j)\hat{d}$ gives the fractional distance $\left|(d_{j,s_j} - d'_{j,s_j})/\sqrt{d_{j,s_j}d'_{j,s_j}}\right|$ for $\hat{d} = \sqrt{\hat{d}_{j,s_j}\hat{d}'_{j,s_j}}$. The "confidence" y_{ij} depends on the length of the line and its curvature—a long straight line in 2-d is more likely to be a straight line in 3D.

2.10.1   Parameter learning

Exact parameter learning based on conditional likelihood for the Laplacian models is intractable, therefore, we use Multi-Conditional Learning (MCL) [15, 87] to divide the learning problem into smaller learning problems for each of the individual densities. MCL is a framework for optimizing graphical models based on a product of several marginal conditional likelihoods each relying on common sets of parameters from an underlying joint model and predicting different subsets of variables conditioned on other subsets.

In detail, we will first focus on learning θ_r given the ground-truth depths d (obtained from our 3D laser scanner, see Section 2.11) and the value of y_{ij} and ν_{i,s_i}. For this, we maximize the conditional pseudo log-likelihood $\log P(\alpha|X, \nu, y, R; \theta_r)$ as

$$\theta_r^* = \arg\max_{\theta_r} \sum_i \log f_1(\alpha_i|X_i, \nu_i, R_i; \theta_r)$$
$$+ \sum_{i,j} \log f_2(\alpha_i, \alpha_j|y_{ij}, R_i, R_j)$$

Now, from Eq. 2.3 note that $f_2(\cdot)$ does not depend on θ_r; therefore the learning problem simplifies to minimizing the L_1 norm, i.e., $\theta_r^* = \arg\min_{\theta_r} \sum_i \sum_{s_i=1}^{S_i} \nu_{i,s_i} \left|\frac{1}{d_{i,s_i}}(x_{i,s_i}^T \theta_r) - 1\right|$. This is efficiently solved using a Linear Program (LP).

In the next step, we learn the parameters ϕ of the logistic regression model for estimating ν in footnote 16. Parameters of a logistic regression model can be estimated by maximizing the conditional log-likelihood. [8] Now, the parameters ψ of

the logistic regression model $P(y_{ij}|\epsilon_{ij};\psi)$ for occlusion boundaries and folds are similarly estimated using the hand-labeled ground-truth ground-truth training data by maximizing its conditional log-likelihood.

2.10.2   MAP inference

MAP inference of the plane parameters α, i.e., maximizing the conditional likelihood $P(\alpha|X,\nu,y,R;\theta)$, is efficiently performed by solving a LP. We implemented an efficient method that uses the sparsity in our problem, so that inference can be performed in about 4-5 seconds for an image having about 2000 superpixels on a single-core Intel 3.40GHz CPU with 2 GB RAM.

Details: When given a new test-set image, we find the MAP estimate of the plane parameters α by maximizing the conditional log-likelihood $\log P(\alpha|X,\nu,Y,R;\theta)$. Note that we solve for α as a continuous variable optimization problem, which is unlike many other techniques where discrete optimization is more popular, e.g., [155]. From Eq. 2.3, we have

$$\alpha^* = \arg\max_\alpha \log P(\alpha|X,\nu,y,R;\theta_r)$$

$$= \arg\max_\alpha \log \frac{1}{Z} \prod_i f_1(\alpha_i|X_i,\nu_i,R_i;\theta_r) \prod_{i,j} f_2(\alpha_i,\alpha_j|y_{ij},R_i,R_j)$$

Note that the partition function Z does not depend on α. Therefore, from Eq. 2.4, 2.6 and 2.7 and for $\hat{d} = x^T\theta_r$, we have

$$
\begin{aligned}
= \arg\min_\alpha \sum_{i=1}^K \Bigg(&\sum_{s_i=1}^{S_i} \nu_{i,s_i} \left| (R_{i,s_i}^T \alpha_i)\hat{d}_{i,s_i} - 1 \right| \\
&+ \sum_{j\in N(i)} \sum_{s_i,s_j\in B_{ij}} y_{ij} \left| (R_{i,s_i}^T \alpha_i - R_{j,s_j}^T \alpha_j)\hat{d}_{s_i,s_j} \right| \\
&+ \sum_{j\in N(i)} \sum_{s_j\in C_j} y_{ij} \left| (R_{j,s_j}^T \alpha_i - R_{j,s_j}^T \alpha_j)\hat{d}_{s_j} \right| \Bigg)
\end{aligned}
$$

where K is the number of superpixels in each image; $N(i)$ is the set of "neighboring" superpixels—one whose relations are modeled—of superpixel i; B_{ij} is the set of pair of

points on the boundary of superpixel i and j that model connectivity; C_j is the center point of superpixel j that model co-linearity and co-planarity; and $\hat{d}_{s_i,s_j} = \sqrt{\hat{d}_{s_i}\hat{d}_{s_j}}$. Note that each of terms is a L_1 norm of a linear function of α; therefore, this is a L_1 norm minimization problem, [13, chap. 6.1.1] and can be compactly written as

$$\arg\min_x \|Ax - b\|_1 + \|Bx\|_1 + \|Cx\|_1$$

where $x \in \mathbb{R}^{3K \times 1}$ is a column vector formed by rearranging the three x-y-z components of $\alpha_i \in \mathbb{R}^3$ as $x_{3i-2} = \alpha_{ix}$, $x_{3i-1} = \alpha_{iy}$ and $x_{3i} = \alpha_{iz}$; A is a block diagonal matrix such that $A\left[(\sum_{l=1}^{i-1} S_l) + s_i, \quad (3i-2) : 3i\right] = R_{i,s_i}^T \hat{d}_{i,s_i} \nu_{i,s_i}$ and $b_1 \in \mathbb{R}^{3K \times 1}$ is a column vector formed from ν_{i,s_i}. B and C are all block diagonal matrices composed of rays R, $\hat{d}$ and y; they represent the cross terms modeling the connected structure, co-planarity and co-linearity properties.

In general, finding the global optimum in a loopy MRF is difficult. However in our case, the minimization problem is an Linear Program (LP), and therefore can be solved exactly using any linear programming solver. (In fact, any greedy method including a loopy belief propagation would reach the global minima.) For fast inference, we implemented our own optimization method, one that captures the sparsity pattern in our problem, and by approximating the L_1 norm with a smooth function:

$$\|x\|_1 \cong \Upsilon_\beta(x) = \tfrac{1}{\beta}\left[\log\left(1 + \exp(-\beta x)\right) + \log\left(1 + \exp(\beta x)\right)\right]$$

Note that $\|x\|_1 = \lim_{\beta \to \infty} \|x\|_\beta$, and the approximation can be made arbitrarily close by increasing β during steps of the optimization. Then we wrote a customized Newton method based solver that computes the Hessian efficiently by utilizing the sparsity. [13]

S1C	34
S1T	34
S1B	34
S1L	34
S1R	34
S2C	34
S2T	34
S2B	34
S2L	34
S2R	34
S3C	34
S3T	34
S3B	34
S3L	34
S3R	34
Shape	14
Total	524
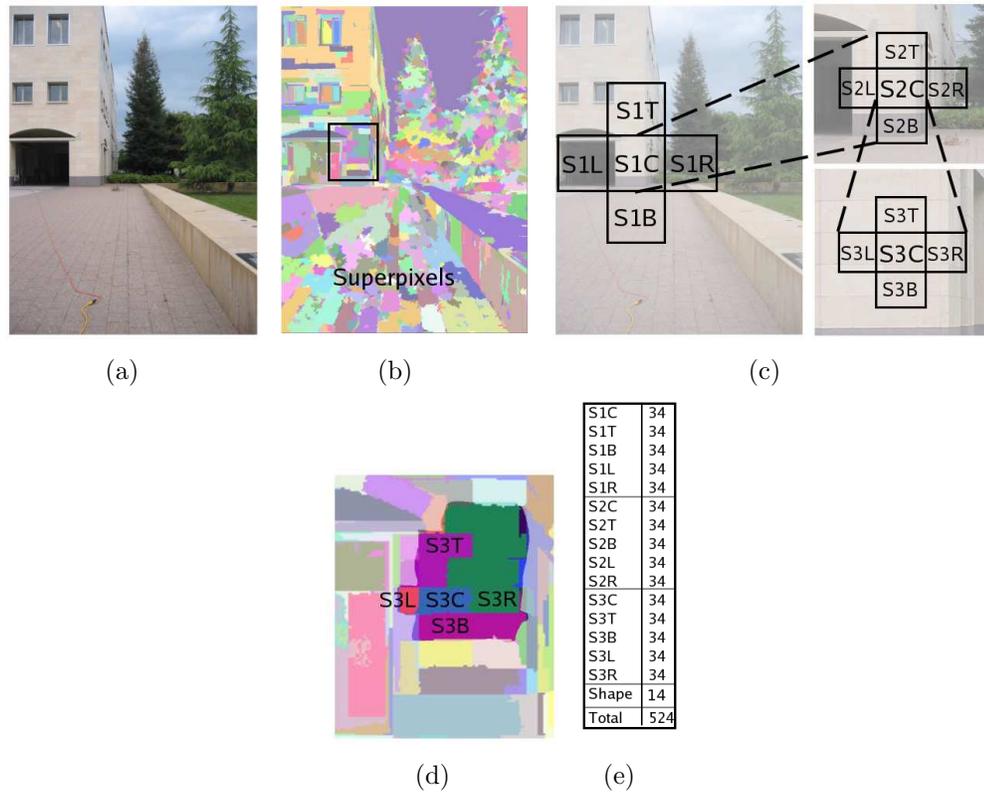
Figure 2.15: The feature vector. (a) The original image, (b) Superpixels for the image, (c) An illustration showing the location of the neighbors of superpixel S3C at multiple scales, (d) Actual neighboring superpixels of S3C at the finest scale, (e) Features from each neighboring superpixel along with the superpixel-shape features give a total of 524 features for the superpixel S3C. (*Best viewed in color.*)

2.11 Data collection

We used a 3D laser scanner to collect images and their corresponding depthmaps (Fig. 2.17). The scanner uses a laser device (SICK LMS-291) which gives depth readings in a vertical column, with a $1.0°$ resolution. To collect readings along the other axis (left to right), the SICK laser was mounted on a panning motor. The motor rotates after each vertical scan to collect laser readings for another vertical column, with a $0.5°$ horizontal angular resolution. We reconstruct the depthmap using the vertical laser scans, the motor readings and known relative position and pose of the

(a) Image   (b) Ground-Truth   (c) Gaussian   (d) Laplacian
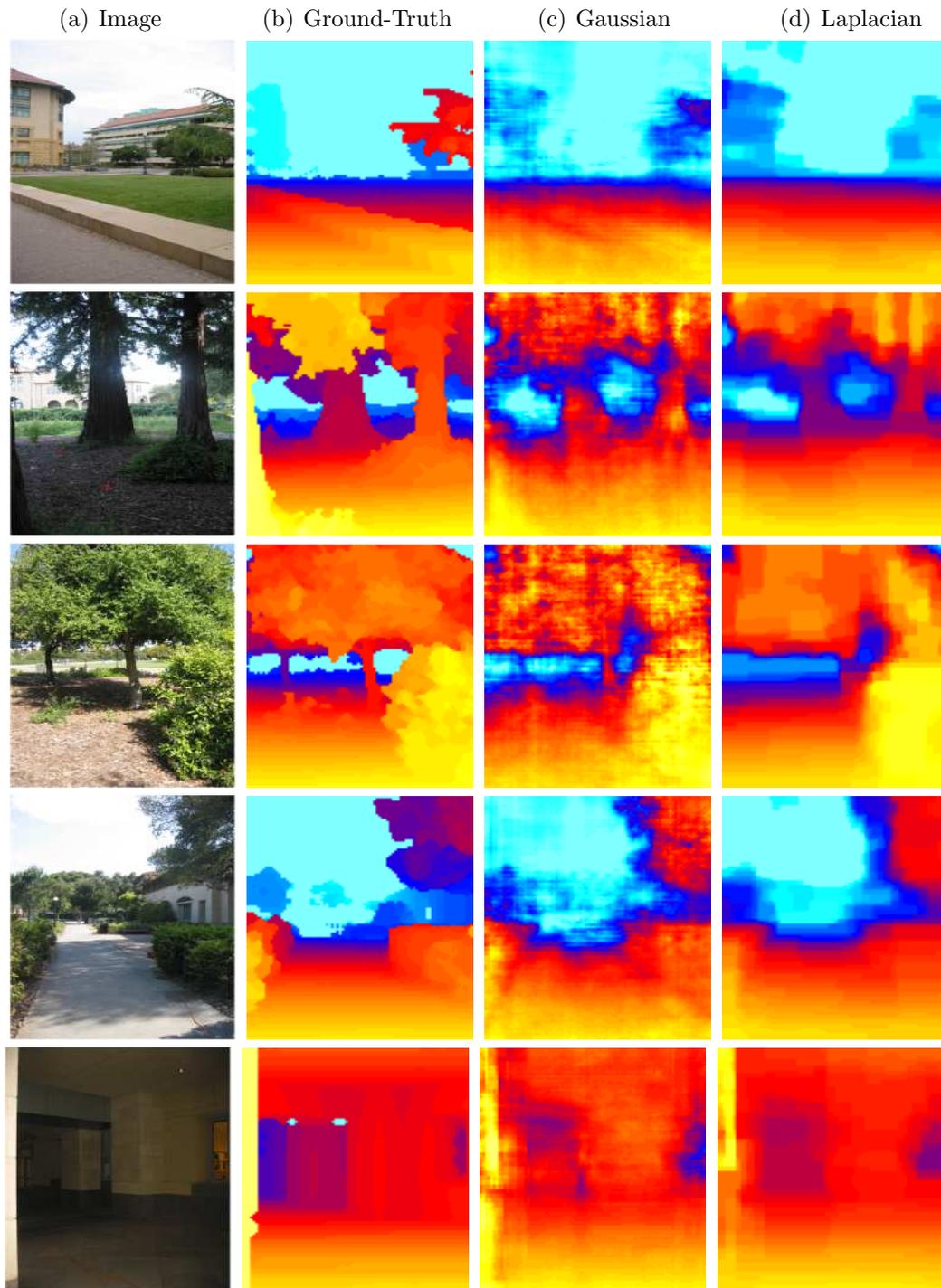
Figure 2.16: Results for a varied set of environments, showing (a) original image, (b) ground truth depthmap, (c) predicted depthmap by Gaussian model, (d) predicted depthmap by Laplacian model. (**Best viewed in color**).

laser device and the camera. We also collected data of stereo pairs with corresponding depthmaps (Section 3.2), by mounting the laser range finding equipment on a LAGR (Learning Applied to Ground Robotics) robot (Fig. 2.18). The LAGR vehicle is equipped with sensors, an onboard computer, and Point Grey Research Bumblebee stereo cameras, mounted with a baseline distance of 11.7cm. [146]

2.12 Experiments: depths

We collected a total of 425 image+depthmap pairs, with an image resolution of 1704x2272 and a depthmap resolution of 86x107. In the experimental results reported here, 75% of the images/depthmaps were used for training, and the remaining 25% for hold-out testing. The images comprise a wide variety of scenes including natural environments (forests, trees, bushes, etc.), man-made environments (buildings, roads, sidewalks, trees, grass, etc.), and purely indoor environments (corridors, etc.). Due to limitations of the laser, the depthmaps had a maximum range of 81m (the maximum range of the laser scanner), and had minor additional errors due to reflections, missing laser scans, and mobile objects. Prior to running our learning algorithms, we transformed all the depths to a log scale so as to emphasize multiplicative rather than additive errors in training. Data used in the experiments is available at:

http://ai.stanford.edu/~asaxena/learningdepth/

We tested our model on real-world test-set images of forests (containing trees, bushes, etc.), campus areas (buildings, people, and trees), and indoor scenes (such as corridors).

Table 2.1 shows the test-set results with different feature combinations of scales, summary statistics, and neighbors, on three classes of environments: forest, campus, and indoor. The *Baseline* model is trained without any features, and predicts the mean value of depth in the training depthmaps. We see that multiscale and column features improve the algorithm's performance. Including features from neighboring patches, which help capture more global information, reduces the error from .162

Figure 2.17: The 3D scanner used for collecting images and the corresponding depthmaps.

Figure 2.18: The custom built 3D scanner for collecting depthmaps with stereo image pairs, mounted on the LAGR robot.
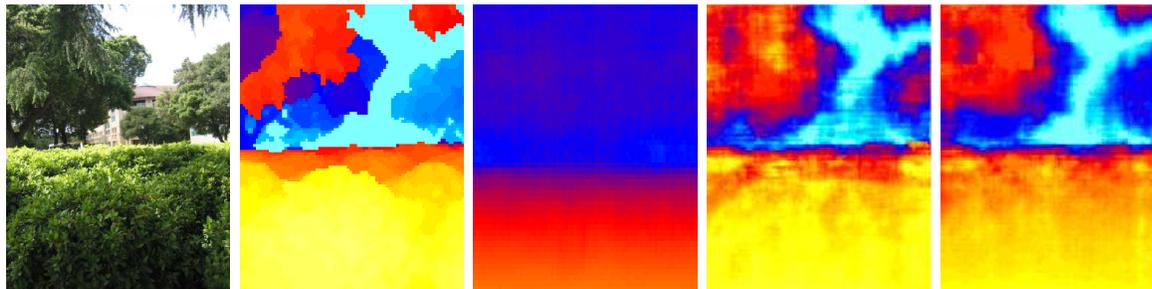
Figure 2.19: (a) original image, (b) ground truth depthmap, (c) "prior" depthmap (trained with no features), (d) features only (no MRF relations), (e) Full Laplacian model. (**Best viewed in color**).

Table 2.1: Effect of multiscale and column features on accuracy. The average absolute errors (RMS errors gave very similar trends) are on a log scale (base 10). H_1 and H_2 represent summary statistics for $k = 1, 2$. S_1, S_2 and S_3 represent the 3 scales. C represents the column features. Baseline is trained with only the bias term (no features).

Feature	All	Forest	Campus	Indoor
Baseline	.295	.283	.343	.228
Gaussian (S_1,S_2,S_3, H_1,H_2,no neighbors)	.162	.159	.166	.165
Gaussian (S_1, H_1,H_2)	.171	.164	.189	.173
Gaussian (S_1,S_2, H_1,H_2)	.155	.151	.164	.157
Gaussian (S_1,S_2,S_3, H_1,H_2)	.144	.144	.143	.144
Gaussian (S_1,S_2,S_3, C, H_1)	.139	.140	.141	.122
Gaussian (S_1,S_2,S_3, C, H_1,H_2)	.133	.135	**.132**	.124
Laplacian	**.132**	**.133**	.142	**.084**

orders of magnitude to .133 orders of magnitude.[17] We also note that the Laplacian model performs better than the Gaussian one, reducing error to .084 orders of magnitude for indoor scenes, and .132 orders of magnitude when averaged over all scenes. Empirically, the Laplacian model does indeed give depthmaps with significantly sharper boundaries (as in our discussion in Section 2.5.2; also see Fig. 2.16).

Fig. 2.19 shows that modeling the spatial relationships in the depths is important. Depths estimated without using the second term in the exponent of Eq. 2.2, i.e., depths predicted using only image features with row-sensitive parameters θ_r, are

---

[17]Errors are on a $\log_{10}$ scale. Thus, an error of ε means a multiplicative error of 10^ε in actual depth. E.g., $10^{.132} = 1.355$, which thus represents an 35.5% multiplicative error.

very noisy (Fig. 2.19d).[18] Modeling the relations between the neighboring depths at multiple scales through the second term in the exponent of Eq. 2.2 also gave better depthmaps (Fig. 2.19e). Finally, Fig. 2.19c shows the model's "prior" on depths; the depthmap shown reflects our model's use of image-row sensitive parameters. In our experiments, we also found that many features/cues were given large weights; therefore, a model trained with only a few cues (e.g., the top 50 chosen by a feature selection method) was not able to predict reasonable depths.

Our algorithm works well in a varied set of environments, as shown in Fig. 2.16 (last column). A number of vision algorithms based on "ground finding" (e.g., [36]) appear to perform poorly when there are discontinuities or significant luminance variations caused by shadows, or when there are significant changes in the ground texture. In contrast, our algorithm appears to be robust to luminance variations, such as shadows (Fig. 2.16, 4th row) and camera exposure (Fig. 2.16, 2nd and 5th rows).

Some of the errors of the algorithm can be attributed to errors or limitations of the training set. For example, the maximum value of the depths in the training and test set is 81m; therefore, far-away objects are all mapped to the distance of 81m. Further, laser readings are often incorrect for reflective/transparent objects such as glass; therefore, our algorithm also often estimates depths of such objects incorrectly. Quantitatively, our algorithm appears to incur the largest errors on images which contain very irregular trees, in which most of the 3D structure in the image is dominated by the shapes of the leaves and branches. However, arguably even human-level performance would be poor on these images.

We note that monocular cues rely on prior knowledge, learned from the training set, about the environment. This is because monocular 3D reconstruction is an inherently ambiguous problem. Thus, the monocular cues may not generalize well to images very different from ones in the training set, such as underwater images or aerial photos.

To test the generalization capability of the algorithm, we also estimated depthmaps of images downloaded from the Internet (images for which camera parameters are not

---

[18]This algorithm gave an overall error of .181, compared to our full model's error of .132.
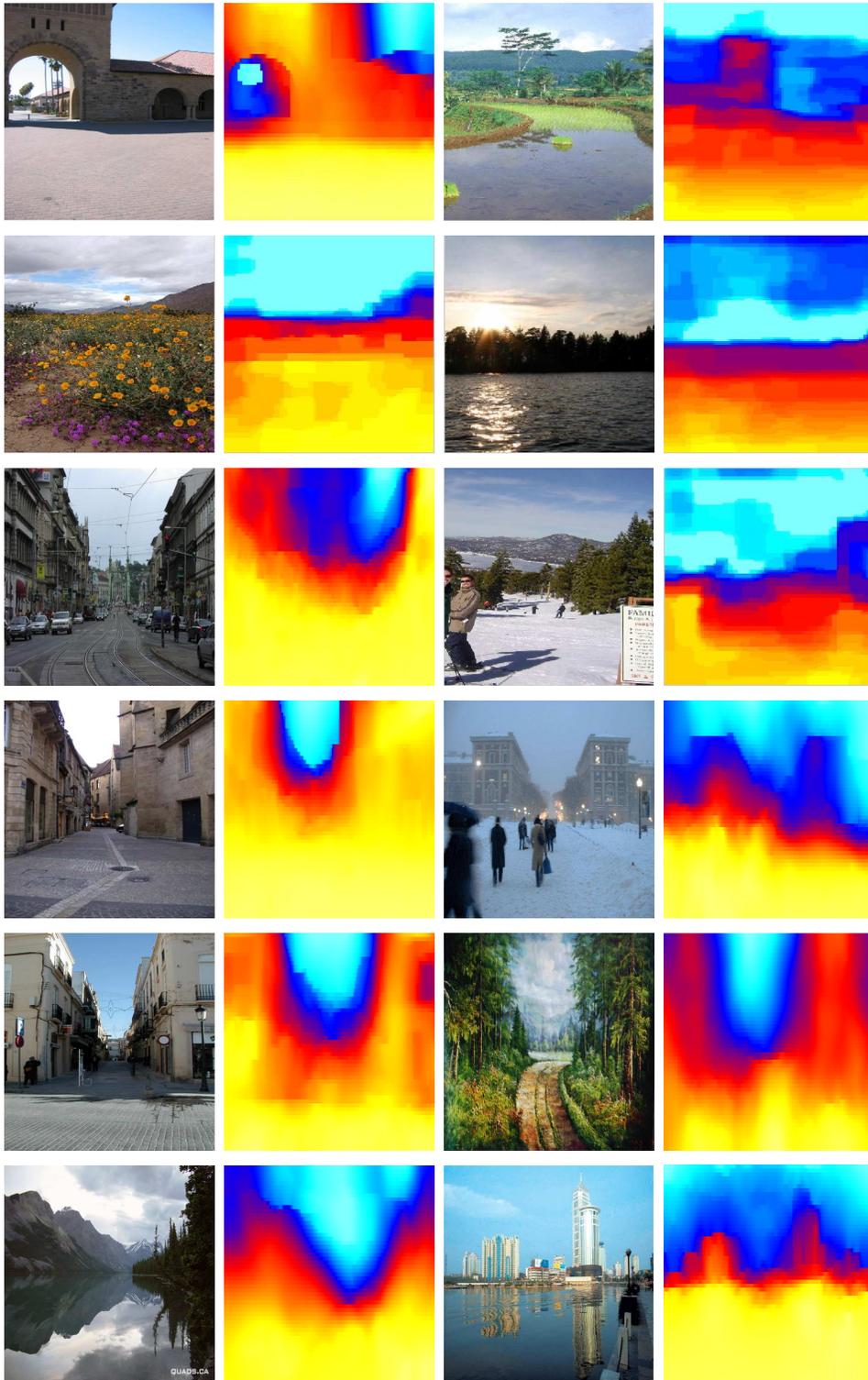
Figure 2.20: Typical examples of the predicted depths for images downloaded from the internet. (**Best viewed in color.**)

known).[19] The model (using monocular cues only) was able to produce reasonable depthmaps on most of the images (Fig. 2.20). Informally, our algorithm appears to predict the relative depths quite well (i.e., their relative distances to the camera);[20] even for scenes very different from the training set, such as a sunflower field, an oil-painting scene, mountains and lakes, a city skyline photographed from sea, a city during snowfall, etc.

2.13   Experiments: visual

We collected a total of 534 images+depths, with an image resolution of 2272x1704 and depths with resolution of 55x305. These images were collected during daytime in a diverse set of urban and natural areas in the city of Palo Alto and its surrounding regions.

| (a) | (b) | (c) | (d) | (e) |
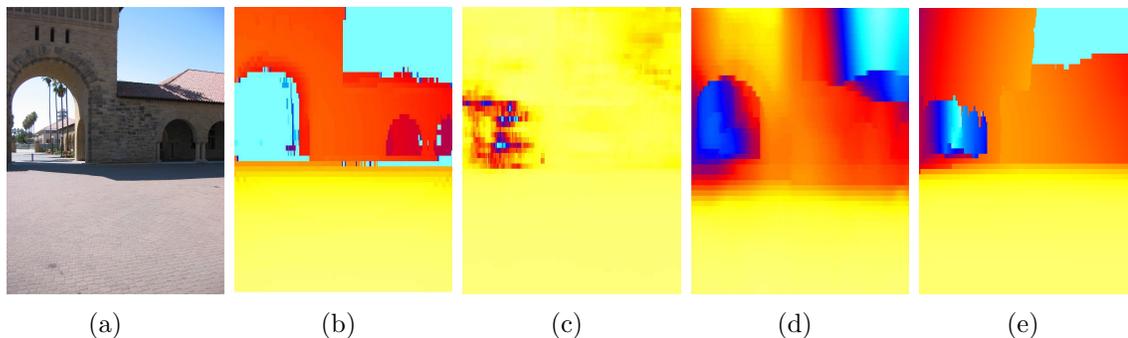
Figure 2.21: (a) Original Image, (b) Ground truth depths, (c) Depth from image features only, (d) Point-wise MRF, (e) Plane parameter MRF. (*Best viewed in color.*)

We tested our model on rest of the 134 images (collected using our 3D scanner),

---

[19]Since we do not have ground-truth depthmaps for images downloaded from the Internet, we are unable to give a quantitative comparisons on these images. Further, in the extreme case of orthogonal cameras or very wide angle perspective cameras, our algorithm would need to be modified to take into account the field of view of the camera.

[20]For most applications such as object recognition using knowledge of depths, robotic navigation, or 3D reconstruction, relative depths are sufficient. The depths could be rescaled to give accurate absolute depths, if the camera parameters are known or are estimated.
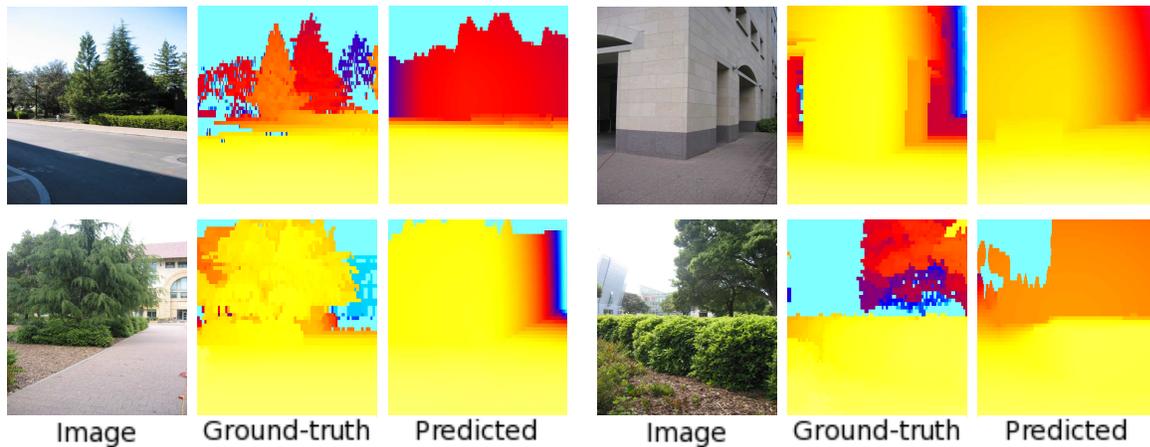
Figure 2.22: Typical depths predicted by our plane parameterized algorithm on hold-out test set, collected using the laser-scanner. (*Best viewed in color.*)
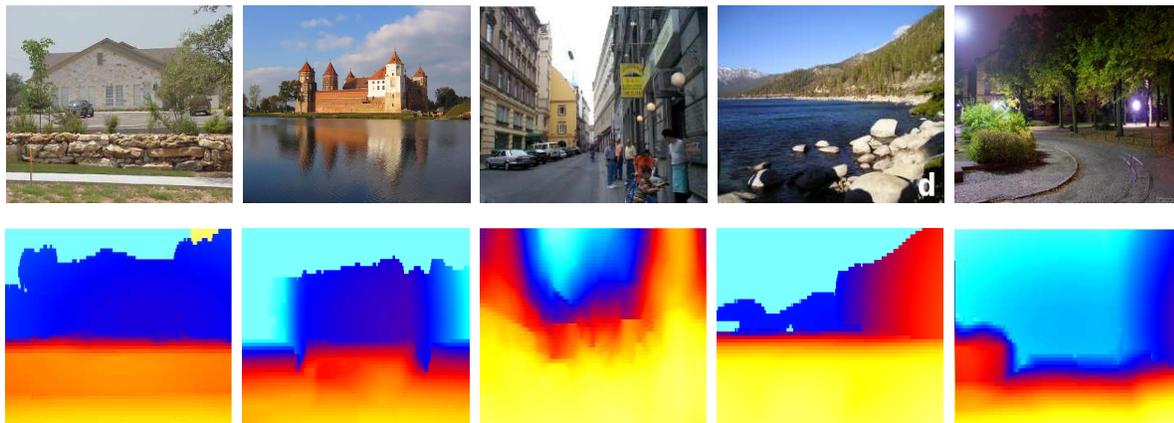
Figure 2.23: Typical results from our algorithm. (Top row) Original images, (Bottom row) depths (shown in log scale, yellow is closest, followed by red and then blue) generated from the images using our plane parameter MRF. (*Best viewed in color.*)

Table 2.2: Results: Quantitative comparison of various methods.

METHOD	CORRECT (%)	% PLANES CORRECT	$\log_{10}$	REL
SCN	NA	NA	0.198	0.530
HEH	33.1%	50.3%	0.320	1.423
BASELINE-1	0%	NA	0.300	0.698
NO PRIORS	0%	NA	0.170	0.447
POINT-WISE MRF	23%	NA	**0.149**	0.458
BASELINE-2	0%	0%	0.334	0.516
NO PRIORS	0%	0%	0.205	0.392
CO-PLANAR	45.7%	57.1%	0.191	0.373
PP-MRF	**64.9%**	**71.2%**	0.187	**0.370**

and also on 588 internet images. The internet images were collected by issuing keywords on Google image search. To collect data and to perform the evaluation of the algorithms in a completely unbiased manner, a person *not* associated with the project was asked to collect images of environments (greater than 800x600 size). The person chose the following keywords to collect the images: campus, garden, park, house, building, college, university, church, castle, court, square, lake, temple, scene. The images thus collected were from places from all over the world, and contained environments that were significantly different from the training set, e.g. hills, lakes, night scenes, etc. The person chose only those images which were of "environments," i.e. she removed images of the geometrical figure 'square' when searching for keyword 'square'; no other pre-filtering was done on the data.

In addition, we manually labeled 50 images with 'ground-truth' boundaries to learn the parameters for occlusion boundaries and folds.

We performed an extensive evaluation of our algorithm on 588 internet test images, and 134 test images collected using the laser scanner.

In Table 2.2, we compare the following algorithms:
(a) Baseline: Both for pointwise MRF (Baseline-1) and plane parameter MRF (Baseline-2). The Baseline MRF is trained without any image features, and thus reflects "prior" depths of sorts.
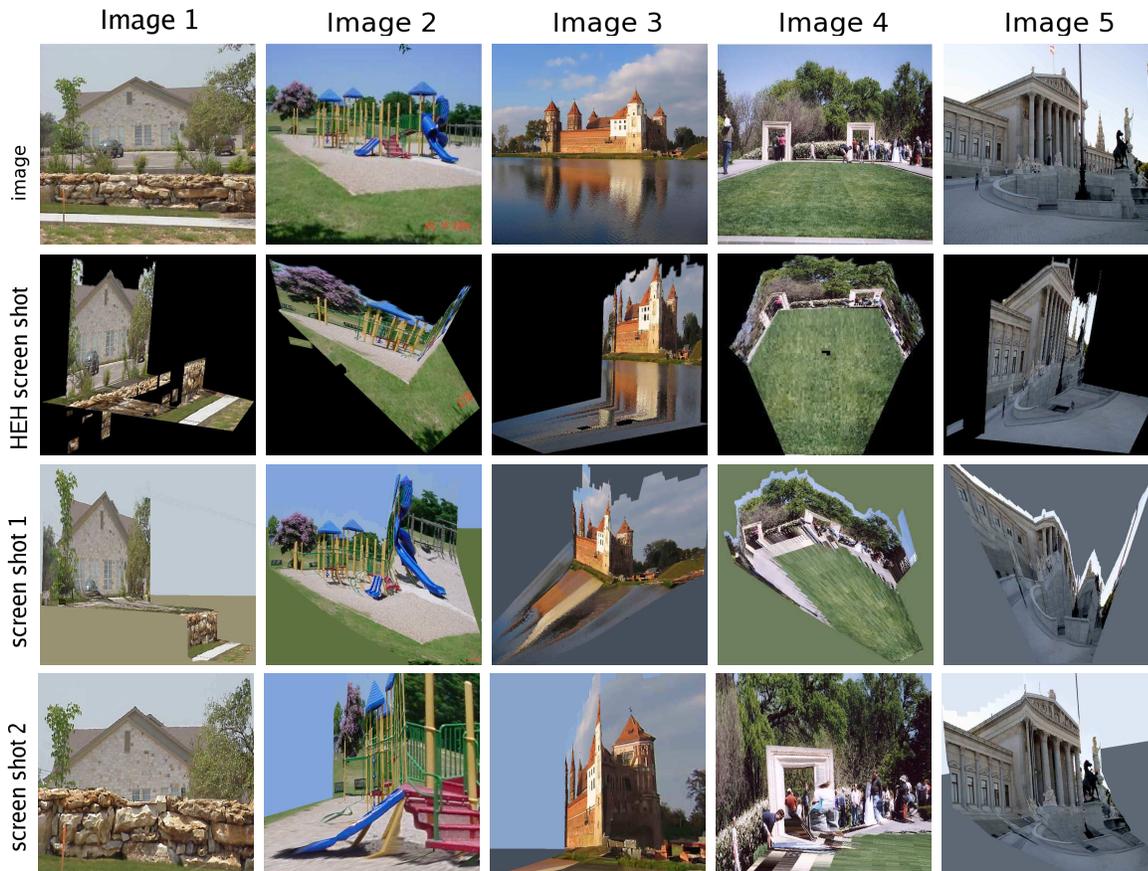
Figure 2.24: Typical results from HEH and our algorithm. **Row 1**: Original Image. **Row 2**: 3D model generated by HEH, **Row 3 and 4**: 3D model generated by our algorithm. (Note that the screenshots cannot be simply obtained from the original image by an affine transformation.) In **image 1**, HEH makes mistakes in some parts of the foreground rock, while our algorithm predicts the correct model; with the rock occluding the house, giving a novel view. In **image 2**, HEH algorithm detects a wrong ground-vertical boundary; while our algorithm not only finds the correct ground, but also captures a lot of non-vertical structure, such as the blue slide. In **image 3**, HEH is confused by the reflection; while our algorithm produces a correct 3D model. In **image 4**, HEH and our algorithm produce roughly equivalent results—HEH is a bit more visually pleasing and our model is a bit more detailed. In **image 5**, both HEH and our algorithm fail; HEH just predict one vertical plane at a incorrect location. Our algorithm predicts correct depths of the pole and the horse, but is unable to detect their boundary; hence making it qualitatively incorrect.

(b) Our Point-wise MRF (described in [152]): with and without constraints (connectivity, co-planar and co-linearity).

(c) Our Plane Parameter MRF (PP-MRF): without any constraint, with co-planar constraint only, and the full model.

(d) Saxena et al. (SCN), [134, 135] applicable for quantitative errors only.

(e) Hoiem et al. (HEH) [46]. For fairness, we scale and shift their depths before computing the errors to match the global scale of our test images. Without the scaling and shifting, their error is much higher (7.533 for relative depth error).

We compare the algorithms on the following metrics:

(a) % of models qualitatively correct,

(b) % of major planes correctly identified,[21]

(c) Depth error $|\log d - \log \hat{d}|$ on a log-10 scale, averaged over all pixels in the hold-out test set,

(d) Average relative depth error $\frac{|d - \hat{d}|}{d}$. (We give these two numerical errors on only the 134 test images that we collected, because ground-truth laser depths are not available for internet images.)

Table 2.2 shows that both of our models (Point-wise MRF and Plane Parameter MRF) outperform the other algorithms in quantitative accuracy in depth prediction. Plane Parameter MRF gives better relative depth accuracy and produces sharper depth discontinuities (Fig. 2.21, 2.22 and 2.23). Table 2.2 also shows that by capturing the image properties of connected structure, co-planarity and co-linearity, the models produced by the algorithm become significantly better. In addition to reducing quantitative errors, PP-MRF does indeed produce significantly better 3D models. When producing 3D flythroughs, even a small number of erroneous planes make the 3D model visually unacceptable, even though the quantitative numbers may still show small errors.

Our algorithm gives qualitatively correct models for 64.9% of images as compared

---

[21]For the first two metrics, we define a model as correct when for 70% of the major planes in the image (major planes occupy more than 15% of the area), the plane is in correct relationship with its nearest neighbors (i.e., the relative orientation of the planes is within 30 degrees). Note that changing the numbers, such as 70% to 50% or 90%, 15% to 10% or 30%, and 30 degrees to 20 or 45 degrees, gave similar trends in the results.
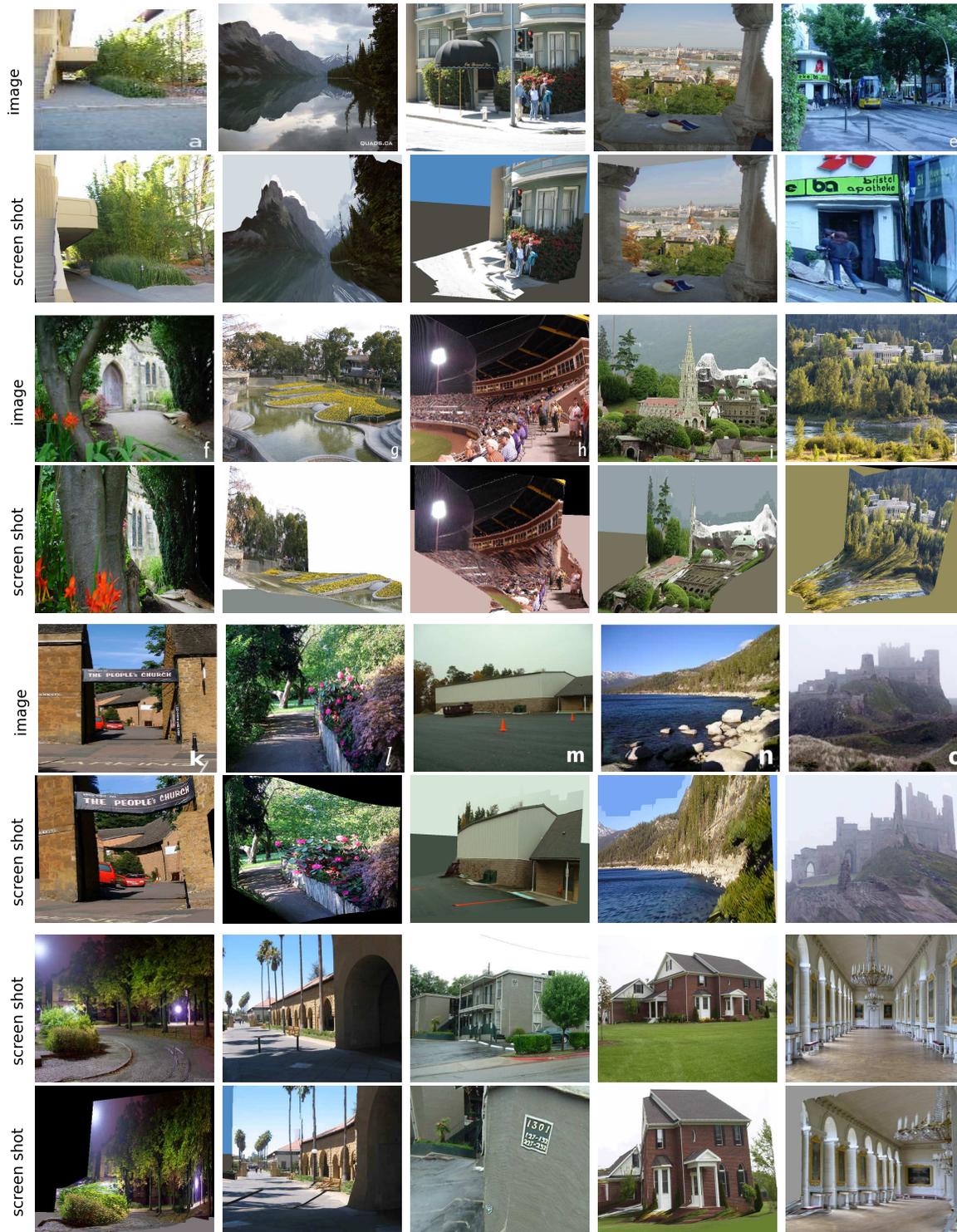
Figure 2.25: Typical results from our algorithm. Original image (top), and a screen-shot of the 3D flythrough generated from the image (bottom of the image). 16 images (a-g,l-t) were evaluated as "correct" and 4 (h-k) were evaluated as "incorrect."

Table 2.3: Percentage of images for which HEH is better, our PP-MRF is better, or it is a tie.

Algorithm	%better
Tie	15.8%
HEH	22.1%
PP-MRF	**62.1%**

to 33.1% by HEH. The qualitative evaluation was performed by a person not associated with the project following the guidelines in Footnote 21. Delage, Lee and Ng [26] and HEH generate a popup effect by folding the images at "ground-vertical" boundaries—an assumption which is not true for a significant number of images; therefore, their method fails in those images. Some typical examples of the 3D models are shown in Fig. 2.24. (Note that all the *test* cases shown in Fig. 5.1, 2.23, 2.24 and 2.25 are from the dataset downloaded from the internet, except Fig. 2.25a which is from the laser-test dataset.) These examples also show that our models are often more detailed, in that they are often able to model the scene with a multitude (over a hundred) of planes.

We performed a further comparison. Even when both algorithms are evaluated as qualitatively correct on an image, one result could still be superior. Therefore, we asked the person to compare the two methods, and decide which one is better, or is a tie.[22] Table 2.3 shows that our algorithm outputs the better model in 62.1% of the cases, while HEH outputs better model in 22.1% cases (tied in the rest).

Full documentation describing the details of the unbiased human judgment process, along with the 3D flythroughs produced by our algorithm, is available online at:

http://make3d.stanford.edu/research

---

[22]To compare the algorithms, the person was asked to count the number of errors made by each algorithm. We define an error when a major plane in the image (occupying more than 15% area in the image) is in wrong location with respect to its neighbors, or if the orientation of the plane is more than 30 degrees wrong. For example, if HEH fold the image at incorrect place (see Fig. 2.24, image 2), then it is counted as an error. Similarly, if we predict top of a building as far and the bottom part of building near, making the building tilted—it would count as an error.

Some of our models, e.g. in Fig. 2.25j, have cosmetic defects—e.g. stretched texture; better texture rendering techniques would make the models more visually pleasing. In some cases, a small mistake (e.g., one person being detected as far-away in Fig. 2.25h, and the banner being bent in Fig. 2.25k) makes the model look bad, and hence be evaluated as "incorrect."

2.14   Large-scale web experiment

Finally, in a large-scale web experiment, we allowed users to upload their photos on the internet, and view a 3D flythrough produced from their image by our algorithm. About 23846 unique users uploaded (and rated) about 26228 images.[23] Users rated 48.1% of the models as good. If we consider the images of scenes only, i.e., exclude images such as company logos, cartoon characters, closeups of objects, etc., then this percentage was 57.3%. We have made the following website available for downloading datasets/code, and for converting an image to a 3D model/flythrough:

http://make3d.stanford.edu

Our algorithm, trained on images taken in daylight around the city of Palo Alto, was able to predict qualitatively correct 3D models for a large variety of environments—for example, ones that have hills or lakes, ones taken at night, and even paintings. (See Fig. 2.25 and the website.) We believe, based on our experiments with varying the number of training examples (not reported here), that having a larger and more diverse set of training images would improve the algorithm significantly.

2.15   Incorporating object information

In this section, we will demonstrate how our model can also incorporate other information that might be available, for example, from object recognizers. In prior work, Sudderth et al. [168] showed that knowledge of objects could be used to get crude

---

[23]No restrictions were placed on the type of images that users can upload. Users can rate the models as good (thumbs-up) or bad (thumbs-down).
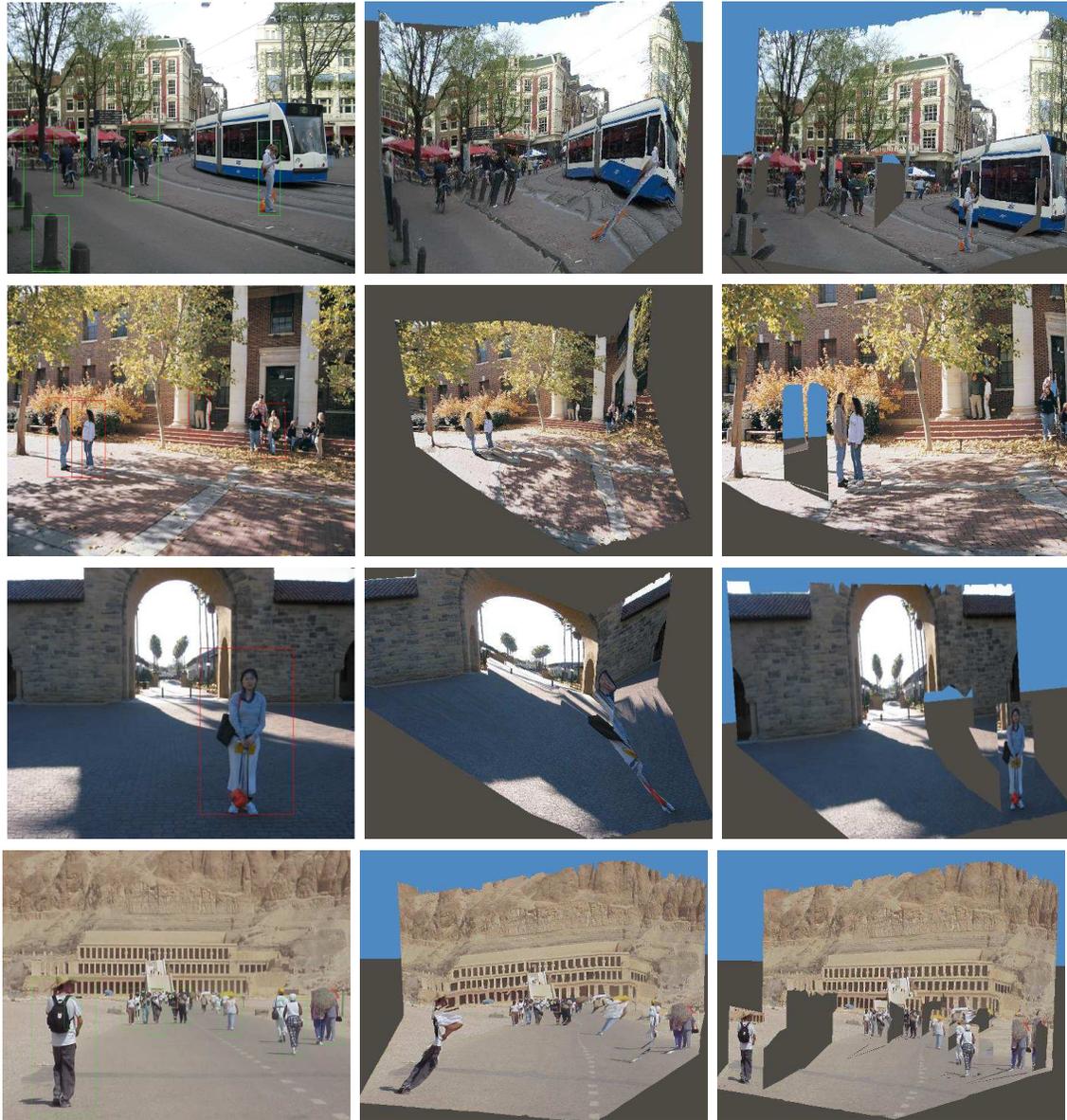
Figure 2.26: (Left) Original Images, (Middle) Snapshot of the 3D model without using object information, (Right) Snapshot of the 3D model that uses object information.

depth estimates, and Hoiem et al. [45] used knowledge of objects and their location to improve the estimate of the horizon. In addition to estimating the horizon, the knowledge of objects and their location in the scene give strong cues regarding the 3D structure of the scene. For example, that a person is more likely to be on top of the ground, rather than under it, places certain restrictions on the 3D models that could be valid for a given image.

Here we give some examples of such cues that arise when information about objects is available, and describe how we can encode them in our MRF:

(a) *"Object A is on top of object B"*

This constraint could be encoded by restricting the points $s_i \in \mathbb{R}^3$ on object A to be on top of the points $s_j \in \mathbb{R}^3$ on object B, i.e., $s_i^T \hat{z} \geq s_j^T \hat{z}$ (if $\hat{z}$ denotes the "up" vector). In practice, we actually use a probabilistic version of this constraint. We represent this inequality in plane-parameter space ($s_i = R_i d_i = R_i/(\alpha_i^T R_i)$). To penalize the fractional error $\xi = \left(R_i^T \hat{z} R_j^T \alpha_j - R_j^T \hat{z} R_i \alpha_i \right) \hat{d}$ (the constraint corresponds to $\xi \geq 0$), we choose an MRF potential $h_{s_i, s_j}(.) = \exp\left(-y_{ij} \left(\xi + |\xi| \right) \right)$, where y_{ij} represents the uncertainty in the object recognizer output. Note that for $y_{ij} \to \infty$ (corresponding to certainty in the object recognizer), this becomes a "hard" constraint $R_i^T \hat{z}/(\alpha_i^T R_i) \geq R_j^T \hat{z}/(\alpha_j^T R_j)$.

In fact, we can also encode other similar spatial-relations by choosing the vector $\hat{z}$ appropriately. For example, a constraint *"Object A is in front of Object B"* can be encoded by choosing $\hat{z}$ to be the ray from the camera to the object.

(b) *"Object A is attached to Object B"*

For example, if the ground-plane is known from a recognizer, then many objects would be more likely to be "attached" to the ground plane. We easily encode this by using our connected-structure constraint.

(c) *Known plane orientation*

If orientation of a plane is roughly known, e.g. that a person is more likely to be "vertical", then it can be easily encoded by adding to Eq. 2.3 a term $f(\alpha_i) = \exp\left(-w_i |\alpha_i^T \hat{z}| \right)$; here, w_i represents the confidence, and $\hat{z}$ represents the up vector.

We implemented a recognizer (based on the features described in Section 5.3.3) for ground-plane, and used the Dalal-Triggs Detector [22] to detect pedestrians. For

these objects, we encoded the (a), (b) and (c) constraints described above. Fig. 2.26 shows that using the pedestrian and ground detector improves the accuracy of the 3D model. Also note that using "soft" constraints in the MRF (Section 2.15), instead of "hard" constraints, helps in estimating correct 3D models even if the object recognizer makes a mistake.

In an extension of our work [43], we addressed the problem of jointly solving object detection and depth reconstruction. We proposed learning a set of related models in such that they both solve their own problem and help each other. We developed a framework called Cascaded Classification Models (CCM), where repeated instantiations of these classifiers are coupled by their input/output variables in a cascade that improves performance at each level. Our method requires only a limited "black box" interface with the models, allowing us to use very sophisticated, state-of-the-art classifiers without having to look under the hood. We demonstrated the effectiveness of our method on a large set of natural images by combining the subtasks of scene categorization, object detection, multiclass image segmentation, and 3d reconstruction. See Fig. 2.27 for a few examples.

2.16 Discussion

The problem of depth perception is fundamental to computer vision, one that has enjoyed the attention of many researchers and seen significant progress in the last few decades. However, the vast majority of this work, such as stereopsis, has used multiple image geometric cues to infer depth. In contrast, single-image cues offer a largely orthogonal source of information, one that has heretofore been relatively underexploited. Given that depth and shape perception appears to be an important building block for many other applications, such as object recognition [176, 45], grasping [136], navigation [88], image compositing [58], and video retrieval [31], we believe that monocular depth perception has the potential to improve all of these applications, particularly in settings where only a single image of a scene is available.

We presented an algorithm for inferring detailed 3D structure from a single still
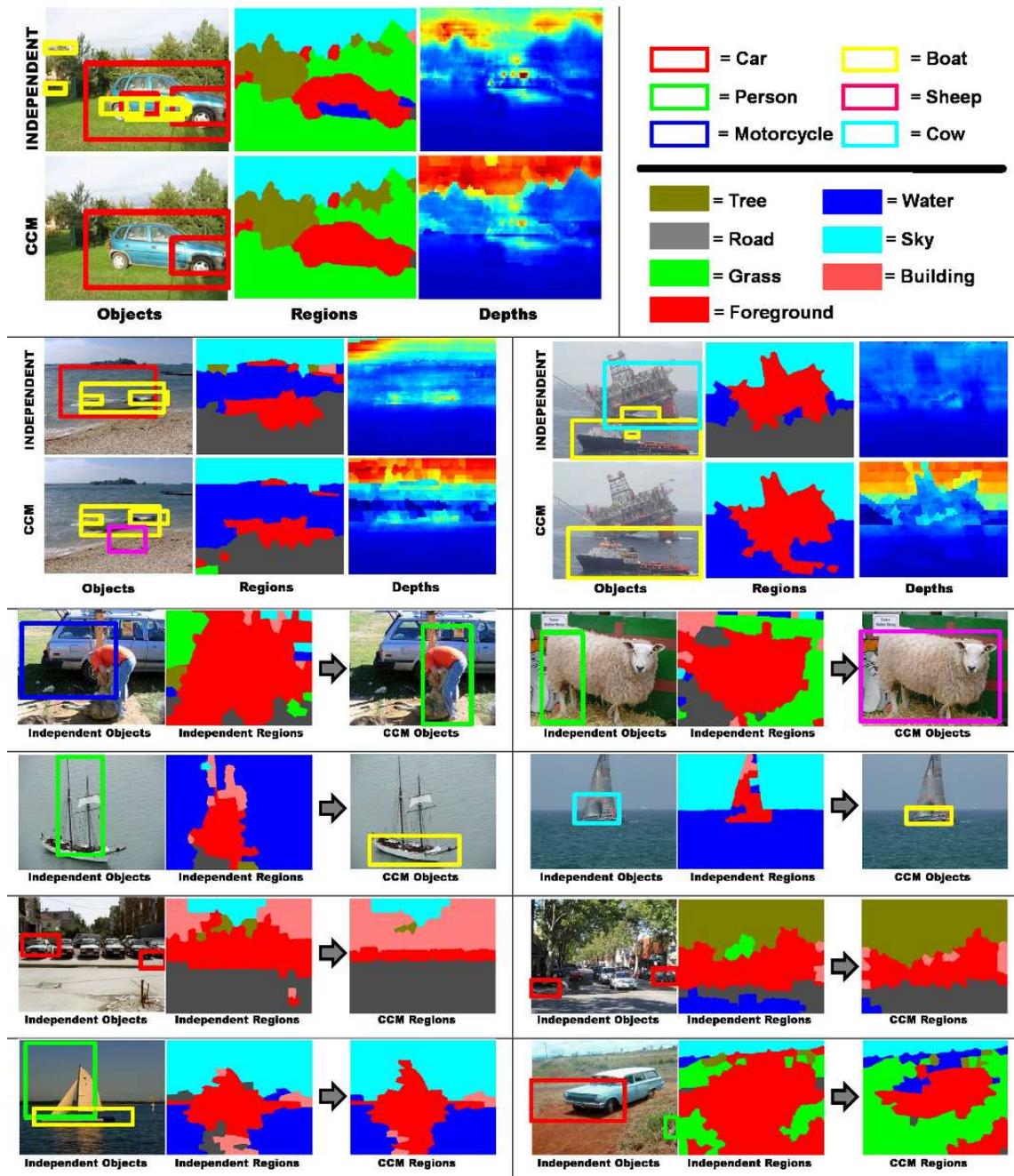
(a)

Figure 2.27: (top two rows) three cases where CCM improved results for all tasks. In the first, for instance, the presence of grass allows the CCM to remove the boat detections. The next four rows show four examples where detections are improved and four examples where segmentations are improved. (*Best viewed in color.*)

image. Compared to previous approaches, our algorithm creates detailed 3D models which are both quantitatively more accurate and visually more pleasing. Our approach begins by over-segmenting the image into many small homogeneous regions called "superpixels" and uses an MRF to infer the 3D position and orientation of each. Other than assuming that the environment is made of a number of small planes, we do not make any explicit assumptions about the structure of the scene, such as the assumption by Delage et al. [26] and Hoiem et al. [46] that the scene comprises vertical surfaces standing on a horizontal floor. This allows our model to generalize well, even to scenes with significant non-vertical structure.

Chapter 3

Multi-view Geometry using Monocular Vision

3.1   Introduction

A 3D model built from a single image will almost invariably be an incomplete model of the scene, because many portions of the scene will be missing or occluded. In this section, we will use both the monocular cues and multi-view triangulation cues to create better and larger 3D models.

Given a sparse set of images of a scene, it is sometimes possible to construct a 3D model using techniques such as structure from motion (SFM) [33, 117], which start by taking two or more photographs, then find correspondences between the images, and finally use triangulation to obtain 3D locations of the points. If the images are taken from nearby cameras (i.e., if the baseline distance is small), then these methods often suffer from large triangulation errors for points far-away from the camera.[1] If, conversely, one chooses images taken far apart, then often the change of viewpoint causes the images to become very different, so that finding correspondences becomes difficult, sometimes leading to spurious or missed correspondences. (Worse, the large baseline also means that there may be little overlap between the images, so that

---

[1]I.e., the depth estimates will tend to be inaccurate for objects at large distances, because even small errors in triangulation will result in large errors in depth.

Figure 3.1: Two images taken from a stereo pair of cameras, and the depthmap calculated by a stereo system.

few correspondences may even exist.) These difficulties make purely geometric 3D reconstruction algorithms fail in many cases, specifically when given only a small set of images.

However, when tens of thousands of pictures are available—for example, for frequently-photographed tourist attractions such as national monuments—one can use the information present in *many* views to reliably discard images that have only few correspondence matches. Doing so, one can use only a small subset of the images available ($\sim15\%$), and still obtain a "3D point cloud" for points that were matched using SFM. This approach has been very successfully applied to famous buildings such as the Notre Dame; the computational cost of this algorithm was significant, and required about a week on a cluster of computers [164].

In the specific case of estimating depth from two images taken from a pair of stereo cameras, (Fig. 3.1) the depths are estimated by triangulation using the two images. Over the past few decades, researchers have developed very good stereo vision systems (see [155] for a review). Although these systems work well in many environments, stereo vision is fundamentally limited by the baseline distance between the two cameras. Specifically, their depth estimates tend to be inaccurate when the distances considered are large (because even very small triangulation/angle estimation errors translate to very large errors in distances). Further, stereo vision also tends to fail for textureless regions of images where correspondences cannot be reliably found.

On the other hand, humans perceive depth by seamlessly combining monocular

cues with stereo cues.[2] We believe that monocular cues and (purely geometric) stereo cues give largely orthogonal, and therefore complementary, types of information about depth. Stereo cues are based on the difference between two images and do not depend on the content of the image. Even if the images are entirely random, it would still generate a pattern of disparities (e.g., random dot stereograms [9]). On the other hand, depth estimates from monocular cues are entirely based on the evidence about the environment presented in a single image. In this chapter, we investigate how monocular cues can be integrated with any reasonable stereo system, to obtain better depth estimates than the stereo system alone.

The reason that many geometric "triangulation-based" methods sometimes fail (especially when only a few images of a scene are available) is that they do not make use of the information present in a single image. Therefore, we will extend our MRF model to seamlessly combine triangulation cues and monocular image cues to build a full photo-realistic 3D model of the scene. Using monocular cues will also help us build 3D model of the parts that are visible only in one view.

3.2   Improving stereo vision using monocular Cues

3.2.1   Disparity from stereo correspondence

Depth estimation using stereo vision from two images (taken from two cameras separated by a baseline distance) involves three steps: First, establish correspondences between the two images. Then, calculate the relative displacements (called "disparity") between the features in each image. Finally, determine the 3D depth of the feature relative to the cameras, using knowledge of the camera geometry.

Stereo correspondences give reliable estimates of disparity, except when large portions of the image are featureless (i.e., correspondences cannot be found). Further,

---

[2]Stereo cues: Each eye receives a slightly different view of the world and stereo vision combines the two views to perceive 3D depth [180]. An object is projected onto different locations on the two retinae (cameras in the case of a stereo system), depending on the distance of the object. The retinal (stereo) disparity varies with object distance, and is inversely proportional to the distance of the object. Disparity is typically not an effective cue for estimating small depth variations of objects that are far away.

for a given baseline distance between cameras, the accuracy decreases as the depth values increase. In the limit of very distant objects, there is no observable disparity, and depth estimation generally fails. Empirically, depth estimates from stereo tend to become unreliable when the depth exceeds a certain distance.

Our stereo system finds good feature correspondences between the two images by rejecting pixels with little texture, or where the correspondence is otherwise ambiguous. More formally, we reject any feature where the best match is not significantly better than all other matches within the search window. We use the sum-of-absolute-differences correlation as the metric score to find correspondences [33]. Our cameras (and algorithm) allow sub-pixel interpolation accuracy of 0.2 pixels of disparity. Even though we use a fairly basic implementation of stereopsis, the ideas in this chapter can just as readily be applied together with other, perhaps better, stereo systems.

3.2.2   Modeling uncertainty in Stereo

The errors in disparity are often modeled as either Gaussian [23] or via some other, heavier-tailed distribution (e.g., [171]). Specifically, the errors in disparity have two main causes: (a) Assuming unique/perfect correspondence, the disparity has a small error due to image noise (including aliasing/pixelization), which is well modeled by a Gaussian. (b) Occasional errors in correspondence cause larger errors, which results in a heavy-tailed distribution for disparity [171].

If the standard deviation is σ_g in computing disparity g from stereo images (because of image noise, etc.), then the standard deviation of the depths[3] will be $\sigma_{d,\text{stereo}} \approx \sigma_g/g$. For our stereo system, we have that σ_g is about 0.2 pixels;[4] this is then used to estimate $\sigma_{d,\text{stereo}}$. Note therefore that $\sigma_{d,\text{stereo}}$ is a function of the estimated depth, and specifically, it captures the fact that variance in depth estimates is larger for distant objects than for closer ones.

---

[3]Using the delta rule from statistics: $\text{Var}(f(x)) \approx (f'(x))^2\text{Var}(x)$, derived from a second order Taylor series approximation of $f(x)$. The depth d is related to disparity g as $d = \log(C/g)$, with camera parameters determining C.

[4]One can also envisage obtaining a better estimate of σ_g as a function of a match metric used during stereo correspondence [14], such as normalized sum of squared differences; or learning σ_g as a function of disparity/texture based features.

3.2.3   Probabilistic model

We use our Markov Random Field (MRF) model, which models relations between depths at different points in the image, to incorporate both monocular and stereo cues. Therefore, the depth of a particular patch depends on the monocular features of the patch, on the stereo disparity, and is also related to the depths of other parts of the image.

In our Gaussian and Laplacian MRFs (Eq. 3.1 and 3.2), we now have an additional term $d_{i,stereo}$, which is the depth estimate obtained from disparity.[5] This term models the relation between the depth and the estimate from stereo disparity. The other terms in the models are similar to Eq. 2.1 and 2.2 in Section 2.5.

$$P_G(d|X;\theta,\sigma) = \frac{1}{Z_G}$$
$$\exp\left(-\frac{1}{2}\sum_{i=1}^{M}\left(\frac{(d_i(1)-d_{i,\text{stereo}})^2}{\sigma_{i,\text{stereo}}^2} + \frac{(d_i(1)-x_i^T\theta_r)^2}{\sigma_{1r}^2} + \sum_{s=1}^{3}\sum_{j\in N_s(i)}\frac{(d_i(s)-d_j(s))^2}{\sigma_{2rs}^2}\right)\right)$$
$$(3.1)$$

$$P_L(d|X;\theta,\lambda) = \frac{1}{Z_L}$$
$$\exp\left(-\sum_{i=1}^{M}\left(\frac{|d_i(1)-d_{i,\text{stereo}}|}{\lambda_{i,\text{stereo}}} + \frac{|d_i(1)-x_i^T\theta_r|}{\lambda_{1r}} + \sum_{s=1}^{3}\sum_{j\in N_s(i)}\frac{|d_i(s)-d_j(s)|}{\lambda_{2rs}}\right)\right)$$
$$(3.2)$$

3.2.4   Results on stereo

For these experiments, we collected 257 stereo pairs+depthmaps in a wide-variety of outdoor and indoor environments, with an image resolution of 1024x768 and a depthmap resolution of 67x54. We used 75% of the images/depthmaps for training, and the remaining 25% for hold-out testing.

---

[5]In this work, we directly use $d_{i,stereo}$ as the stereo cue. In [136], we use a library of features created from stereo depths as the cues for identifying a grasp point on objects.

Figure 3.2: Results for a varied set of environments, showing one image of the stereo pairs (column 1), ground truth depthmap collected from 3D laser scanner (column 2), depths calculated by stereo (column 3), depths predicted by using monocular cues only (column 4), depths predicted by using both monocular and stereo cues (column 5). The bottom row shows the color scale for representation of depths. Closest points are 1.2 m, and farthest are 81m. (*Best viewed in color*)

Table 3.1: The average errors (RMS errors gave very similar trends) for various cues and models, on a log scale (base 10).

Algorithm	All	Campus	Forest	Indoor
Baseline	.341	.351	.344	.307
Stereo	.138	.143	.113	.182
Stereo (smooth)	.088	.091	.080	.099
Mono (Gaussian)	.093	.095	.085	.108
Mono (Lap)	.090	.091	.082	.105
Stereo+Mono (Lap)	**.074**	**.077**	**.069**	**.079**

We quantitatively compare the following classes of algorithms that use monocular and stereo cues in different ways:

(i) **Baseline**: This model, trained without any features, predicts the mean value of depth in the training depthmaps.

(ii) **Stereo**: Raw stereo depth estimates, with the missing values set to the mean value of depth in the training depthmaps.

(iii) **Stereo (smooth)**: This method performs interpolation and region filling; using the Laplacian model without the second term in the exponent in Eq. 3.2, and also without using monocular cues to estimate λ_2 as a function of the image.

(iv) **Mono (Gaussian)**: Depth estimates using only monocular cues, without the first term in the exponent of the Gaussian model in Eq. 3.1.

(v) **Mono (Lap)**: Depth estimates using only monocular cues, without the first term in the exponent of the Laplacian model in Eq. 3.2.

(vi) **Stereo+Mono**: Depth estimates using the full model.

Table 3.1 shows that although the model is able to predict depths using monocular cues only ("Mono"), the performance is significantly improved when we combine both mono and stereo cues. The algorithm is able to estimate depths with an error of .074 orders of magnitude, (i.e., 18.6% of multiplicative error because $10^{.074} = 1.186$) which represents a significant improvement over stereo (smooth) performance of .088.

Fig. 3.2 shows that the model is able to predict depthmaps (column 5) in a variety of environments. It also demonstrates how the model takes the best estimates from
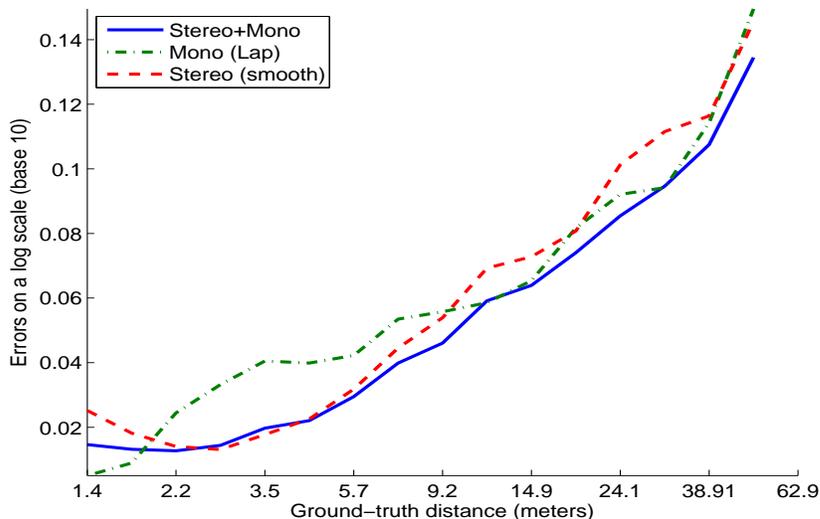
Figure 3.3: The average errors (on a log scale, base 10) as a function of the distance from the camera.

both stereo and monocular cues to estimate more accurate depthmaps. For example, in row 6 (Fig. 3.2), the depthmap generated by stereo (column 3) is very inaccurate; however, the monocular-only model predicts depths fairly accurately (column 4). The combined model uses both sets of cues to produce a better depthmap (column 5). In row 3, stereo cues give a better estimate than monocular ones. We again see that our combined MRF model, which uses both monocular and stereo cues, gives an accurate depthmap (column 5) correcting some mistakes of stereo, such as some far-away regions that stereo predicted as close.

In Fig. 3.3, we study the behavior of the algorithm as a function of the 3D distance from the camera. At small distances, the algorithm relies more on stereo cues, which are more accurate than the monocular cues in this regime. However, at larger distances, the performance of stereo degrades, and the algorithm relies more on monocular cues. Since our algorithm models uncertainties in both stereo and monocular cues, it is able to combine stereo and monocular cues effectively.

We note that monocular cues rely on prior knowledge, learned from the training set, about the environment. This is because monocular 3D reconstruction is an

inherently ambiguous problem. Thus, the monocular cues may not generalize well to images very different from ones in the training set, such as underwater images or aerial photos. In contrast, the stereopsis cues we used are are purely geometric, and therefore should work well even on images taken from very different environments. For example, the monocular algorithm fails sometimes to predict correct depths for objects which are only partially visible in the image (e.g., Fig. 3.2, row 2: tree on the left). For a point lying on such an object, most of the point's neighbors lie outside the image; hence the relations between neighboring depths are less effective here than for objects lying in the middle of an image. However, in many of these cases, the stereo cues still allow an accurate depthmap to be estimated (row 2, column 5).

3.3   Larger 3D models from multiple images

In the previous section, we showed that combining monocular cues and triangulation cues gives better depth estimates. However, the method was applicable to stereo pairs that are calibrated and placed closely together.

Now, we consider creating 3D models from multiple views (which are not necessarily from a calibrated stereo pair) to create a larger 3D model. This problem is harder because the cameras could be farther apart, and many portions could be visible from one camera only.

One of other goals is to create 3D models that are quantitatively correct as well as visually pleasing. We will use plane parameter representation for this purpose.

3.3.1   Representation

Given two small plane (superpixel) segmentations of two images, there is no guarantee that the two segmentations are "consistent," in the sense of the small planes (on a specific object) in one image having a one-to-one correspondence to the planes in the second image of the same object. Thus, at first blush it appears non-trivial to build a 3D model using these segmentations, since it is impossible to associate the planes in one image to those in another. We address this problem by using our MRF to reason
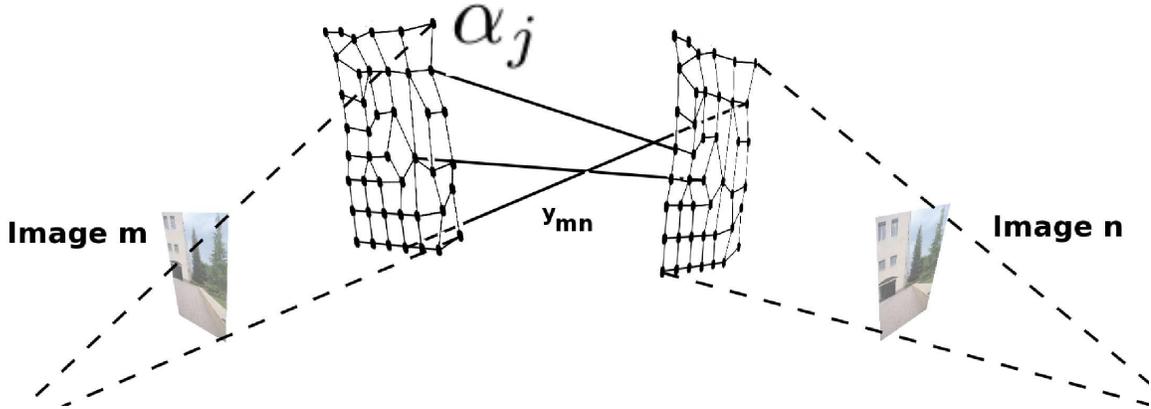
Figure 3.4: An illustration of the Markov Random Field (MRF) for inferring 3D structure. (Only a subset of edges and scales shown.)

simultaneously about the position and orientation of every plane in every image. If two planes lie on the same object, then the MRF will (hopefully) infer that they have exactly the same 3D position. More formally, in our model, the plane parameters α_i^n of each small i^{th} plane in the nth image are represented by a node in our Markov Random Field (MRF). Because our model uses L_1 penalty terms, our algorithm will be able to infer models for which $\alpha_i^n = \alpha_j^m$, which results in the two planes exactly overlapping each other.

3.3.2 Probabilistic model

In addition to the image features/depth, co-planarity, connected structure, and co-linearity properties, we will also consider the depths obtained from triangulation (SFM)—the depth of the point is more likely to be close to the triangulated depth. Similar to the probabilistic model for 3D model from a single image, most of these cues are noisy indicators of depth; therefore our MRF model will also reason about our "confidence" in each of them, using latent variables y_T (Section 3.3.3).

Let $Q^n = [\text{Rotation}, \quad \text{Translation}] \in \mathbb{R}^{3 \times 4}$ (technically SE(3)) be the camera pose when image n was taken (w.r.t. a fixed reference, such as the camera pose of the first image), and let d_T be the depths obtained by triangulation (see Section 3.3.3). We

formulate our MRF as

$$P(\alpha|X, Y, d_T; \theta) \propto \prod_n f_1(\alpha^n | X^n, \nu^n, R^n, Q^n; \theta^n)$$
$$\prod_n f_2(\alpha^n | y^n, R^n, Q^n)$$
$$\prod_n f_3(\alpha^n | d_T^n, y_T^n, R^n, Q^n) \tag{3.3}$$

where, the superscript n is an index over the images, For an image n, α_i^n is the plane parameter of superpixel i in image n. Sometimes, we will drop the superscript for brevity, and write α in place of α^n when it is clear that we are referring to a particular image.

The first term $f_1(\cdot)$ and the second term $f_2(\cdot)$ capture the monocular properties, and are same as in Eq. 2.3. We use $f_3(\cdot)$ to model the errors in the triangulated depths, and penalize the (fractional) error in the triangulated depths d_{Ti} and $d_i = 1/(R_i^T \alpha_i)$. For K^n points for which the triangulated depths are available, we therefore have

$$f_3(\alpha|d_T, y_T, R, Q) \propto \prod_{i=1}^{K^n} \exp\left(-y_{Ti} \left|d_{Ti} R_i^T \alpha_i - 1\right|\right). \tag{3.4}$$

This term places a "soft" constraint on a point in the plane to have its depth equal to its triangulated depth.

MAP Inference: For MAP inference of the plane parameters, we need to maximize the conditional log-likelihood $\log P(\alpha|X, Y, d_T; \theta)$. All the terms in Eq. 3.3 are L_1 norm of a linear function of α; therefore MAP inference is efficiently solved using a Linear Program (LP).

3.3.3   Triangulation matches

In this section, we will describe how we obtained the correspondences across images, the triangulated depths d_T and the "confidences" y_T in the $f_3(\cdot)$ term in Section 3.3.2.

We start by computing 128 SURF features [4], and then calculate matches based on the Euclidean distances between the features found. Then to compute the camera

Figure 3.5: An image showing a few matches (left), and the resulting 3D model (right) without estimating the variables y for confidence in the 3D matching. The noisy 3D matches reduce the quality of the model. (Note the cones erroneously projecting out from the wall.)

poses $Q = [\text{Rotation}, \quad \text{Translation}] \in \mathbb{R}^{3 \times 4}$ and the depths d_T of the points matched, we use bundle adjustment [77] followed by using monocular approximate depths to remove the scale ambiguity. However, many of these 3D correspondences are noisy; for example, local structures are often repeated across an image (e.g., Fig. 3.5, 3.8 and 3.10).[6] Therefore, we also model the "confidence" y_{Ti} in the ith match by using logistic regression to estimate the probability $P(y_{Ti} = 1)$ of the match being correct. For this, we use neighboring 3D matches as a cue. For example, a group of spatially consistent 3D matches is more likely to be correct than a single isolated 3D match. We capture this by using a feature vector that counts the number of matches found

---

[6]Increasingly many cameras and camera-phones come equipped with GPS, and sometimes also accelerometers (which measure gravity/orientation). Many photo-sharing sites also offer geo-tagging (where a user can specify the longitude and latitude at which an image was taken). Therefore, we could also use such geo-tags (together with a rough user-specified estimate of camera orientation), together with monocular cues, to improve the performance of correspondence algorithms. In detail, we compute the approximate depths of the points using monocular image features as $\hat{d} = x^T \theta$; this requires only computing a dot product and hence is fast. Now, for each point in an image B for which we are trying to find a correspondence in image A, typically we would search in a band around the corresponding epipolar line in image A. However, given an approximate depth estimated from from monocular cues, we can limit the search to a rectangular window that comprises only a subset of this band. (See Fig. 3.6.) This would reduce the time required for matching, and also improve the accuracy significantly when there are repeated structures in the scene, e.g., in Fig. 3.7 (See [150] for more details.)

in the present superpixel and in larger surrounding regions (i.e., at multiple spatial scales), as well as measures the relative quality between the best and second best match.
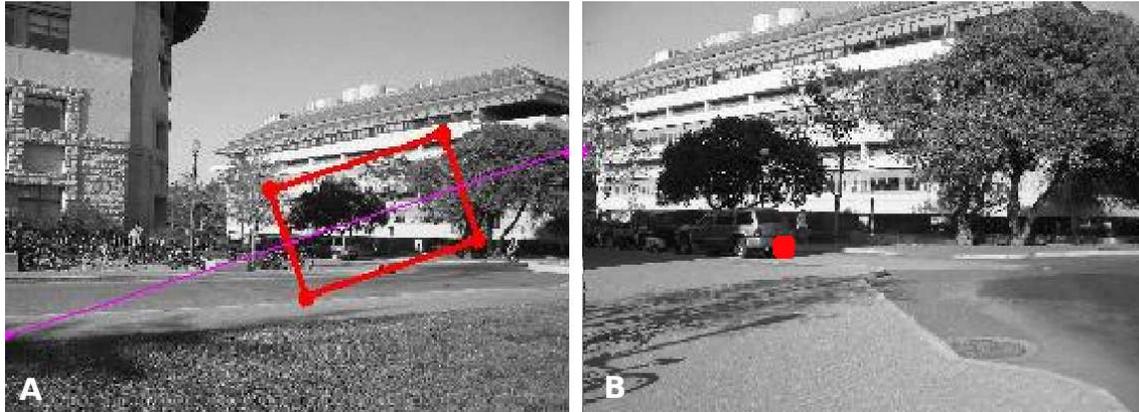
Figure 3.6: Approximate monocular depth estimates help to limit the search area for finding correspondences. For a point (shown as a red dot) in image B, the corresponding region to search in image A is now a rectangle (shown in red) instead of a band around its epipolar line (shown in purple) in image A.

3.3.4   Phantom planes

This cue enforces occlusion constraints across multiple cameras. Concretely, each small plane (superpixel) comes from an image taken by a specific camera. Therefore,
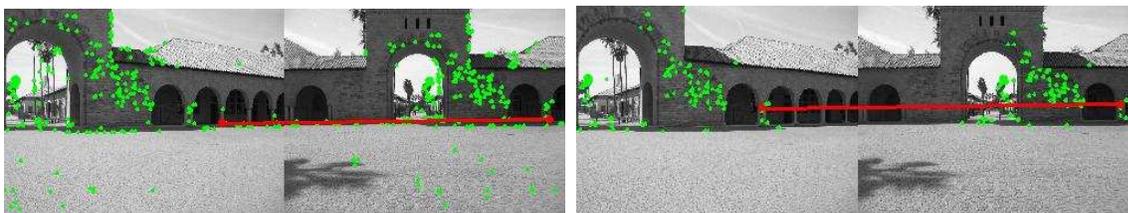
Figure 3.7: (a) Bad correspondences, caused by repeated structures in the world. (b) Use of monocular depth estimates results in better correspondences. Note the these correspondences are still fairly sparse and slightly noisy, and are therefore insufficient for creating a good 3D model if we do not additionally use monocular cues.

there must be an unoccluded view between the camera and the 3D position of that small plane—i.e., the small plane must be visible from the camera location where its picture was taken, and it is not plausible for any other small plane (one from a different image) to have a 3D position that occludes this view. This cue is important because often the connected structure terms, which informally try to "tie" points in two small planes together, will result in models that are inconsistent with this occlusion constraint, and result in what we call "phantom planes"—i.e., planes that are not visible from the camera that photographed it. We penalize the distance between the offending phantom plane and the plane that occludes its view from the camera by finding additional correspondences. This tends to make the two planes lie in exactly the same location (i.e., have the same plane parameter), which eliminates the phantom/occlusion problem.

(a) (b) (c) (d)

(e) (f)
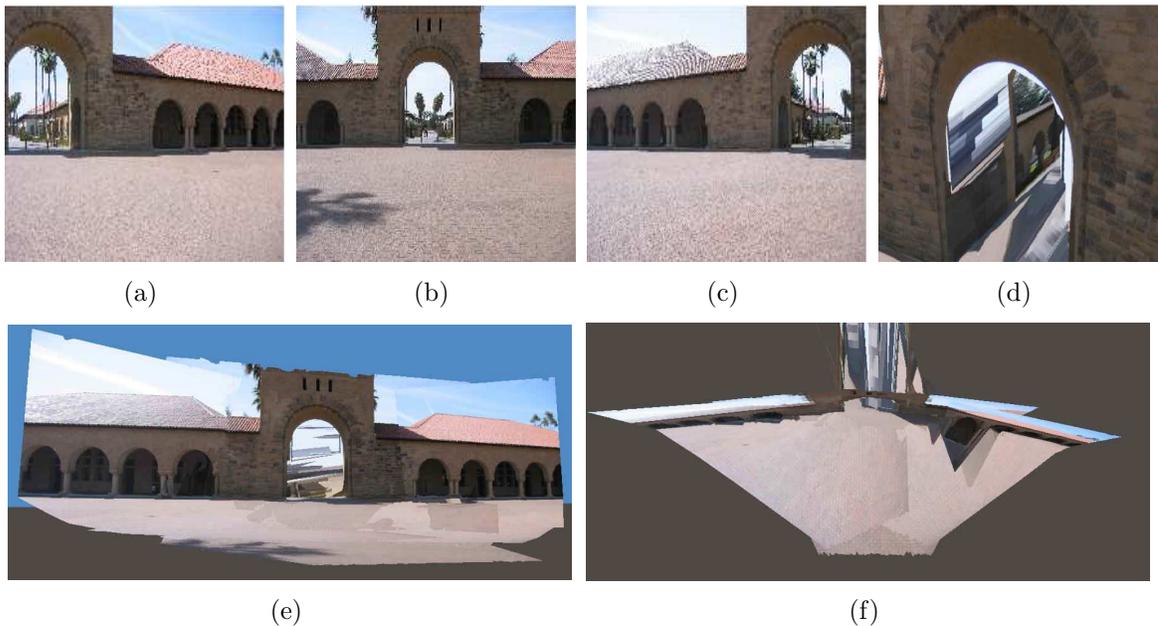
Figure 3.8: (a,b,c) Three original images from different viewpoints; (d,e,f) Snapshots of the 3D model predicted by our algorithm. (f) shows a top-down view; the top part of the figure shows portions of the ground correctly modeled as lying either within or beyond the arch.
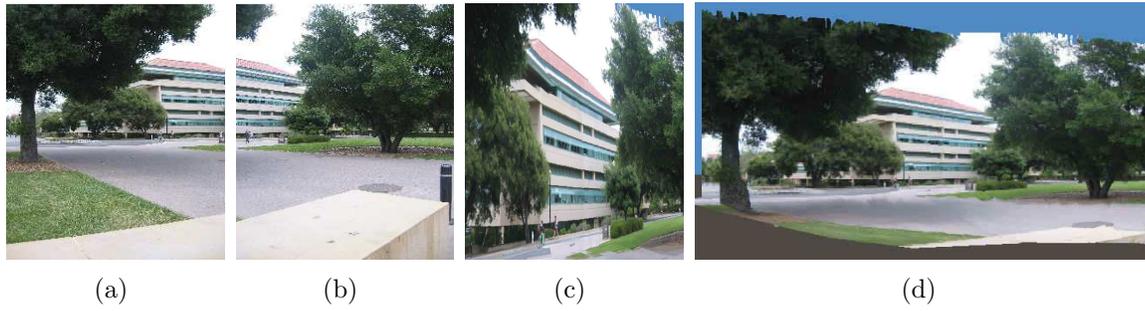
Figure 3.9: (a,b) Two original images with only a little overlap, taken from the same camera location. (c,d) Snapshots from our inferred 3D model.
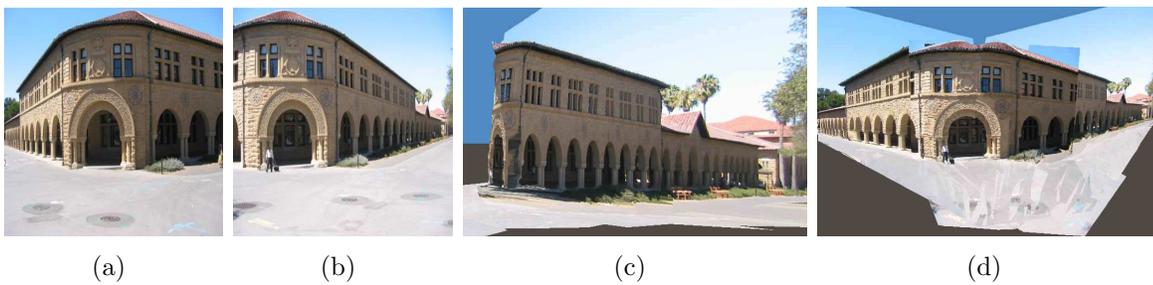
Figure 3.10: (a,b) Two original images with many repeated structures; (c,d) Snapshots of the 3D model predicted by our algorithm.
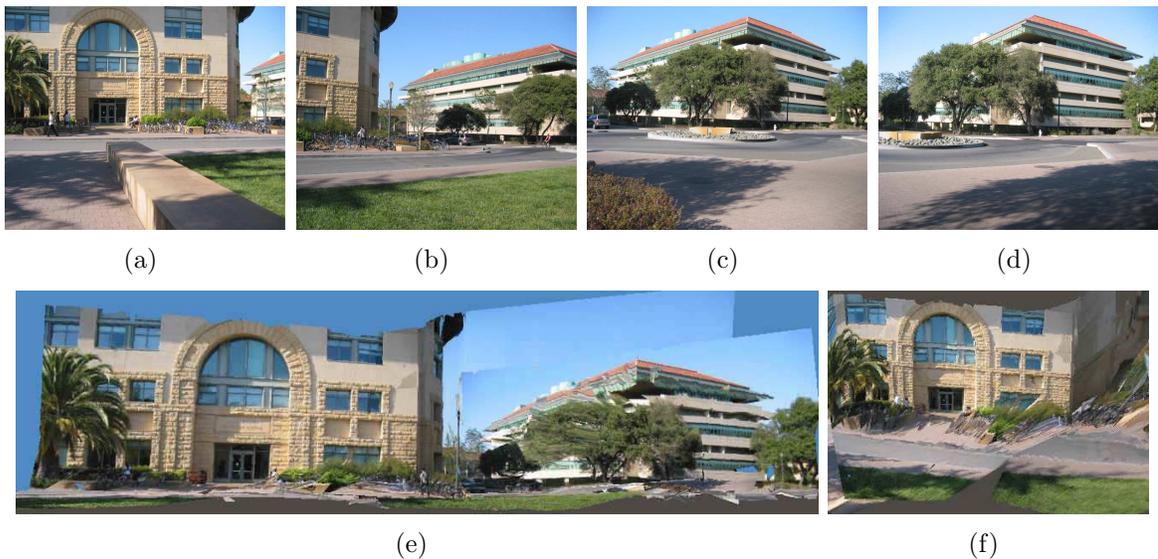
Figure 3.11: (a,b,c,d) Four original images; (e,f) Two snapshots shown from a larger 3D model created using our algorithm.

3.3.5   Experiments

In this experiment, we create a photo-realistic 3D model of a scene given only a few images (with unknown location/pose), even ones taken from very different viewpoints or with little overlap. Fig. 3.8, 3.9, 3.10 and 3.11 show snapshots of some 3D models created by our algorithm. Using monocular cues, our algorithm is able to create full 3D models even when large portions of the images have no overlap (Fig. 3.8, 3.9 and 3.10). In Fig. 3.8, monocular predictions (not shown) from a single image gave approximate 3D models that failed to capture the arch structure in the images. However, using both monocular and triangulation cues, we were able to capture this 3D arch structure. The models are available at:

http://make3d.stanford.edu/research

3.4   Discussion

We have presented a learning algorithm based on MRFs that captures monocular cues and incorporates them into a stereo system so as to obtain significantly improved depth estimates. The monocular cues and (purely geometric) stereo cues largely orthogonal, and therefore complimentary, types of information about depth. We show that by using both monocular and stereo cues, we obtain significantly more accurate depth estimates than is possible using either monocualr or stereo cues alone.

We then preesented an algorithm that builds large 3D reconstructions of outdoor environments, given only a small number of images. Our algorithm segments each of the images into a number of small planes, and simultaneously infers the 3D position and orientation of each plane in every image. Using an MRF, it reasons about both monocular depth cues and triangulation cues, while further taking into account various properties of the real world, such as occlusion, co-planarity, and others.

Chapter 4

Obstacle Avoidance using Monocular Vision

4.1 Introduction

Consider the problem of a small robot trying to navigate through a cluttered environment, for example, a small electric car through a forest (see Figure 4.1). In this chapter, we consider the problem of obstacle avoidance for a remote control car in *unstructured* outdoor environments.

We present an approach in which supervised learning is first used to estimate depths from a single monocular image. For the purposes of driving a car, we do not need to estimate depths at each point in the image (as we did in chapter 2). It is sufficient to be able to estimate a single depth value for each steering direction (i.e., distance to the obstacle in each steering direction). Our learning algorithm is trained using images labeled with ground-truth distances to the closest obstacles. Since we are estimating depth only in each steering direction (a 1D problem) instead of each point in the image (a 2D problem), the learning model is significantly simplified. The resulting algorithm is able to learn monocular vision cues that accurately estimate the relative depths of obstacles in a scene. Reinforcement learning/policy search is then applied within a simulator that renders synthetic scenes. This learns a control policy that selects a steering direction as a function of the vision system's output.

We present results evaluating the predictive ability of the algorithm both on held out test data, and in actual autonomous driving experiments.

Figure 4.1: (a) The remote-controlled car driven autonomously in various cluttered unconstrained environments, using our algorithm. (b) A view from the car, with the chosen steering direction indicated by the red square; the estimated distances to obstacles in the different directions are shown by the bar graph below the image.

Most work on obstacle detection using vision has focused on binocular vision/steoreopsis. In this paper, we present a monocular vision obstacle detection algorithm based on supervised learning. Our motivation for this is two-fold: First, we believe that single-image monocular cues have not been effectively used in most obstacle detection systems; thus we consider it an important open problem to develop methods for obstacle detection that exploit these monocular cues. Second, the dynamics and inertia of high speed driving (5m/s on a small remote control car) means that obstacles must be perceived at a distance if we are to avoid them, and the distance at which standard binocular vision algorithms can perceive them is fundamentally limited by the "baseline" distance between the two cameras and the noise in the observations [24], and thus difficult to apply to our problem.[1]

---

[1]While it is probably possible, given sufficient effort, to build a stereo system for this task, the one commercial system we evaluated (because of vibrations and motion blur) was unable to reliably detect obstacles even at 1m range.

We collected a dataset of several thousand images, each correlated with a laser range scan that gives the distance to the nearest obstacle in each direction. After training on this dataset (using the laser range scans as the ground-truth target labels), a supervised learning algorithm is then able to accurately estimate the distances to the nearest obstacles in the scene. This becomes our basic vision system, the output of which is fed into a higher level controller trained using reinforcement learning.

An addition motivation for our work stems from the observation that the majority of successful robotic applications of RL (including autonomous helicopter flight, some quadruped/biped walking, snake robot locomotion, etc.) have relied on model-based RL [59], in which an accurate model or simulator of the MDP is first built.[2] In control tasks in which the perception problem is non-trivial—such as when the input sensor is a camera—the quality of the simulator we build for the sensor will be limited by the quality of the computer graphics we can implement. We were interested in asking: Does model-based reinforcement learning still make sense in these settings?

Since many of the cues that are useful for estimating depth can be re-created in synthetic images, we implemented a graphical driving simulator. We ran experiments in which the real images and laser scan data was replaced with synthetic images. Our learning algorithm can be trained either on real camera images labeled with ground-truth distances to the closest obstacles, or on a training set consisting of synthetic graphics images. This is interesting in scenarios where real data for training might not be readily available. We also used the graphical simulator to train a reinforcement learning algorithm, and ran experiments in which we systematically varied the level of graphical realism. We show that, surprisingly, even using low- to medium-quality synthetic images, it is often possible to learn depth estimates that give reasonable results on real camera test images. We also show that, by combining information learned from both the synthetic and the real datasets, the resulting vision system performs better than one trained on either one of the two. Similarly, the reinforcement learning controller trained in the graphical simulator also performs well in real world autonomous driving. Videos showing the system's performance in real world driving experiments (using the vision system built with supervised

---

[2]Some exceptions to this include [63, 66].

learning, and the controller learned using reinforcement learning) are available at `http://ai.stanford.edu/~asaxena/rccar/`

4.2 Related work

The most commonly studied for this problem is stereo vision [155]. Depth-from-motion or optical flow is based on motion parallax [3]. Both methods require finding correspondence between points in multiple images separated over space (stereo) or time (optical flow). Assuming that accurate correspondences can be established, both methods can generate very accurate depth estimates. However, the process of searching for image correspondences is computationally expensive and error prone, which can dramatically degrade the algorithm's overall performance.

Gini [36] used single camera vision to drive a indoor robot, but relied heavily on the known color and texture of the ground, and hence does not generalize well and will fail in unstructured outdoor environments. In [118], monocular vision and apprenticeship learning (also called imitation learning) was used to drive a car, but only on highly structured roads and highways with clear lane markings, in which the perception problem is much easier. [72] also successfully applied imitation learning to driving in richer environments, but relied on stereo vision.

4.3 Vision system

For the purposes of driving a car, we do not need to estimate depths at each point in the image. It is sufficient to be able to estimate a single depth value for each steering direction (i.e., distance to the obstacle in each steering direction).

Therefore, we formulate the vision problem as one of depth estimation over vertical stripes of the image. The output of this system will be fed as input to a higher level control algorithm.

In detail, we divide each image into vertical stripes. These stripes can informally be thought of as corresponding to different steering directions. In order to learn to estimate distances to obstacles in outdoor scenes, we collected a dataset of several

Figure 4.2: Laser range scans overlaid on the image. Laser scans give one range estimate per degree.

thousand outdoor images (Fig. 4.2). Each image is correlated with a laser range scan (using a SICK laser range finder, along with a mounted webcam of resolution 352x288, Fig. 4.3), that gives the distance to the nearest obstacle in each stripe of the image. We create a spatially arranged vector of local features capturing monocular depth cues. To capture more of the global context, the feature vector of each stripe is augmented with the features of the stripe to the left and right. We use linear regression on these features to learn to predict the relative distance to the nearest obstacle in each of the vertical stripes.

Since many of the cues that we used to estimate depth can be re-created in synthetic images, we also developed a custom driving simulator with a variable level of graphical realism. By repeating the above learning method on a second data set consisting of synthetic images, we learn depth estimates from synthetic images alone. Additionally, we combine information learned from the synthetic dataset with that learned from real images, to produce a vision system that performs better than either does individually.

Figure 4.3: Rig for collecting correlated laser range scans and real camera images.

4.3.1   Feature vector

Each image is divided into 16 stripes, each one labeled with the distance to the nearest obstacle. In order to emphasize multiplicative rather than additive errors, we converted each distance to a log scale. Early experiments training with linear distances gave poor results (details omitted due to space constraints).

Each stripe is divided into 11 vertically overlapping windows (Fig. 4.4). We denote each window as W_{sr} for $s = 1...16$ stripes and $r = 1...11$ windows per stripe. For each window, coefficients representing texture energies and texture gradients are calculated as described below. Ultimately, the feature vector for a stripe consists of coefficients for that stripe, the stripe to its left, and the stripe to its right. Thus, the spatial arrangement in the feature vector for the windows allows some measure of global structure to be encoded in it.
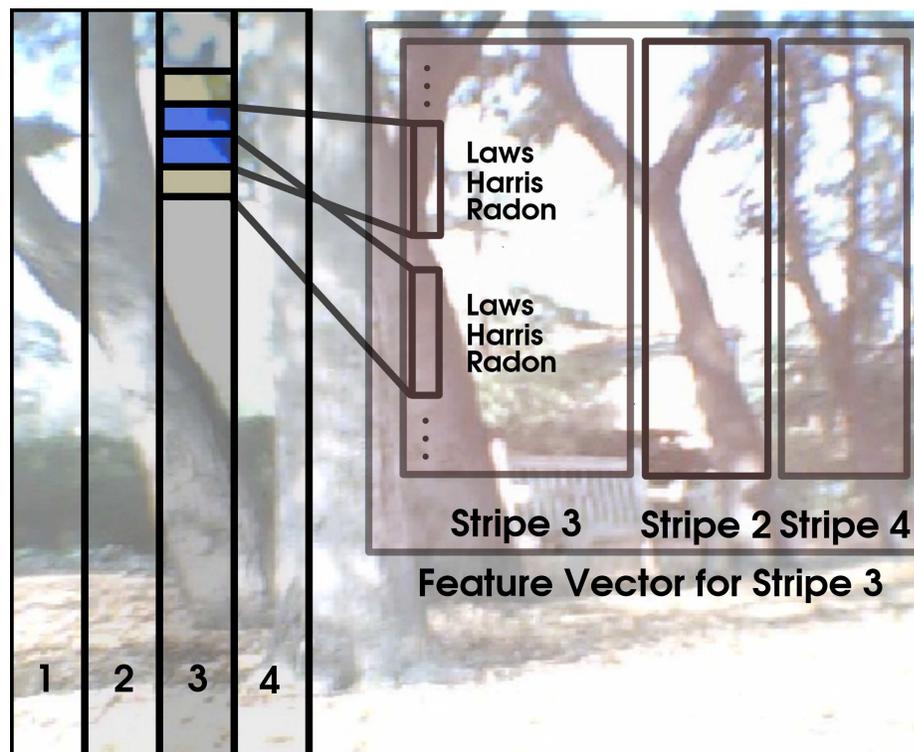
Figure 4.4: For each overlapping window W_{sr}, statistical coefficients (Law's texture energy, Harris angle distribution, Radon) are calculated. The feature vector for a stripe consists of the coefficients for itself, its left column and right column.

Computation speed is the limiting criterion while designing features for this problem—
the vision system has to operate at more than about 10 fps in order for the car to
successfully avoid obstacles. Therefore, we only use the first nine of the filters in
(Fig. 2.4). We apply these 9 masks to the intensity channel image I_Y, and for the
color channels Cb and Cr, we calculate the local averaging mask M_1 only; this gives
a total of 11 texture energy coefficients for each window. Lastly, we estimate the
texture energy E_n for the rth window in the sth stripe (denoted as W_{sr}) as

$$E_n(s, r) = \sum_{x,y \in W_{sr}} |F_n(x, y)| \tag{4.1}$$

In order to calculate a texture gradient that is robust to noise in the image and that
can capture a greater variety of textures, we use a variant of the Radon transform[3] and
a variant of the Harris corner detector.[4] These features measure how the directions
of edges in each window are distributed.

4.3.2   Training

Using the labeled data and the feature vector for each column as described above, we
trained linear models to estimate the log distance to the nearest obstacle in a stripe
of the image. In the simplest case, standard linear regression was used in order to

---

[3]The Radon transform is a standard method for estimating the density of edges at various ori-
entations. [24] We found the distribution of edges for each window W_{sr} using a variant of the this
method. The Radon transform maps an image $I(x, y)$ into a new (θ, ρ) coordinate system, where θ
corresponds to possible edge orientations. Thus the image is now represented as $I_{\mathrm{radon}}(\theta, \rho)$ instead
of $I(x, y)$. For each of 15 discretized values of θ, we pick the highest two values of $I_{\mathrm{radon}}(\theta, \rho)$ by
varying ρ, i.e., $R(\theta) = \mathrm{toptwo}_\rho(g(\theta, \rho))$

[4]A second approach to finding distributions of directional edges over windows in the image is to
use a corner detector. [24] More formally, given an image patch p (in our case 5x5 pixels), the Harris
corner detector first computes the 2x2 matrix M of intensity gradient covariances, and then compares
the relative magnitude of the matrix's two eigenvalues. [24] We use a slightly different variant of this
algorithm, in which rather than thresholding on the smaller eigenvalue, we first calculate the angle
represented by each eigenvector and put the eigenvalues into bins for each of 15 angles. We then
sum up all of the eigenvalues in each bin over a window and use the sums as the input features to
the learning algorithm.

find weights w for N total images and S stripes per image

$$w = \arg\min_w \sum_{i=1}^{N} \sum_{s=1}^{S} \left(w^T x_{is} - \ln(d_i(s)) \right)^2 \tag{4.2}$$

where, x_{is} is the feature vector for stripe s of the ith image, and $d_i(s)$ is the distance to the nearest obstacle in that stripe.

Additionally, we experimented with outlier rejection methods such as robust regression (iteratively re-weighted using the "fair response" function, Huber, 1981) . For this task, this method did not provide significant improvements over linear regression, and simple minimization of the sum of squared errors produced nearly identical results to the more complex methods. All of the results below are given for linear regression.

4.3.3   Synthetic graphics data

We created graphics datasets of synthetic images of typical scenes that we expect to see while driving the car. Graphics data with various levels of detail were created, ranging from simple two-color trees of uniform height without texture; through complex scenes with five different kind of trees of varying heights, along with texture, haze and shadows (Fig. 4.5). The scenes were created by placing trees randomly throughout the environment (by sampling tree locations from a uniform distribution). The width and height of each tree was again chosen uniformly at random between a minimum and maximum value.

There are two reasons why it is useful to supplement real images with synthetic ones. First, a very large amount of synthetica data can be inexpensively created, spanning a variety of environments and scenarios. Comparably large and diverse amounts of real world data would have been much harder to collect. Second, in the synthetic data, there is no noise in the labels of the ground truth distances to obstacles.
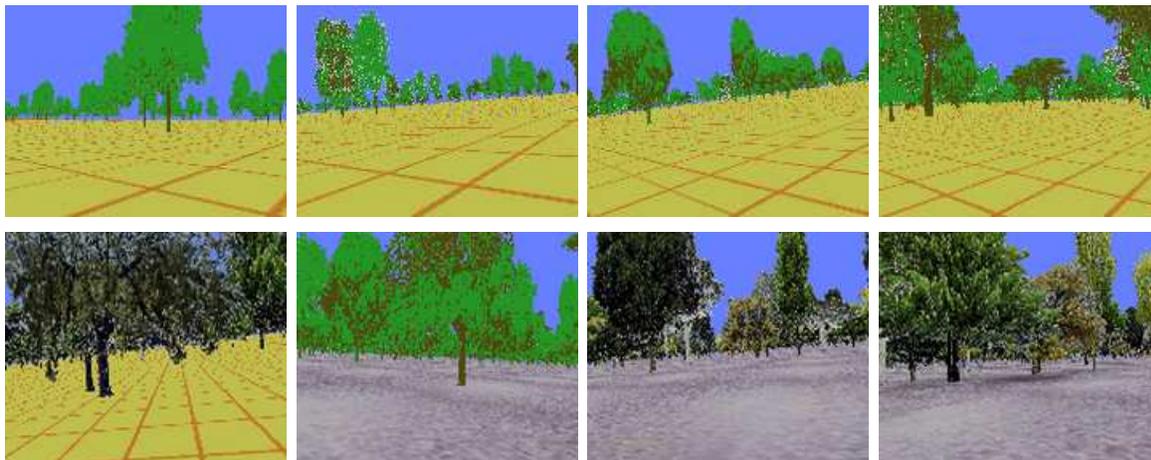
Figure 4.5: Graphics images in order of increasing level of detail. (In color, where available.) (a) Uniform trees, (b) Different types of trees of uniform size, (c) Trees of varying size and type, (d) Increased density of trees, (e) Texture on trees only, (f) Texture on ground only, (g) Texture on both, (h) Texture and Shadows.

4.3.4   Error metrics

Since the ultimate goal of these experiments is to be able to drive a remote control car autonomously, the estimation of distance to nearest obstacle is really only a proxy for true goal of choosing the best steering direction. Successful distance estimation allows a higher level controller to navigate in a dense forest of trees by simply steering in the direction that is the farthest away. The real error metric to optimize in this case should be the mean time to crash. However, since the vehicle will be driving in unstructured terrain, experiments in this domain are not easily repeatable.

In the ith image, let α be a possible steering direction (with each direction corresponding to one of the vertical stripes in the image), let α_{chosen} be steering direction chosen by the vision system (chosen by picking the direction cooresponding to the farthest predicted distance), let $d_i(\alpha)$ be the actual distance to the obstacle in direction α, and let $\hat{d}_i(\alpha)$ be the distance predicted by the learning algorithm. We use the following error metrics:

Depth. The mean error in log-distance estimates of the stripes is defined as

$$E_{depth} = \frac{1}{N}\frac{1}{S}\sum_{i=1}^{N}\sum_{s=1}^{S}\left|\ln(d_i(s)) - \ln(\hat{d}_i(s))\right| \tag{4.3}$$

Relative Depth To calculate the relative depth error, we remove the mean from the true and estimated log-distances for each image. This gives a free-scaling constraint to the depth estimates, reducing the penalty for errors in estimating the scale of the scene as a whole.

Choosen Direction Error. When several steering directions α have nearly identical $d_i(\alpha)$, it is it is unreasonable to expect the learning algorithm to reliably pick out the single best direction. Instead, we might wish only to ensure that $d_i(\alpha_{chosen})$ is nearly as good as the best possible direction. To measure the degree to which this holds, we define the error metric

$$E_\alpha = \frac{1}{N}\sum_{i=1}^{N}\left|\ln(\max_{s}(d_i(s)) - \ln(d_i(\alpha_{chosen}))\right| \tag{4.4}$$

This gives us the difference between the true distance in the chosen direction and the true distance in the actual best direction.

Hazard Rate. Because the car is driving at a high speed (about 5m/s), it will crash into an obstacle when the vision system chooses a column containing obstacles less than about 5m away. Letting d_{Hazard}=5m denote the distance at which an obstacle becomes a hazard (and writing $1\{\cdot\}$ to denote the indicator function), we define the hazard-rate as

$$\text{HazardRate} = \frac{1}{N}\sum_{i=1}^{N}1\{d_i(\alpha_{chosen}) < d_{Hazard}\}, \tag{4.5}$$

4.3.5   Combined vision system

We combined the system trained on synthetic data with the one trained on real images in order to reduce the hazard rate error. The system trained on synthetic images alone had a hazard rate of 11.0%, while the best performing real-image system had a hazard

rate of 2.69%. Training on a combined dataset of real and synthetic images did not produce any improvement over the real-image system, even though the two separately trained algorithms make the same mistake only 1.01% of the time. Thus, we chose a simple voting heuristic to improve the accuracy of the system. Specifically, we ask each system output its top two steering directions, and a vote is taken among these four outputs to select the best steering direction (breaking ties by defaulting to the algorithm trained on real images). This results in a reduction of the hazard rate to 2.04% (Fig. 4.7).[5]

4.4   Control and reinforcement learning

In order to convert the output of the vision system into actual steering commands for the remote control car, a control policy must be developed. The policy controls how aggressively to steer, what to do if the vision system predicts a very near distance for all of the possible steering directions in the camera-view, when to slow down, etc. We model the RC car control problem as a Markov decision process (MDP) [169]. We then used the PEGASUS policy search algorithm [101] (with locally greedy search) to learn the the parameters of our control policy. Here is a description of the six learned parameters[6] of our control policy:

- θ_1: σ of the Gaussian used for spatial smoothing of the predicted distances.

- θ_2: if $\hat{d}_i(\alpha_{chosen}) < \theta_2$, take evasive action rather than steering towards α_{chosen}.

- θ_3: the maximum change in steering angle at any given time step.

- θ_4, θ_5: parameters used to choose which direction to turn if no location in the image is a good steering direction (using the current steering direction and the predicted distances of the left-most and right-most stripes of the image).

- θ_6: the percent of max throttle to use during an evasive turn.

---

[5]The other error metrics are not directly applicable to this combined output.

[6]Since the simulator did not accurately represent complex ground contact forces that act on the car when driving outdoors, two additional parameters were set manually: the maximum throttle, and a turning scale parameter

The reward function was given as

$$R(s) = -|v_{desired} - v_{actual}| - K \cdot \text{Crashed} \tag{4.6}$$

where $v_{desired}$ and v_{actual} are the desired and actual speeds of the car, Crashed is a binary variable stating whether or not the car has crashed in that time step. In our experiments, we used $K = 1000$. Thus, the vehicle attempts to maintain the desired forward speed while minimizing contact with obstacles.

Each trial began with the car initialized at (0,0) in a randomly generated environment and ran for a fixed time horizon. The learning algorithm converged after 1674 iterations of policy search.

Figure 4.6: Experimental setup showing the car and the instruments to control it remotely.

4.5   Experiments

4.5.1   Experimental setup

Our test vehicle is based on an off-the-shelf remote control car (the Traxxas Stampede measuring 12"x16"x9" with wheel clearance of about 2"). We mounted on it a Drag-onFly spy camera (from PointGrey Research, 320x240 pixel resolution, 4mm focal length). The camera transmits images at up to 20 frames per second to a receiver attached to a laptop running a 1.6GHz Pentium. Steering and throttle commands are sent back to the RC transmitter from the laptop via a custom-built serial interface. Finally, the RC transmitter sends the commands directly to the car. The system ran at about 10 frames-per-second (fps).

4.5.2   Results

Table 4.2 shows the test-set error rates obtained using different feature vectors, when training on real camera images. As a baseline for comparing the performance of different features, the first row in the table shows the error obtained by using no feature at all (i.e., always predict the mean distance and steer randomly). The results indicate that texture energy and texture gradient have complementary information, with their combination giving better performance than either alone. Table 4.3 shows the error rates for synthetic images at various degrees of detail. The performance of the vision system trained on synthetic images alone first goes up as we add texture in the images, and add variety in the images, by having different types and sizes of trees. However, it drops when we add shadows and haze. We attribute this to the fact that real shadows and haze are very different from the ones in synthetic images, and the learning algorithm significantly over-fits these images (showing virtually no error at all on synthetic test images).

Figure 4.7 shows hazard rates for the vision system relying on graphics-trained weights alone, real images trained weights alone, and improvement after combining the two systems.

We drove the car in four different locations and under different weather conditions
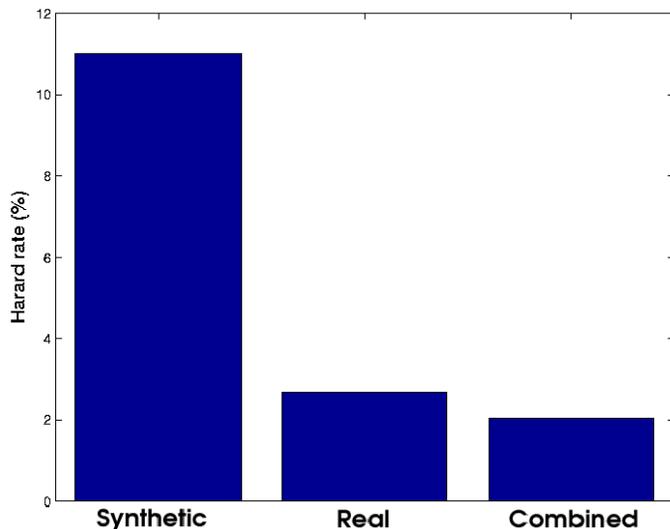
Figure 4.7: Reduction in hazard rate by combining systems trained on synthetic and real data.

Table 4.1: Real driving tests under different terrains

	Obstacle Density	Obstacle Type	Ground	Mean Time (sec)	Max Time (sec)	Avg Speed (m/s)
1	High	Sculptures, Trees, Bushes, Rocks	Uneven Ground, Rocks, Leaves	19	24	4
2	Low	Trees, man-made benches	Even, with Leaves	40	80	6
3	Medium	Trees	Rough with	80	>200	2
4	Medium	Uniform Trees, benches	Leaves, tiles	40	70	5

(see Fig. 4.8 for a few snapshots). The learned controller parameters were directly ported from the simulator to the real world (rescaled to fit in the range of steering and throttle commands allowed by the real vehicle). For each location, the mean time[7]

---

[7]Some of the test locations were had nearby roads, and it was unsafe to allow the car to continue driving outside the bounds of our testing area. In these cases, the car was given a restart and the

Figure 4.8: An image sequence showing the car avoiding a small bush, a difficult obstacle to detect in presence of trees which are larger visually yet more distant. (a) First row shows the car driving to avoid a series of obstacles (b) Second row shows the car view. (Blue square is the largest distance direction. Red square is the chosen steering direction by the controller.) (Best viewed in color)

before crash is given in Table 4.1. All the terrain types had man-made structures (like cars and buildings) in the background. The unstructured testing sites were limited to areas where no training or development images were taken. Videos of the learned controller driving autonomously are available on the web at the URL given in Section 4.1. Performance while driving in the simulator was comparable to the real driving tests.

4.6   Discussion

We have used supervised learning to learn monocular depth estimates, and used them to drive an RC car at high speeds through the environment. Further, the experiments with the graphical simulator show that model-based RL holds great promise even in settings involving complex environments and complex perception. In particular, a vision system trained on computer graphics was able to give reasonable depth estimate on real image data, and a control policy trained in a graphical simulator worked well on real autonomous driving.

---

time was accumulated.

Table 4.2: Test error on 3124 images, obtained on different feature vectors after training data on 1561 images. For the case of "None" (without any features), the system predicts the same distance for all stripes, so relative depth error has no meaning.

FEATURE	E_{depth}	REL DEPTH	E_α	HAZARD RATE
NONE	.900	-	1.36	23.8%
Y (INTENSITY)	.748	.578	.792	9.58%
LAWS ONLY	.648	.527	.630	2.82%
LAWS	.640	.520	.594	2.75%
RADON	.785	.617	.830	6.47%
HARRIS	.687	.553	.713	4.55%
LAW+HARRIS	.612	.508	.566	2.69%
LAWS+RADON	.626	.519	.581	2.88%
HARRIS+RADON	.672	.549	.649	3.20%
LAW+HAR+RAD	.604	.508	.546	2.69%

Table 4.3: Test error on 1561 real images, after training on 667 graphics images, with different levels of graphics realism using Laws features.

TRAIN	E_{depth}	REL DEPTH	E_α	HAZARD RATE
NONE	.900	-	1.36	23.8%
LASER BEST	.604	.508	.546	2.69%
GRAPHICS-8	.925	.702	1.23	15.6%
GRAPHICS-7	.998	.736	1.10	12.8%
GRAPHICS-6	.944	.714	1.04	14.7%
GRAPHICS-5	.880	.673	.984	11.0%
GRAPHICS-4	1.85	1.33	1.63	34.4%
GRAPHICS-3	.900	.694	1.78	38.2%
GRAPHICS-2	.927	.731	1.72	36.4%
GRAPHICS-1	1.27	1.00	1.56	30.3%
G-5 (L+HARRIS)	.929	.713	1.11	14.5%

Chapter 5

Robotic Grasping: Identifying Grasp-point

5.1 Introduction

We consider the problem of grasping novel objects, specifically ones that are being seen for the first time through vision. The ability to grasp a previously unknown object is a challenging yet important problem for robots. For example, a household robot can use this ability to perform tasks such as cleaning up a cluttered room and unloading items from a dishwasher. In this chapter, we will present a learning algorithm that enables a robot to grasp a wide-variety of commonly found objects, such as plates, tape-rolls, jugs, cellphones, keys, screwdrivers, staplers, a thick coil of wire, a strangely shaped power horn, and others.

Modern-day robots can be carefully hand-programmed or "scripted" to carry out many complex manipulation tasks, ranging from using tools to assemble complex machinery, to balancing a spinning top on the edge of a sword [162]. However, autonomously grasping a previously unknown object still remains a challenging problem. If we are trying to grasp a previously known object, or if we are able to obtain a full 3D model of the object, then various approaches such as ones based on friction cones [85], form- and force-closure [7], pre-stored primitives [89], or other methods can be applied. However, in practical scenarios it is often very difficult to obtain a full and
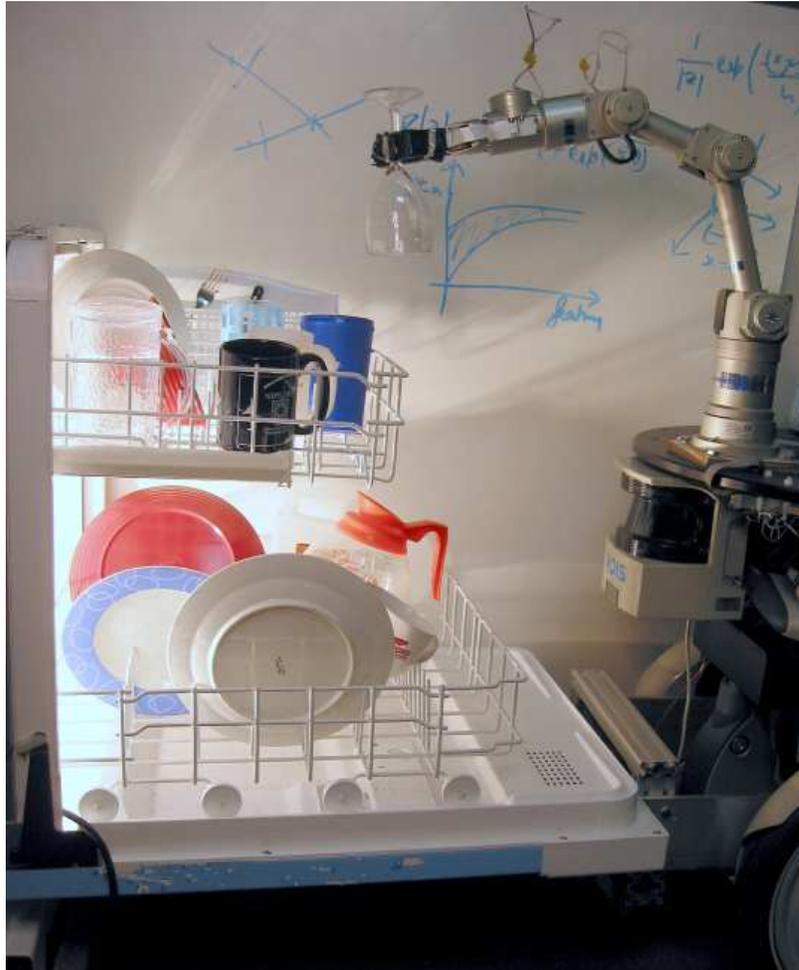
Figure 5.1: Our robot unloading items from a dishwasher.

accurate 3D reconstruction of an object seen for the first time through vision. For stereo systems, 3D reconstruction is difficult for objects without texture, and even when stereopsis works well, it would typically reconstruct only the visible portions of the object. Even if more specialized sensors such as laser scanners (or active stereo) are used to estimate the object's shape, we would still only have a 3D reconstruction of the front face of the object.

In contrast to these approaches, we propose a learning algorithm that neither requires, nor tries to build, a 3D model of the object. Instead it predicts, directly as a function of the images, a point at which to grasp the object. Informally, the

Figure 5.2: Some examples of objects on which the grasping algorithm was tested.

algorithm takes two or more pictures of the object, and then tries to identify a point within each 2D image that corresponds to a good point at which to grasp the object. (For example, if trying to grasp a coffee mug, it might try to identify the mid-point of the handle.) Given these 2D points in each image, we use triangulation to obtain a 3D position at which to actually attempt the grasp. Thus, rather than trying to triangulate every single point within each image in order to estimate depths—as in dense stereo—we only attempt to triangulate one (or at most a small number of) points corresponding to the 3D point where we will grasp the object. This allows us to grasp an object without ever needing to obtain its full 3D shape, and applies even to textureless, translucent or reflective objects on which standard stereo 3D reconstruction fares poorly (see Figure 5.5).

To the best of our knowledge, our work represents the first algorithm capable of grasping novel objects (ones where a 3D model is not available), including ones from novel object classes, that we are perceiving for the first time using vision.

This paper focuses on the task of grasp identification, and thus we will consider only objects that can be picked up without performing complex manipulation.[1] We

---

[1] For example, picking up a heavy book lying flat on a table might require a sequence of complex manipulations, such as to first slide the book slightly past the edge of the table so that the manipulator can place its fingers around the book.

will attempt to grasp a number of common office and household objects such as toothbrushes, pens, books, cellphones, mugs, martini glasses, jugs, keys, knife-cutters, duct-tape rolls, screwdrivers, staplers and markers (see Figure 5.2). We will also address the problem of unloading items from dishwashers.

The remainder of this chapter is structured as follows. In Section 5.2, we describe related work. In Section 5.3, we describe our learning approach, as well as our probabilistic model for inferring the grasping point. In Section 5.5, we describe our robotic manipulation platforms. In Section 5.6, we describe the motion planning/trajectory planning for moving the manipulator to the grasping point. In Section 5.7, we report the results of extensive experiments performed to evaluate our algorithm, and Section 5.8 concludes.

5.2 Related work

Most work in robot manipulation assumes availability of a complete 2D or 3D model of the object, and focuses on designing control and planning methods to achieve a successful and stable grasp. Here, we will discuss in detail prior work that uses learning or vision for robotic manipulation, and refer the reader to [7, 85, 161] for a more general survey of past work in robotic manipulation.

In simulation environments (without real world experiments), learning has been applied to robotic manipulation for several different purposes. For example, [107] used Support Vector Machines (SVM) to estimate the quality of a grasp given a number of features describing the grasp and the object. [48, 49] used partially observable Markov decision processes (POMDP) to choose optimal control policies for two-fingered hands. They also used imitation learning to teach a robot whole-body grasps. [89] used heuristic rules to generate and evaluate grasps for three-fingered hands by assuming that the objects are made of basic shapes such as spheres, boxes, cones and cylinders each with pre-computed grasp primitives. All of these methods assumed full knowledge of the 3D model of the object. Further, these methods were not tested through real-world experiments, but were instead modeled and evaluated in a simulator.

If a 3D model of the object is given, then one can use techniques such as form- and force-closure to compute stable grasps, i.e., the grasps with complete restraint. A grasp with complete restraint prevents loss of contact and thus is very secure. A form-closure of a solid object is a finite set of wrenches (force-moment combinations) applied on the object, with the property that any other wrench acting on the object can be balanced by a positive combination of the original ones [85]. Intuitively, form closure is a way of defining the notion of a "firm grip" of the object [82]. It gurantees maintenance of contact as long as the links of the hand and the object are well approximated as rigid and as long as the joint actuators are sufficiently strong [123]. [75] proposed an efficient algorithm for computing all n-finger form-closure grasps on a polygonal object based on a set of sufficient and necessary conditions for form-closure.

The primary difference between form-closure and force-closure grasps is that force-closure relies on contact friction. This translates into requiring fewer contacts to achieve force-closure than form-closure[73, 47, 90]. [131] demonstrated that a necessary and sufficient condition for force-closure is that the primitive contact wrenches resulted by contact forces positively span the entire wrench space. [102] considered force-closure for planar grasps using friction points of contact. Ponce et al. presented several algorithms for computing force-closure grasps. In [119], they presented algorithms for computing force-closure grasps for two fingers for curved 2D objects. In [120], they considered grasping 3D polyhedral objects with a hand equipped with four hard fingers and assumed point contact with friction. They proved necessary and sufficient conditions for equilibrium and force-closure, and present a geometric characterization of all possible types of four-finger equilibrium grasps. [6] investigated the form and force closure properties of the grasp, and proved the equivalence of force-closure analysis with the study of the equilibria of an ordinary differential equation. All of these methods focussed on inferring the configuration of the hand that results in a stable grasp. However, they assumed that the object's full 3D model is available in advance.

Some work has been done on using vision for real world grasping experiments; however most were limited to grasping 2D planar objects. For uniformly colored planar objects lying on a uniformly colored table top, one can find the 2D contour

(a) Martini glass   (b) Mug   (c) Eraser   (d) Book   (e) Pencil
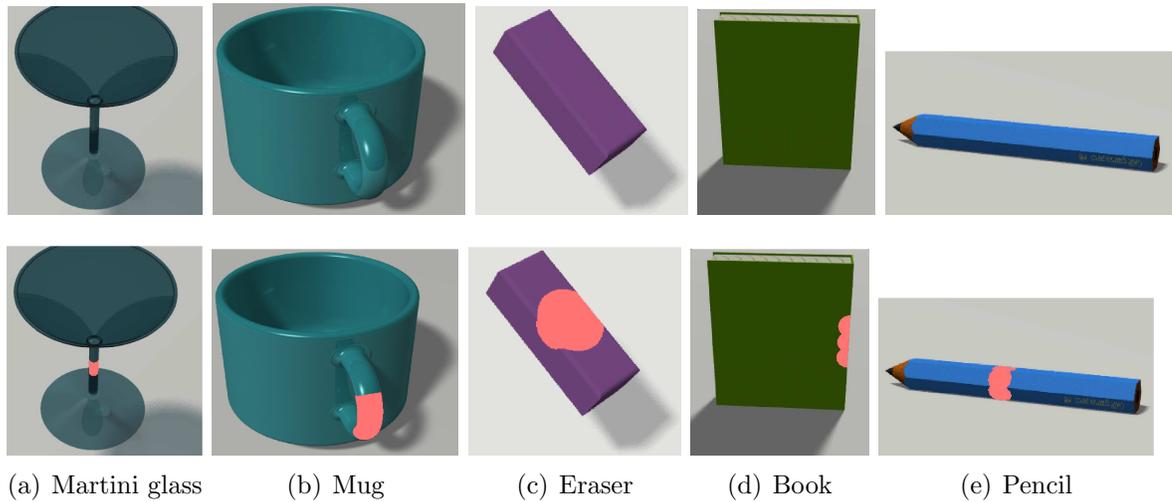
Figure 5.3: The images (top row) with the corresponding labels (shown in red in the bottom row) of the five object classes used for training. The classes of objects used for training were martini glasses, mugs, whiteboard erasers, books and pencils.

of the object quite reliably. Using local visual features (based on the 2D contour) and other properties such as form- and force-closure, the methods discussed below decide the 2D locations at which to place (two or more) fingertips to grasp the object. [113, 19] estimated 2D hand orientation using K-means clustering for simple objects (specifically, square, triangle and round "blocks"). [94, 95] calculated 2D positions of three-fingered grasps from 2D object contours based on feasibility and force closure criteria. [12] also considered the grasping of planar objects and chose the location of the three fingers of a hand by first classifying the object as circle, triangle, square or rectangle from a few visual features, and then using pre-scripted rules based on fuzzy logic. [56] used Q-learning to control the arm to reach towards a spherical object to grasp it using a parallel plate gripper.

If the desired location of the grasp has been identified, techniques such as visual servoing that align the gripper to the desired location [69] or haptic feedback [109] can be used to pick up the object. [114] learned to sequence together manipulation gaits for four specific, known 3D objects. However, they considered fairly simple scenes, and used online learning to associate a controller with the height and width of the

bounding ellipsoid containing the object. For grasping known objects, one can also use Learning-by-Demonstration [54], in which a human operator demonstrates how to grasp an object, and the robot learns to grasp that object by observing the human hand through vision.

The task of identifying where to grasp an object (of the sort typically found in the home or office) involves solving a difficult perception problem. This is because the objects vary widely in appearance, and because background clutter (e.g., dishwasher prongs or a table top with a pattern) makes it even more difficult to understand the shape of a scene. There are numerous robust learning algorithms that can infer useful information about objects, even from a cluttered image. For example, there is a large amount of work on recognition of known object classes (such as cups, mugs, etc.), e.g., [157]. The performance of these object recognition algorithms could probably be improved if a 3D model of the object were available, but they typically do not require such models. For example, that an object is cup-shaped can often be inferred directly from a 2D image. Our approach takes a similar direction, and will attempt to infer grasps directly from 2D images, even ones containing clutter. [134, 149] also showed that given just a single image, it is often possible to obtain the 3D structure of a scene. While knowing the 3D structure by no means implies knowing good grasps, this nonetheless suggests that most of the information in the 3D structure may already be contained in the 2D images, and suggests that an approach that learns directly from 2D images holds promise. Indeed, [83] showed that humans can grasp an object using only one eye.

Our work also takes inspiration from [16], which showed that cognitive cues and previously learned knowledge both play major roles in visually guided grasping in humans and in monkeys. This indicates that learning from previous knowledge is an important component of grasping novel objects. Further, [38] showed that there is a dissociation between recognizing objects and grasping them, i.e., there are separate neural pathways that recognize objects and that direct spatial control to reach and grasp the object. Thus, given only a quick glance at almost any rigid object, most primates can quickly choose a grasp to pick it up, even without knowledge of the object type. Our work represents perhaps a first step towards designing a vision

grasping algorithm which can do the same.

5.3 Learning the grasping point

We consider the general case of grasping objects—even ones not seen before—in 3D cluttered environments such as in a home or office. To address this task, we will use an image of the object to identify a location at which to grasp it.

Because even very different objects can have similar sub-parts, there are certain visual features that indicate good grasps, and that remain consistent across many different objects. For example, jugs, cups, and coffee mugs all have handles; and pens, white-board markers, toothbrushes, screw-drivers, etc. are all long objects that can be grasped roughly at their mid-point (Figure 5.3). We propose a learning approach that uses visual features to predict good grasping points across a large range of objects.

In our approach, we will first predict the 2D location of the grasp in each image; more formally, we will try to identify the projection of a good grasping point onto the image plane. Then, given two (or more) images of an object taken from different camera positions, we will predict the 3D position of a grasping point. If each of these points can be perfectly identified in each image, then we can easily "triangulate" from these images to obtain the 3D grasping point. (See Figure 5.7a.) In practice it is difficult to identify the projection of a grasping point into the image plane (and, if there are multiple grasping points, then the correspondence problem—i.e., deciding which grasping point in one image corresponds to which point in another image— must also be solved). This problem is further exacerbated by imperfect calibration between the camera and the robot arm, and by uncertainty in the camera position if the camera was mounted on the arm itself. To address all of these issues, we develop a probabilistic model over possible grasping points, and apply it to infer a good position at which to grasp an object.

5.3.1 Grasping point

For most objects, there is typically a small region that a human (using a two-fingered pinch grasp) would choose to grasp it; with some abuse of terminology, we will informally refer to this region as the "grasping point," and our training set will contain labeled examples of this region. Examples of grasping points include the center region of the neck for a martini glass, the center region of the handle for a coffee mug, etc. (See Figure 5.3.)

For testing purposes, we would like to evaluate whether the robot successfully picks up an object. For each object in our test set, we define the successful grasp region to be the region where a human/robot using a two-fingered pinch grasp would (reasonably) be expected to successfully grasp the object. The error in predicting the grasping point (reported in Table 5.1) is then defined as the distance of the predicted point from the closest point lying in this region. (See Figure 5.4; this region is usually somewhat larger than that used in the training set, defined in the previous paragraph.) Since our gripper (Figure 5.1) has some passive compliance because of attached foam/rubber, and can thus tolerate about 0.5cm error in positioning, the successful grasping region may extend slightly past the surface of the object. (E.g., the radius of the cylinder in Figure 5.4 is about 0.5cm greater than the actual neck of the martini glass.)

5.3.2 Synthetic data for training

We apply supervised learning to identify patches that contain grasping points. To do so, we require a labeled training set, i.e., a set of images of objects labeled with the 2D location of the grasping point in each image. Collecting real-world data of this sort is cumbersome, and manual labeling is prone to errors. Thus, we instead chose to generate, and learn from, synthetic data that is automatically labeled with the correct grasps. (We also used synthetic data to learn distances in natural scenes for obstacle avoidance system, see 4.3.3.)

In detail, we generate synthetic images along with correct grasps (Figure 5.3)
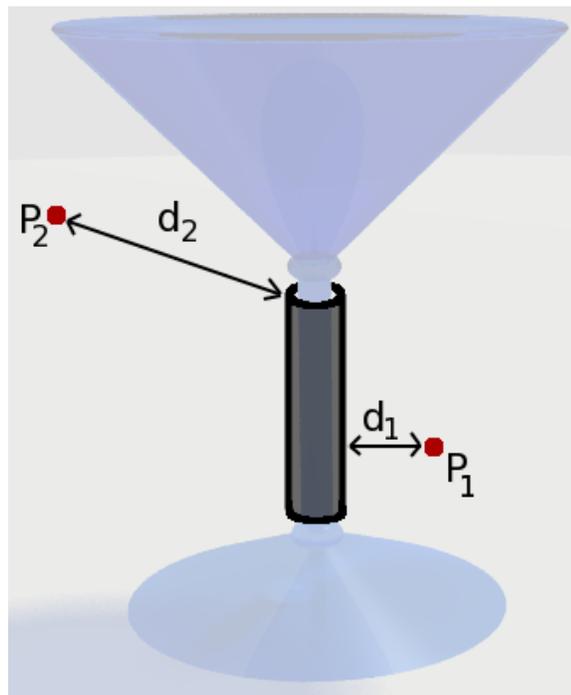
Figure 5.4: An illustration showing the grasp labels. The labeled grasp for a martini glass is on its neck (shown by a black cylinder). For two predicted grasping points P_1 and P_2, the error would be the 3D distance from the grasping region, i.e., d_1 and d_2 respectively.

using a computer graphics ray tracer.[2] There is a relation between the quality of the synthetically generated images and the accuracy of the algorithm. The better the quality of the synthetically generated images and graphical realism, the better the accuracy of the algorithm. Therefore, we used a ray tracer instead of faster, but cruder, OpenGL style graphics. OpenGL style graphics have less realism, the learning performance sometimes *decreased* with added graphical details in the rendered images (see 4.3.3).

The advantages of using synthetic images are multi-fold. First, once a synthetic model of an object has been created, a large number of training examples can be automatically generated by rendering the object under different (randomly chosen) lighting conditions, camera positions and orientations, etc. In addition, to increase the diversity of the training data generated, we randomized different properties of the objects such as color, scale, and text (e.g., on the face of a book). The time-consuming part of synthetic data generation was the creation of the mesh models of the objects. However, there are many objects for which models are available on the internet that can be used with only minor modifications. We generated 2500 examples from synthetic data, comprising objects from five object classes (see Figure 5.3). Using synthetic data also allows us to generate perfect labels for the training set with the exact location of a good grasp for each object. In contrast, collecting and manually labeling a comparably sized set of real images would have been extremely time-consuming.

We have made the data available online at:

http://ai.stanford.edu/~asaxena/learninggrasp/data.html

5.3.3   Features

In our approach, we begin by dividing the image into small rectangular patches, and for each patch predict if it contains a projection of a grasping point onto the image

---

[2]Ray tracing [37] is a standard image rendering method from computer graphics. It handles many real-world optical phenomenon such as multiple specular reflections, textures, soft shadows, smooth curves, and caustics. We used PovRay, an open source ray tracer.

(a)   (b)
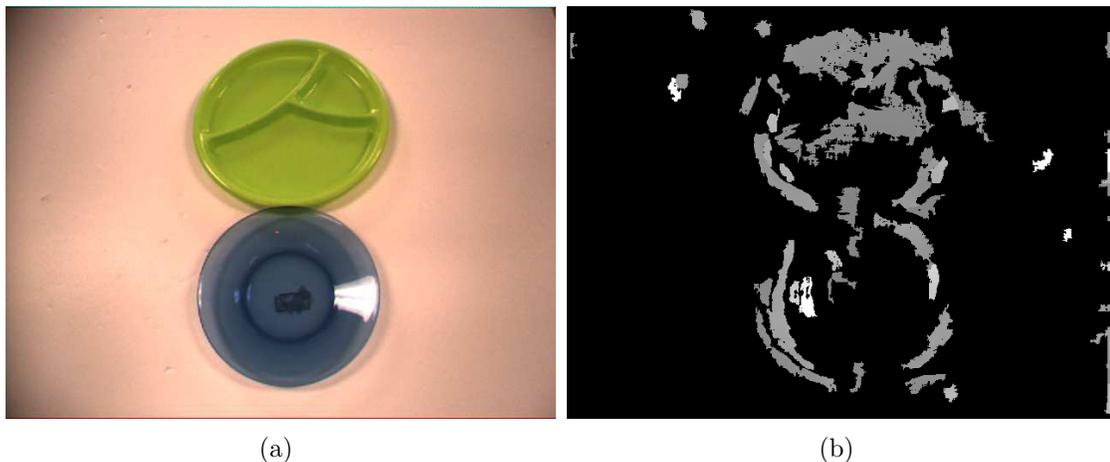
Figure 5.5: (a) An image of textureless/transparent/reflective objects. (b) Depths estimated by our stereo system. The grayscale value indicates the depth (darker being closer to the camera). Black represents areas where stereo vision failed to return a depth estimate.

plane.

Instead of relying on a few visual cues such as presence of edges, we will compute a battery of features for each rectangular patch. By using a large number of different visual features and training on a huge training set (Section 5.3.2), we hope to obtain a method for predicting grasping points that is robust to changes in the appearance of the objects and is also able to generalize well to new objects.

We start by computing features for three types of local cues: edges, textures, and color. [146, 135] We transform the image into YCbCr color space, where Y is the intensity channel, and Cb and Cr are color channels. We compute features representing edges by convolving the intensity channel with 6 oriented edge filters (Figure 2.4). Texture information is mostly contained within the image intensity channel, so we apply 9 Laws' masks to this channel to compute the texture energy. For the color channels, low frequency information is most useful for identifying grasps; our color features are computed by applying a local averaging filter (the first Laws mask) to the 2 color channels. We then compute the sum-squared energy of each of these filter outputs. This gives us an initial feature vector of dimension 17.

To predict if a patch contains a grasping point, local image features centered on

the patch are insufficient, and one has to use more global properties of the object. We attempt to capture this information by using image features extracted at multiple spatial scales (3 in our experiments) for the patch. Objects exhibit different behaviors across different scales, and using multi-scale features allows us to capture these variations. In detail, we compute the 17 features described above from that patch as well as the 24 neighboring patches (in a 5x5 window centered around the patch of interest). This gives us a feature vector x of dimension $1 * 17 * 3 + 24 * 17 = 459$.

| (a) Coffee Pot | (b) Duct tape | (c) Marker | (d) Mug | (e) Synthetic Martini Glass |
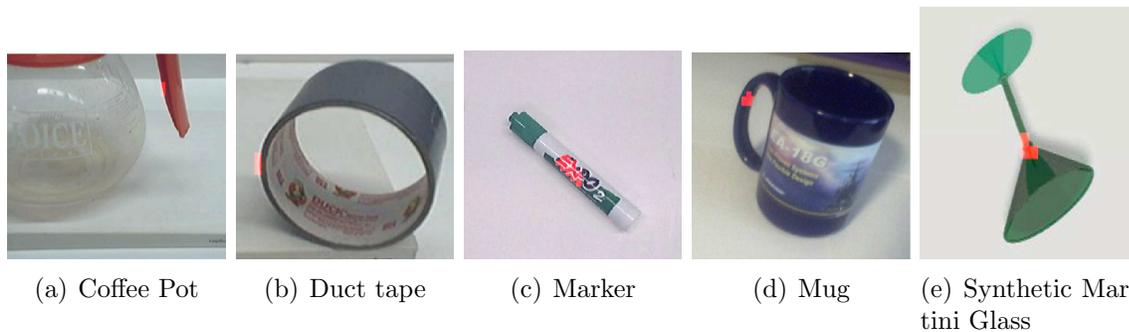
Figure 5.6: Grasping point classification. The red points in each image show the locations most likely to be a grasping point, as predicted by our logistic regression model. (Best viewed in color.)

Although we rely mostly on image-based features for predicting the grasping point, some robots may be equipped with range sensors such as a laser scanner or a stereo camera. In these cases (Section 5.7.3), we also compute depth-based features to improve performance. More formally, we apply our texture based filters to the depth image obtained from a stereo camera, append them to the feature vector used in classification, and thus obtain a feature vector $x_s \in \mathbb{R}^{918}$. Applying these texture based filters this way has the effect of computing relative depths, and thus provides information about 3D properties such as curvature. However, the depths given by a stereo system are sparse and noisy (Figure 5.5) because many objects we consider are textureless or reflective. Even after normalizing for the missing depth readings, these features improved performance only marginally.

5.3.4 Probabilistic model

Using our image-based features, we will first predict whether each region in the image contains the projection of a grasping point. Then in order to grasp an object, we will statistically "triangulate" our 2D predictions to obtain a 3D grasping point.

In detail, on our manipulation platforms (Section 5.5), we have cameras mounted either on the wrist of the robotic arm (Figure 5.11) or on a frame behind the robotic arm (Figure 5.9). When the camera is mounted on the wrist, we command the arm to move the camera to two or more positions, so as to acquire images of the object from different viewpoints. However, there are inaccuracies in the physical positioning of the arm, and hence there is some slight uncertainty in the position of the camera when the images are acquired. We will now describe how we model these position errors.

Formally, let C be the image that would have been taken if the actual pose of the camera was exactly equal to the measured pose (e.g., if the robot had moved exactly to the commanded position and orientation, in the case of the camera being mounted on the robotic arm). However, due to positioning error, instead an image $\hat{C}$ is taken from a slightly different location. Let (u, v) be a 2D position in image C, and let $(\hat{u}, \hat{v})$ be the corresponding image position in $\hat{C}$. Thus $C(u, v) = \hat{C}(\hat{u}, \hat{v})$, where $C(u, v)$ is the pixel value at (u, v) in image C. The errors in camera position/pose should usually be small,[3] and we model the difference between (u, v) and $(\hat{u}, \hat{v})$ using an additive Gaussian model: $\hat{u} = u + \epsilon_u$, $\hat{v} = v + \epsilon_v$, where $\epsilon_u, \epsilon_v \sim N(0, \sigma^2)$.

Now, to predict which locations in the 2D image are grasping points (Figure 5.6), we define the class label $z(u, v)$ as follows. For each location (u, v) in an image C, $z(u, v) = 1$ if (u, v) is the projection of a grasping point onto the image plane, and $z(u, v) = 0$ otherwise. For a corresponding location $(\hat{u}, \hat{v})$ in image $\hat{C}$, we similarly define $\hat{z}(\hat{u}, \hat{v})$ to indicate whether position $(\hat{u}, \hat{v})$ represents a grasping point in the image $\hat{C}$. Since (u, v) and $(\hat{u}, \hat{v})$ are corresponding pixels in C and $\hat{C}$, we assume

---

[3]The robot position/orientation error is typically small (position is usually accurate to within 1mm), but it is still important to model this error. From our experiments (see Section 5.7), if we set $\sigma^2 = 0$, the triangulation is highly inaccurate, with average error in predicting the grasping point being 15.4 cm, as compared to 1.8 cm when appropriate σ^2 is chosen.

$\hat{z}(\hat{u}, \hat{v}) = z(u, v)$. Thus:

$$P(z(u, v) = 1|C) = P(\hat{z}(\hat{u}, \hat{v}) = 1|\hat{C})$$
$$= \int_{\epsilon_u} \int_{\epsilon_v} P(\epsilon_u, \epsilon_v) P(\hat{z}(u + \epsilon_u, v + \epsilon_v) = 1|\hat{C}) d\epsilon_u d\epsilon_v \qquad (5.1)$$

Here, $P(\epsilon_u, \epsilon_v)$ is the (Gaussian) density over ϵ_u and ϵ_v. We then use logistic regression to model the probability of a 2D position $(u + \epsilon_u, v + \epsilon_v)$ in $\hat{C}$ being a good grasping point:

$$P(\hat{z}(u + \epsilon_u, v + \epsilon_v) = 1|\hat{C}) = P(\hat{z}(u + \epsilon_u, v + \epsilon_v) = 1|x; \theta)$$
$$= 1/(1 + e^{-x^T \theta}) \qquad (5.2)$$

where $x \in \mathbb{R}^{459}$ are the features for the rectangular patch centered at $(u + \epsilon_u, v + \epsilon_v)$ in image $\hat{C}$ (described in Section 5.3.3). The parameter of this model $\theta \in \mathbb{R}^{459}$ is learned using standard maximum likelihood for logistic regression: $\theta^* = \arg\max_\theta \prod_i P(z_i|x_i; \theta)$, where (x_i, z_i) are the synthetic training examples (image patches and labels), as described in Section 5.3.2. Figure 5.6a-d shows the result of applying the learned logistic regression model to some real (non-synthetic) images.

3D grasp model: Given two or more images of a new object from different camera positions, we want to infer the 3D position of the grasping point. (See Figure 5.7.) Because logistic regression may have predicted multiple grasping points per image, there is usually ambiguity in the correspondence problem (i.e., which grasping point in one image corresponds to which graping point in another). To address this while also taking into account the uncertainty in camera position, we propose a probabilistic model over possible grasping points in 3D space. In detail, we discretize the 3D work-space of the robotic arm into a regular 3D grid $G \subset \mathbb{R}^3$, and associate with each grid element j a random variable y_j, so that $y_j = 1$ if grid cell j contains a grasping point, and $y_j = 0$ otherwise.

From each camera location $i = 1, ..., N$, one image is taken. In image C_i, let the ray passing through (u, v) be denoted $R_i(u, v)$. Let $G_i(u, v) \subset G$ be the set of grid-cells through which the ray $R_i(u, v)$ passes. Let $r_1, ...r_K \in G_i(u, v)$ be the indices of
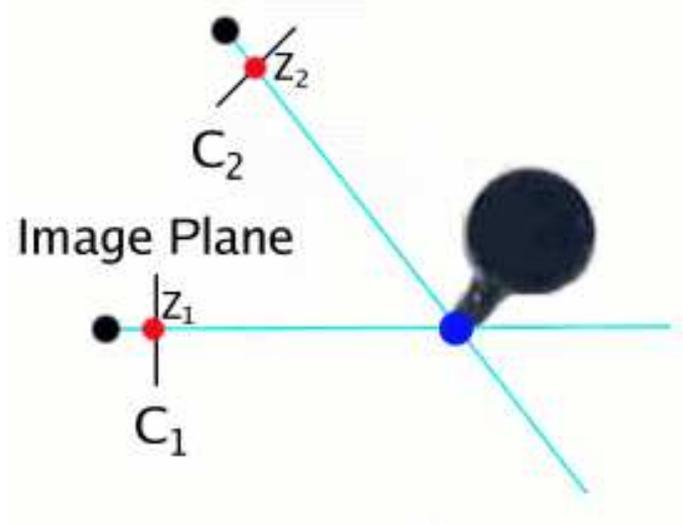
Figure 5.7: (a) Diagram illustrating rays from two images C_1 and C_2 intersecting at a grasping point (shown in dark blue). (Best viewed in color.)

the grid-cells lying on the ray $R_i(u, v)$.

We know that if any of the grid-cells r_j along the ray represent a grasping point, then its projection is a grasp point. More formally, $z_i(u, v) = 1$ if and only if $y_{r_1} = 1$ or $y_{r_2} = 1$ or ... or $y_{r_K} = 1$. For simplicity, we use a (arguably unrealistic) naive Bayes-like assumption of independence, and model the relation between $P(z_i(u, v) = 1|C_i)$ and $P(y_{r_1} = 1$ or ... or $y_{r_K} = 1|C_i)$ as

$$P(z_i(u, v) = 0|C_i) = P(y_{r_1} = 0, ..., y_{r_K} = 0|C_i)$$
$$= \prod_{j=1}^{K} P(y_{r_j} = 0|C_i) \tag{5.3}$$

Assuming that any grid-cell along a ray is equally likely to be a grasping point, this therefore gives

$$P(y_{r_j} = 1|C_i) = 1 - (1 - P(z_i(u, v) = 1|C_i))^{1/K} \tag{5.4}$$

Next, using another naive Bayes-like independence assumption, we estimate the

probability of a particular grid-cell $y_j \in G$ being a grasping point as:

$$P(y_j = 1|C_1, ..., C_N) = \frac{P(y_j = 1)P(C_1, ..., C_N|y_j = 1)}{P(C_1, ..., C_N)}$$

$$= \frac{P(y_j = 1)}{P(C_1, ..., C_N)} \prod_{i=1}^{N} P(C_i|y_j = 1)$$

$$= \frac{P(y_j = 1)}{P(C_1, ..., C_N)} \prod_{i=1}^{N} \frac{P(y_j = 1|C_i)P(C_i)}{P(y_j = 1)}$$

$$\propto \prod_{i=1}^{N} P(y_j = 1|C_i) \tag{5.5}$$

where $P(y_j = 1)$ is the prior probability of a grid-cell being a grasping point (set to a constant value in our experiments). One can envision using this term to incorporate other available information, such as known height of the table when the robot is asked to pick up an object from a table. Using Equations 5.1, 5.2, 5.4 and 5.5, we can now compute (up to a constant of proportionality that does not depend on the grid-cell) the probability of any grid-cell y_j being a valid grasping point, given the images.[4]

Stereo cameras: Some robotic platforms have stereo cameras (e.g., the robot in Figure 5.10); therefore we also discuss how our probabilistic model can incorporate stereo images. From a stereo camera, since we also get a depth value $w(u, v)$ for each location (u, v) in the image,[5] we now obtain a 3D image $C(u, v, w(u, v))$ for 3D positions (u, v, w).

However because of camera positioning errors (as discussed before), we get $\hat{C}(\hat{u}, \hat{v}, \hat{w}(\hat{u}, \hat{v}))$ instead of actual image $C(u, v, w(u, v))$. We again model the difference between

---

[4]In about 2% of the trials described in Section 5.7.2, grasping failed because the algorithm found points in the images that did not actually correspond to each other. (E.g., in one image the point selected may correspond to the midpoint of a handle, and in a different image a different point may be selected that corresponds to a different part of the same handle.) Thus, triangulation using these points results in identifying a 3D point that does not lie on the object. By ensuring that the pixel values in a small window around each of the corresponding points are similar, one would be able to reject some of these spurious correspondences.

[5]The depths estimated from a stereo camera are very sparse, i.e., the stereo system finds valid points only for a few pixels in the image. (see Figure 5.5) Therefore, we still mostly rely on image features to find the grasp points. The pixels where the stereo camera was unable to obtain a depth are treated as regular (2D) image pixels.

(u, v, w) and $(\hat{u}, \hat{v}, \hat{w})$ using an additive Gaussian: $\hat{u} = u + \epsilon_u$, $\hat{v} = v + \epsilon_v$, $\hat{w} = w + \epsilon_w$, where $\epsilon_u, \epsilon_v \sim N(0, \sigma_{uv}^2)$. $\epsilon_w \sim N(0, \sigma_w^2)$. Now, for our class label $z(u, v, w)$ in the 3D image, we have:

$$
\begin{aligned}
P(z(u, v, w) = 1|C) &= P(\hat{z}(\hat{u}, \hat{v}, \hat{w}) = 1|\hat{C}) \\
&= \int_{\epsilon_u} \int_{\epsilon_v} \int_{\epsilon_w} P(\epsilon_u, \epsilon_v, \epsilon_w) \\
&\quad P(\hat{z}(u + \epsilon_u, v + \epsilon_v, w + \epsilon_w) = 1|\hat{C}) d\epsilon_u d\epsilon_v d\epsilon_w
\end{aligned}
\tag{5.6}
$$

Here, $P(\epsilon_u, \epsilon_v, \epsilon_w)$ is the (Gaussian) density over ϵ_u, ϵ_v and ϵ_w. Now our logistic regression model is

$$
\begin{aligned}
P(\hat{z}(\hat{u}, \hat{v}, \hat{w}(\hat{u}, \hat{v})) = 1|\hat{C}) &= P(\hat{z}(\hat{u}, \hat{v}, \hat{w}(\hat{u}, \hat{v})) = 1|x_s; \theta_s) \\
&= 1/(1 + e^{-x_s^T \theta_s})
\end{aligned}
\tag{5.7}
$$

where $x_s \in \mathbb{R}^{918}$ are the image and depth features for the rectangular patch centered at $(\hat{u}, \hat{v})$ in image $\hat{C}$ (described in Section 5.3.3). The parameter of this model $\theta_s \in \mathbb{R}^{918}$ is learned similarly.

Now to use a stereo camera in estimating the 3D grasping point, we use

$$
P(y_j = 1|C_i) = P(z_i(u, v, w(u, v)) = 1|C_i)
\tag{5.8}
$$

in Eq. 5.5 in place of Eq. 5.4 when C_i represents a stereo camera image with depth information at (u, v).

This framework allows predictions from both regular and stereo cameras to be used together seamlessly, and also allows predictions from stereo cameras to be useful even when the stereo system failed to recover depth information at the predicted grasp point.

5.3.5   MAP inference

Given a set of images, we want to infer the most likely 3D location of the grasping point. Therefore, we will choose the grid cell j in the 3D robot workspace that maximizes the conditional log-likelihood $\log P(y_j = 1|C_1, ..., C_N)$ in Eq. 5.5. More formally, let there be N_C regular cameras and $N - N_C$ stereo cameras. Now, from Eq. 5.4 and 5.8, we have:

$$
\arg\max_j \ \log P(y_j = 1|C_1, ..., C_N)
$$

$$
= \arg\max_j \ \log \prod_{i=1}^{N} P(y_j = 1|C_i)
$$

$$
= \arg\max_j \ \sum_{i=1}^{N_C} \log\left(1 - (1 - P(z_i(u,v) = 1|C_i))^{1/K}\right)
$$

$$
+ \sum_{i=N_C+1}^{N} \log\left(P(z_i(u,v,w(u,v)) = 1|C_i)\right) \tag{5.9}
$$

where $P(z_i(u,v) = 1|C_i)$ is given by Eq. 5.1 and 5.2 and $P(z_i(u,v,w(u,v)) = 1|C_i)$ is given by Eq. 5.6 and 5.7.

A straightforward implementation that explicitly computes the sum above for every single grid-cell would give good grasping performance, but be extremely inefficient (over 110 seconds). Since there are only a few places in an image where $P(\hat{z}_i(u,v) = 1|C_i)$ is significantly greater than zero, we implemented a counting algorithm that explicitly considers only grid-cells y_j that are close to at least one ray $R_i(u,v)$. (Grid-cells that are more than 3σ distance away from all rays are highly unlikely to be the grid-cell that maximizes the summation in Eq. 5.9.) This counting algorithm efficiently accumulates the sums over the grid-cells by iterating over all N images and rays $R_i(u,v)$,[6] and results in an algorithm that identifies a 3D grasping position in 1.2 sec.

---

[6]In practice, we found that restricting attention to rays where $P(\hat{z}_i(u,v) = 1|C_i) > 0.1$ allows us to further reduce the number of rays to be considered, with no noticeable degradation in performance.

5.4 Predicting orientation

In [141], we presented an algorithm for predicting the 3D orientation of an object from its image. Here, for the sake of simplicity, we will present the learning algorithm in the context of grasping using our 5-dof arm on STAIR 1.

As discussed in Section 5.6, our task is to predict the 2D wrist orientation θ of the gripper (at the predicted grasping point) given the image. For example, given a picture of a closed book (which we would like to grasp at its edge), we should choose an orientation in which the robot's two fingers are parallel to the book's surfaces, rather than perpendicular to the book's cover.

Since our robotic arm has a parallel plate gripper comprising two fingers that close in parallel, a rotation of π results in similar configuration of the gripper. This results in a discontinuity at $\theta = \pi$, in that the orientation of the gripper at $\theta = \pi$ is equivalent to $\theta = 0$. Therefore, to handle this symmetry, we will represent angles via $y(\theta) = [\cos(2\theta), \sin(2\theta)] \in \mathbb{R}^2$. Thus, $y(\theta + N\pi) = [\cos(2\theta + 2N\pi), \sin(2\theta + 2N\pi)] = y(\theta)$.

Now, given images features x, we model the conditional distribution of y as a multi-variate Gaussian:

$$P(y|x; w, K) = (2\pi)^{-n/2}|K|^{1/2}$$
$$\exp\left[-\frac{1}{2}(y - w^T x)^T K(y - w^T x)\right]$$

where K^{-1} is a covariance matrix. The parameters of this model w and K are learnt by maximizing the conditional log likelihood $\log \prod_i P(y_i|x_i; w)$.

Now when given a new image, our MAP estimate for y is given as follows. Since $||y||_2 = 1$, we will choose

$$y^* = \arg\max_{y:||y||_2=1} \log P(y|x; w, K) = \arg\max_{y:||y||_2=1} y^T K q$$

for $q = w^T x$. (This derivation assumed $K = \sigma^2 I$ for some σ^2, which will roughly hold true if θ is chosen uniformly in the training set.) The closed form solution of this is $y = Kq/||Kq||_2$.

In our robotic experiments, typically 30° accuracy is required to successfully grasp an object, which our algorithm almost always attains. In an example, Figure 5.8 shows the predicted orientation for a pen.
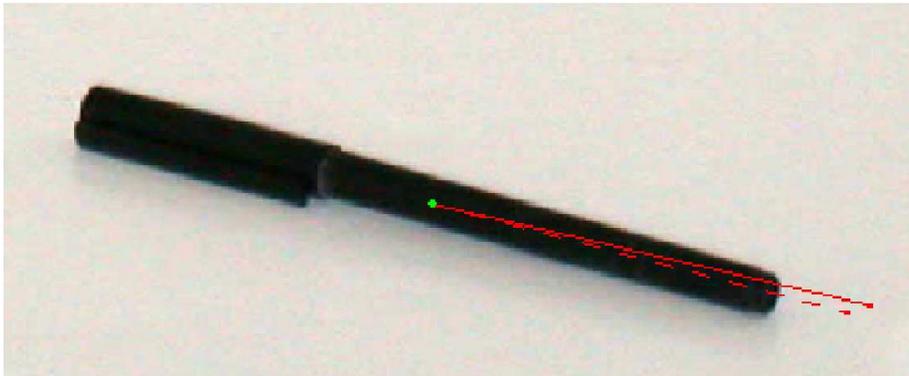
Figure 5.8: Predicted orientation at the grasping point for pen. Dotted line represents the true orientation, and solid line represents the predicted orientation.

5.5   Robot platforms

Our experiments were performed on two robots built for the STAIR (STanford AI Robot) project.[7] Each robot has an arm and other equipment such as cameras, computers, etc. (See Fig. 5.9 and 5.10.) The STAIR platforms were built as part of a project whose long-term goal is to create a general purpose household robot that can navigate in indoor environments, pick up and interact with objects and tools, and carry out tasks such as tidy up a room or prepare simple meals. Our algorithms for grasping novel objects represent perhaps a small step towards achieving some of these goals.

STAIR 1 uses a harmonic arm (Katana, by Neuronics), and is built on top of a Segway robotic mobility platform. Its 5-dof arm is position-controlled and has a parallel plate gripper. The arm has a positioning accuracy of ± 1 mm, a reach of 62cm, and can support a payload of 500g. Our vision system used a low-quality webcam (Logitech Quickcam Pro 4000) mounted near the end effector and a stereo

---

[7]See http://www.cs.stanford.edu/group/stair for details.

camera (Bumblebee, by Point Grey Research). In addition, the robot has a laser scanner (SICK LMS-291) mounted approximately 1m above the ground for navigation purposes. (We used the webcam in the experiments on grasping novel objects, and the Bumblebee stereo camera in the experiments on unloading items from dishwashers.)

STAIR 2 sits atop a holonomic mobile base, and its 7-dof arm (WAM, by Barrett Technologies) can be position or torque-controlled, is equipped with a three-fingered hand, and has a positioning accuracy of ± 0.6 mm. It has a reach of 1m and can support a payload of 3kg. Its vision system uses a stereo camera (Bumblebee2, by Point Grey Research) and a Swissranger camera [170].

We used a distributed software framework called Switchyard [125] to route messages between different devices such as the robotic arms, cameras and computers. Switchyard allows distributed computation using TCP message passing, and thus provides networking and synchronization across multiple processes on different hardware platforms. Now, we are using its second version called ROS [126].

5.6 Planning

After identifying a 3D point at which to grasp an object, we need to find an arm pose that realizes the grasp, and then plan a path to reach that arm pose so as to pick up the object.

Given a grasping point, there are typically many end-effector orientations consistent with placing the center of the gripper at that point. The choice of end-effector orientation should also take into account other constraints, such as location of nearby obstacles, and orientation of the object.

5-dof arm. When planning in the absence of obstacles, we found that even fairly simple methods for planning worked well. Specifically, on our 5-dof arm, one of the degrees of freedom is the wrist rotation, which therefore does not affect planning to avoid obstacles. Thus, we can separately consider planning an obstacle-free path using the first 4-dof, and deciding the wrist rotation. To choose the wrist rotation, using a simplified version of our algorithm in [141], we learned the 2D value of the 3D grasp orientation projected onto the image plane (see Appendix). Thus, for example,

Figure 5.9: STAIR 1 platform. This robot is equipped with a 5-dof arm and a parallel plate gripper.

Figure 5.10: STAIR 2 platform. This robot is equipped with 7-dof Barrett arm and three-fingered hand.

if the robot is grasping a long cylindrical object, it should rotate the wrist so that the parallel-plate gripper's inner surfaces are parallel (rather than perpendicular) to the main axis of the cylinder. Further, we found that using simple heuristics to decide the remaining degrees of freedom worked well.[8]

When grasping in the presence of obstacles, such as when unloading items from a dishwasher, we used a full motion planning algorithm for the 5-dof as well as for the opening of the gripper (a 6th degree of freedom). Specifically, having identified possible goal positions in configuration space using standard inverse kinematics [85], we plan a path in 6-dof configuration space that takes the end-effector from the starting position to a goal position, avoiding obstacles. For computing the goal orientation of the end-effector and the configuration of the fingers, we used a criterion that attempts to minimize the opening of the hand without touching the object being grasped or other nearby obstacles. Our planner uses Probabilistic Road-Maps (PRMs) [159], which start by randomly sampling points in the configuration space. It then constructs a "road map" by finding collision-free paths between nearby points, and finally finds a shortest path from the starting position to possible target positions in this graph. We also experimented with a potential field planner [61], but found the PRM method gave better results because it did not get stuck in local optima.

7-dof arm. On the STAIR 2 robot, which uses a 7-dof arm, we use the full algorithm in [141], for predicting the 3D orientation of a grasp, given an image of an object. This, along with our algorithm to predict the 3D grasping point, determines six of the seven degrees of freedom (i.e., the end-effector location and orientation). For deciding the seventh degree of freedom, we use a criterion that maximizes the distance of the

---

[8]Four degrees of freedom are already constrained by the end-effector 3D position and the chosen wrist angle. To decide the fifth degree of freedom in uncluttered environments, we found that most grasps reachable by our 5-dof arm fall in one of two classes: *downward* grasps and *outward* grasps. These arise as a direct consequence of the shape of the workspace of our 5 dof robotic arm (Figure 5.11). A "downward" grasp is used for objects that are close to the base of the arm, which the arm will grasp by reaching in a downward direction (Figure 5.11, first image), and an "outward" grasp is for objects further away from the base, for which the arm is unable to reach in a downward direction (Figure 5.11, second image). In practice, to simplify planning we first plan a path towards an approach position, which is set to be a fixed distance away from the predicted grasp point towards the base of the robot arm. Then we move the end-effector in a straight line forward towards the target grasping point. Our grasping experiments in uncluttered environments (Section 5.7.2) were performed using this heuristic.

arm from the obstacles. Similar to the planning on the 5-dof arm, we then apply a PRM planner to plan a path in the 7-dof configuration space.

5.7 Experiments

Table 5.1: Mean absolute error in locating the grasping point for different objects, as well as grasp success rate for picking up the different objects using our robotic arm. (Although training was done on synthetic images, testing was done on the real robotic arm and real objects.)

<div align="center">

OBJECTS SIMILAR TO ONES TRAINED ON

TESTED ON	MEAN ABS ERROR (CM)	GRASP-SUCCESS RATE
MUGS	2.4	75%
PENS	0.9	100%
WINE GLASS	1.2	100%
BOOKS	2.9	75%
ERASER/CELLPHONE	1.6	100%
OVERALL	1.80	90.0%

NOVEL OBJECTS

TESTED ON	MEAN ABS ERROR (CM)	GRASP-SUCCESS RATE
STAPLER	1.9	90%
DUCT TAPE	1.8	100%
KEYS	1.0	100%
MARKERS/SCREWDRIVER	1.1	100%
TOOTHBRUSH/CUTTER	1.1	100%
JUG	1.7	75%
TRANSLUCENT BOX	3.1	75%
POWERHORN	3.6	50%
COILED WIRE	1.4	100%
OVERALL	1.86	87.8%

</div>

Figure 5.11: The robotic arm picking up various objects: screwdriver, box, tape-roll, wine glass, a solder tool holder, coffee pot, powerhorn, cellphone, book, stapler and coffee mug. (See Section 5.7.2.)

5.7.1   Experiment 1: Synthetic data

We first evaluated the predictive accuracy of the algorithm on synthetic images (not contained in the training set). (See Figure 5.6e.) The average accuracy for classifying whether a 2D image patch is a projection of a grasping point was 94.2% (evaluated on a balanced test set comprised of the five objects in Figure 5.3). Even though the accuracy in classifying 2D regions as grasping points was only 94.2%, the accuracy in predicting 3D grasping points was higher because the probabilistic model for inferring a 3D grasping point automatically aggregates data from multiple images, and therefore "fixes" some of the errors from individual classifiers.

5.7.2   Experiment 2: Grasping novel objects

We tested our algorithm on STAIR 1 (5-dof robotic arm, with a parallel plate gripper) on the task of picking up an object placed on an uncluttered table top in front of the robot. The location of the object was chosen randomly (and we used cardboard boxes to change the height of the object, see Figure 5.11), and was completely unknown to the robot. The orientation of the object was also chosen randomly from the set of orientations in which the object would be stable, e.g., a wine glass could be placed vertically up, vertically down, or in a random 2D orientation on the table surface (see Figure 5.11). (Since the training was performed on synthetic images of objects of different types, none of these scenarios were in the training set.)

In these experiments, we used a web-camera, mounted on the wrist of the robot, to take images from two or more locations. Recall that the parameters of the vision algorithm were trained from synthetic images of a small set of five object classes, namely books, martini glasses, white-board erasers, mugs/cups, and pencils. We performed experiments on coffee mugs, wine glasses (empty or partially filled with water), pencils, books, and erasers—but all of different dimensions and appearance than the ones in the training set—as well as a large set of objects from novel object classes, such as rolls of duct tape, markers, a translucent box, jugs, knife-cutters, cellphones, pens, keys, screwdrivers, staplers, toothbrushes, a thick coil of wire, a strangely shaped power horn, etc. (See Figures 5.2 and 5.11.) We note that many of

these objects are translucent, textureless, and/or reflective, making 3D reconstruction difficult for standard stereo systems. (Indeed, a carefully-calibrated Point Gray stereo system, the Bumblebee BB-COL-20,—with higher quality cameras than our web-camera—fails to accurately reconstruct the visible portions of 9 out of 12 objects. See Figure 5.5.)

In extensive experiments, the algorithm for predicting grasps in images appeared to generalize very well. Despite being tested on images of real (rather than synthetic) objects, including many very different from ones in the training set, it was usually able to identify correct grasp points. We note that test set error (in terms of average absolute error in the predicted position of the grasp point) on the real images was only somewhat higher than the error on synthetic images; this shows that the algorithm trained on synthetic images transfers well to real images. (Over all 5 object types used in the synthetic data, average absolute error was 0.81cm in the synthetic images; and over all the 14 real test objects, average error was 1.84cm.) For comparison, neonate humans can grasp simple objects with an average accuracy of 1.5cm. [11]

Table 5.1 shows the errors in the predicted grasping points on the test set. The table presents results separately for objects which were similar to those we trained on (e.g., coffee mugs) and those which were very dissimilar to the training objects (e.g., duct tape). For each entry in the table, a total of four trials were conducted except for staplers, for which ten trials were conducted. In addition to reporting errors in grasp positions, we also report the grasp success rate, i.e., the fraction of times the robotic arm was able to physically pick up the object. For a grasp to be counted as successful, the robot had to grasp the object, lift it up by about 1ft, and hold it for 30 seconds. On average, the robot picked up the novel objects 87.8% of the time.

For simple objects such as cellphones, wine glasses, keys, toothbrushes, etc., the algorithm performed perfectly in our experiments (100% grasp success rate). However, grasping objects such as mugs or jugs (by the handle) allows only a narrow trajectory of approach—where one "finger" is inserted into the handle—so that even a small error in the grasping point identification causes the arm to hit and move the object, resulting in a failed grasp attempt. Although it may be possible to improve the algorithm's accuracy, we believe that these problems can best be solved by using

a more advanced robotic arm that is capable of haptic (touch) feedback.

In many instances, the algorithm was able to pick up completely novel objects (a strangely shaped power-horn, duct-tape, solder tool holder, etc.; see Figures 5.2 and 5.11). Perceiving a transparent wine glass is a difficult problem for standard vision (e.g., stereopsis) algorithms because of reflections, etc. However, as shown in Table 5.1, our algorithm successfully picked it up 100% of the time. Videos showing the robot grasping the objects are available at

http://ai.stanford.edu/~asaxena/learninggrasp/

5.7.3   Experiment 3: Unloading items from dishwashers

The goal of the STAIR project is to build a general purpose household robot. As a step towards one of STAIR's envisioned applications, in this experiment we considered the task of unloading items from dishwashers (Figures 5.1 and 5.14). This is a difficult problem because of the presence of background clutter and occlusion between objects—one object that we are trying to unload may physically block our view of a second object. Our training set for these experiments also included some hand-labeled real examples of dishwasher images (Figure 5.12), including some images containing occluded objects; this helps prevent the algorithm from identifying grasping points on the background clutter such as dishwasher prongs. Along with the usual features, these experiments also used the depth-based features computed from the depth image obtained from the stereo camera (Section 5.3.3). Further, in these experiments we did not use color information, i.e., the images fed to the algorithm were grayscale.

In detail, we asked a person to arrange several objects neatly (meaning inserted over or between the dishwasher prongs, and with no pair of objects lying flush against each other; Figures 10 and 11 show typical examples) in the upper tray of the dishwasher. To unload the items from the dishwasher, the robot first identifies grasping points in the image. Figure 5.13 shows our algorithm correctly identifying grasps on multiple objects even in the presence of clutter and occlusion. The robot then uses
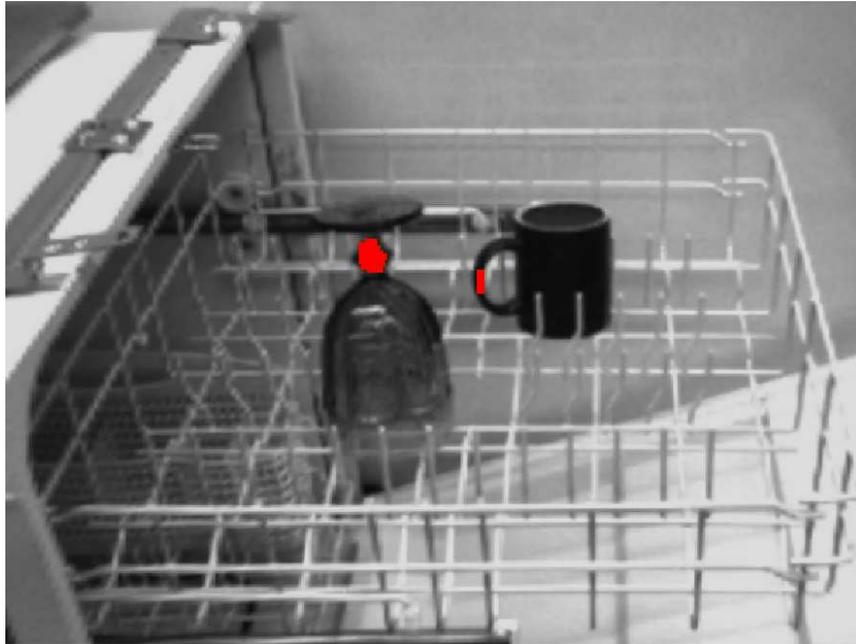
Figure 5.12: Example of a real dishwasher image, used for training in the dishwasher experiments.

these grasps and the locations of the obstacles (perceived using a stereo camera) to plan a path while avoiding obstacles, to pick up the object. When planning a path to a grasping point, the robot chooses the grasping point that is most accessible to the robotic arm using a criterion based on the grasping point's height, distance to the robot arm, and distance from obstacles. The robotic arm then removes the first object (Figure 5.14) by lifting it up by about 1ft and placing it on a surface on its right using a pre-written script, and repeats the process above to unload the next

Table 5.2: Grasp-success rate for unloading items from a dishwasher.

Tested on	Grasp-Success-rate
Plates	100%
Bowls	80%
Mugs	60%
Wine Glass	80%
Overall	80%

Figure 5.13: Grasping point detection for objects in a dishwasher. (Only the points in top five grasping regions are shown.)

item. As objects are removed, the visual scene also changes, and the algorithm will find grasping points on objects that it had missed earlier.

We evaluated the algorithm quantitatively for four object classes: plates, bowls, mugs and wine glasses. (We have successfully unloaded items from multiple dishwashers; however we performed quantitative experiments only on one dishwasher.) We performed five trials for each object class (each trial used a different object). We achieved an average grasping success rate of 80.0% in a total of 20 trials (see Table 5.2). Our algorithm was able to successfully pick up objects such as plates and wine glasses most of the time. However, due to the physical limitations of the 5-dof arm with a parallel plate gripper, it is not possible for the arm to pick up certain objects, such as bowls, if they are in certain configurations (see Figure 5.15). For mugs, the grasp success rate was low because of the problem of narrow trajectories discussed in Section 5.7.2.

We also performed tests using silverware, specifically objects such as spoons and forks. These objects were often placed (by the human dishwasher loader) against the corners or walls of the silverware rack. The handles of spoons and forks are only about 0.5cm thick; therefore a larger clearance than 0.5cm was needed for them to be grasped using our parallel plate gripper, making it physically extremely difficult to do so. However, if we arrange the spoons and forks with part of the spoon or fork at least 2cm away from the walls of the silverware rack, then we achieve a grasping success rate of about 75%.

Some of the failures were because some parts of the object were not perceived; therefore the arm would hit and move the object resulting in a failed grasp. In such cases, we believe an algorithm that uses haptic (touch) feedback would significantly increase grasp success rate. Some of our failures were also in cases where our algorithm correctly predicts a grasping point, but the arm was physically unable to reach that grasp. Therefore, we believe that using an arm/hand with more degrees of freedom, will significantly improve performance for the problem of unloading a dishwasher.

Figure 5.14: Dishwasher experiments (Section 5.7.3): Our robotic arm unloads items from a dishwasher.

Figure 5.15: Dishwasher experiments: Failure case. For some configurations of certain objects, it is impossible for our 5-dof robotic arm to grasp it (even if a human were controlling the arm).

5.7.4 Experiment 4: Grasping kitchen and office objects

Our long-term goal is to create a useful household robot that can perform many different tasks, such as fetching an object in response to a verbal request and cooking simple kitchen meals. In situations such as these, the robot would know which object it has to pick up. For example, if the robot was asked to fetch a stapler from an office, then it would know that it needs to identify grasping points for staplers only. Therefore, in this experiment we study how we can use information about object type and location to improve the performance of the grasping algorithm.

Consider objects lying against a cluttered background such as a kitchen or an office. If we predict the grasping points using our algorithm trained on a dataset containing all five objects, then we typically obtain a set of reasonable grasping point predictions (Figure 5.16, left column). Now suppose we know the type of object we want to grasp, as well as its approximate location in the scene (such as from an object recognition algorithm [39]). We can then restrict our attention to the area of the image containing the object, and apply a version of the algorithm that has been trained using only objects of a similar type (i.e., using object-type specific parameters, such as using bowl-specific parameters when picking up a cereal bowl, using spoon-specific parameters when picking up a spoon, etc). With this method, we obtain object-specific grasps, as shown in Figure 5.16 (right column).

Achieving larger goals, such as cooking simple kitchen meals, requires that we combine different algorithms such as object recognition, navigation, robot manipulation, etc. These results demonstrate how our approach could be used in conjunction with other complementary algorithms to accomplish these goals.

5.8 Discussion

We proposed an algorithm for enabling a robot to grasp an object that it has never seen before. Our learning algorithm neither tries to build, nor requires, a 3D model of the object. Instead it predicts, directly as a function of the images, a point at which to grasp the object. In our experiments, the algorithm generalizes very well to
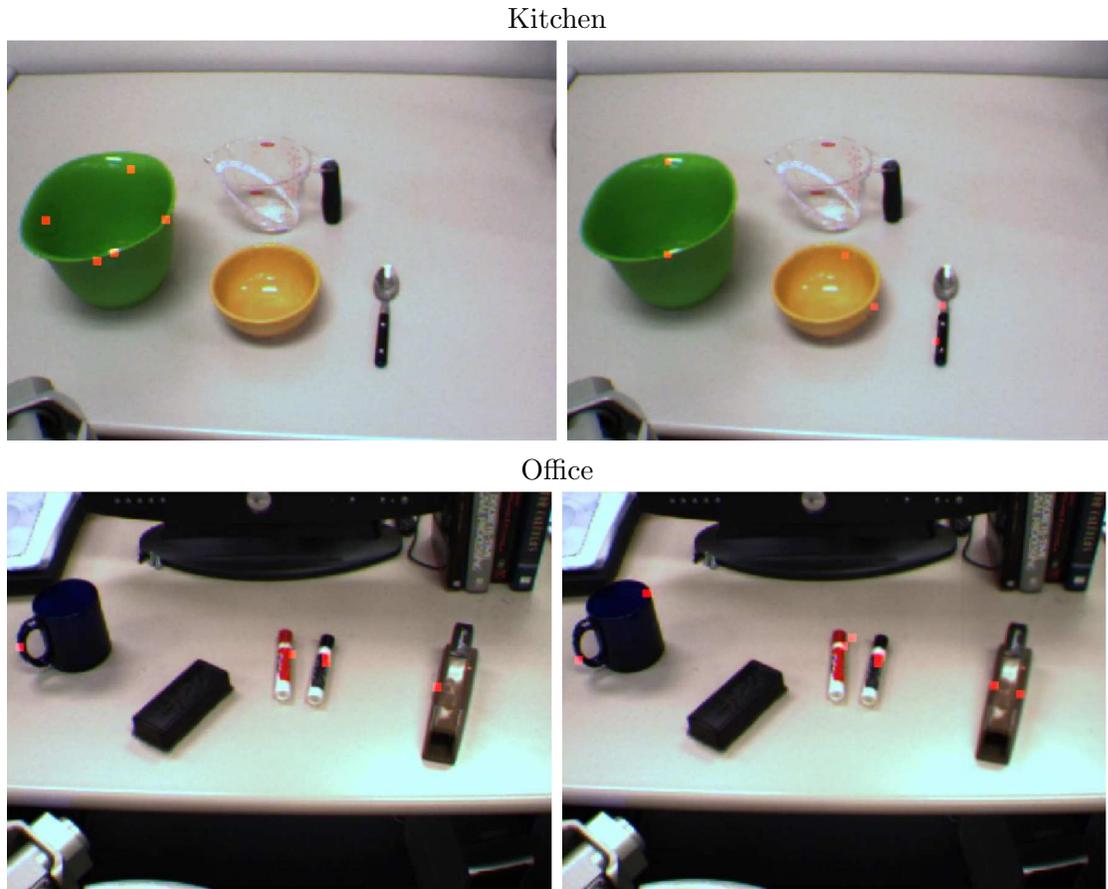
Figure 5.16: Grasping point classification in kitchen and office scenarios: (Left Column) Top five grasps predicted by the grasp classifier alone. (Right Column) Top two grasps for three different object-types, predicted by the grasp classifier when given the object types and locations. The red points in each image are the predicted grasp locations. (Best viewed in color.)

novel objects and environments, and our robot successfully grasped a wide variety of objects in different environments such as dishwashers, office and kitchen.

Chapter 6

Robotic Grasping: Full Hand Configuration

6.1 Introduction

Consider the problem of grasping novel objects, in the presence of significant amounts of clutter. A key challenge in this setting is that a full 3D model of the scene is typically not available. Instead, a robot's sensors (such as a camera or a depth sensor) can usually estimate only the shape of the visible portions of the scene. In this chapter, we present an algorithm that, given such partial models of the scene, selects a grasp—that is, a configuration of the robot's arm and fingers—to try to pick up an object.

Our algorithm in chapter 5 applied supervised learning to identify visual properties that indicate good grasps, given a 2-d image of the scene. However, it only chose a 3D "grasp point"—that is, the 3D position (and 3D orientation; Saxena et al. [139]) of the center of the end-effector. Thus, it did not generalize well to more complex arms and hands, such as to multi-fingered hands where one has to not only choose the 3D position (and orientation) of the hand, but also address the high dof problem of choosing the positions of all the fingers.

If a full 3D model (including the occluded portions of a scene) were available, then methods such as form- and force-closure [86, 7, 116] and other grasp quality

metrics [107, 48, 18] can be used to try to find a good grasp. However, given only the point cloud returned by stereo vision or other depth sensors, a straightforward application of these ideas is impossible, since we do not have a model of the occluded portions of the scene.

In prior work that used vision for real-world grasping experiments, most were limited to grasping 2-d planar objects. For a uniformly colored planar object lying on a uniformly colored table top, one can find the 2-d contour of the object quite reliably. Using local visual features (based on the 2-d contour) and other properties such as form- and force-closure, [19, 17, 12, 93] computed the 2-d locations at which to place (two or three) fingertips to grasp the object. In more general settings (i.e., non-planar grasps), Edsinger and Kemp ([28]) grasped cylindrical objects using a power grasp by using visual servoing and Platt et al. ([115]) used schema structured learning for grasping simple objects (spherical and cylindrical) using power grasps; however, this does not apply to grasping general shapes (e.g., a cup by its handle) or to grasping in cluttered environments.

In detail, we will consider a robot that uses a camera, together with a depth sensor, to perceive a scene. The depth sensor returns a "point cloud," corresponding to 3D locations that it has found on the front unoccluded surfaces of the objects. (See Fig. 6.1.) Such point clouds are typically noisy (because of small errors in the depth estimates); but more importantly, they are also incomplete.[1]

Our approach begins by computing a number of features of grasp quality, using both 2-d image and the 3D point cloud features. For example, the 3D data is used to compute a number of grasp quality metrics, such as the degree to which the fingers are exerting forces normal to the surfaces of the object, and the degree to which they enclose the object. Using such features, we then apply a supervised learning algorithm to estimate the degree to which different configurations of the full arm and fingers reflect good grasps.

Our grasp planning algorithm that relied on long range vision sensors (such as cameras, swissranger), errors and uncertainty ranging from small deviations in the

---

[1]For example, standard stereo vision fails to return depth values for textureless portions of the object, thus its point clouds are typically very sparse. Further, the Swissranger gives few points only because of its low spatial resolution of 144×176.
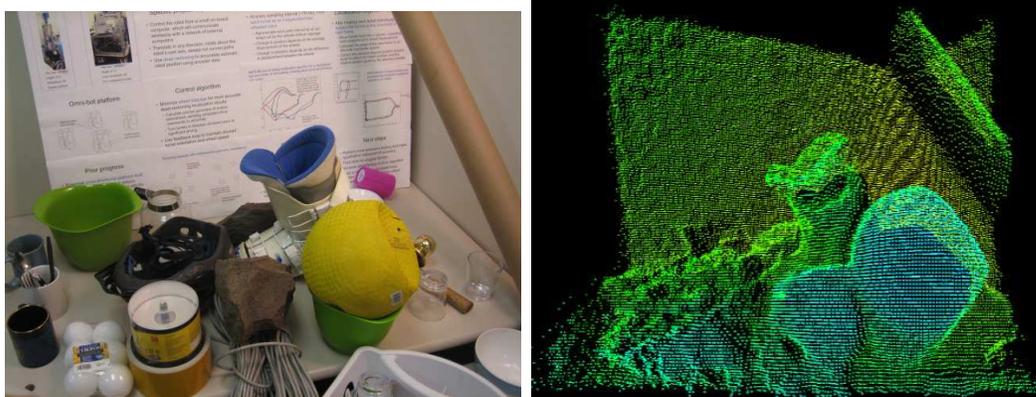
Figure 6.1: Image of an environment (left) and the 3D point-cloud (right) returned by the Swissranger depth sensor.

object's location to occluded surfaces significantly limited the reliability of these open-loop grasping strategies. (We found that approximately 65% of the grasp failures were because we used only long range sensors and lacked a reactive controller with sufficient local surface pose information.) Therefore, we also present new optical proximity sensors mounted on robot hand's fingertips and learning algorithms to enable the robot to reactively adjust the grasp. These sensors are a very useful complement to long-range sensors (such as vision and 3D cameras), and provide a sense of "pre-touch" to the robot, thus helping in the grasping performance.

We test our algorithm on two robots, on a variety of objects of shapes very different from ones in the training set, including a ski boot, a coil of wire, a game controller, and others. Even when the objects are placed amidst significant clutter, our algorithm often selects successful grasps.

We present an additional application of our algorithms to opening new doors. This capability enables our robot to navigate in spaces that were earlier blocked by doors and elevators.

This chapter is organized as follows. We first give a brief overview of the 3D sensor used in Section 6.2. We then describe our method for grasping using this 3D data in Section 6.3. We describe our experiments and results in Section 6.4. We then describe the optical proximity sensors in Section 6.5. Finally, we describe the door
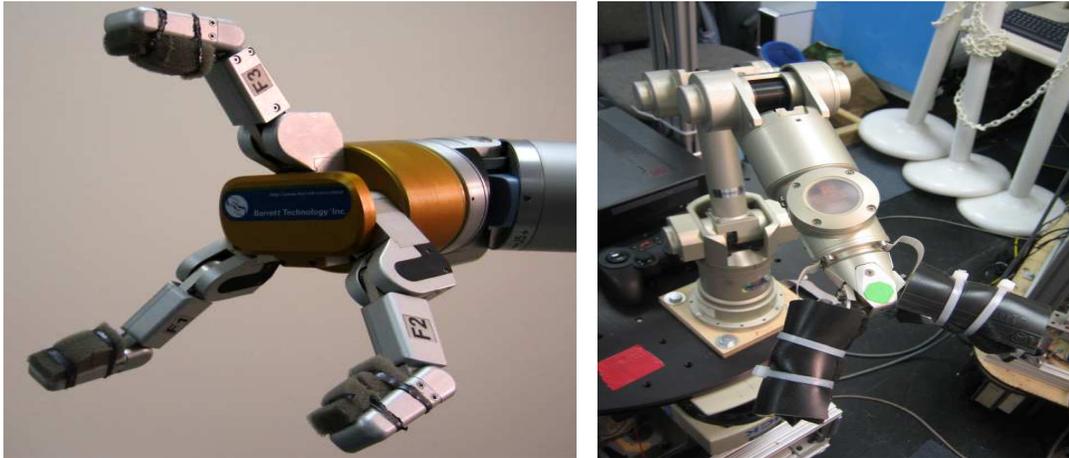
Figure 6.2: (Left) Barrett 3-fingered hand. (Right) Katana parallel plate gripper.

opening application in Section 6.6.

6.2   3D sensor

The SwissRanger camera is a time-of-flight depth sensor that returns a 144×176 array of depth estimates, spanning a $47.5° \times 39.6°$ field of view, with a range of about 8m. Using an infrared structured light source, each pixel in the camera independently measures the arrival time of the light reflected back by the objects. Its depth estimates are typically accurate to about 2cm. However, it also suffers from systematic errors, in that it tends not to return depth estimates for dark objects, nor for surfaces lying at a large angle relative to the camera's image plane. (See Fig. 6.1 for a sample 3D scan.)

6.3   Grasping Strategy

There are many different properties that make certain grasps preferable to others. Examples of such properties include form- and force-closure (to minimize slippage), sufficient contact with the object, distance to obstacles (to increase robustness of the grasp), and distance between the center of the object and the grasping point

(to increase stability). In real world grasping, however, such properties are difficult to compute exactly, because of the quality of sensor data. Our algorithm will first compute a variety of features that attempt to capture some of these properties. Using these features, we then apply supervised learning to predict whether or not a given arm/finger configuration reflects a good grasp.

Definition of grasp: We will infer the full goal configuration of the arm/fingers that is required to grasp an object. For example, STAIR 1 uses an arm with 5 joints and a parallel plate gripper (with one degree of freedom); for this robot, the configuration is given by $\alpha \in \mathbb{R}^6$. The second robot STAIR 2 uses an arm with 7 joints, equipped with a three-fingered hand that has 4 joints (Fig. 6.2); for this robot, our algorithm infers a configuration $\alpha \in \mathbb{R}^{11}$. We will informally refer to this goal configuration as a "grasp."

We will then use a motion planner algorithm [159] to plan a path (one that avoids obstacles) from the initial configuration to this goal configuration.

6.3.1   Probabilistic model

We will use both the point-cloud R and the image I taken of a scene to infer a goal configuration α of the arm/fingers.

In chapter 5, we classified each 2-d point in a given image as a 1 (candidate grasp) or 0. For example, for an image of a mug, it would try to classify the handle and rim as candidate grasping points. In our approach, we use a similar classifier that computes a set of image features and predicts the probability $P(y = 1|\alpha, I) \in [0, 1]$ of each point in the image being a candidate grasping point. However, a remaining difficulty is that this algorithm does not take into account the arm/finger kinematics; thus, many of the 2-d points it selects are physically impossible for the robot to reach.

To address this problem, we use a second classifier that, given a configuration α and a point-cloud R, predicts the probability $P(y|\alpha, R)$ that the grasp will succeed. This classifier will compute features that capture a variety of properties that are indicative of grasp quality. Our model will combine these two classifiers to estimate the probability $P(y|\alpha, R, I)$ of a grasp α succeeding. Let $y \in \{0, 1\}$ indicate whether

$\{\alpha, R, I\}$ is a good grasp. We then have:

$$P(y|\alpha, R, I) \propto P(R, I|y, \alpha)P(y, \alpha) \tag{6.1}$$

We assume conditional independence of R and I, and uniform priors $P(\alpha)$, $P(I)$, $P(R)$ and $P(y)$. Hence,

$$
\begin{aligned}
P(y|\alpha, R, I) &\propto P(R|y, \alpha)P(I|y, \alpha)P(y, \alpha) \\
&\propto P(y|\alpha, R)P(y|\alpha, I)
\end{aligned}
\tag{6.2}
$$

Here, the $P(y|\alpha, I)$ is the 2-d image classifier term similar to the one in chapter 5. We also use $P(y|\alpha, R; \theta) = 1/\exp\left(1 + \psi(R, \alpha)^T\theta\right)$, where $\psi(R, \alpha)$ are the features discussed in the next section.

Inference: From Eq. 6.2, the inference problem is:

$$
\begin{aligned}
\alpha^* &= \arg\max_{\alpha} \log P(y = 1|\alpha, R, I) \\
&= \arg\max_{\alpha} \log P(y = 1|\alpha, R) + \log P(y = 1|\alpha, I)
\end{aligned}
$$

Now, we note that a configuration α has a very small chance of being the optimal configuration if either one of the two terms is very small. Thus, for efficiency, we implemented an image-based classifier $P(y|I, \alpha)$ that returns only a small set of 3D points with a high value. We use this to restrict the set of configurations α to only those in which the hand-center lies on one of the 3D location output by the image-based classifier. Given one such 3D location, finding a full configuration α for it now requires solving only an $n - 3$ dimensional problem (where $n = 6$ or 11, depending on the arm). Further, we found that it was sufficient to consider only a few locations of the fingers, which further reduces the search space; e.g., in the goal configuration, the gap between the finger and the object is unlikely to be larger than a certain value. By sampling randomly from this space of "likely" configurations (similar to sampling in PRMs) and evaluating the grasp quality only of these samples, we obtain an inference algorithm that is computationally tractable and that also

typically obtains good grasps.

Algorithm 1 Grasping an Object

 1: Get a set of candidate 2-d grasp points in the image using a classifier based on [137]
 2: Use depth sensors (stereo/swissranger) to find corresponding 3D grasp point
 3: **for** Each grasp point in candidate 3D grasp points set **do**
 4:     Use arm inverse kinematics to generate configuration(s) with hand center near the 3D grasp point
 5:     Add configurations(s) for which arm/finger does not hit obstacles to the candidate valid configuration set
 6: **end for**
 7: **for** Each configuration in candidate valid configuration set **do**
 8:     Compute features using the point-cloud and the configuration
 9:     Use features to classify if grasp will be good/bad
10:     $Score[grasp] \Leftarrow ScoreForCandidateGrasp$ (from classifier)
11: **end for**
12: Execute $grasp^* = \arg\max Score[grasp]$
13: Plan path for $grasp^*$ using PRM motion planner

6.3.2   Features

Below, we describe the features that make up the feature vector $\psi(R, \alpha)$ used to estimate a good grasp. The same features were used for both robots.

Presence / Contact: For a given finger configuration, some part of an object should be inside the volume enclosed by the fingers. Intuitively, more enclosed points indicate that the grasp contains larger parts of the object, which generally decreases the difficulty of grasping it (less likely to miss). For example, for a coil of wire, it is better to grasp a bundle rather than a single wire. To robustly capture this, we calculate a number of features—the number of points contained in spheres of different sizes located at the hand's center, and also the number of points located inside the volume enclosed within the finger-tips and the palm of the hand.

Symmetry / Center of Mass: Even if many points are enclosed by the hand, their distribution is also important. For example, a stick should be grasped at the middle instead of at the tip, as slippage might occur in the latter case due to a greater torque

<center>(a)   (b)</center>
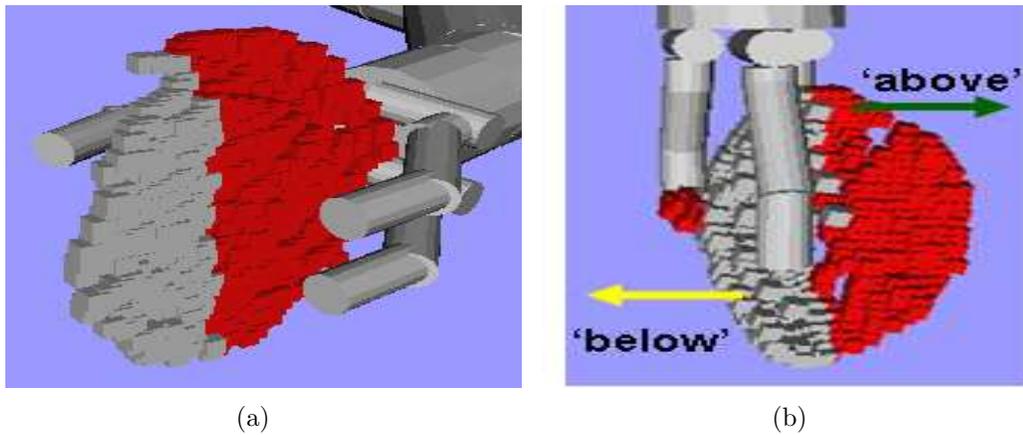
Figure 6.3: Features for symmetry / evenness.

induced by gravity. To capture this property, we calculate a number of features based on the distribution of points around the hand's center along an axis perpendicular to the line joining the fingers. To ensure grasp stability, an even distribution (1:1 ratio) of points on both sides of the axis is desirable. More formally, if there are N points on one side and N' on the other side, then our feature would be $|N - N'|/(N + N')$. Again, to increase robustness, we use several counting methods, such as counting all the points, and counting only those points not enclosed by the hand.

Local Planarity / Force-Closure: One needs to ensure a few properties to avoid slippage, such as a force closure on the object (e.g., to pick up a long tube, a grasp that lines up the fingers along the major axis of the tube would likely fail). Further, a grasp in which the finger direction (i.e., direction in which the finger closes) is perpendicular to the local tangent plane is more desirable, because it is more likely to lie within the friction cone, and hence is less likely to slip. For example, when grasping a plate, it is more desirable that the fingers close in a direction perpendicular to the plate surface.

To capture such properties, we start with calculating the principal directions of a 3D point cloud centered at the point in question. This gives three orthonormal component directions u_i, with u_1 being the component with largest variance, followed by u_2 and u_3. (Let σ_i be the corresponding variances.) For a point on the rim of a

Figure 6.4: Demonstration of PCA features.

circular plate, u_1 and u_2 would lie in the plane in which the plate lies, with u_1 usually tangent to the edge, and u_2 facing inwards. Ideally, the finger direction should be orthogonal to large variance directions and parallel to the small variance ones. For f_j as the finger direction ($j = 1$ for parallel gripper, and $j = 1, 2$ for three-fingered hand), we would calculate the following features: (a) Directional similarity, $s_{ij} = |u_i \cdot f_j|$, and (b) Difference from ideal, $(\frac{\sigma_1 - \sigma_i}{\sigma_1 - \sigma_3} - s_{ij})^2$.

6.4   Experiments

We performed three sets of extensive experiments: grasping with our three-fingered 7-dof arm in uncluttered as well as cluttered environments, and on our 5-dof arm with a parallel plate gripper for unloading items in a cluttered dishwasher.

6.4.1   Grasping single novel objects

We considered several objects from 13 novel object classes in a total of 150 experiments. These object classes varied greatly in shape, size, and appearance, and are very different from the plates, bowls, and rectangular blocks used in the training set. During the experiments, objects were placed at a random location in front of our robot. Table 6.1 shows the results: "Prediction" refers to the percentage of cases the final grasp and plan were good, and "Grasp success" is the percentage of cases in which the robot predicted a good grasp and actually picked up the object as well
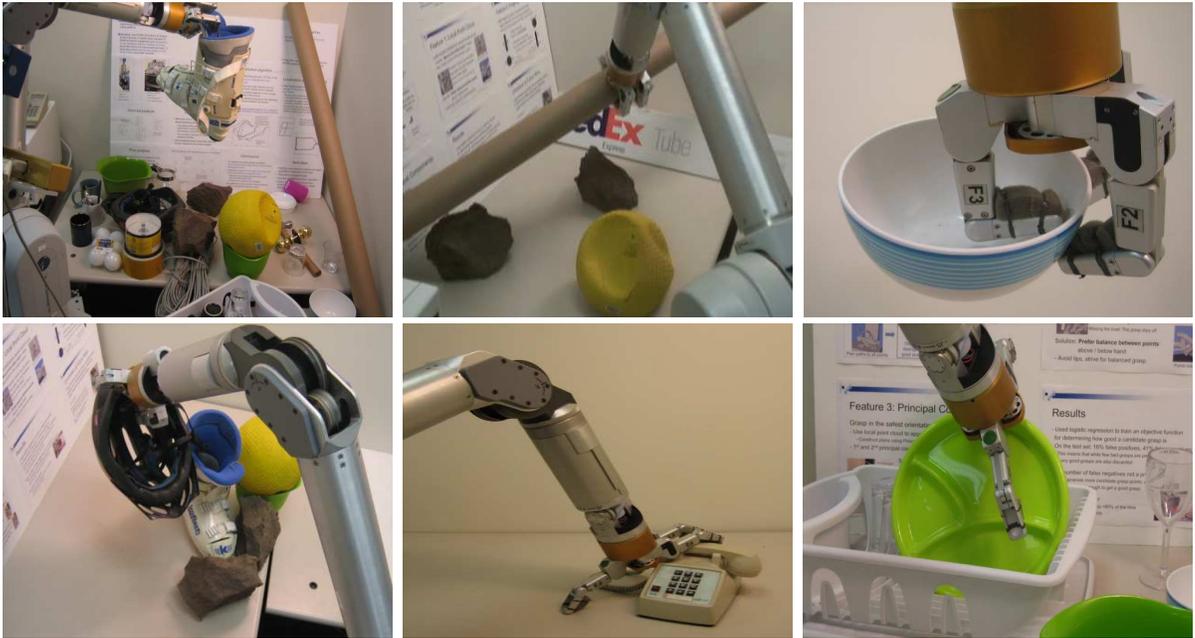
Figure 6.5: Snapshots of our robot grasping novel objects of various sizes/shapes.

(i.e., if the object slipped and was not picked up, then it counted as failure).

In 5, we required that objects be neatly placed on a "rack." However, we consider the significantly harder task of grasping randomly placed objects in any orientation. Further, many objects such as ski boots, helmets, etc. require more intricate grasps (such as inserting a finger in the ski boot because it is too big to hold as a power grasp).

For each object class, we performed 10-20 trials, with different instances of each object for each class (e.g., different plates for "plates" class). The average "Prediction" accuracy was 81.3%; and the actual grasp success rate was 76%. The success rate was different depending on the size of the object.[2] Both prediction and actual grasping were best for medium sizes, with success rates of 92% and 86% respectively. Even though handicapped with a significantly harder experimental trial (i.e., objects

---

[2]We defined an object to be small if it could be enclosed within the robot hand, medium if it was approximately 1.5-3 times the size of the hand, and large otherwise (some objects were even 4ft long).

Table 6.1: STAIR 2. Grasping single objects. (150 trials.)

Object Class	Size	Prediction	Grasp success
Ball	small	80%	80%
Apple	small	90%	80%
Gamepad	small	85%	80%
CD container	small	70%	60%
Helmet	med	100%	100%
Ski boot	med	80%	80%
Plate bundle	med	100%	80%
Box	med	90%	90%
Robot arm link	med	90%	80%
Cardboard tube	large	70%	65%
Foam	large	60%	60%
Styrofoam	large	80%	70%
Coil of wire	large	70%	70%

not neatly stacked, but thrown in random places and a larger variation of object sizes/shapes considered) as compared to Saxena et al., our algorithm surpasses their success rate by 6%.

6.4.2   Grasping in cluttered scenarios

In this case, in addition to the difficulty in perception, manipulation and planning became significantly harder in that the arm had to avoid all other objects while reaching for the predicted grasp; this significantly reduced the number of feasible candidates and increased the difficulty of the task.

In each trial, more than five objects were placed in random locations (even where objects touched each other, see some examples in Fig. 6.5). Using only the 2-d image-based method of Saxena et al. (with PRM motion planning), but not our algorithm that considers finding all arm/finger joints from partial 3D data, success was below 5%. In a total of 40 experiments, our success rate was 75% (see Table 6.2). We believe our robot is the first one to be able to automatically grasp objects, of types never seen before, placed randomly in such heavily cluttered environments.

Figure 6.6: Barrett arm grasping an object using our algorithm.

Table 6.2: STAIR 2. Grasping in cluttered scenes. (40 trials.)

Environment	Object	Prediction	Grasp Success
Terrain	Tube	87.5%	75%
Terrain	Rock	100%	75%
Kitchen	Plate	87.5%	75%
Kitchen	Bowl	75%	75%

Table 6.3: STAIR 1. Dishwasher unloading results. (50 trials.)

Object Class	Prediction Good	Actual Success
Plate	100%	85%
Bowl	80%	75%
Mug	80%	60%

We have made our grasping movies available at:
http://stair.stanford.edu/multimedia.php

6.5   Optical proximity sensors

In our grasp planning algorithm that relied on long range vision sensors (such as cameras, swissranger), errors and uncertainty ranging from small deviations in the object's location to occluded surfaces significantly limited the reliability of these open-loop grasping strategies. We found that approximately 65% of the grasp failures were because we used only long range sensors and lacked a reactive controller with sufficient local surface pose information.

Tactile sensing has been employed as a means to augment the initial grasp and manipulation strategies by addressing inconsistencies in the contact forces during object contact and manipulation [173]. However, tactile sensors have to actually touch the object in order to provide useful information. Because current sensor technologies are not sensitive enough to detect finger contacts before causing significant object motion, their use is limited to either minor adjustments of contact forces at pre-computed

Figure 6.7: Three-fingered Barrett Hand with our optical proximity sensors mounted on the finger tips.

grasp configurations, or to planning algorithms that require iterative re-grasping of the object in order to grasp successfully [109, 172, 50]. While the latter approach has shown substantial improvements in grasp reliability, it requires a significant amount of time and frequently causes lighter objects to be knocked over during the repeated grasp attempts.

The limitations of tactile-sensing-augmented grasp planning can be overcome by 'pre-touch' sensing. This modality has recently become a popular means of bridging the gap in performance between long range vision and tactile sensors. In pre-touch sensing, gripper-mounted, short-range (0-4cm) proximity sensors are used to estimate the absolute distance and orientation (collectively called surface pose) of a desired contact location without requiring the robot to touch the object [163]. The vast majority of these pre-touch proximity sensors use optical methods because of their high precision [10, 106, 179, 111, 112]. Optical sensors, however, are highly sensitive to surface reflection properties. Alternatively, capacitive-based proximity sensors have also been used [183]. While invariant to surface properties, these capacitive-based

sensors have difficulty detecting materials with low dielectric contrast, such as fabrics and thin plastics. Unfortunately, in both cases, present sensor calibration and modeling techniques have yet to produce pose estimates that are robust enough to be useful across the range of surface textures, materials, and geometries encountered in unstructured environments. Furthermore, the finger tips of typical robotic grippers are too small to accommodate the sensors used in previous work.

Therefore, in [51] we developed an optical proximity sensor, embedded in the fingers of the robot (see Fig. 6.7), and show how it can be used to estimate local object geometries and perform better reactive grasps. This allows for on-line grasp adjustments to an initial grasp configuration without the need for premature object contact or re-grasping strategies. Specifically, a design for a low cost, pose-estimating proximity sensor is presented that meets the small form factor constraints of a typical robotic gripper.

The data from these sensors is interpreted using empirically derived models and the resulting pose estimates are provided to a reactive grasp closure controller that regulates contact distances even in the absence of reliable surface estimates. This allows the robot to move the finger tip sensors safely into configurations where sensor data for the belief-state update can be gathered. Simultaneously, the pose estimates of the sensor interpretation approach can provide the necessary information to adjust the grasp configuration to match the orientation of the object surface, increasing the likelihood of a stable grasp.

We perform a series of grasping experiments to validate the system using a dexterous robot consisting of a 7-DOF Barrett arm and multi-fingered hand. In these tests, we assume that an approximate location of the object to be grasped has already been determined either from long range vision sensors [136], or through human interaction [60]. From the initial grasp positions, the system exhibits improved grasping on a variety of household objects with varying materials, surface properties, and geometries.

Figure 6.8: Normalized Voltage Output vs. Distance for a TCND5000 emitter-detector pair.

6.5.1 Optical Sensor Hardware

The purpose of the optical sensor hardware is to provide data that can be used by the modeling algorithm to construct pose estimates of nearby surfaces. The design is driven by a series of constraints, including size, sensor response, and field of view. The hardware design was mostly due to Paul Nangeroni [51]; however, we provide the full design details for completeness.

A basic optical sensor consists of an emitter, photo-receiver, and signal processing circuitry. The light from the emitter is reflected by nearby surfaces and received by the photo-receiver. The amplitude and phase of the light vary as a function of the distance to the surface, its orientation, and other properties of the surface material (reflectance, texture, etc.) [10]. In amplitude-modulated proximity sensor design, the most commonly preferred method, these variations in amplitude can be converted
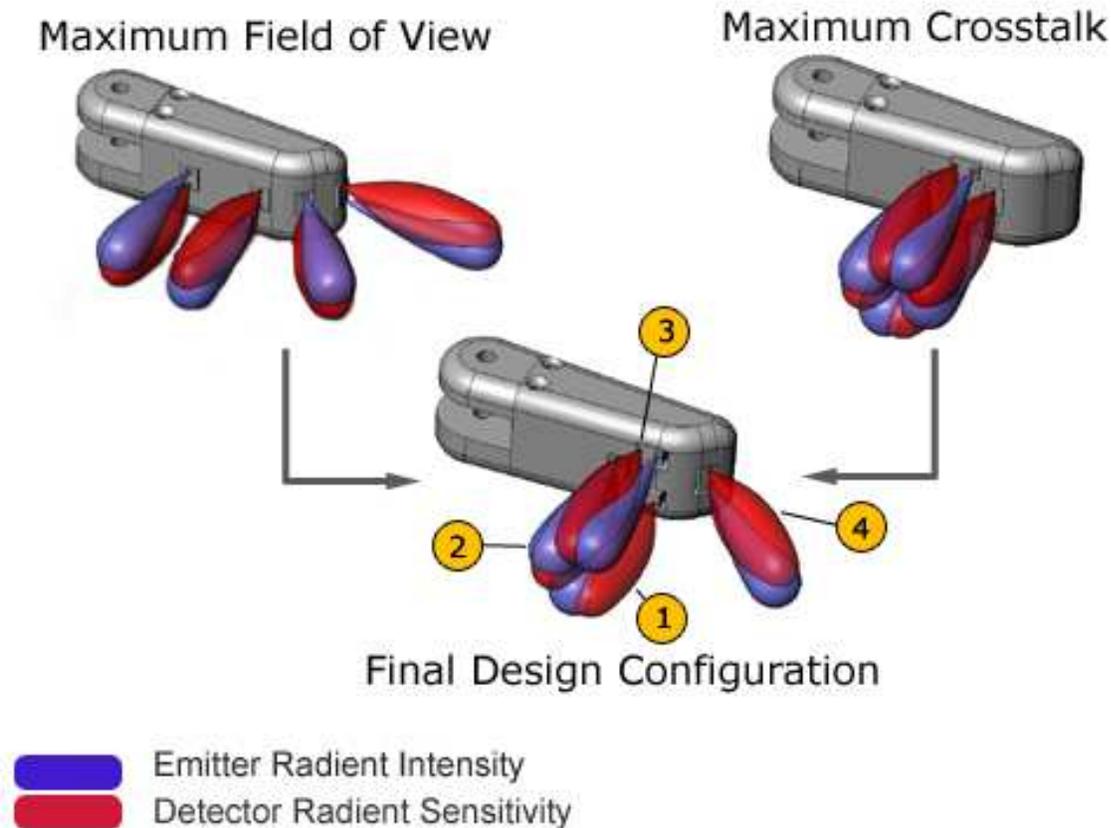
Figure 6.9: Possible fingertip sensor configurations.

into pose estimates by measuring the response from constellations of at least three receivers focused at the same point on the target surface [10, 127, 179].

Although conceptually simple, modeling the pose of unknown surfaces is difficult because of the non-monotonic behavior of the proximity sensor receivers. The response of a single sensor is a function of the distance to the target surface and the baseline between the emitter and receiver, as shown in Fig. 6.8. While the response in the far-field varies as the inverse square of the distance [105], the response of the sensor in the near field is far more complex. The decrease in received light energy in the near field is governed not only by the reflective properties of the surface, but also by the geometric baseline between the emitter and receiver. As distance approaches

zero in the near field, the amount of energy that can be reflected between the emitter and receiver decreases because the overlap between the emitter and receiver cones decreases. This results in a sharp drop-off of received light intensity. To avoid complications in modeling the sensor data, most approaches, including ours, offset the sensors from the surface of the gripper to maximize the far field sensor response. Although this sacrifices the higher sensitivity of the near field, the longer range of the far field is better suited to grasping applications [10].

The selection of specific sensor hardware and signal processing electronics is severely constrained by the limited space in typical robotic fingers (for instance, our Barrett fingers have a 1cm x 2cm x 1cm cavity volume). Bonen and Walker used optical fibers to achieve the desired geometric arrangement of emitters and receivers in their respective sensors. However, the bend radius and large terminations of the fibers violate the space constraints in this application. Instead, our design uses four low-cost, off-the-shelf infrared emitter/receiver pairs (Vishay TCND50000) on each finger. The small size (6 x 3.7 x 3.7 mm) meets the volume requirements of the design and the range (2-40mm) is ideal for pre-touch sensing.

The arrangement of these sensors represents yet another design tradeoff between field of view and pose estimation accuracy. Large fields of view, both out from and in front of the face of the fingertip, as shown in Fig. 6.9a, are advantageous for detecting oncoming objects. Unfortunately, the sensor spacing needed to achieve this reduces the overlap between adjacent sensor pairs and lowers the signal to noise ratio. Conversely, arranging the sensors to focus the emitters and maximize crosstalk, as shown in Fig. 6.9b, improves local pose estimation accuracy at the expense of broader range data to nearby objects. The final configuration, shown in Fig. 6.9c, consists of three sensors arranged in a triangle on the face of the finger to estimate pose with a fourth sensor at a 45° angle to the finger tip to increase the field of view. Although surface area exists for placing additional sensors, the quantity is limited to four by the available space in the Barrett finger cavity for pre-amplifier circuitry. The sensors are inset into the finger to minimize near-field effects, and the aluminum housing is matte anodized to decrease internal specular reflection. The crosstalk between

Emitters in Blue
Detectors in Red
Sensor #
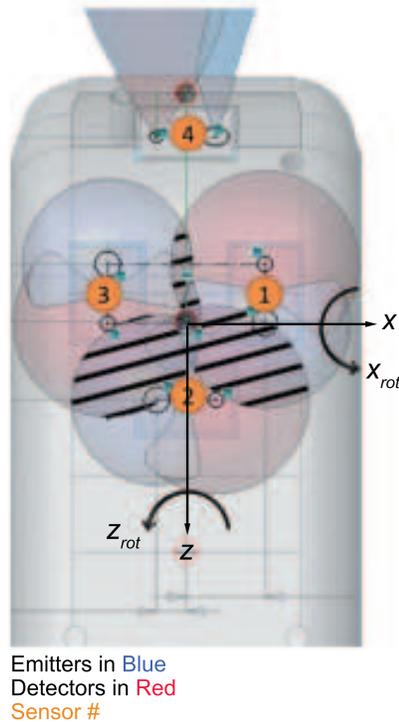
Figure 6.10: Front view of sensor constellation. Hatches show the crosstalk between adjacent sensor pairs.

adjacent sensors is illustrated by the hatched area in Fig. 6.10.[3]

The complete proximity sensing suite consists of twelve total emitter-detector pairs, four on each finger. The emitters are pulsed in sequence by a PIC 18F4550 micro-controller located on the wrist of the robot (as shown in Fig. 6.7) so that 16 readings are generated for each finger on each cycle. The collected sensor data is pre-amplified by circuitry in the finger tip and then sampled by a 10-bit A/D converter before streaming the data back to the robot over a serial link. In spite of the 950nm operating frequency, raw sensor readings remain sensitive to ambient light effects. Background subtraction was used to remove this ambient component and increase the signal to noise ratio.

---

[3]While the crosstalk between sensors 1-3 can be useful when the primary sensor values saturate, which occurs on many light-colored surfaces, experimental testing showed the crosstalk to be below the noise floor on many of the object surfaces we encountered. Since this work assumes that the nature of the surface is unknown a priori, our current work ignores the crosstalk and only uses the primary sensor values.

6.5.2   Sensor model

Given a series of observations from the sensors on each finger, our goal is to estimate the surface pose (distance and orientation) of the surface in front of the finger.

More formally, let $o = (o_1, o_2, o_3) \in \mathbb{R}^3$ be the readings from the three sensors grouped in a triangle,[4] and $s = (d, x_{rot}, z_{rot})$ be the surface pose of the local surface (approximated as a plane). Here, d is the straight-line distance from the center of the three sensors to the surface of the object (along the vector pointing outwards from the finger surface) sensors. x_{rot} and z_{rot} are the relative orientations of the surface around the finger's x-axis (pitch) and z-axis (roll) respectively, as shown in Fig. 6.10. (They are equal to 90° when the object surface is parallel to the finger surface.)

One of the major challenges in the use of optical sensors is that intrinsic surface properties (such as reflectivity, diffusivity, etc.) cause the relationship between the raw sensor signal and the surface pose to vary significantly across different surface types. For that reason, prior work using short-range proximity sensors to find surface pose has focused on using multiple direct models obtained by performing regression on empirical data. This data is gathered by recording sensor array readings for each relevant surface in turn, placed at a comprehensive set of known surface poses [10]. In particular, [10, 106, 179] both use a least-squares polynomial fit of data taken for a known surface or group of similar surfaces to directly estimate surface pose given sensor readings. However, acquiring enough data to successfully model a new surface is extremely time-consuming, and having to do so for every potential surface that might be encountered is prohibitive. In practical grasping scenarios, we need to be able to deal with unknown and never-before-encountered surfaces.

Reference Forward Model

As opposed to fully characterizing every surface with a separate model, we use a single reference model that is scaled with an estimate of the object's IR reflectivity

---

[4]Sensor 4, which is offset by 45° to increase the field of view, is not included in this sensor model because it rarely focuses on the same flat surface as the other three. As a stand-alone sensor, it is nevertheless independently useful, particularly when the peak expected value (object surface reflectivity) is known. For instance, it can be used to prevent unexpected collisions with objects while moving, or even to move to a fixed distance from a table or other surface with known orientation.

parameter in order to obtain an approximate forward model for the specific object of interest.[5] By scaling with the estimated peak value, the model becomes roughly invariant to surface brightness.

For this work, the calibration data was taken using a Kodak grey card, which is a surface often used to adjust white balance in photography. The data consists of 1904 samples of o taken at distance values ranging from 0.5 cm to 3.4 cm, x_{rot} values ranging from 30° to 110°, and z_{rot} values ranging from 40° to 140°. Fig. 6.11 shows the calibration data.
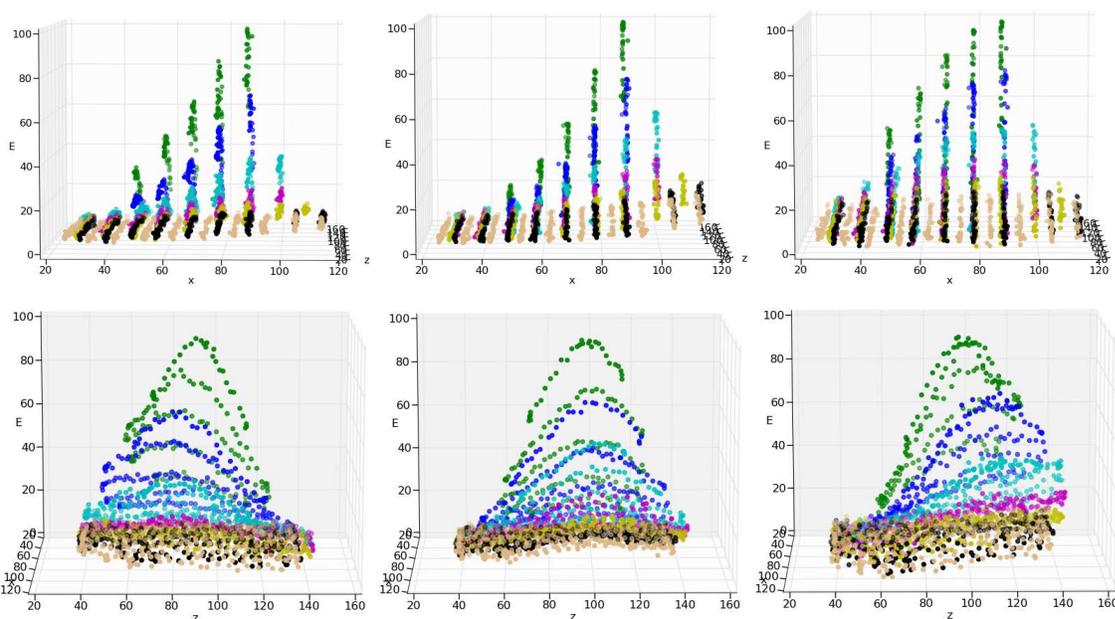
Figure 6.11: Calibration Data. (Top row) plots xrot vs. energy for all three sensors, with binned distances represented by different colors (distance bin centers in mm: green=5, blue=9, cyan=13, magenta=19, yellow=24, black=29, burlywood=34). (Bottom row) shows zrot vs. energy for all three sensors.

---

[5]Estimation of the surface reflectivity value is not particularly onerous to collect, as a single grasp of the object, particularly with the use of raw values to attempt to align at least one finger with the surface of the object, is sufficient to collect the required value. Alternatively, an estimate of the peak value could be obtained by observing the object with an IR camera or with a laser rangefinder that provides IR intensity values.
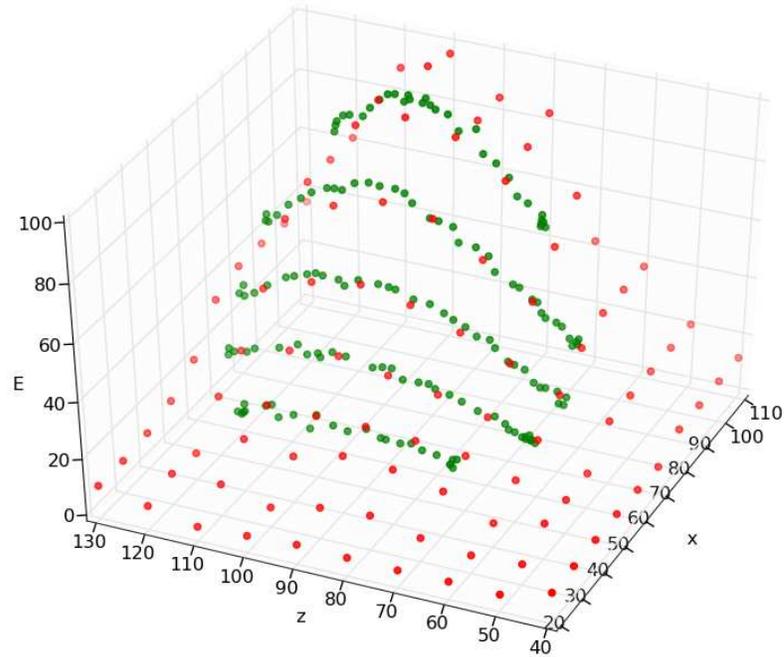
Figure 6.12: Locally weighted linear regression on calibration data. The plot shows energy values for one sensor at a fixed distance and varying x and z orientation. Green points are recorded data; red points show the interpolated and extrapolated estimates of the model.

Belief State Model

In order to address the sensitivity of the direct model to the object surface characteristics, we use an empirical forward sensor model and a probabilistic estimation process to derive the current best estimate for s. In our setting, as our robot hand is executing a grasp trajectory, we take many sensor readings with fingers at different (known) locations, and use the sensor readings, the finger positions, and our empirical sensor model to update a belief state (a probability distribution over possible values of s) at each time step.

Here, we assume that the object is static during a grasp. This is a reasonable assumption due to the fact that because we are using proximity or 'pre-touch' sensors, the actual grasp and sensing procedure does not make contact with the object prior to complete closure of the grasp and thus does not actively cause any object

displacements. In addition, we assume that the surface seen under each finger is locally planar throughout the entire grasp process.[6]

For each finger, let S be the set of all possible states s, and let s_t, o_t and a_t be the state, sensor readings (observation), and hand pose (actions) at time t, respectively. At all times, we will track the belief state $b_t := P(s_0|o_1...o_t, a_1...a_t)$, the probability distribution over all possible surface poses in S seen at the first time step, given all observations and hand poses since. We assume that our sensor observations at different times are conditionally independent given the surface pose:

$$P(s_0|o_1...o_T, a_1...a_T) = \prod_{t=1}^{T} P(s_0|o_t, a_t) \qquad (6.3)$$

Observation Model

An essential part of finding the most likely surface pose is having a model for how likely it is that we could have seen the current sensor readings if that pose were true. Specifically, the quantity we are looking for is $P(o_t|a_t, s_t)$, the probability of seeing the sensor readings we obtained at time t (o_t) given the current hand configuration (a_t) and a particular surface pose (s_t). For this, we need a function mapping states to observations, $C(s) = o$. We use locally weighted linear regression on our scaled calibration data set to estimate o given s. An example of the values obtained using locally weighted linear regression is shown in Fig. 6.12, where the green points are actual data and the red points are estimated values. Each estimated point uses a plane derived from only the 8 closest actual data points. Also, because extrapolation using locally weighted linear regression is extremely poor, the estimated point is clipped to be no greater than the highest of those 8 values, and no less than the lowest.

We then assume that the estimated model values are correct (for a given s_t, we expect to see the model-estimated o_t), and that any deviations we see in the actual sensor readings, $\epsilon = o_t - C(s_t)$, are due to Gaussian noise, $\epsilon \sim N(0, \sigma^2)$. This assumption is wildly inaccurate, as the errors are in fact systematic, but this assumption

---

[6]Even when the surface under the finger changes and the old data becomes erroneous, the estimates would get progressively better as more data from the new surface is observed. It is also simple to place a higher weight on new data, to make the adjustment faster.

nonetheless allows one to find the closest alignment of the observed o points to the scaled calibration model without any assumptions about surface characteristics. For our experiments, sensor readings were scaled to vary from 1 to 100, and σ was set to be 25, since we expect significant deviations from our model values.

Inference

At each time step, we wish to compute an estimate $\hat{s}_t$ of the current state. The states, S, are discretized to a uniform grid. We can represent b_t as a grid of probabilities that sum to 1, and update the belief state probabilities at each time step using our actions and observations as we would any Bayesian filter.

More specifically, the belief state update is performed as follows:

$$
\begin{aligned}
b_t &= P(s_0|o_1...o_t, a_1...a_t) \\
&= \frac{P(o_t|a_0, a_t, s_0)P(s_0|o_1...o_{t-1}, a_1...a_{t-1})}{P(o_t)} \\
&= \frac{P(o_t|a_t, s_t)b_{t-1}}{P(o_t)}
\end{aligned}
\tag{6.4}
$$

We assume a uniform prior on the sensor readings, and thus the denominator can be normalized out. We then find the expected value of the state s_0 as follows:

$$
\hat{s}_0 = E(s_0) = \sum_{s_0} P(s_0|o_1...o_t, a_1...a_t)s_0
\tag{6.5}
$$

We can compute $\hat{s}_t$ from $\hat{s}_0$ using the hand kinematics and the known hand positions a_t and a_0.

The advantage of combining observations in this manner is that, while the actual sensor readings we expect to see vary greatly from surface to surface, readings across different sensors in the array vary in a similar way with respect to orientation changes for all surfaces. Thus, a number of observations over a wide range of different values of s can be used to align a new set of observations with a model that is not quite right. The s with expected (model) observations that align best with the actual observations is generally the one with the highest likelihood, even with large model error.
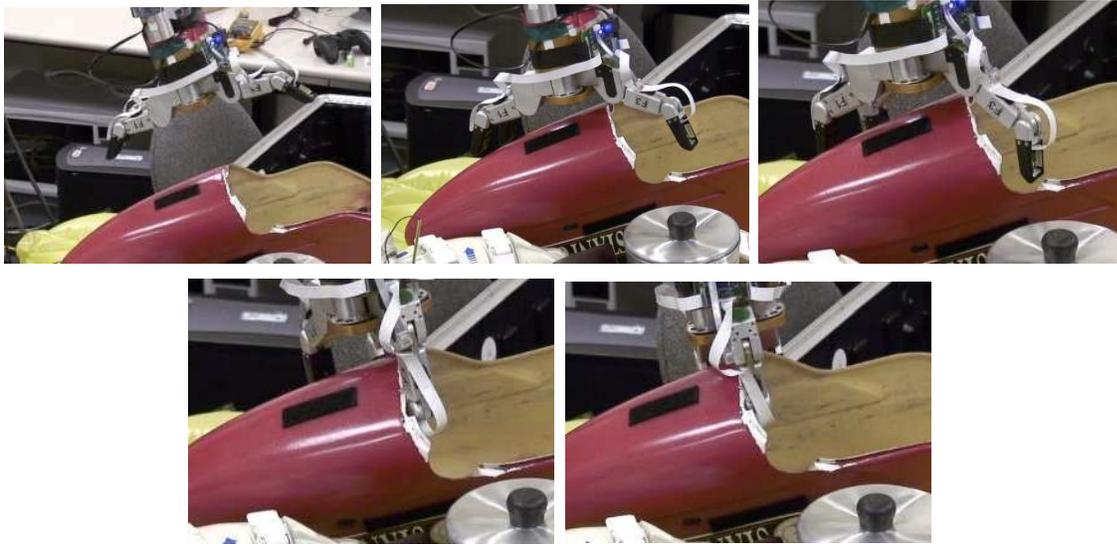
Figure 6.13: A sequence showing the grasp trajectory chosen by our algorithm. Initially, the fingers are completely open; as more data comes, the estimates get better, and the hand turns and closes the finger in such a way that all the fingers touch the object at the same time.

6.5.3   Reactive grasp closure controller

To verify the usefulness of our 'pre-grasp' proximity sensors, a reactive grasp closure controller was designed and implemented that uses the proximity data to move the fingers such that: 1) they do not collide with the object, and 2) they close simultaneously, forming a symmetric grasp configuration around a pre-specified grasp point. This ability is aimed at achieving successful grasps even in the presence of significant uncertainty about the object geometry, and at allowing the sensor interpretation approach to safely collect data to improve the surface estimates.

The hierarchical control architecture used here composes two reactive control elements that run asynchronously and that control separate aspects of the grasp closure process. At the bottom level, a finger distance controller controls the fingers to maintain approximately equal distances to the object surface. On top of this control element, a kinematic conditioning controller controls the arm to center the object within the hand and to cause the fingers to be parallel to the surface.

Figure 6.14: Different objects on which we present our analysis of the model predictions. See Section 6.5.4 for details.

Table 6.4: Model Test Experiment Error

	DIST(CM)	$x_{rot}(°)$	$z_{rot}(°)$
BROWN WOOD BOWL	.35	4.4	13.5
BEIGE PLASTIC BOWL	.34	4.4	23.0
BLACK PLASTIC BOX	.49	10.3	16.5
BLUE PLASTIC BOX	.59	5.3	21.7
CARDBOARD BOX	.20	3.8	14.4
YELLOW BOX	.43	3.6	17.3
AVERAGE	.40	5.3	17.7

6.5.4 Experiments

We will first evaluate the ability of our algorithm to model sensor data in Section 6.5.4. Then we will use the model to perform grasping experiments in Section 6.5.4.

The sensors described in Section 6.5.1 were mounted onto the finger tips of a Barrett hand. This manipulator has a total of 4 degrees of freedom (three fingers each with one degree of freedom, and a finger "spread"). The hand was attached to a 7-dof arm (WAM, by Barrett Technologies) mounted on the STAIR (STanford AI Robot) platform.

Prediction Model

As a test of our estimation method, we placed the Barrett hand at known positions relative to a number of commonly found objects of various materials. (See Fig. 6.14 for their pictures.) Our test data consisted of three different orientations for each object: parallel to the fingers, +10° rotation, and −20° rotation. The fingers were

then made to close slowly around the object until each finger nearly touched the surface. Sets of 100 raw sensor readings were taken every 1.2 degrees of finger base joint bend, and for every 1.2 degrees of finger spread (up to at most 17 degrees) and averaged. Results showing errors in the final estimates of (d, x_{rot}, z_{rot}) are in Table 6.4.

Table 6.4 shows that our model is able to predict the distances with an average error of 0.4cm. We are also able to estimate x_{rot} reasonably accurately, with an average error of 5.3°. The high error in x_{rot} in the case of the black plastic box can be attributed to the fact that the first finger in one run did not see the same surface the entire time, which is a requirement for reasonable predictions with our model. Note that these objects have significantly different surface properties, and other than the peak sensor value, no other surface characteristics were assumed. High-reflectance surfaces tended to do worse than matte surfaces due to significant deviation from the calibration model. Nonetheless, our experiments show that we are able to make reasonable predictions by the time the fingers touch the surface.

The higher errors in z_{rot} are a consequence of the movements used during the data collection and estimation process. Since the sensor interpretation approach uses belief updates to determine the maximum likelihood orientation, the quality of the resulting estimates depends on the proximity data encountered along the fingers' trajectories, with larger variations in the local geometry resulting in more accurate estimates. In the procedure used here to gather the test data, a wide range of x_{rot} angles was encountered within the useful range of the sensors due to the strong curling of the Barrett fingers. On the other hand, only a very limited range of z_{rot} angles were correctly observed due to the significantly smaller variation available using the hand's spread angle. Furthermore, most spread actions moved the fingers to positions significantly further from the object, often resulting in sensor readings that were no longer observable. While this is a problem in the test data and illustrates the advantage of active sensing strategies, its cause should be largely alleviated when using the grasp closure controller to establish the initial grasp, due to the ability of the finger distance controller to maintain the fingers at a distance that provides usable results throughout the finger spread operation. Additionally, the inclusion of arm motions

through the use of the kinematic conditioning controller should further enhance the range of z_{rot} angles encountered during a grasp, and thus allow for somewhat better z_{rot} estimates.

Grasping Experiments

Our goal was to focus on the final approach of grasping using proximity sensors. Our system could be used in a variety of settings, including the point-and-click approach of [60], where a laser pointer is used to highlight an object for a robot to pick it up, or combined with long range vision sensors that select optimal grasp points [136].

In this experiment, we placed a variety of objects (weighing less than 3 pounds) in known locations on a table (see Fig. 6.16), with some objects flush against each other. These objects are of a variety of shapes, ranging from simple boxes or bowls, to more complicated shapes such as a ski boot.

The robot moves the hand to the approximate center of the object and executes a grasp strategy,[7] using our controller to move the hand and the fingers in response to estimated distances from the sensors and the pose estimation algorithm. The robot then picks up the object and moves it to verify that the grasp is stable. (See Fig. 6.13 and Fig. 6.16 for some images of the robot picking up the objects.)
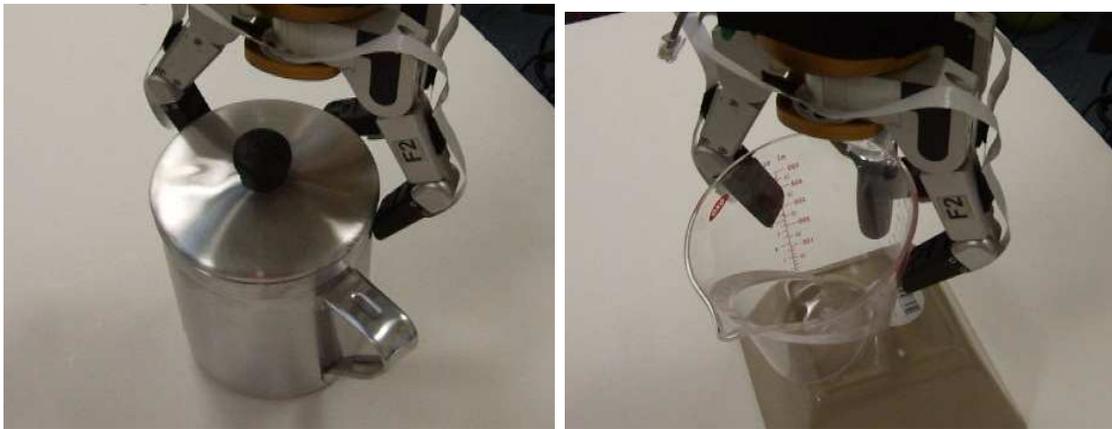
Figure 6.15: Failure cases: (a) Shiny can, (b) Transparent cup

---

[7]One could envision using better strategies, such as those based on vision-based learning [140, 141], for moving the hand to a grasping point on the object.
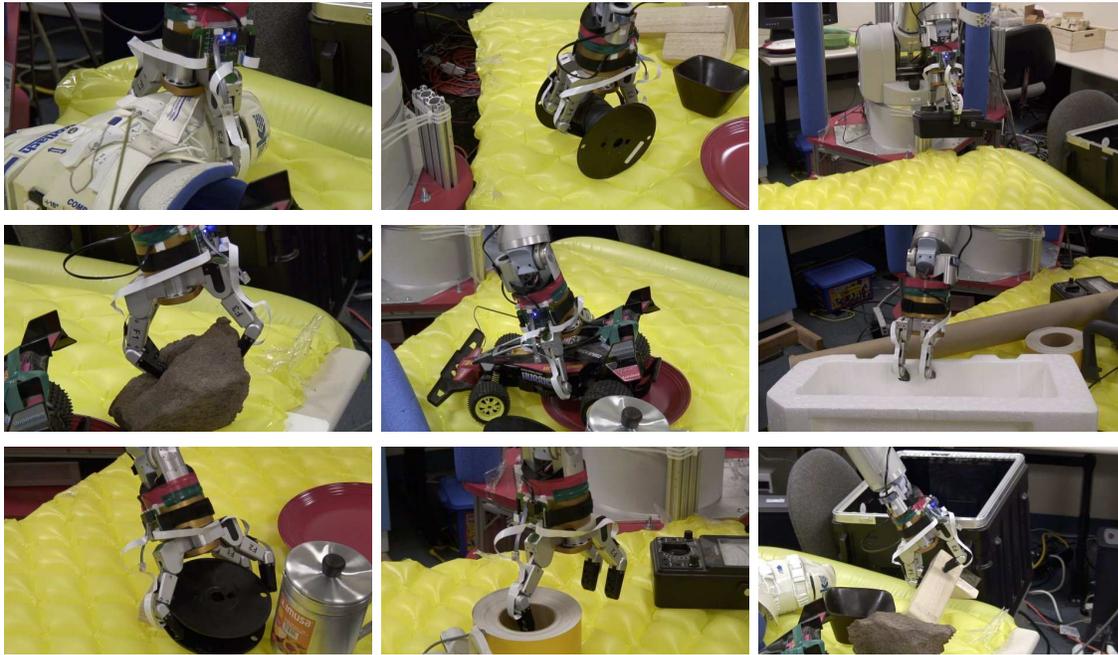
Figure 6.16: Snapshots of the robot picking up different objects.

Note that our objects are of a variety of shapes and made of a variety of materials. Out of the 26 grasping trials performed on 21 unique objects,[8] our grasping system failed three times. Two of these failures were due to extreme surface types: a transparent cup and a highly reflective aluminum can (Fig. 6.15). To address these cases, our optical-based proximity sensors could be combined with a capacitive-based system that is good at detecting metallic or glass surfaces. The third object was a ski-boot, for which our approach worked perfectly; however, due to the object's weight, our robot was unable to lift it up.

A video of the grasp experiments is available at the following url:

http://stair.stanford.edu/proximitygrasping/

In the video, the hand rotates the fingers in many cases to be approximately

---

[8]*Procedure used to choose the objects*: We asked a person not associated with the project to bring objects larger than about 8 inches in length and less than 3 pounds from our lab and different offices. No other selection criterion was specified, and therefore we believe that our objects represented a reasonably unbiased sample of commonly found objects.

parallel to the object surface and causes the fingers to contact at nearly the same time, thereby improving the grasp.

6.6 Applications to opening new doors

Recently, there is growing interest in using robots not only in controlled factory environments but also in unstructured home and office environments. In the past, successful navigation algorithms have been developed for robots in these environments. However, to be practically deployed, robots must also be able to manipulate their environment to gain access to new spaces. In Klingbeil, Saxena, and Ng [64], we presented our work towards enabling a robot to autonomously navigate anywhere in a building by opening doors, even those it has never seen before.

Most prior work in door opening (e.g., [122, 110]) assumes that a detailed 3D model of the door (and door handle) is available, and focuses on developing the control actions required to open one specific door. In practice, a robot must rely on only its sensors to perform manipulation in a new environment.

Our methods for grasping, described in previous chapters use a vision-based learning approach to choose a point at which to grasp an object. However, a task such as opening a door is more involved in that it requires a series of manipulation tasks; the robot must first plan a path to reach the handle and then apply a series of forces/torques (which may vary in magnitude and direction) to open the door. The vision algorithm must be able to infer more information than a single grasp point to allow the robot to plan and execute such a manipulation task.

In this chapter, we focus on the problem of manipulation in novel environments where a detailed 3D model of the object is not available. We note that objects, such as door handles, vary significantly in appearance, yet similar function implies similar form. Therefore, we will design vision-based learning algorithms that attempt to capture the visual features shared across different objects that have similar functions. To perform a manipulation task, such as opening a door, the robot end-effector must move through a series of way-points in cartesian space, while achieving the desired orientation at each way-point, in order to turn the handle and open the door. A

small number of keypoints such as the handle's axis of rotation and size provides sufficient information to compute such a trajectory. We use vision to identify these "visual keypoints," which are required to infer the actions needed to perform the task. To open doors or elevators, there are various types of actions a robot can perform; the appropriate set of actions depends on the type of control object the robot must manipulate, e.g., left/right turn door handle, spherical doorknob, push-bar door handle, elevator button, etc. Our algorithm learns the visual features that indicate the appropriate type of control action to use.

Finally, to demonstrate the robustness of our algorithms, we provide results from extensive experiments on 20 different doors in which the robot was able to reliably open new doors in new buildings, even ones which were seen for the first time by the robot (and the researchers working on the algorithm).

6.6.1   Related work

There has been a significant amount of work done in robot navigation [174]. Many of these use a SLAM-like algorithm with a laser scanner for robot navigation. Some of these works have, in fact, even identified doors [34, 184, 166, 5, 1]. However, all of these works assumed a *known* map of the environment (where they could annotate doors); and more importantly none of them considered the problem of enabling a robot to autonomously open doors.

In robotic manipulation, most work has focused on developing control actions for different tasks, such as grasping objects [7], assuming a perfect knowledge of the environment (in the form of a detailed 3D model). Recently, some researchers have started using vision-based algorithms for some applications, e.g. [69]. These algorithms do not apply to manipulation problems where one needs to estimate a full trajectory of the robot and also consider interactions with the object being manipulated.

There has been some recent work in opening doors using manipulators [128, 108, 62, 122, 110, 55]; however, these works focused on developing control actions assuming a pre-surveyed location of a known door handle. In addition, these works implicitly assumed some knowledge of the type of door handle, since a turn lever door handle

must be grasped and manipulated differently than a spherical door knob or a push-bar door.

Little work has been done in designing autonomous elevator-operating robots. Notably, [91] demonstrated their robot navigating to different floors using an elevator, but their training phase (which requires that a human must point out where the appropriate buttons are and the actions to take for a given context) used the same elevator as the one used in their test demonstration. Other researchers have addressed the problem of robots navigating in elevators by simply having the robot stand and wait until the door opens and then ask a human to press the correct floor button [29].

Table 6.5: Visual keypoints for some manipulation tasks.

Manipulation task	Visual Keypoints
Turn a door handle	1. Location of the handle
	2. Its axis of rotation
	3. Length of the handle
	4. Type (left-turn, right-turn, etc.)
Press an elevator button	1. Location of the button
	2. Normal to the surface
Open a dishwasher tray	1. Location of the tray
	2. Direction to pull or push it

In the application of elevator-operating robots, some robots have been deployed in places such as hospitals [130, 30]. However, expensive modifications must be made to the elevators, so that the robot can use a wireless communication to command the elevator. For opening doors, one can also envision installing automatic doors, but our work removes the need to make these expensive changes to the many elevators and doors in a typical building.

In contrast to many of these previous works, our work does not assume existence of a known model of the object (such as the door, door handle, or elevator button) or a precise knowledge of the location of the object. Instead, we focus on the problem of manipulation in novel environments, in which a model of the objects is not available, and one needs to rely on noisy sensor data to identify visual keypoints. Some of these keypoints need to be determined with high accuracy for successful manipulation

(especially in the case of elevator buttons).

6.6.2   Algorithm

Consider the task of pressing an elevator button. If our perception algorithm is able to infer the location of the button and a direction to exert force in, then one can design a control strategy to press it. Similarly, in the task of pulling a drawer, our perception algorithm needs to infer the location of a point to grasp (e.g., a knob or a handle) and a direction to pull. In the task of turning a door handle, our perception algorithm needs to infer the size of the door handle, the location of its axis, and a direction to push, pull or rotate.

More generally, for many manipulation tasks, the perception algorithm needs to identify a set of properties, or "visual keypoints" which define the action to be taken. Given these visual keypoints, we use a planning algorithm that considers the kinematics of the robot and the obstacles in the scene, to plan a sequence of control actions for the robot to carry out the manipulation task.

Dividing a manipulation task into these two parts: (a) an algorithm to identify visual keypoints, and (b) an algorithm to plan a sequence of control actions, allows us to easily extend the algorithm to new manipulation tasks, such as opening a dishwasher. To open a dishwasher tray, the visual keypoints would be the location of the tray and the desired direction to move it. This division acts as a bridge between state of the art methods developed in computer vision and the methods developed in robotics planning and control.

6.6.3   Identifying visual keypoints

Objects such as door handles vary significantly in appearance, yet similar function implies similar form. Our learning algorithms will, therefore, try to capture the visual features that are shared across different objects having similar function.

In this paper, the tasks we consider require the perception algorithm to: (a) locate the object, (b) identify the particular sub-category of object (e.g., we consider door handles of left-turn or right-turn types), and (c) identify some properties such as the

surface normal or door handle axis of rotation. An estimate of the surface normal helps indicate a direction to push or pull, and an estimate of the door handle's axis of rotation helps in determining the action to be performed by the arm.

In the field of computer vision, a number of algorithms have been developed that achieve good performance on tasks such as object recognition [160, 132]. Perception for robotic manipulation, however, goes beyond object recognition in that the robot not only needs to locate the object but also needs to understand what task the object can perform and how to manipulate it to perform that task. For example, if the intention of the robot is to enter a door, it must determine the type of door handle (i.e., left-turn or right-turn) and an estimate of its size and axis of rotation, in order to compute the appropriate action (i.e., to turn the door handle left and push/pull).

Manipulation tasks also typically require more accuracy than what is currently possible with most classifiers. For example, to press an elevator button, the 3D location of the button must be determined within a few millimeters (which corresponds to a few pixels in the image), or the robot will fail to press the button. Finally, another challenge in designing perception algorithms is that different sensors are suitable for different perception tasks. For example, a laser range finder is more suitable for building a map for navigation, but a 2D camera is a better sensor for finding the location of the door handle. We will first describe our image-based classifier.

Object Recognition

To capture the visual features that remain consistent across objects of similar function (and hence appearance), we start with a 2D sliding window classifier. We use a supervised learning algorithm that employs boosting to compute a dictionary of Haar features.

In detail, the supervised training procedure first randomly selects ten small windows to produce a dictionary of Haar features [176]. In each iteration, it trains decision trees using these features to produce a model while removing irrelevant features from the dictionary.

There are a number of contextual properties that we take advantage of to improve the classification accuracy. Proximity of objects to each other and spatial cues, such

Figure 6.17: Results on test set. The green rectangles show the raw output from the classifiers, and the blue rectangle is the one after applying context.

as that a door handle is less likely to be found close to the floor, can be used to learn a location based prior (partly motivated by [176]).

We experimented with several techniques to capture the fact that the labels (i.e., the category of the object found) have correlation. An algorithm that simply uses non-maximal suppression of overlapping windows for choosing the best candidate locations resulted in many false positives—12.2% on the training set and 15.6% on the test set. Thus, we implemented an approach that takes advantage of the context for the particular objects we are trying to identify. For example, we know that doors (and elevator call panels) will always contain at least one handle (or button) and never more than two handles. We can also expect that if there are two objects, they will lie in close proximity to each other and they will likely be horizontally aligned (in the case of door handles) or vertically aligned (in the case of elevator call buttons). This approach resulted in much better recognition accuracy.

Figure 3 shows some of the door handles identified using our algorithm. Table 6.6

Table 6.6: Accuracies for recognition and localization.

	RECOGNITION	LOCALIZATION
DOOR HANDLE	94.5%	93.2%
ELEVATOR BUTTONS	92.1%	91.5%

shows the recognition and localization accuracies. "Localization accuracy" is computed by assigning a value of 1 to a case where the estimated location of the door handle or elevator button was within 2 cm of the correct location and 0 otherwise. An error of more than 2 cm would cause the robot arm to fail to grasp the door handle (or push the elevator button) and open the door.

Once the robot has identified the location of the object in an image, it needs to identify the object type and infer control actions from the object properties to know how to manipulate it. Given a rectangular patch containing an object, we classify what action to take. In our experiments, we considered three types of actions: turn left, turn right, and press. The accuracy of the classifier that distinguishes left-turn from right-turn handles was 97.3%.

Estimates from 3D data

Our object recognition algorithms give a 2D location in the image for the visual keypoints. However, we need their corresponding 3D locations to be able to plan a path for the robot.

In particular, once an approximate location of the door handle and its type is identified, we use 3D data from the stereo camera to estimate the axis of rotation of the door handle. Since, the axis of a right- (left-) turn door handle is the left- (right-) most 3D point on the handle, we build a logistic classifier for door-axis using two features—distance of the point from the door and its distance from the center (towards left or right). Similarly, we use PCA on the local 3D point cloud to estimate the orientation of the surface—required in cases such as elevator buttons and doors for identifying the direction to apply force.

However, the data obtained from a stereo sensor is often noisy and sparse in that

the stereo sensor fails to give depth measurements when the areas considered are textureless, e.g., blank elevator walls [146]. Therefore, we also present a method to fuse the 2D image location (inferred by our object recognition algorithm), with a horizontal laser scanner (available on many mobile robots) to obtain the 3D location of the object. Here, we make a ground-vertical assumption—that every door is vertical to a ground-plane [26].[9] This enables our approach to be used on robots that do not have a 3D sensor such as a stereo camera (that are often more expensive).

Planning and Control

Given the visual keypoints and the goal, we build upon a Probabilistic RoadMap (PRM) [159] motion planning algorithm for obtaining a smooth, collision-free path for the robot to execute.

6.6.4   Experiments

We used a Voronoi-based global planner for navigation; this enabled the robot to localize itself in front of and facing a door within 20cm and ± 20 degrees. An experiment began with the robot starting at a random location within 3m of the door. It used lasers to navigate to the door, and our vision-based classifiers to find the handle.

In the experiments, our robot saw all of our test locations for the first time. The training images for our vision-based learning algorithm were collected in completely separate buildings, with different doors and door handle shapes, structure, decoration, ambient lighting, etc. We tested our algorithm on two different buildings on a total

---

[9]In detail, a location in the image corresponds to a ray in 3D, which would intersect the plane in which the door lies. Let the planar laser readings be denoted as $l_i = (x(\theta_i), y(\theta_i))$. Let the origin of the camera be at $c \in \mathbb{R}^3$ in arm's frame, and let $r \in \mathbb{R}^3$ be the unit ray passing from the camera center through the predicted location of the door handle in the image plane. I.e., in the robot frame, the door handle lies on a line connecting c and $c + r$.

Let $T \in \mathbb{R}^{2 \times 3}$ be a projection matrix that projects the 3D points in the arm frame into the plane of the laser. In the laser plane, therefore, the door handle is likely to lie on a line passing through Tc and $T(c + r)$.

$$t^* = \min_t \sum_{i \in \Psi} ||T(c + rt) - l_i||_2^2 \tag{6.6}$$

where Ψ is a small neighborhood around the ray r. Now the location of the 3D point to move the end-effector to is given by $s = c + rt^*$.

Table 6.7: Error rates obtained for the robot opening the door in a total number of 34 trials.

Door Type	Num of trials	Recog. (%)	Class. (%)	Localiza-tion (cm)	Success-rate
Left	19	89.5%	94.7%	2.3	84.2%
Right	15	100%	100%	2.0	100%
Total	34	94.1%	97.1%	2.2	91.2%

of five different floors (about *20 different* doors). Many of the test cases were also run where the robot localized at different angles, typically between -30 and +30 degrees with respect to the door, to verify the robustness of our algorithms.

In a total of 34 experiments, our robot was able to successfully open the doors *31 out of 34* times. Table 6.7 details the results; we achieved an average recognition accuracy of 94.1% and a classification accuracy of 97.1%. We define the localization error as the mean error (in cm) between the predicted and actual location of the door handle. This led to a success-rate (fraction of times the robot actually opened the door) of 91.2%. Notable failures among the test cases included glass doors (erroneous laser readings), doors with numeric keypads, and very dim/poor lighting conditions. In future work, we plan to use active vision [57], which takes visual feedback into account while objects are being manipulated and thus provides complementary information that would hopefully improve the performance on these tasks.

Videos of the robot opening new doors are available at:

http://ai.stanford.edu/∼asaxena/openingnewdoors

6.7 Discussion

We presented a learning algorithm that, given partial models of the scene from 3-d sensors, selects a grasp—that is, a configuration of the robot's arm and fingers—to try to pick up an object. We tested our approach on a robot grasping a number of objects cluttered backgrounds.

We also presented a novel gripper-mounted optical proximity sensor that enables stable grasping of common objects using pre-touch pose estimation. components. We

Figure 6.18: Some experimental snapshots showing our robot opening different types of doors.

converted the data provided by these sensors into pose estimates of nearby surfaces by a probabilistic model that combines observations over a wide range of finger/hand configurations. We validated the system with experiments using a Barrett hand, which showed improvements in reactive grasping.

To navigate and perform tasks in unstructured environments, robots must be able to perceive their environments to identify what objects to manipulate and how they can be manipulated to perform the desired tasks. We presented a framework that identifies some visual keypoints using our vision-based learning algorithms. Our robot was then able to use these keypoints to plan and execute a path to perform the desired task. This strategy enabled our robot to open a variety of doors, even ones it had not seen before.

Chapter 7

Conclusions

One of the most basic perception abilities for a robot is perceiving the 3D shape of an environment. This is also useful in a number of robotics applications such as in navigation and in manipulation.

In this dissertation, we described learning algorithms for depth perception from a single still image. The problem of depth perception is fundamental to computer vision, one that has enjoyed the attention of many researchers and seen significant progress in the last few decades. However, the vast majority of this work, such as stereopsis, has used multiple image geometric cues to infer depth. In contrast, single-image cues offer a largely orthogonal source of information, one that had heretofore been relatively underexploited.

We proposed an algorithm for single image depth perception that used a machine learning technique called Markov Random Field (MRF). It successfully modeled the spatial dependency between different regions in the image. The estimated 3D models were both quantitatively accurate and visually pleasing. Further, our algorithm was able to accurately estimate depths from images of a wide variety of environments. We also studied the performance of our algorithm through a live deployment in the form of a web service called Make3D. Tens of thousands of users successfully used it to automatically convert their own pictures into 3D models.

Given that depth and shape perception appears to be an important building block for many other applications, such as object recognition, image compositing and video

retrieval, we believe that monocular depth perception has the potential to improve all of these applications, particularly in settings where only a single image of a scene is available.

One such application that we described in this dissertation was to improve the performance of 3D reconstruction from multiple views. In particular, we considered stereo vision, where the depths are estimated by triangulation using two images. Over the past few decades, researchers have developed very good stereo vision systems. Although these systems work well in many environments, stereo vision is fundamentally limited by the baseline distance between the two cameras. We found that monocular cues and (purely geometric) stereo cues give largely orthogonal, and therefore complementary, types of information about depth. Stereo cues are based on the difference between two images and do not depend on the content of the image. On the other hand, depth estimates from monocular cues are entirely based on the evidence about the environment presented in a single image.

We investigated how monocular cues can be integrated with any reasonable stereo system, to obtain better depth estimates than the stereo system alone. Our learning algorithm (again using Markov Random Fields) modeled the depth as a function of both single image features as well as the depths obtained from stereo triangulation if available, and combined them to obtain better depth estimates. We then also created 3D models from multiple views (which are not necessarily from a calibrated stereo pair) to create a larger 3D model.

We applied these ideas of depth perception to a challenging problem of obstacle avoidance for a small robot navigating through a cluttered environment. For small-sized robots, the size places a frugal payload restriction on weight and power supply. In these robots, only light-weight small sensors that take less power could be used. Cameras are an attractive option because they are small, passive sensors with low power requirements.

We used supervised learning to learn monocular depth estimates (distances to obstacles in each steering direction) from a single image captured by the onboard camera, and used them to drive an RC car at high speeds through the environment. The vision system was able to give reasonable depth estimate on real image data,

and a control policy trained in a graphical simulator worked well on real autonomous driving. Our method was able to avoid obstacles and drive autonomously in novel environments in various locations.

We then considered the problem of grasping novel objects. Grasping is an important ability for robots and is necessary in many tasks, such as for household robots cleaning up a room or unloading items from a dishwasher. Most of the prior work had assumed that a known (and registered) full 3D model of the object is available. In reality, it is quite hard to obtain such a model. We presented a supervised learning algorithm that predicts, directly as a function of the images, a point at which to grasp the object. In our experiments, the algorithm generalized very well to novel objects and environments, and our robot successfully grasped a wide variety of objects in different environments such as dishwashers, office and kitchen.

Next, we considered more complex arms and hands (e.g., multi-fingered hands where one has to not only choose a 3D position to grasp the object at, but also address the high degree-of-freedom problem of choosing the positions of all the fingers. This is further complicated by presence of significant amounts of clutter. A key challenge in this setting is that a full 3D model of the scene is typically not available. Instead, a robot's sensors (such as a camera or a depth sensor) can usually estimate only the shape of the visible portions of the scene. We presented an algorithm that, given such partial models of the scene, selects a grasp—that is, a configuration of the robot's arm and fingers—to try to pick up an object.

In order to improve performance of grasping, we also presented a novel gripper-mounted optical proximity sensor that enabled stable grasping of common objects using pre-touch pose estimation. These sensors are a nice complement to the long-range vision sensors (such as the cameras).

A household robot not only needs to pick up objects, but it also needs to navigate and perform other manipulation tasks. One such manipulation task is the task of opening new doors. We presented a method that identifies some visual keypoints using vision. This enabled our robot to navigate to places inaccessible before by opening doors, even the ones it had not seen before.

Bibliography

[1] D. Anguelov, D. Koller, E. Parker, and S. Thrun. Detecting and modeling doors with mobile robots. In *ICRA*, 2004.

[2] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. SCAPE: shape completion and animation of people. *ACM Trans. Graph.*, 24 (3):408–416, 2005. ISSN 0730-0301.

[3] J.L. Barron, D.J. Fleet, and S.S. Beauchemin. Performance of optical flow techniques. *IJCV*, 12:43–77, 1994.

[4] H. Bay, T. Tuytelaars, and L.V. Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, 2006.

[5] P. Biber and T. Duckett. Dynamic maps for long-term operation of mobile service robots. In *RSS*, 2005.

[6] A. Bicchi. On the closure properties of robotic grasping. *IJRR*, 14(4):319–334, 1995.

[7] A. Bicchi and V. Kumar. Robotic grasping and contact: a review. In *International Conference on Robotics and Automation (ICRA)*, 2000.

[8] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] I. Blthoff, H. Blthoff, and P. Sinha. Top-down influences on stereoscopic depth-perception. *Nature Neuroscience*, 1:254–257, 1998.

[10] A. Bonen. Development of a robust electro-optical proximity-sensing system. *Ph.D. Dissertation, Univ. Toronto*, 1995.

[11] T. G. R. Bower, J. M. Broughton, and M. K. Moore. Demonstration of intention in the reaching behaviour of neonate humans. *Nature*, 228:679–681, 1970.

[12] D.L. Bowers and R. Lumia. Manipulation of unmodeled objects using intelligent grasping schemes. *IEEE Transactions on Fuzzy Systems*, 11(3), 2003.

[13] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[14] M.Z. Brown, D. Burschka, and G.D. Hager. Advances in computational stereo. *IEEE Trans on Pattern Analysis and Machine Intelligence (PAMI)*, 25(8):993–1008, 2003. ISSN 0162-8828.

[15] M. Kelm C. Paul, X. Wang and A. McCallum. Multi-conditional learning for joint probability models with latent variables. In *NIPS Workshop Advances Structured Learning Text and Speech Processing*, 2006.

[16] U. Castiello. The neuroscience of grasping. *Nature Reviews Neuroscience*, 6: 726–736, 2005.

[17] E. Chinellato, R.B. Fisher, A. Morales, and A.P. del Pobil. Ranking planar grasp configurations for a three-finger hand. In *ICRA*, 2003.

[18] M. Ciocarlie, C. Goldfeder, and P. Allen. Dimensionality reduction for hand-independent dexterous robotic grasping. In *IROS*, 2007.

[19] J. Coelho, J. Piater, and R. Grupen. Developing haptic and visual perceptual categories for reaching and grasping with a humanoid robot. *Robotics and Autonomous Systems*, 37:195–218, 2001.

[20] N. Cornelis, B. Leibe, K. Cornelis, and L. Van Gool. 3d city modeling using cognitive loops. In *Video Proceedings of CVPR (VPCVPR)*, 2006.

[21] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. *International Journal of Computer Vision (IJCV)*, 40:123–148, 2000.

[22] N Dalai and B Triggs. Histogram of oriented gradients for human detection. In *Computer Vision and Pattern Recognition (CVPR)*, 2005.

[23] S. Das and N. Ahuja. Performance analysis of stereo, vergence, and focus as depth cues for active vision. *IEEE Trans Pattern Analysis & Machine Intelligence (IEEE-PAMI)*, 17:1213–1219, 1995.

[24] E.R. Davies. Laws' texture energy in TEXTURE. In *Machine Vision: Theory, Algorithms, Practicalities 2nd Edition*. Academic Press, San Diego, 1997.

[25] E. Delage, H. Lee, and A.Y. Ng. Automatic single-image 3d reconstructions of indoor manhattan world scenes. In *International Symposium on Robotics Research (ISRR)*, 2005.

[26] E. Delage, H. Lee, and A.Y. Ng. A dynamic bayesian network model for autonomous 3d reconstruction from a single indoor image. In *Computer Vision and Pattern Recognition (CVPR)*, 2006.

[27] J. Diebel and S. Thrun. An application of markov random fields to range sensing. In *NIPS*, 2005.

[28] A. Edsinger and C. Kemp. Manipulation in human environments. In *Int'l Conf Humanoid Robotics*, 2006.

[29] Simmons et al. Grace and george: Autonomous robots for the aaai robot challenge. In *AAAI Robot challenge*, 2004.

[30] J.M. Evans. Helpmate: an autonomous mobile robot courier for hospitals. In *IROS*, 1992.

[31] R. Ewerth, M. Schwalb, and B. Freisleben. Using depth features to retrieve monocular video shots. In *ACM International Conference on Image and Video Retrieval*, 2007.

[32] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59, 2004.

[33] D.A. Forsyth and J. Ponce. *Computer Vision : A Modern Approach*. Prentice Hall, 2003.

[34] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *JAIR*, 1999.

[35] C. Frueh and A. Zakhor. Constructing 3D city models by merging ground-based and airborne views. In *Computer Vision and Pattern Recognition (CVPR)*, 2003.

[36] G. Gini and A. Marchi. Indoor robot navigation with single camera vision. In *PRIS*, 2002.

[37] A. S. Glassner. *An Introduction to Ray Tracing*. Morgan Kaufmann Publishers, Inc., San Francisco, 1989.

[38] M. A. Goodale, A. D. Milner, L. S. Jakobson, and D. P. Carey. A neurological dissociation between perceiving objects and grasping them. *Nature*, 349:154–156, 1991.

[39] S. Gould, J. Arfvidsson, A. Kaehler, B. Sapp, M. Meissner, G. Bradski, P. Baumstarck, S. Chung, and A. Y. Ng. Peripheral-foveal vision for real-time object recognition and tracking in video. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[40] F. Han and S.C. Zhu. Bayesian reconstruction of 3d shapes and scenes from a single image. In *ICCV Workshop Higher-Level Knowledge in 3D Modeling Motion Analysis*, 2003.

[41] T. Hassner and R. Basri. Example based 3d reconstruction from single 2d images. In *CVPR workshop on Beyond Patches*, 2006.

[42] X. He, R.S. Zemel, and M.A. Carreira-Perpinan. Multiscale conditional random fields for image labeling. In *CVPR*, 2004.

[43] G. Heitz, S. Gould, A. Saxena, and D. Koller. Cascaded classification models: Combining models for holistic scene understanding. In *Neural Information Processing Systems (NIPS)*, 2008.

[44] A. Hertzmann and S.M. Seitz. Example-based photometric stereo: Shape reconstruction with general, varying brdfs. *IEEE Trans Pattern Analysis and Machine Intelligence (PAMI)*, 27(8):1254–1264, 2005.

[45] D. Hoiem, A.A. Efros, and M. Hebert. Putting objects in perspective. In *Computer Vision and Pattern Recognition (CVPR)*, 2006.

[46] D. Hoiem, A.A. Efros, and M. Herbert. Geometric context from a single image. In *International Conference on Computer Vision (ICCV)*, 2005.

[47] J. Hong, G. Lafferiere, B. Mishra, and X. Tan. Fine manipulation with multi-finger hands. In *ICRA*, 1990.

[48] K. Hsiao, L. Kaelbling, and T. Lozano-Perez. Grasping POMDPs. In *International Conference on Robotics and Automation (ICRA)*, 2007.

[49] K. Hsiao and T. Lozano-Perez. Imitation learning of whole-body grasps. In *IEEE/RJS International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[50] K. Hsiao, T. Lozano-Perez, and L. P. Kaelbling. Robust belief-based execution of manipulation programs. In *WAFR*, 2008.

[51] K. Hsiao, P. Nangeroni, M. Huber, A. Saxena, and A.Y. Ng. Reactive grasping using optical proximity sensors. In *International Conference on Robotics and Automation (ICRA)*, 2009.

[52] J. Huang, A.B. Lee, and D. Mumford. Statistics of range images. *Computer Vision and Pattern Recognition (CVPR)*, 2000.

[53] P.J. Huber. *Robust Statistics*. Wiley, New York, 1981.

[54] M. Hueser, T. Baier, and J. Zhang. Learning of demonstrated grasping skills by stereoscopic tracking of human hand configuration. In *International Conference on Robotics and Automation (ICRA)*, 2006.

[55] A. Jain and C.C. Kemp. Behaviors for robust door opening and doorway traversal with a force-sensing mobile manipulator. In *RSS Workshop on Robot Manipulation: Intelligence in Human*, 2008.

[56] I. Kamon, T. Flash, and S. Edelman. Learning to grasp using visual information. In *International Conference on Robotics and Automation (ICRA)*, 1996.

[57] D. Katz and O. Brock. Manipulating articulated objects with interactive perception. In *ICRA*, 2008.

[58] M. Kawakita, K. Iizuka, T. Aida, T. Kurita, and H. Kikuchi. Real-time three-dimensional video image composition by depth information. In *IEICE Electronics Express*, 2004.

[59] M. Kearns and S. Singh. Finite-sample rates of convergence for q-learning and indirect methods. In *NIPS 11*, 1999.

[60] C.C. Kemp, C. Anderson, H. Nguyen, A. Trevor, and Z. Xu. A point-and-click interface for the real world: Laser designation of objects for mobile manipulation. In *HRI*, 2008.

[61] O. Khatib. The potential field approach and operational space formulation in robot control. *Adaptive and Learning Systems: Theory and Applications*, pages 367–377, 1986.

[62] D. Kim, J.H. Kang, C.S. Hwang, and G.T. Park. Mobile robot for door opening in a house. *LNAI*, 3215:596–602, 2004.

[63] M. Kim and W. Uther. Automatic gait optimisation for quadruped robots. In *Proc. Australasian Conf. on Robotics and Automation*, pages 1–9, 2003.

[64] E. Klingbeil, A. Saxena, and A.Y. Ng. Learning to open new doors. In *Robotics Science and Systems (RSS) workshop on Robot Manipulation*, 2008.

[65] R Koch, M Pollefeys, and L Van Gool. Multi viewpoint stereo from uncalibrated video sequences. In *European Conference on Computer Vision (ECCV)*, 1998.

[66] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proc. IEEE Int'l Conf. Robotics and Automation*, 2004.

[67] V. Kolmogorov, A. Criminisi, A. Blake, G. Cross, and C. Rother. Probabilistic fusion of stereo with color and contrast for bilayer segmentation. *IEEE Pattern Analysis and Machine Intelligence (PAMI)*, 28(9):1480–1492, 2006.

[68] S. Konishi and A. Yuille. Statistical cues for domain specific image segmentation with performance analysis. In *Computer Vision and Pattern Recognition (CVPR)*, 2000.

[69] D. Kragic and H. I. Christensen. Robust visual servoing. *International Journal of Robotics Research (IJRR)*, 22:923–939, 2003.

[70] S. Kumar and M. Hebert. Discriminative fields for modeling spatial dependencies in natural images. In *Neural Information Processing Systems (NIPS) 16*, 2003.

[71] J. Lafferty, A. McCallum, and F. Pereira. Discriminative fields for modeling spatial dependencies in natural images. In *ICML*, 2001.

[72] Y. LeCun. Presentation at Navigation, Locomotion and Articulation workshop. Washington DC, 2003.

[73] J.W. Li, M.H. Jin, and H. Liu. A new algorithm for three-finger force-closure grasp of polygonal objects. In *ICRA*, 2003.

[74] T. Lindeberg and J. Garding. Shape from texture from a multi-scale perspective. In *ICCV*, 1993.

[75] Y.H. Liu. Computing n-finger form-closure grasps on polygonal objects. *IJRR*, 19(2):149–158, 2000.

[76] J.M. Loomis. Looking down is looking up. *Nature News and Views*, 414:155–156, 2001.

[77] M. Lourakis and A. Argyros. A generic sparse bundle adjustment c/c++ package based on the levenberg-marquardt algorithm. Technical report, Foundation for Research and Technology - Hellas, 2006.

[78] Y. Lu, J.Z. Zhang, Q.M.J. Wu, and Z.N. Li. A survey of motion-parallax-based 3-d reconstruction algorithms. *IEEE Tran on Systems, Man and Cybernetics, Part C*, 34:532–548, 2004.

[79] A. Maki, M. Watanabe, and C. Wiles. Geotensity: Combining motion and lighting for 3d surface reconstruction. *International Journal of Computer Vision (IJCV)*, 48(2):75–90, 2002.

[80] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *Journal of the Optical Society of America A*, 7(5):923–932, 1990.

[81] J. Malik and R. Rosenholtz. Computing local surface orientation and shape from texture for curved surfaces. *International Journal of Computer Vision (IJCV)*, 23(2):149–168, 1997.

[82] X. Markenscoff, L. Ni, and C.H. Papadimitriou. The geometry of grasping. *IJRR*, 9(1):61–74, 1990.

[83] J. J. Marotta, T. S. Perrot, D. Nicolle3, P. Servos, and M. A. Goodale. Adapting to monocular vision: grasping with one eye. 104:107–114, 2004.

[84] D.R. Martin, C.C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color and texture cues. *IEEE Trans Pattern Analysis and Machine Intelligence*, 26, 2004.

[85] M. T. Mason and J. K. Salisbury. Manipulator grasping and pushing operations. In *Robot Hands and the Mechanics of Manipulation*. The MIT Press, Cambridge, MA, 1985.

[86] M.T. Mason and J.K. Salisbury. *Robot Hands and the Mechanics of Manipulation*. MIT Press, Cambridge, MA, 1985.

[87] A. McCallum, C. Pal, G. Druck, and X. Wang. Multi-conditional learning: generative/discriminative training for clustering and classification. In *AAAI*, 2006.

[88] J. Michels, A. Saxena, and A.Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2005.

[89] A. T. Miller, S. Knoop, P. K. Allen, and H. I. Christensen. Automatic grasp planning using shape primitives. In *International Conference on Robotics and Automation (ICRA)*, 2003.

[90] B. Mishra, J.T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica, Special Issue: Robotics*, 2(4):541–558, 1987.

[91] J. Miura, Y. Yano, K. Iwase, and Y. Shirai. Task model-based interactive teaching. In *IROS*, 2004.

[92] T.M. Moldovan, S. Roth, and M.J. Black. Denoising archival films using a learned bayesian model. In *International Conference on Image Processing (ICIP)*, 2006.

[93] A. Morales, E. Chinellato, P.J. Sanz, A.P. del Pobil, and A.H. Fagg. Learning to predict grasp reliability for a multifinger robot hand by using visual features. In *Int'l Conf AI Soft Comp.*, 2004.

[94] A. Morales, P. J. Sanz, and A. P. del Pobil. Vision-based computation of three-finger grasps on unknown planar objects. In *IEEE/RSJ Intelligent Robots and Systems Conference*, 2002.

[95] A. Morales, P. J. Sanz, A. P. del Pobil, and A. H. Fagg. An experiment in constraining vision-based finger contact selection with gripper geometry. In *IEEE/RSJ Intelligent Robots and Systems Conference*, 2002.

[96] E.N. Mortensen, H. Deng, and L. Shapiro. A SIFT descriptor with global context. In *Computer Vision and Pattern Recognition (CVPR)*. 2005.

[97] K. Murphy, A. Torralba, and W.T. Freeman. Using the forest to see the trees: A graphical model relating features, objects, and scenes. In *Neural Information Processing Systems (NIPS) 16*, 2003.

[98] T. Nagai, T. Naruse, M. Ikehara, and A. Kurematsu. Hmm-based surface reconstruction from single images. In *Proc IEEE International Conf Image Processing (ICIP)*, volume 2, 2002.

[99] S.G. Narasimhan and S.K. Nayar. Shedding light on the weather. In *Computer Vision and Pattern Recognition (CVPR)*, 2003.

[100] O. Nestares, R. Navarro, J. Portilia, and A. Tabernero. Efficient spatial-domain implementation of a multiscale image representation based on gabor functions. *Journal of Electronic Imaging*, 7(1):166–173, 1998.

[101] A.Y. Ng and M. Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proc. 16th Conf. UAI*, 2000.

[102] V.D. Nguyen. Constructing stable force-closure grasps. In *ACM Fall joint computer conference*, 1986.

[103] A. Oliva and A. Torralba. Building the gist of a scene: the role of global image features in recognition. *IEEE Trans Pattern Analysis and Machine Intelligence (PAMI)*, 155:23–36, 2006.

[104] B.A. Olshausen and D.J. Field. Sparse coding with an over-complete basis set: A strategy employed by v1? *Vision Research*, 37:3311–3325, 1997.

[105] O. Partaatmadja, B. Benhabib, E. Kaizerman, and M. Q. Dai. A two-dimensional orientation sensor. *Journal of Robotic Systems*, 1992.

[106] O. Partaatmadja, B. Benhabib, A. Sun, and A. A. Goldenberg. An electrooptical orientation sensor for robotics. *Robotics and Automation, IEEE Transactions on*, 8:111–119, 1992.

[107] R. Pelossof, A. Miller, P. Allen, and T. Jebara. An svm learning approach to robotic grasping. In *International Conference on Robotics and Automation (ICRA)*, 2004.

[108] L. Petersson, D. Austin, and D. Kragic. High-level control of a mobile manipulator for door opening. In *IROS*, 2000.

[109] A. Petrovskaya, O. Khatib, S. Thrun, and A. Y. Ng. Bayesian estimation for autonomous object manipulation based on tactile sensors. In *International Conference on Robotics and Automation*, 2006.

[110] A. Petrovskaya and A.Y. Ng. Probabilistic mobile manipulation in dynamic environments, with application to opening doors. In *IJCAI*, 2007.

[111] G. Petryk and M. Buehler. Dynamic object localization via a proximity sensor network. In *Int'l Conf Multisensor Fusion and Integration for Intelligent Systems*, 1996.

[112] G. Petryk and M. Buehler. Robust estimation of pre-contact object trajectories. In *Fifth Symposium on Robot Control*, 1997.

[113] J. H. Piater. Learning visual features to predict hand orientations. In *ICML Workshop on Machine Learning of Spatial Knowledge*, 2002.

[114] R. Platt, A. H. Fagg, and R. Grupen. Reusing schematic grasping policies. In *IEEE-RAS International Conference on Humanoid Robots, Tsukuba, Japan*, 2005.

[115] R. Platt, R. Grupen, and A. Fagg. Improving grasp skills using schema structured learning. In *ICDL*, 2006.

[116] N.S. Pollard. Closure and quality equivalence for efficient synthesis of grasps from examples. *IJRR*, 23(6), 2004.

[117] M. Pollefeys. Visual modeling with a hand-held camera. *International Journal of Computer Vision (IJCV)*, 59, 2004.

[118] D. Pomerleau. An autonomous land vehicle in a neural network. In *NIPS 1*. Morgan Kaufmann, 1989.

[119] J. Ponce, D. Stam, and B. Faverjon. On computing two-finger force-closure grasps of curved 2d objects. *IJRR*, 12(3):263–273, 1993.

[120] J. Ponce, S. Sullivan, A. Sudsang, J.D. Boissonnat, and J.P. Merlet. On computing four-finger equilibrium and force-closure grasps of polyhedral objects. *IJRR*, 16(1):11–35, 1997.

[121] J. Porrill, J.P. Frisby, W.J. Adams, and D. Buckley. Robust and optimal use of information in stereo vision. *Nature*, 397:63–66, 1999.

[122] M. Prats, P. Sanz, and A.P. del Pobil. Task planning for intelligent robot manipulation. In *IASTED Artificial Intel App*, 2007.

[123] D. Prattichizzo and J.C. Trinkle. *Springer Handbook of Robotics*, pages 671–698. Springer-Verlag New York, LLC, 2008.

[124] M. Quartulli and M. Datcu. Bayesian model based city reconstruction from high resolution ISAR data. In *IEEE/ISPRS Joint Workshop Remote Sensing and Data Fusion over Urban Areas*, 2001.

[125] M. Quigley. Switchyard. Technical report, Stanford University, http://ai.stanford.edu/∼mquigley/switchyard, 2007. URL `http://ai.stanford.edu/ mquigley/switchyard/`.

[126] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *Open-Source Software workshop of ICRA*, 2009.

[127] P. P. L. Regtien. Accurate optical proximity detector. *Instrumentation and Measurement Technology Conference, 1990. IMTC-90. Conference Record., 7th IEEE*, pages 141–143, 1990.

[128] C. Rhee, W. Chung, M. Kim, Y. Shim, and H. Lee. Door opening control using the multi-fingered robotic hand for the indoor service robot. In *ICRA*, 2004.

[129] Kotz S, Kozubowski T, and Podgorski K. *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance.* Birkhauser, 2001.

[130] T. Sakai, H. Nakajima, D. Nishimura, H. Uematsu, and K. Yukihiko. Autonomous mobile robot system for delivery in hospital. In *MEW Technical Report*, 2005.

[131] J.K. Salisbury and B. Roth. Kinematic and force analysis of articulated hands. *ASME J. Mechanisms, Transmissions, Automat. Design*, 105:33–41, 1982.

[132] B. Sapp, A. Saxena, and A.Y. Ng. A fast data collection and augmentation procedure for object recognition. In *AAAI*, 2008.

[133] A. Saxena, A. Anand, and A. Mukerjee. Robust facial expression recognition using spatially localized geometric model. In *International Conf Systemics, Cybernetics and Informatics (ICSCI)*, 2004.

[134] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *Neural Information Processing Systems (NIPS) 18*, 2005.

[135] A. Saxena, S.H. Chung, and A.Y. Ng. 3-D depth reconstruction from a single still image. *International Journal of Computer Vision (IJCV)*, 76(2):157–173, 2007.

[136] A. Saxena, J. Driemeyer, J. Kearns, and A.Y. Ng. Robotic grasping of novel objects. In *Neural Information Processing Systems (NIPS) 19*, 2006.

[137] A. Saxena, J. Driemeyer, J. Kearns, and A.Y. Ng. Robotic grasping of novel objects. In *NIPS*, 2006.

[138] A. Saxena, J. Driemeyer, J. Kearns, C. Osondu, and A. Y. Ng. Learning to grasp novel objects using vision. In *10th International Symposium of Experimental Robotics (ISER)*, 2006.

[139] A. Saxena, J. Driemeyer, and A. Y. Ng. Learning 3-d object orientation from images. In *NIPS workshop on Robotic Challenges for Machine Learning*, 2007.

[140] A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic grasping of novel objects using vision. *IJRR*, 27:157–173, 2008.

[141] A. Saxena, J. Driemeyer, and A. Y. Ng. Learning 3-d object orientation from images. In *ICRA*, 2009.

[142] A. Saxena, A. Gupta, and A. Mukerjee. Non-linear dimensionality reduction by locally linear isomaps. In *Proc 11th Int'l Conf on Neural Information Processing*, 2004.

[143] A. Saxena, G. Gupta, V. Gerasimov, and S. Ourselin. In use parameter estimation of inertial sensors by detecting multilevel quasi-static states. In *KES*, 2005.

[144] A. Saxena and A.Y. Ng. Learning sound location from a single microphone. In *ICRA*, 2009.

[145] A. Saxena, S. Ray, and R.K. Varma. A novel electric shock protection system based on contact currents on skin surface. In *NPSC*, 2002.

[146] A. Saxena, J. Schulte, and A.Y. Ng. Depth estimation using monocular and stereo cues. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[147] A. Saxena and A. Singh. A microprocessor based speech recognizer for isolated hindi digits. In *IEEE ACE*, 2002.

[148] A. Saxena, S.G. Srivatsan, V. Saxena, and S. Verma. Bioinspired modification of polystyryl matrix: Single-step chemical evolution to a moderately conducting polymer. *Chemistry Letters*, 33(6):740–741, 2004.

[149] A. Saxena, M. Sun, and A. Y. Ng. Learning 3-d scene structure from a single still image. In *ICCV workshop on 3D Representation for Recognition (3dRR-07)*, 2007.

[150] A. Saxena, M. Sun, and A.Y. Ng. 3-d reconstruction from sparse views using monocular vision. In *ICCV workshop on Virtual Representations and Modeling of Large-scale environments (VRML)*, 2007.

[151] A. Saxena, M. Sun, and A.Y. Ng. Make3d: Depth perception from a single still image. In *AAAI*, 2008.

[152] A. Saxena, M. Sun, and A.Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Transactions of Pattern Analysis and Machine Intelligence (PAMI)*, 31(5):824–840, 2009.

[153] A. Saxena, L. Wong, and A.Y. Ng. Learning grasp strategies with partial shape information. In *AAAI*, 2008.

[154] A. Saxena, L. Wong, M. Quigley, and A.Y. Ng. A vision-based system for grasping novel objects in cluttered environments. In *ISRR*, 2007.

[155] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision (IJCV)*, 47, 2002.

[156] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition (CVPR)*, 2003.

[157] H. Schneiderman and T. Kanade. Probabilistic modeling of local appearance and spatial relationships for object recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 1998.

[158] S.H. Schwartz. *Visual Perception 2nd ed.* Appleton and Lange, Connecticut, 1999.

[159] F. Schwarzer, M. Saha, and J.-C. Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Transactions on Robotics and Automation*, 21:338–353, 2005.

[160] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *Computer Vision and Pattern Recognition (CVPR)*, 2005.

[161] K. Shimoga. Robot grasp synthesis: a survey. *International Journal of Robotics Research (IJRR)*, 15:230–266, 1996.

[162] T. Shin-ichi and M. Satoshi. Living and working with robots. *Nipponia*, 2000.

[163] J.R. Smith, E. Garcia, R. Wistort, and G. Krishnamoorthy. Electric field imaging pretouch for robotic graspers. In *International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[164] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM SIGGRAPH*, 25(3):835–846, 2006.

[165] S.P. Soundararaj, A.K. Sujeeth, and A. Saxena. Autonomous indoor helicopter flight using a single onboard camera. In *IROS*, 2009.

[166] C. Stachniss and W. Burgard. Mobile robot mapping and localization in non-static environments. In *AAAI*, 2005.

[167] G. Strang and T. Nguyen. *Wavelets and filter banks.* Wellesley-Cambridge Press, Wellesley, MA, 1997.

[168] E.B. Sudderth, A. Torralba, W.T. Freeman, and A.S. Willisky. Depth from familiar objects: A hierarchical model for 3D scenes. In *Computer Vision and Pattern Recognition (CVPR)*, 2006.

[169] R.S. Sutton and A.G. Barto. *Reinforcement Learning*. MIT Press, 1998.

[170] Swissranger. Mesa imaging. In *http://www.mesa-imaging.ch*, 2005.

[171] R. Szelinski. Bayesian modeling of uncertainty in low-level vision. In *International Conference on Computer Vision (ICCV)*, 1990.

[172] T. Takahashi, T. Tsuboi, T. Kishida, Y. Kawanami, S. Shimizu, M. Iribe, T. Fukushima, and M. Fujita. Adaptive grasping by multi fingered hand with tactile sensor based on robust force and position control. In *ICRA*, 2008.

[173] J. Tegin and J. Wikander. Tactile sensing in intelligent robotic manipulation - a review. *Industrial Robot: An International Journal*, pages 264–271, 2005.

[174] S. Thrun and M. Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, 2006.

[175] S. Thrun and B. Wegbreit. Shape from symmetry. In *International Conf on Computer Vision (ICCV)*, 2005.

[176] A. Torralba. Contextual priming for object detection. *International Journal of Computer Vision*, 53(2):161–191, 2003.

[177] A. Torralba and A. Oliva. Depth estimation from image structure. *IEEE Trans Pattern Analysis and Machine Intelligence (PAMI)*, 24(9):1–13, 2002.

[178] L. Torresani and A. Hertzmann. Automatic non-rigid 3D modeling from video. In *European Conference on Computer Vision (ECCV)*, 2004.

[179] S. Walker, K. Loewke, M. Fischer, C. Liu, and J. K. Salisbury. An optical fiber proximity sensor for haptic exploration. *ICRA*, 2007.

[180] B.A. Wandell. *Foundations of Vision.* Sinauer Associates, Sunderland, MA, 1995.

[181] A.E. Welchman, A. Deubelius, V. Conrad, H.H. Blthoff, and Z. Kourtzi. 3D shape perception from combined depth cues in human visual cortex. *Nature Neuroscience*, 8:820–827, 2005.

[182] A.S. Willsky. Multiresolution markov models for signal and image processing. *IEEE*, 2002.

[183] R. Wistort and J.R. Smith. Electric field servoing for robotic manipulation. In *IROS*, 2008.

[184] D. Wolf and G. S. Sukhatme. Online simultaneous localization and mapping in dynamic environments. In *ICRA*, 2004.

[185] B. Wu, T.L. Ooi, and Z.J. He. Perceiving distance accurately by a directional process of integrating ground information. *Letters to Nature*, 428:73–77, 2004.

[186] R. Zhang, P.S. Tsai, J.E. Cryer, and M. Shah. Shape from shading: A survey. *IEEE Trans Pattern Analysis & Machine Intelligence (IEEE-PAMI)*, 21:690–706, 1999.

[187] W. Zhao, R. Chellappa, P.J. Phillips, and A. Rosenfield. Face recognition: A literature survey. *ACM Computing Surveys*, 35:399–458, 2003.