Toward Autonomous Robot Interaction and Use of Objects

Oleg O. Sushkov

Declarations

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Signed: ..... Date: 01/10/12

Copyright Statement

Signed: ..... Date: 01/10/12

Authenticity Statement

I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.

Signed: ..... Date: 01/10/12

Abstract

This thesis is focused on the skills and tasks that ~~need to~~ must be performed ~~for~~ by an autonomous robot to interact with objects in a complex environment. For a robot to manipulate and interact effectively with an object it must first learn the appearance of the object, determine the shape of the object, be able to recognise and localise the object in the environment, and determine the physical properties of the object. We investigate and improve upon aspects of each of these skills in turn.

~~For object recognition~~, ~~A~~ a new method for local image feature matching of object recognition ~~is~~ has been developed that is more accurate than existing methods, as well as being more efficient in some circumstances. Next we developed a system that combines robot initiated object motion and long term image feature tracking to accurately extract object features from complex scene images. This allows a robot to learn to recognise previously unseen objects in the presence of clutter, noise and background motion. The object feature matching and segmentation methods are then combined with 3D reconstruction methods to determine the object's shape. This is done by stitching together multiple views of the object.

Physical properties (weight, friction, centre of mass, etc) are important factors in determining how a robot can use an object. However, unlike shape and appearance, these may be impossible to determine by passive observation. We have developed a method in which the robot performs experiments on the object, and uses the outcomes to update its knowlege of the object's physical properties. To perform the most informative experiments, a physics simulator is used to internally rehearse each experiment and its potential outcomes. The outcome of each experiment, after being performed on the physical object, is input back into the simulator forming a hypothesis-experiment-refinement loop. In this way the robot effectively learns the internal properties of an object. Finally, the robot uses this knowledge to plan and carry out a simple task in which the object is used as a tool.

Acknowledgements

They say “it takes a village to raise a child”, and in my case this thesis is the proverbial child, and to raise it I needed the help of a proverbial village. First and foremost I must thank Claude Sammut, who provided much needed guidance and supervision. I’ve lost count of the number of times I was confused about which direction to take with my work, but after a talk with Claude things became much clearer and the path forward less obscured. Similarly I must thank Bernhard Hengst for helping me make sense of Bayesian probability and always having time to talk. Others, like Malcolm Ryan and Will Uther, always had their door open and willing to share their wisdom with a confused PhD student.

I also thank all of my peers in the lab who often provided much need distraction from the day to day toils of a research student, and on occasions some surprising words of wisdom. Thank you Nawid and Anna for the advice on PhD life as well as providing the environment in which the Procrastination Vortex could form, far from my desk. Thanks to Rudi and Matt for keeping the hardware running, and thanks to Jayen for keeping the software running. Without you I am sure everything would fall apart in an instant. Finally thanks to everybody else in the lab, Brad, Bhuman, Tim Wiley, Tim Cerexe, Jenny, Mike, Adrian, and Dave, for making the Level 3 lab a friendly and fun place to work.

A PhD is more than just the lab. Without a balanced life, friends and family, this long journey would have been impossible, or at least far from enjoyable. Thank you to all of my friends, your good company kept me sane. Most of all, thank you Emily for being there when I needed, being patient and understanding.

Finally the biggest thanks goes to my family, without whose support this would no be possible.

Contents

List of Algorithms

List of Figures

Chapter 1

Introduction

This thesis deals with the steps required for an autonomous robot to progress from encountering a new object in the environment to being able to recognise, manipulate, and finally use the object to complete a task. This thesis encompasses several fields ranging from computer vision, object recognition and reconstruction, to simulation and planning. We focus on maximising robot autonomy by developing methods to allow a robot to be self-sufficient in tasks that would otherwise require human operator intervention or the availability of pre-processed data.

The initial step is to develop a vision system capable of learning to recognise a new object in a complex environment, with no human intervention or pre-processed training data. This consists of two separate tasks. The first is to learn the object's appearance by separating its image features from the background. This is challenging as the robot has no *a priori* knowledge of the object, and segmentation may be complicated by a cluttered and dynamic scene background. The second is to effectively match the learned object's appearance model to an image to recognise and localise the object in the scene.

The next step is to learn the object's 3D shape and the full 360° appearance model. This allows the robot to recognise an object in a scene from any angle and to determine its pose. Knowing the shape of the object enables a robot to perform grasp and manipulation planning. The robot learns the full 3D model of the object

by ~~stitching together and~~ combining views of the object from different directions.

The final step is for the robot to learn the physical and internal properties of an object through experimentation and interaction. Some properties of an object cannot be determined by passive observation (eg: centre of mass, coefficient of friction, etc). To do this, the robot must actively interact with the object, performing experiments and observing the results. By doing this, it can build a model of the internal properties of the object. The challenges include choosing an appropriate representation to model the properties of the object, performing the experiments that provide the most information about the object, and correctly inferring the object's properties from the outcomes of the experiments. By doing this, the robot can efficiently build an accurate model of the object, taking into account its shape, appearance, internal and physical properties. This allows the robot to effectively manipulate and use the object to accomplish tasks.

The end result is a robot system ~~which~~ that can autonomously progress from a first encounter with an object to effectively using the object as a tool.

1.1 Motivation

Robots are fast becoming ubiquitous and the range of applications in which they are used is growing wider [1]. One of the first widespread applications of robots was in factories to automate assembly and construction [2]. In these cases the robots operated in structured environments and performed structured tasks, often simply repeating a pre-programmed motion. Over time robots became mobile, gained a degree of autonomy, and were being used in less structured environments [3, 4]. Extrapolating this trend leads to many new potential application areas, for example household service robots (example in Figure 1.1.1). These are robots which could be used to perform tasks around the house and provide assistance for the elderly and disabled. This application is of particular importance given the aging population in many countries [5]. One of the main challenges [6] to applying robotics to these

Figure 1.1.1: A Willow Garage PR2 Robot performing a household task. (Image courtesy of [7])

applications is autonomy and self-sufficiency.

Consider the following example scenario: a household service robot is deployed into an elderly person's home. A typical tasks that it may need to perform is retrieving medicine. This seemingly simple task involves several steps that are complicated by the need for autonomy and to operate in a complex, human-centric environment.

First, the robot must be able to recognise and locate the correct medicine container in the house. There are many different approaches to object perception using robot sensors (discussed in detail in Chapter 2), however many of them rely on the robot having *a priori* knowledge of the target object. For example, this knowledge can be in the form of segmented views of the object. The problem with this is that an autonomous robot, operating in a complex environment, may encounter an endless array of objects. It is not feasible to provide the robot, prior to deployment, with images of every possible object it may need to recognise. Instead, the robot must be able to autonomously learn the appearance of new objects. In the case of a household service robot, it would learn the appearance of various objects around the

house after it has been deployed. In this way the robot would no longer be reliant on pre-programmed data, but would be able to dynamically learn to recognise and localise novel objects. A related task is reconstructing the 3D shape of an object. Knowing the shape of an object allows a robot to more effectively interact with it, being able to choose optimal grasp points [8], as well as allowing the robot to perform motion planning and reason about the potential tool uses of the object[9]. Similar to learning the object's appearance, a robot should be able to learn the 3D shape of an object autonomously.

Consider a different scenario in which a household service robot needs to prop open a door with some object. In this case the properties of the object will determine if it is a suitable tool to use for this task. For example, a light or slippery object may not be suitable, whereas a heavy and rough object would be. Similar to the problem of object recognition, it is not feasible to pre-program a robot with knowledge of the physical properties of every object it may encounter in an unstructured environment such as a home. Instead the robot must be able to autonomously discover object properties such as weight, centre of mass, coefficient of friction, etc. To do this it may need to perform various experiments to build a model of the object.

Many of the problems this thesis addresses are based on minimising the role of human intervention and pre-programmed knowledge, instead maximising robot autonomy. The aim is a level of autonomy which would allow a robot to be deployed into an unknown and unstructured environment, and to be able to discover new objects and use these as tools to solve some tasks.

1.2 Contributions

The main achievements presented in this theses are:

- A new method for local image feature matching, correlating scene image features to a database of learned object image features for object recognition and localisation. The presented algorithm is more accurate than the standard

There is more than one method and none that is "standard".

method, as well as computationally more efficient in certain circumstances. This enables a robot to recognise and localise objects in a scene with greater accuracy and speed.

- A method for a robot to learn to recognise a previously unseen object by using motion and long term feature tracking to segment object features from the background, generating a database of object image features in the process. We solve the problem of generating segmented snapshots of the target object in the presense of a high degree of background clutter and motion. Object snapshots are combined to learn the full 3D aspect graph and shape of the object by stitching together multiple object views.

- A novel method for a robot to learn the physical properties of an object by active experimentation. We use a physics simulator to generate hypotheses and guide the robot toward the experiments with the highest information gain. We solve the problem of choosing the optimal experiment by internally rehearsing each experiment in simulation to determine the posterior probability and the associated expected entropy. The optimal experiment is then carried out by the robot on the object, and the results provide information about the internal and physical properties of the object.

- The learned model, encapsulating the appearance, shape and the physical properties of the object, is used to plan and execute a tool-use task. The task is planned using internal rehearsal in simulation, and then carried out by the robot.

The result of this thesis is a robot system with the ability to encounter an unknown object, learn to recognise the object, determine its 3D shape, its physical properties, and use the object to complete a task requiring tool use.

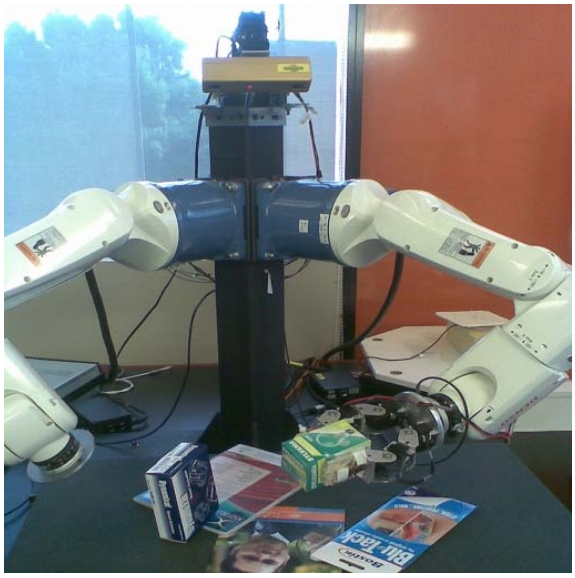

Figure 1.3.1: The robot platform used for this thesis. The robot is composed of a six degrees-of-freedom industrial arm, a two fingered gripper, and a camera mounted on a pan-tilt unit. A tablet-top workspace is accessible in front of the robot.

1.3 Robot Platform

The platform used to test and evaluate the ~~presented~~ presented here methods and algorithms consists of a camera on a pan-tilt unit and a six degrees of freedom robot arm with a gripper attachment. This is arranged in a humanoid configuration, the pan-tilt unit and camera are placed on top of a metal spine and the arm is attached below (see Figure 1.3.1). The spine is fixed to a table, which provides the robot with a flat workspace in front. The camera unit is located 0.8 metres above the worspace surface, while the arm is fixed to a point 0.5 metres above the surface.

The camera unit is a Point Grey Bumblebee2 stereo camera[1], which incorporates ~~It is composed of~~ two RGB cameras with 43° field of view and a 12cm baseline distance. We used a resolution of 640 × 480 at a refresh rate of 10 frames per second. For some ~~parts of the thesis~~ experiments this camera is replaced with a Microsoft Kinect[2] RGB-D ~~depth~~ camera. The Kinect is a sensor unit combining a traditional RGB camera with an infrared camera and projector which, using structured light, is able to provide 11 bits of depth information per pixel. The result is a 640 × 480 resolution image, with each

[1] http://www.ptgrey.com/products/bumblebee2/
[2] http://www.xbox.com/en-US/kinect

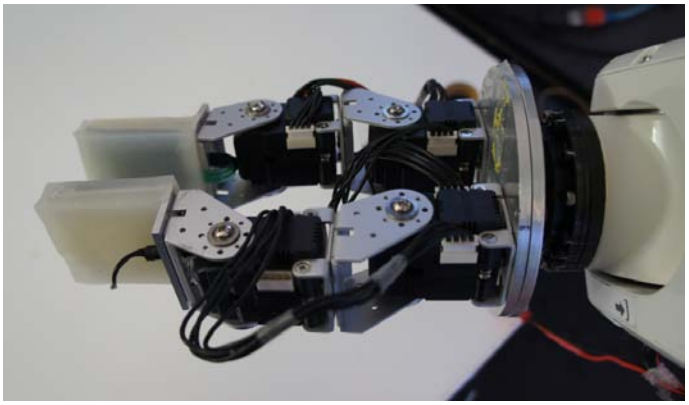

Figure 1.3.2: The two-fingered gripper attachment on the end of the robot arm. Each finger is composed of two servos, the finger tips consist of metal plates covered in silicone to increase friction.

pixel containing RGB color information as well as a depth value. The Kinect sensor is described in greater detail in Chapter 6.

The robot arm is a Denso Robotics VP-6[3] robot, which is able to orient the end-point with six degrees of freedom and is composed of six joints. At the end of the robot arm we attached a two fingered gripper (see Figure 1.3.2). Each finger consists of two servos with a silicone coated pad at the tip for improved grip.

1.4 Thesis Outline

- **Chapter 2** provides an overview of the related work in the fields of computer vision, 3D reconstruction and active robot learning.
- **Chapter 3** describes a new method of image feature matching for object recognition, improving upon existing methods in both speed and accuracy.
- **Chapter 4** describes a method for a robot to learn the appearance of a new object autonomously and in a complex environment. This allows the robot to learn to recognise objects after it has been deployed, rather than being limited to those objects it has been trained to recognise during development.
- **Chapter 5** combines the methods developed in Chapters 3 and 4 with 3D

[3]http://www.densorobotics.com/products_vp_5_6axis.php

reconstruction techniques into a system allowing a robot to learn the full 3D aspect graph of an object as well as its shape.

- **Chapter 6** presents a technique for a robot to learn the physical and internal properties of an object using interaction and experimentation. The learned object model is then used for planning and carrying out a tool-use task.

- **Chapter 7** suggests avenues for future work as well as concluding the thesis.

Chapter 2

Background

Need to be clear what localising means since it could be confused with SLAM

It's up to the examiners to decide if it's an improvement.

investigates

This thesis ~~improves upon~~ the primary skills an autonomous robot needs to interact with and use objects in its environment. These skills are: learning the appearance of a novel object, recognising and localising the object in a cluttered scene, reconstructing the 3D shape of the object, and learning the physical properties of the object to accomplish a task. In this chapter we discuss the background and existing literature in each of these areas. We present ~~the relevant~~ related work and the ~~shortcomings~~ strengths and weaknesses of the existing approaches, followed by an outline of how these weaknesses are addressed in the following chapters.

2.1 Object Perception

Object recognition and localisation is a key aspect of any robot system that needs to interact with objects in an unstructured environment. ~~There are~~ Many different sensor modalities ~~that~~ have been used for robot object recognition ~~in literature~~. Haptic feedback (touch) has been used to determine the shape of an object from sparse surface contact points that are then used to match the shape of known objects [10]. Scene range data from a time-of-flight sensor, such as a laser range scanner, can also be used to recognise and losalise objects in a scene by matching known shapes [11]. Magnetic markers [12] and acoustic signatures [13] are further examples of the

wide range of sensors and modalities that have been applied to object tracking and recognition. However, the most common sensor ~~modality~~ for object recognition and localisation is vision ~~using a video camera~~.

Video cameras are ~~a~~ popular sensor because of their low cost, high resolution, and the large amount of scene data contained in the camera image stream. The output of most camera ~~systems~~ is ~~in the form of~~ a stream of images, where each image is a two dimensional array of pixels. Most computer vision techniques for recognising and localising an object in the scene by examining the pixel values of the camera images.

There has been a large amount of research in ~~the domain of~~ computer vision addressing the problem of object recognition and localisation, applying a variety of techniques ~~and approaches~~ [14, 15]. We present an overview of the different ~~categories of~~ approaches ~~which~~ have been used for object recognition and localisation in computer vision, going into detail for the more relevant research literature.

In general, the development of ~~the various~~ object recognition algorithms has been driven by the need to address two main problems. First, the appearance of ~~the~~ object in a scene image may be different from its appearance during training. This can be due to ~~factors such as~~ changed lighting conditions, partial occlusion of the object, and viewing the object from a different perspective. The second problem is background clutter. The scene image may contain background regions ~~which~~ appear similar to the the target object. This can make accurate object recognition and localisation ~~a~~ difficult ~~task~~.

2.1.1 Model Based Recognition

3D Model based recognition relies on a known 3D model of an object to recognise and localise it ~~in a scene image~~. A typical representation of the model is a 3D Computer Aided Design (CAD) model that defines the faces and edges of the object. Edges [16] and contours extracted from ~~the scene~~ image can be matched against the 3D model to determine the object's pose in the scene [17]. However, one of the

problems with this approach is finding the initial correspondence between scene edges and the object model. A simple solution is to initialise the scene object in a particular orientation[17], but this is not suitable when the object's pose is unknown. An improved approach is to use image feature matching to determine the initial correspondence between the object model and the scene [18].

Have you defined what an image feature is?

Despite being able to accurately track the pose of an object, model based object tracking has several drawbacks. An accurate 3D CAD model of the object is required, which may be difficult and time consuming to obtain. Additionally, the objects must have well defined contours and edges for optimal performance. Objects with complex patterns and textures may not be suitable for recognition and tracking using this approach.

2.1.2 Appearance Based Recognition

In contrast to model based recognition, appearance based methods store a representation of the object's visual appearance and features from different points of view. The stored representation is then matched against a scene image to recognise the object in the environment. The different appearance based approaches are characterised by how the object's appearance features are represented, and how this model is matched to the scene image. We can divide these into two classes: methods using global image features, and methods using local image features.

Global Image Features

Global image features are a function of either the entire image, or a large part of it. Global features can be used to form a very compact representation of an image, ~~representing the image~~ as a vector in a high dimensional space.

Colour and colour histograms are useful global features for object perception. If the object has a relatively uniform colour, distinct from the background, object recognition and localisation can be performed by labeling as object pixels if they

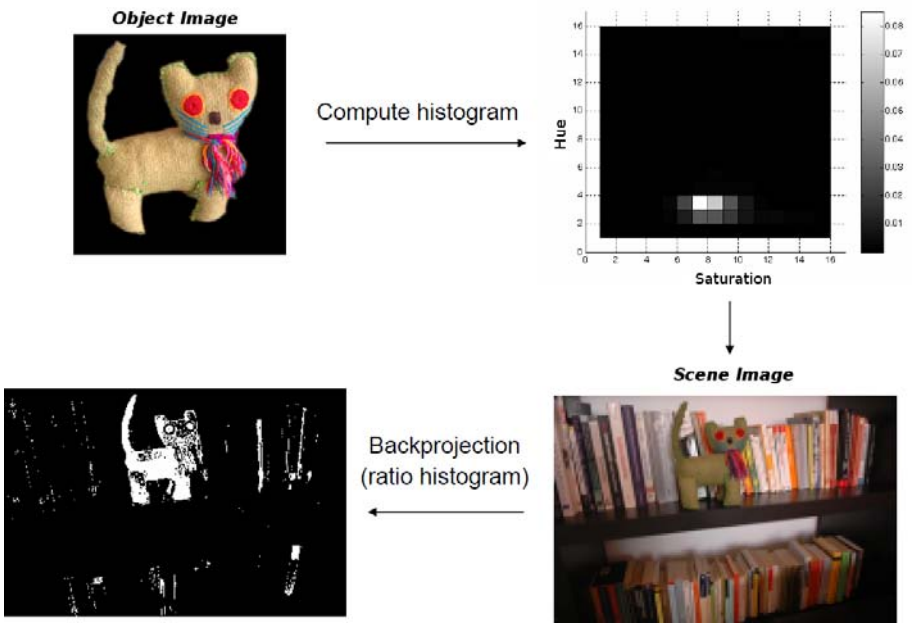

Figure 2.1.1: For colour histogram object detection, a training object image (top left) is used to generate a representative colour histogram (top right). This is then used to label the pixels in the scene image (bottom right), highlighting pixels that have a high probability of belonging to the target object (bottom left). (Images courtesy of [22])

match a predefined colour-space region [19]. Object pixel regions determine the location of the target object in the image. Alternatively, an object view can be characterised by it's colour histogram. A histogram is a representation of the ~~occurance~~ frequency ~~of~~ each colour in an image. This can be used ~~for~~ classifying the content of an image [20], but also for object detection and localisation. If we calculate the colour histogram of an object's appearance using training images, we can find the image regions ~~which~~ have a similar colour to the object by back projecting the object's histogram onto the scene image [21]. An example of this is shown in Figure 2.1.1.

This approach can work well if the object's colours are distinct from the background. It is also able to handle rotation and scale changes, as a histogram is invariant to such image transforms. However, if the background image regions contain similar colours to the target object, this approach will fail. A more robust approach is to combine colour with other image features such as shape [23].

Shape based approaches to object recognition ~~describe~~ the contour or silhouette

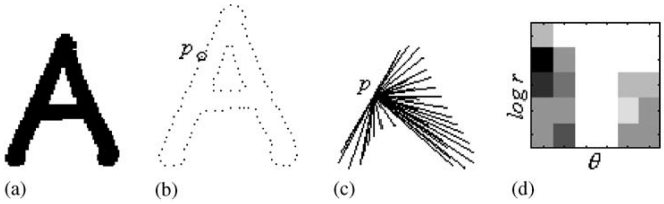

Figure 2.1.2: A shape context description vector is formed for an object (a) by taking its silhouette (b) and drawing vectors out from points along the contour to other points on the contour (c). The result for a point p can be represented as a log-polar histogram (e). The shape context of the entire object is a vector of the histograms of the points along the contour. (Image courtesy of [24])

of an object as a description vector. This description vector can be comprised of shape metrics such as area, eccentricity, etc [24]. However, improved object discrimination has been achieved by using shape context features [25]. These features are formed by taking points along the contour of an object (which can be found using edge detection [16]) and drawing vectors out to other points along the contour. The length and direction of the vectors form the shape context description of the object (see Figure 2.1.2). For matching shapes, a minimum error mapping is found between the points comprising the shape contexts of a learned model and the scene image, where each point is characterised by its vectors. This approach is effective for objects with well defined shapes and silhouettes. However, as background clutter and partial object occlusion is introduced, this method becomes less effective.

Template matching [26] is another approach to object recognition. The most simple variant is to store the object's appearance as the raw pixel values of a snapshot from a particular point of view. To detect and localise the object in the scene, the stored snapshot pixel values are directly compared to the scene image. This method, however, does not cope well with changes in perspective and lighting, as well as partial occlusions. We can improve upon raw pixel intensity template matching by using Principal Component Analysis [27] (PCA) to perform dimensionality reduction. PCA performs eigen-decomposition to find the similarities and differences in the template pixel data across many different sample views of the object. This allows the most statistically significant and invariant pixel components of the

Have a look at recent work at Google for recognising objects in vast databases based an equally vast training data. I think Andrew Moore is the main guy.

templates to be stored and allows matching to be performed on scene images in which the object may have a slightly different appearance as compared to the training data (due to lighting and perspective changes, noise, etc). This approach has been successfully used to perform very fast object recognition of a set of 100 objects [28]. It has also been applied to facial recognition tasks [29].

There are many other global feature approaches to object perception that have been studied in literature. However, despite benefits such as matching and classification speed, object recognition techniques based on global image features have several weaknesses. For scenes where the target object is partially occluded, or scenes with a lot of background clutter, global image feature techniques can struggle. Techniques such as PCA templates can also require a large amount of training data to build a reliable statistical model of the object's appearance under different lighting conditions. References to justify these claims?

Local Image Features

In contrast to global image features, local image features refer to small image patches. There are typically very many local image features that comprise an image. Each local image feature can be characterised by an image position (a position in the image?) and in some cases a vector that describes some aspects of the neighbourhood of pixels around the feature. Object recognition and localisation is performed by matching the local image features extracted from training images of the object to the scene image features.

An example of a simple local image feature is a corner. A corner is defined as an image region where two edges intersect, or alternatively as a region with two dominant edge

Why describe this one and not some other one?

efficient algorithm for extracting corner local image features from an image. However, there are several factors that make it unsuitable for object recognition. First, each corner is characterised only by its position, making matching corners between a learned object appearance model and a scene image difficult. Second, corners are not scale

invariant. Part of an object may appear as a corner at one scale, but not at another. These shortcomings are addressed by interest point detectors.

An interest point detector searches an image for stable points of interest, and generates a description vector that encapsulates the structure of the image in a neighbourhood around the point. Interest points can typically be characterised as having a well defined position in an image, are stable and reproducible under different lighting conditions, rotations, and small perspective changes.

There are numerous local image detector algorithm [31, 32, 33], each with their own strengths and weaknesses. We will give a detailed overview of the Scale Invariant Feature Transform (SIFT) algorithm [34, 35] as it is one of the most popular interest point detectors for object recognition and localisation in cluttered and complex scenes.

SIFT Algorithm The Scale Invariant Feature Transform (SIFT) is a local image descriptor developed by ~~D.~~ Lowe[original ref]. It combines the detection of stable local insterest points at multiple image scales with a rotation and scale invariant descriptor that describes the neighbourhood around each interest point. The SIFT algorithm can output many features for an image, depending on its content. Object recognition and localisation is performed by matching SIFT features from reference images of the target object to the scene image features.

SIFT features have been used for 3D object recognition and pose localisation for robot manupulation [36, 37], localisation [38] and stereo correspondence matching [39]. It is a popular approach for object recognition because it is able to successfully deal with scene clutter, partial object occlusion, lighting and perspective changes, and is rotation and scale invariant. However, one of the weaknesses of SIFT and other similar features, is it does not handle plain untextured objects well. SIFT features are generated in areas of high texture, around edges and corners. A plain object may generate very few or no features at all, making object recognition impossible using this method. Figure 2.1.3 demonstrates object recognition using SIFT

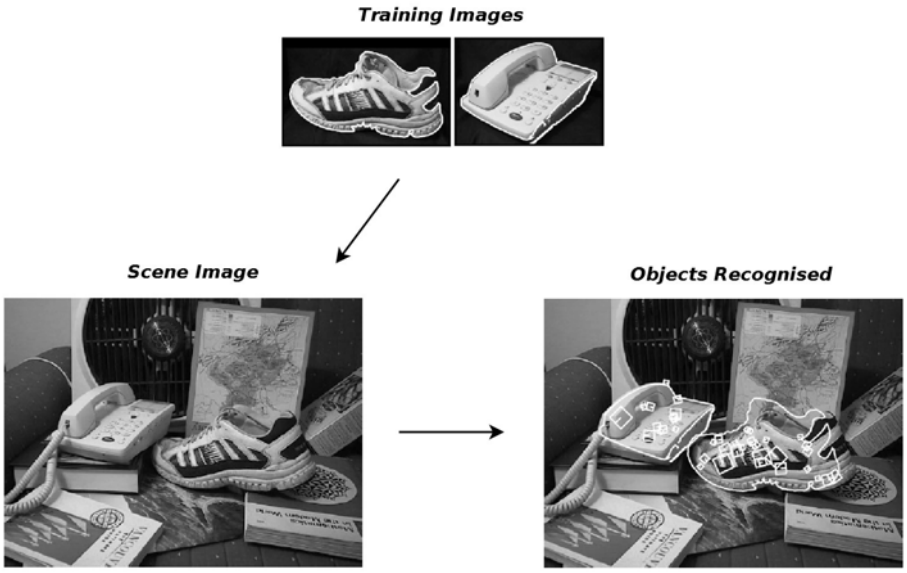

Figure 2.1.3: SIFT features can be used to effectively recognise and localise objects in highly cluttered and complex scenes. (Images courtesy of [35])

features in a cluttered environment.

The SIFT algorithm can be broken down into several steps. These steps first detect and localise stable interest points in an image, and then generate a description of the pixel neighbourhood around each interest point. The SIFT extraction process for an image is as follows:

1. **Generate a Scale-Space pyramid.** SIFT finds features at all image scales. To do this, a scale-space pyramid is built by repeatedly convolving the original image with a Gaussian kernel, effectively blurring the image. The levels of the pyramid correspond to observing the scene at different scales.

2. **Build a Difference of Gaussians pyramid.** The Difference of Gaussians is an approximation of the Laplacian function high-pass filter on the image. This is computed by taking the difference of adjacent pairs of images from the scale-space pyramid.

3. **Find the extrema points in the Difference of Gaussians pyramid.** Each point in the Difference of Gaussians pyramid has 26 neighbours, 8 on the same scale and 9 each in the above and below scale. The points that are the

minimum or maximum of their 26 neighbours are the extrema points. These extrema are candidates for stable interest points. These are typically found near corners and edges in an image at a given scale.

4. **Candidate point localisation.** The extrema point condidates are localised using sub-pixel interpolation. A Taylor expansion of the Difference of Gaussians around each extrema point is found, and the stationary point defines the sub-pixel position of the candidate point.

5. **Rejection of unstable extrema points.** Points with a low absolute value in the Difference of Gaussians pyramid or points that are too *edge-like* are considered to have poor repeatability and unstable image locations in the presence of noise. Principal curvature is used to measure the edge response at the extrema point. If the difference in principal curvature in the edge direction is very different to the perpendicular direction, then the point is considered to be poorly localised and is rejected.

6. **Orientation assignment.** Each of the remaining points is assigned a principal orientation. This is the most prominent gradient direction of a small neighbourhood of pixels around the interest point. Assigning an orientation to each point allows the key-point description vector (described in the next step) to be represented in a rotation invariant manner.

7. **Generation of key-point description vectors.** For each key-point, a neighbourhood of pixels is used to build an array of histograms of gradients. The histogram is oriented relative to the principal orientation of the key-point (calculated in the previous step), making the descriptor rotation invariant. Additionally, the histogram is normalised on the gradient magnitude of the pixel neighbourhood to increase robustness to changes in contrast and lighting. The result is a 128-dimensional description vector for each key-point.

The final result of extracting SIFT features from an image is a set of features. Each

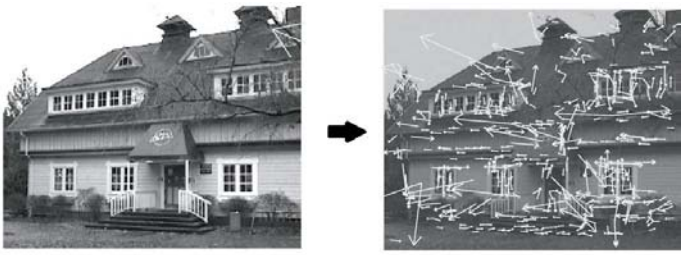

Figure 2.1.4: The SIFT features (right) generated for an image of a house (left). Each feature is represented by an arraw. The length of the arraw represents the scale of the feature, the direction represents the feature's orientation. (Image courtesy of [34])

feature is characterised by the following data: an (x, y) sub-pixel image coordinate, an orientation vector describing the direction of the principal gradient, a scale, and a 128-dimensional description vector describing the image neighbourhood around the feature point. Figure 2.1.4 shows the resulting extracted SIFT features on a sample scene image.

Object recognition is performed by matching SIFT features from a stored database of object features (extracted from sample image views of the object) to the scene image features. This is done by first matching each scene image feature to its nearest neighbour object database feature based on the 128-dimensional description vector. Spurious matches are rejected by examining the ratio of the distance to the nearest neighbour and the distance to the second-nearest neighbour. If this ratio is above a certain threshold then the match is considered spurious and is rejected [34]. This is done because some SIFT features are more discriminatory than others. The ratio of the nearest and second-nearest match distances approximates how discriminatory the feature is.

This isn't clear. Do mean each point is individually looked up in the database?

The next step is to use a geometric consistency and model fitting method, such as the Hough transform [40] or the RANSAC method [41], to find the location and pose of a detected object in the scene image. The SIFT feature matching process is described in detail in Chapter 3.

Say a bit more about it here

One of the main challenges of this approach is efficiently performing feature matching between the scene and database features. Nearest neighbour matching in

a low dimensional space can be performed efficiently using a k-d tree [42]. However, SIFT descriptors are 128-dimensional, and as a result of the high dimensionality of the search space k-d trees perform poorly. An alternate approach is to use an approximate nearest neighbout matching method using the best-bin-first modification of the k-d tree search [43] to significantly speed up feature matching.

There has been some work in improving SIFT. One of the weaknesses of SIFT is that it does not take colour into account, as the base algorithm deals with grey-scale image. CSIFT [44] is an extension of SIFT that takes colour into account in the feature description vector. PCA-SIFT [45] is another variant, applying PCA on the 128-dimensional description vector of each feature to reduce its dimensionality. In some cases this can improve the feature matching accuracy and speed. SURF [46] is a related interest point detector, providing much faster image feature extraction performance as compared to SIFT, and improved matching under object transformation. If SURF is better, why didn't you describe it instead and use it as the basis of your work?

~~Presented~~ Improvements

Chapter 3 presents an improved method for matching learned object SIFT features to scene image features for object recognition and localisation. We improve on the existing nearest neighbour method by taking into account the geometric consistency of matched features concurrently with their description vector similarity. This is in contrast to the existing method where features are first matched using only their description vectors, followed by removing inconsistent matched in a later stage. Our approach results in a greater number of features matches, as well as allowing a significant improvement in matching speed in some situations. Is this good or bad?

2.2 Learning an Object's Appearance

The object recognition and localisation techniques reviewed in Section 2.1 depend on the availability of training data for the target object. If a robot is to be able

to interact with an object, it needs this training data (typically in the form of cleanly segmented views of the object) to learn it's appearance and build an internal representation of the object. This is then matched to a scene image depending on the particular recognition approach. For example, in the case of SIFT feature matching, features are extracted from cleanly segmented training images of the object, and then inserted into a database for later use in recognition and localisation tasks [34].

In the case of an autonomous robot, a problem can arise if a previously unseen object must be recognised. In some applications it may be possible to provide the robot *a priori* with training data for all objects that can be encountered in the environment. However, this is not always possible to do. For example, in the case of a house-hold service robot, there are ~~a countless number~~ many of different objects it may encounter, making it infeasible to provide training views of every possible object for recognition and localisation.

How are training data pre-programmed?

Rather than rely on pre-programmed training data, the robot should instead be able to autonomously generate training data for new objects encountered in the environment. This task involves observing the object and separating the object image regions and features from the background. This problem can be solved through image segmentation, separating image into background and object segments. The problem of image segmentation has been extensively studied in literature [14, 47], with many different approaches and methods. We can separate these into two classes, static image segmentation and dynamic segmentation.

This paragraph is not clear. It's dense and not clear what you are trying to say.

2.2.1 Static Image Segmentation

Static image segmentation refers to methods that use information from a single image. The aim is to the use brightness, colour, contrast, and texture data contained in the image to determine the object and background regions. This can then be used by a robot to build an appearance model of the object for later recognition and localisation.

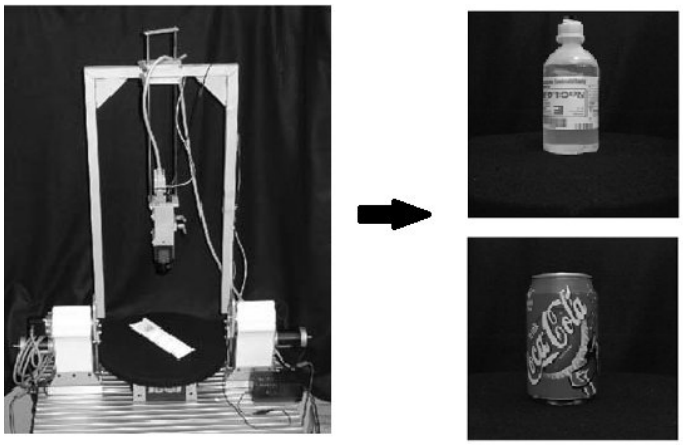

Figure 2.2.1: The environment may be engineered to make the background easy separable from the object image regions. In this case, the object is placed on a turn-table with a very dark colour, making threshold segmentation possible. (Image courtesy of [49])

One of the simplest approaches is to use an intensity threshold value to separate the background pixels from the object pixels [14, 48]. The choice of threshold value can be determined in several ways, for example choosing a fixed constant or by examining the image intensity histogram to find a suitable threshold value. This lightweight approach may be suitable for situations where the background colour is distinct from the object's appearance. An example of this is if the object is placed on a turn-table engineered to have a fixed colour distinct from the object [49, 50] (see Figure 2.2.1). However, in the case of an uncontrolled environment this approach is not effective as the background scene can have similar colour and intensity to the target object.

Region growing [15] is an approach to image segmentation that considers the relationship between pixels in a region and grows the region to include adjacent pixels that have similar properties. There are two categories of region growing algorithms, seeded [51] and unseeded [52]. In the case of seeded region growing, the algorithm is initialised with a set number of starting image locations. Each region is then grown outwards from the seed points by considering the neighbouring pixels at the region border. If a neighbouring pixel has similar properties to the region, then it is included in the region. An example criteria for including a neighbour pixel

is if its intensity is within a threshold distance of the mean insensity of the region's pixels.

One of the issues with this approach is the choice of seed locations can significantly affect the final segmentation. An alternative is unseeded region growing. In this case an arbitrary point is chosen in the image to start growing a region. When a neighbouring pixel differs from the current regions by more than a threshold amount, it becomes a seed for a new region.

A different approach to ~~the~~ segmentation ~~problem~~ is to represent the image as a weighted, undirected graph [53]. Each pixel is represented by a graph vertex, and adjacent pixels are connected by graph edges with weights determined by a similarity measure, such that similar pixels are connected by heavier weighted edges. This graph is then partitioned in a way that minimises some energy function using a graph-cut [54, 55, 56]. A cut is a parititioning of the graph vertices into two disjoint subsets by removing a cut-set of edges. A min-cut is a cut such that the sum of the edge weights of the cut-set is minimum. This can be used for image segmentation by introducing two extra vertices that can represent, for example, the background and foreground image regions. These are joined to all of the pixel vertices by edges weighted by the *a priori* confidence that the corresponding pixel is in the foreground or background. A min-cut is then performed to separate the vertices into two disjoint sets. The vertices that are in the set connected to the foreground represent the foreground image region pixels, the remainder the background. This process is shown in Figure 2.2.2.

Another method of segmentation is to use a watershed transform. This is done by considering the gradient magnitude of the image as a topographic surface that defines the *altitude* of each pixel. The image is then treated as a landscape, and a water precipitation process is simulated to calculate the point for each pixel to which the water will flow down to. Using ~~this~~ these data, a watershed transform can be performed [57, 58], grouping together image regions that share a catchment point.

There are many other image segmentation approaches and algorithms. However,

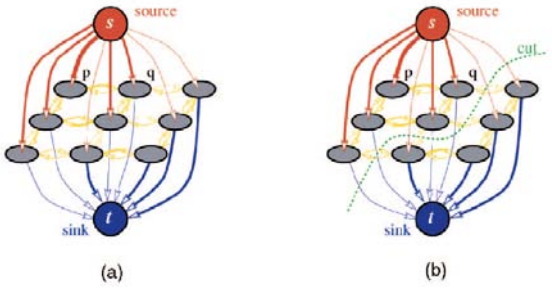

Figure 2.2.2: Pixels can be represented as graph vertices and adjacent pixels joined by weighted edges. A cut is performed to separate the graph vertices into disjoint sets linked to a sink and source nodes that can represent the background and foreground components. Image courtesy of [55].

Figure 2.2.3: Static image segmentation can over-segment a scene image. This is because the boundary of semantic objects in the scene may not correspond to colour, texture, or contrast boundaries in the image. (Image courtey of [53])

static image segmentation for the purposes of separating a target object from a complex background, with no *a priori* knowledge of the object's appearance, is fundamentally reliant on the assumption that the object boundaries correspond to discontinuities in the brightness, colour, texture, or contrast in the image. However, for complex objects in cluttered scenes, this assumption ~~will~~ does not hold. A single image may not contain sufficient information to resolve ambiguities and separate a complex object from a cluttered background. Figure 2.2.3 shows an example of a complex scene with the resulting segmentation. The people in the foreground are oversegmented, with the region boundaries located at image discontinuities rather than at semantic object boundaries.

Rather than relying on a single image to perform scene segmentation, a robot can use multiple image dynamic segmentation to ~~successfully~~ separate the target

object image regions from the background.

2.2.2 Dynamic Segmentation

We refer to scene segmentation methods that use more than one image, or a stream of images, as dynamic. The aim is to use the temporal domain to gather more scene information to improve the effectiveness of the segmentation.

One particular class of dynamic segmentation approaches is background subtraction. First a background model of a scene is constructed, and then when the target object is placed in the scene, the background model is subtracted from the scene image. The remaining regions are the foreground object. Figure 2.2.4 shows an example of this process. There are many variations of background subtraction [59], differing in how the background is modeled and how it is used for extracting the foreground. One approach is to model the background colour of each pixel as a Gaussian probability distribution function [60]. The colour value of each pixel is tracked over time and a Gaussian function is fitted to the values. When performing segmentation, the current value of each pixel is compared to it's probability function to determine the likelihood that the pixel belongs to the background. If the likelihood is over a threshold, then the pixel is considered to be a background pixel. This approach works well for situations where the scene background is static. However, if there is regular movement in the background, such as swaying trees, a uni-modal Gaussian is a poor model for the value of a background pixel. An improved approach is to model each pixel as a mixture of Gaussians [61]. In this way regular background movement can be taken into account.

Using purely background subtraction for foreground segmentation can be problematic in cases where the foreground object does not move uniformly. This may be the case for a person who is moving only their arms and head, but not their body. In this case the body would be erroneously labeled as background, while the limbs and head as foreground. By combining colour and contrast with motion cues, more

Figure 2.2.4: Foreground segmentation using background subtraction. The background model is learned over time (top right). When foreground objects enter the scene (top left), the background model is subtracted from the image (botom left). This allows the foreground image regions to be extracted (bottom right). Image courtesy of [62].

effective segmentation in such cases can be achieved [63]. Another approach is to segment the scene into different motion layers, rather than simply foreground and background [64].

Nonetheless, background subtraction ~~approaches~~ assume that the ~~scene~~ background is only slowly changing or changes in a periodic manner. Large unpredictable movement in the background would result in that movement being mistaken for the foreground object.

Another dynamic object segmentation approach is to combine pixel based segmentation and feature tracking to generate object image snapshots. In the work ~~by~~ of Southey *et al* [65], the scene is observed using a stereo camera pair to generate a depth map. The depth information for each pixel is used in combination with the pixel intensity to perform a segmentation of the scene using a normalised cut algorithm [56]. This, however, leads to an oversegmentation of the target objects (see Figure 2.2.5). This problem is addressed by moving the target objects and tracking

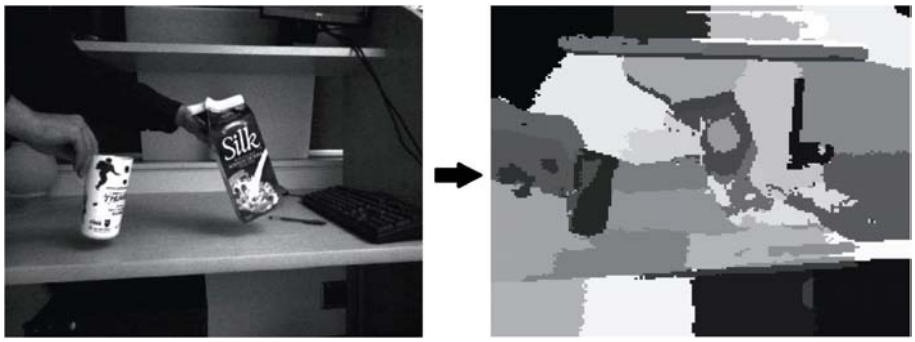

Figure 2.2.5: In the method presented by Southey *et al* [65], the scene image (left) is first segmented using pixel intensity and depth information using a normlized cut algorithm. This results in an oversegmentation of the image (right) as the pixel intensity boundaries do not correspond to object boundaries. (Images courtesy of [65])

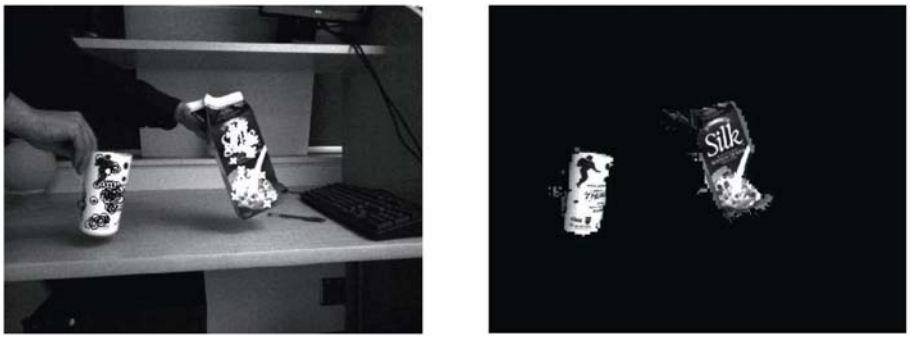

Figure 2.2.6: In the method presented by Southey *et al* [65], tracked SIFT features (left) are used to join together segmented image regions that move together. The merged regions correspond to the target objects. (Image courtesy of [65])

the motion of the SIFT features. This information is then used to merge together the oversegmented regions by considering that SIFT features moving together in adjacent regions should be part of the same segment. The result is correctly segmented target object image regions (see Figure 2.2.6). However, this method does not effectively address the issue of background motion as moving background objects can be mistaken for the foreground.

Active Robot Segmentation

Active robot segmentation refers to methods that rely on the robot actively manipulating the target object in the environment to separate it from the background. For example, the robot can nudge the object to generate movement and by detect-

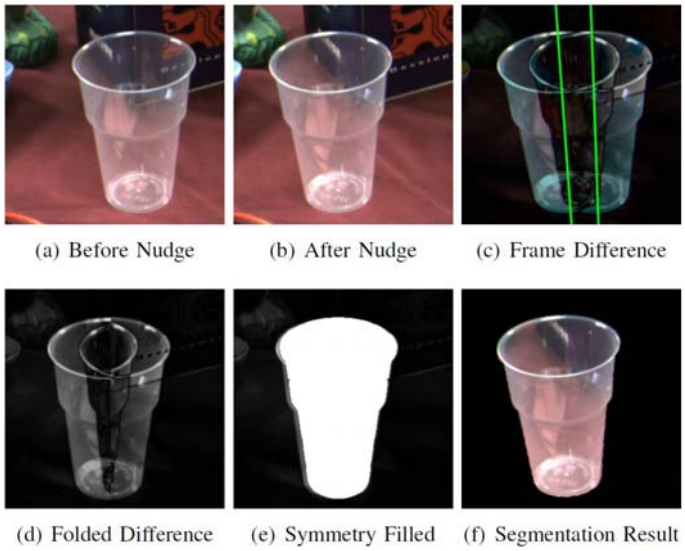

Figure 2.2.7: Nudging a symmetrical object while tracking the axis of summetry line allows the object to be segmented from the background. (Image courtesy of [66])

ing the movement, segment the object image regions. One particular approach uses object symmetry to track the object motion and determine the object displacement [66]. Symmetric regions are found in the scene and nudged by the robot manipulator perpendicular to the camera viewing direction. The displacement of the axis of symmetry in the scene image is used as a cue to determine the object region (see Figure 2.2.7). However, this approach is only applicable to near-symmetric objects.

A more general approach to object segmentation through active robot manipulation is presented by Fitzpatrick *et al* [67, 68]. In this approach, the robot uses it's manipulator to sweep the area containing the target object. This is done while tracking the scene motion by using per pixel frame differencing. When the robot's manipulator makes contact with the target object, a burst of motion is generated due to the movement of the object (see Figure 2.2.8). The burst of motion is located around the edges of the object, and can have discontinuities. To extract the full object image region, a min-cut [55, 69] algorithm is used (see Figure 2.2.9). Min-cut uses the sparse object motion information along the edges to extract the foreground image region. ~~The~~ Object segmentation can be refining by repeating the process multiple times [70].

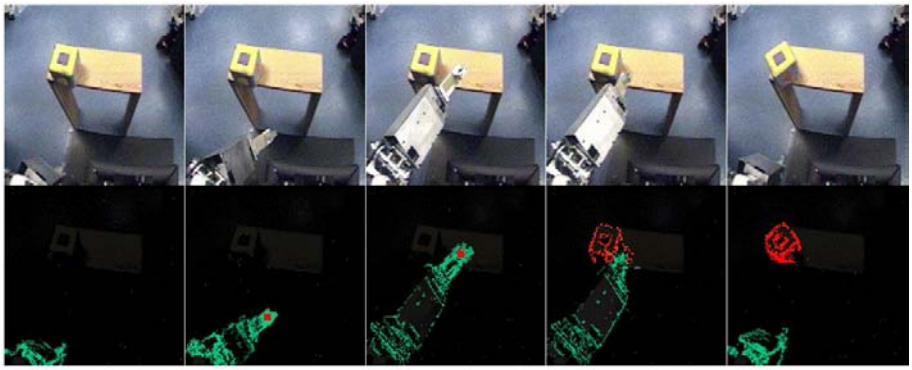

Figure 2.2.8: In the system presented by Fitzpatrick *et al*, the robot sweeps the scene with the manipulator, tracking the image motion using frame differencing. When the robot's end-effector makes contact with the objet, a burst of motion is detected. This can then be used for object region segmentation. (Image courtesy of [68])

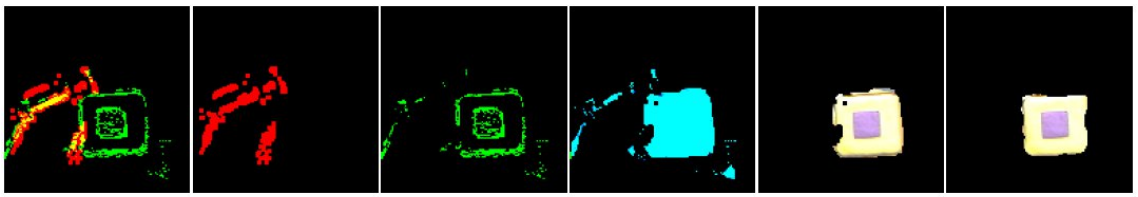

Figure 2.2.9: The robot manipulator bumps the target object. The motion due to the manipulator is filtered out, and the object regions are segmented using a min-cut algorithm. The result is a clean segmentation of the object from the background. (Image courtesy of [67])

This approach of robot-induced object motion for foreground segmentation works well for scenes with a static background. However, any significant and unpredictable background motion can result in ~~an~~ incorrect segmentation. If a background object moves at the same time as the target object, the resulting segmentation could include the background object as well as the target object image regions. Another issue is the nature of the manipulation of the object. The method presented by Fitzpatrick *et al* [67, 68] requires that the object is placed in the scene and then bumped by the robot manipulator. This may be sufficient for learning a single aspect of the object, however, learning multiple aspects from different angles may be difficult. For example, in the case of a cube shaped object, there are only six orientations in which it can be placed on a flat surface. To observe the object from some view points, the robot ~~will need to~~ must orient its body and camera appropriately, rather than orienting the object. Ideally, the robot should hold the object and be able to orient it appropriately to view and learn the various aspects.

~~Presented~~ Improvements

In Chapter 4 we present a method for a robot to autonomously segment object features from the background to build a model for object recognition. We address the issue of background motion, background clutter, and the ability to observe different aspects of the object. Our approach is to track individual SIFT features in the scene while the robot moves the object. We use the long term trajectory data for each feature to segment the object features from the static background, as well as from any background motion. The segmented object SIFT features can then form the basis of object recognition as well as for reconstructing the object's 3D shape.

2.3 Reconstructing an Object's Shape

After learning an object's appearance, the next step for a robot to be able to interact with the object is to reconstruct it's 3D shape. Knowing an object's shape allows the

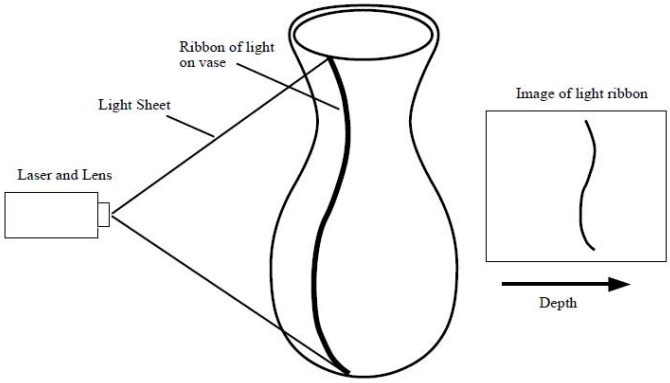

Figure 2.3.1: By projecting a strip of light onto an object and observing the resulting image, the contour of the object can be determined. (Image courtesy of [73])

robot to perform grasp planning [71, 8] and motion planning [72] effectively. There is a large amount of existing literature on the topic of 3D object reconstruction, using a wide variety of approaches. These approaches vary in ~~terms of~~ their speed, accuracy, assumptions about the shape of the object, and suitability for an autonomous robot to reconstruct an object in a complex environment.

A technique used for very high quality 3D reconstruction is to use a laser to project a line of light onto an object placed on a turn-table [73, 74]. A camera is used to detect the reflected line of light on the surface of the object. The shape and location of the line in the camera image specifies the contour of the object (see Figure). The object is rotated on the turn-table, allowing the full 360° shape of the object to be reconstructed. This method can output an extremely accurate and dense 3D reconstruction of the object. However, it requires that the object is placed in a carefully engineered environment, with a turn-table and calibrated laser projector and camera.

Instead of using a specialised laser projector to determine an object's shape, its appearance in a camera image can be used for reconstruction. For example, the object's silhouette can be used to reconstruct its shape [75, 76]. A single silhouette image cannot be used to determine the object's volume, but the areas of the scene that are not part of the object volume (the image areas outside the silhouette). By combining ~~this~~ these data from many images of the object from different view points, the

object volume can be determined. The problem with this approach is that for some objects the visual hull is not equal to the shape of the object. For example, concave indents on the surface of the object cannot be accounted for in the silhouette, and thus cannot be reconstructed using this technique.

The shape of an object can be inferred from a series of images in which the camera is moving relative to the object. The observed motion field of the series of images gives clues as to the structure of the shape. This technique of 3D shape recovery is known as structure from motion [77, 78].

Another technique is to use SIFT features for reconstruction. Each feature is highly discriminatory, and therefore allows a correspondence to be built between different images of a scene. This has been used for such as robot SLAM [79], and can be applied for object reconstruction by observing the object from multiple view points and correlating the SIFT features between the different images. By doing this, the relative 3D positions of the SIFT features can be determined, and used to build a 3D point cloud of the surface of the object [80, 81]. Figure 2.3.2 shows an example of an object point cloud recovered from correlated SIFT features. Another method uses matched SIFT features between images as a basis to perform further shape recovery of the scene using Delaunay triangulation and graph cuts [82].

Local image features other than SIFT can be used as well. Yamazaki *et al* [83] presented a method for a wheeled robot to reconstruct an object by driving around it and tracking the object's image features using a KLT tracker [84, 85]. The data from the motion was used to construct a point cloud representation of the object's surface. The problem with these approaches is that SIFT features (and most other local image features) are found in highly textured image regions, around corners and edges. Plain coloured regions will not generate many SIFT features, or none at all, resulting in insufficient shape information for parts of the object.

Structured light is one method that can be used to recover 3D shape in areas that lack texture [86]. This class of techniques use a projector to project a light pattern (typically in the IR spectrum) onto the scene. This light pattern is then

Figure 2.3.2: In the system presented by Skrypnyk *et al*, a point cloud is generated by correlating SIFT features from multiple images of the scene. The left image shows the cup placed in the environment, the right image shows the resulting SIFT feature point cloud and the camera view points used the generate it. (Image courtesy of [80])

detected by a camera and can be used to determine the depth of the scene at each image point. This approach has been incorporated in consumer devices such as the Microsoft Kinect[1]. This provides an inexpensive and reliable sensor for finding the depth information of a scene as viewed from the camera. Accurate and fast scene reconstruction is possible using a moving Kinect camera [87]. A Kinect sensor outputs a series of frame images, each image is composed of a standard RGB colour component and a depth value for each pixel. This data can be used to create a 3D model of the scene. However, a single frame only provides partial information, as many parts of the scene may be occluded or out of view. To compensate for this, the Kinect is moved around the scene, and the individual scene snapshots are stitched together using an Iterative Closest Point (ICP) algorithm [88]. ICP is a method of aligning two 3D point clouds by repeatedly shifting them to minimise the distance between corresponding point pairs.

For a robot to effectively reconstruct an object, it can grasp the object and manipulate it to view it from different points of view. However, this introduces the problem of filtering out the robot manipulator from the 3D model data. Otherwise the robot arm may be mistaken for the object. A secondary issue is how to optimally orient the object to view it from all of the necessary angles, as well as to account

[1]http://www.xbox.com/en-US/kinect

Figure 2.3.3: Robot reconstructing a held object using a depth camera (left). The object is observed from multiple points of view (top right) to account for occlusions. The shape uncertainty of the object (red in bottom right) decreases as more observations are made. (Image courtesy of [90])

for the robot gripper and arm occluding certain parts of the object. Krainin *et al* [89] address these issues by first learning an accurate 3D model of the arm to filter out arm segments, and by maintaining a confidence distribution over the object's surface to indicate the areas that need to be observed to learn the complete model of the object (see Figure 2.3.3).

In Chapter 5 we present a system that combines existing 3D reconstruction techniques with the segmentation and feature matching methods developed in Chapter 3 and Chapter 4, to allow a robot to autonomously recover the shape of an object in a complex environment.

2.4 Learning an Object's Properties

Learning an object's appearance and shape is not always sufficient for a robot to effectively use the object. Tasks such as grasping, manipulation, and tool use [91, 92, 6] may require the robot to learn properties of the object other than shape and appearance, as they may affect the outcome of some robot actions. Furthermore, these properties may not be discoverable through passive observation, but may instead require active robot interaction with the object. For example, an object's centre of mass cannot be determined by passive observation, as it may depend on the inter-

nal mass distribution of the object. Instead the robot can interact with the object, performing experiments to determine the centre of mass (for example: by dropping the object from different orientations and observing the outcome).

Affordance Learning

There has been some previous research in the area of learning object properties, much of it in the context of affordances. An affordance is a term first introduced by J.J. Gibson [93] in the field of cognitive psychology. It refers to a property of an object that allows an action to be performed. For example, a door knob affords being grasped and a button affords being pressed. ~~The relevant~~ work in this area ~~is~~ explor~~ing~~ how a robot can interact with an object, by performing actions and observing the outcomes, to determine the affordances and properties of the object.

In the work by Griffith *et al* [94], a robot categorises objects into container and non-container categories. The robot does this by dropping a small block over the object, and then pushing the object (see Figure 2.4.1). If the robot detects that the block and object move together, then the robot's confidence that it is a container increases. After performing these experiments on a number of different objects, the robot is able to learn the visual features that distinguish container and non-container objects, allowing it to categorise a novel object using its depth image.

In other work, the robot determines whether an object is rigid or soft-bodied using exploratory poking actions [95]. An object is placed on a table and its image based skeleton is extracted. The robot then performs a poking action and compares the initial image based skeleton to the resulting skeleton. The difference between the two is used to determine if the object is rigid or non-rigid, as well as to find the location of any joints. A similar approach has been used to locate and classify the joints of an articulated object such as a pair of scissors [96].

In addition to the previous work focusing on learning the object properties, there has also been work in learning properties inherent to the relationship between the

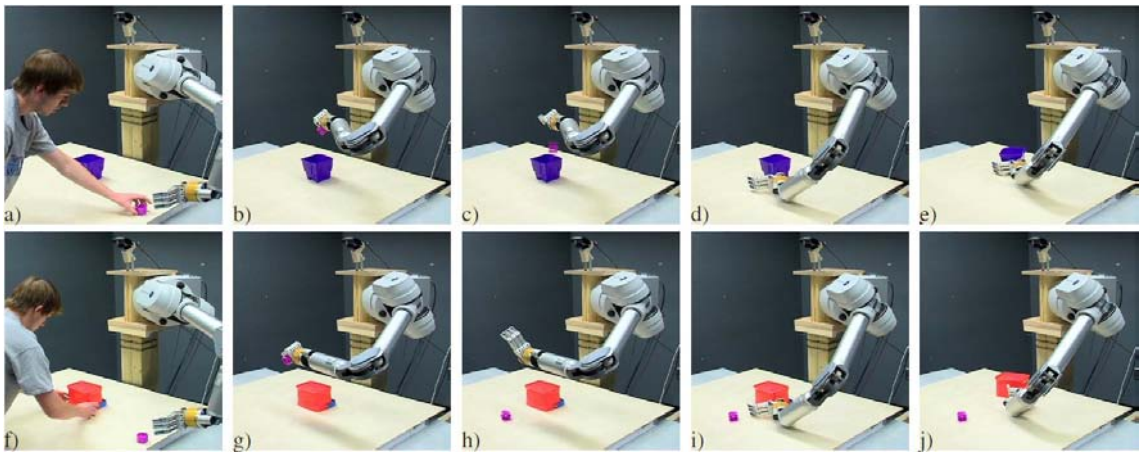

Figure 2.4.1: Griffith *et al* developed a method for a robot to classify objects into container and non-container classes by dropping a block over each of them and observing if subsequently the block and the object moved together. (Image courtesy of [94])

robot and the object. The learning of grasping affordances of various objects is explored by Kraft *et al* [97, 98]. In this work a robot repeatedly performs experiments on various objects, attemping to grasp them at different points. The success and failure of these grasps allows the robot to build a model of the object that defines the areas of the object that can be successfully grasped.

Stoytchev presented work on grounding the affordance representation of an object in the context of a robot's behaviours [99, 100]. The robot in this case learns how various objects extend its reach. The objects in question are stick tools of different shapes. The robot performs behavioural babbling with each tool, moving them around the workspace while observing the effect on a puck object (see Figure 2.4.2). The resulting movement of the puck when it is manipulated with the different tools allows the robot to build an affordance model of each object define how the tool can be used to move a puck. Brown [9] further extended the concept of grounding an objects affordances and properties by incorporating active learning and inductive logic to build a symbolic planner based description of a tool object. This allows a level of generalisation to be built into the affordance representation.

Fitzpatrick *et al* developed a method for a robot to learn the motion model of an object in response to a prodding action [101]. The robot uses its manipulator

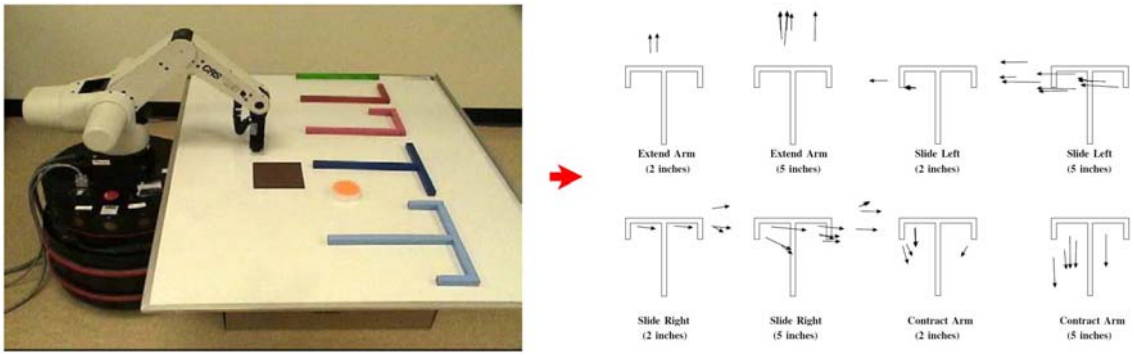

Figure 2.4.2: Work by Stoytchev involved a robot learning the reach extension tool affordance model of various stick objects (left). The reach extension model is represented as how the position of a puck is affected by moving the stick object (right). (Image courtesy of [99])

to bump various objects from different directions (see Figure 2.4.3). The resulting movement is used to construct a model of the motion of the object, represented as a motion vector distribution parametrised by the poke angle. Different objects will have different motion models. For example, a cylinder shaped object will roll when bumped from some directions, but not from others. A ball, on the other hand, will roll regardless of the direction of the bump. ~~This~~ These data ~~is~~ are then used to categorise objects, and to choose an appropriate action to make a particular object move.

The main shortcoming of the approach presented by Fitzpatrick is that the representation of the object's model does not generalise to different situations. For each object, the model is in the form of an explicit distribution over motion vectors, learned from the robot's explorative poking actions. Such a model is capable of predicting the object's motion only in a similar environment as the learning environment. However, if the environment is changed, for example by putting the object on a slope, the learned model will be not make accurate predictions.

There has been other similar work, using a neural network to model the motion as a result of a robot poking action [103]. Another method of modeling an object's affordance is to use a Bayesian network [104]. In this case, the Bayesian network represents the probability of various outcomes when different actions are performed on objects with certain properties (size, colour, shape). The Bayesian network

single 'l' is American. Double "ll" is real English.

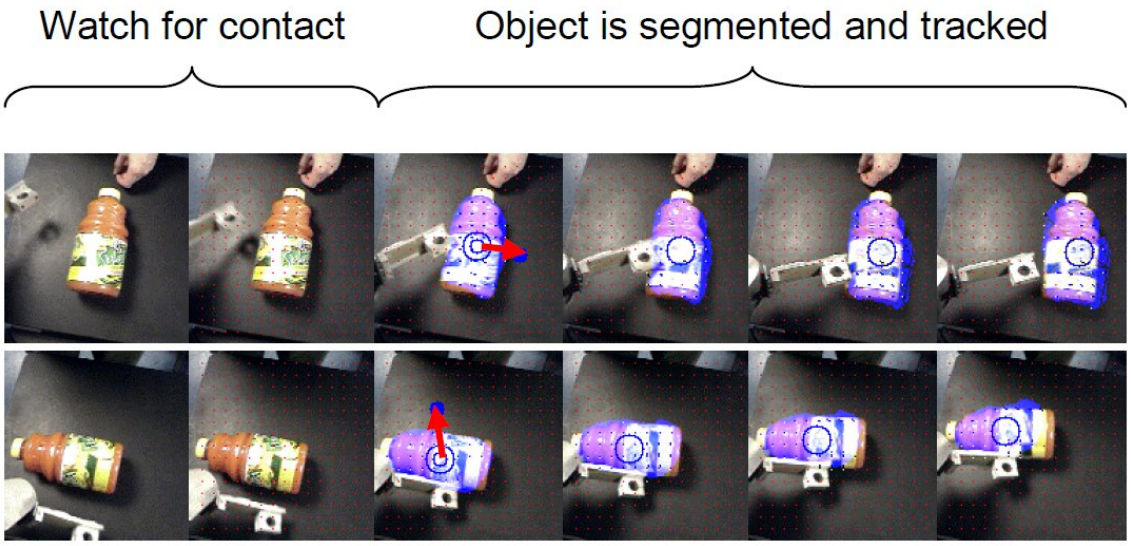

Figure 2.4.3: The approach by Fitzpatrick involved the robot poking an object from different angles and tracking the resulting movement. This movement data ~~is~~ are used to build a motion model of each object. (Image courtesy of [102])

did you actually ask for permission to use it? That's what "courtesy of" means

is built from empirical data obtained by the robot performing many trials of the actions on different objects and observing the results. Using a Bayesian network to model the object properties is a more general representation capable of expressing a variety of outcomes and actions as compared to explicitly modeling the motion vector [101]. However, this does not solve the problem of applying the learned model to environments significantly different to the learning environment.

The choice of robot exploratory actions is a further short coming of existing methods. Many of the existing methods use a behavioural babbling approach [105, 106] where the robot performs either random actions or a fixed list of predetermined actions. The disadvantage of this is that some actions may be more informative than others and therefore they should be prioritised ahead of the less informative actions. Furthermore, how informative a certain action is dependent on the results of the previous actions. Performing random or a fixed list of actions is not an effecient way of learning an object's properties. Alternatively a robot can learn an object's properties by demonstration and imitation [107], where a human interacts with the object and the robot infers the properties by observing the outcomes.

System Identification

A related concept to discovering the properties of an object is system identification [108, 109]. System identification is the process of modeling a dynamic system using statistical methods, such as linear regression, based on some experimental measurements of the system. This also includes elements of optimal experimental design [110, 111] to generate useful and informative measurements for model fitting. System identification is commonly applied in control engineering applications for building a model of a complex system to be used in a control algorithm. An example application of this is to model the behaviour of a small unmanned helicopter [112]. In this case, the helicopter is flown using remote control by an experienced operator. The reponse of the helicopter to various inputs is recorded, including acceleration, roll, pitch, yaw, etc. This experimental data is then used to build a model of the helicopter dynamics, to allow automatic control of the helicopter. The dynamic model representation varies depending on the particular application, one example is to model the helicopter dynamics using a neural network [113].

One of the weaknesses of using a system identification approach for a robot to model an object is it does not incorporate a feedback mechanism between the results of an experiment and the choice of the next experiment to carry out. Typically a predetermined list of experiments is performed to gather data, and a model is then fitted to the results. A better approach is for the choice of the next experiment to carry out on a system or object to be based on the results of the previous experiments.

~~Presented~~ Improvements

In Chapter 6 we present a method for a robot to autonomously learn the properties of an object using active interaction. We use a physics simulator to model the object, as well as to generate hypotheses about an object's properties and predictions of the outcome of a given robot action. When the robot performs and experiment, we

Is "confidence" a better term here?

update the certainty distribution over the object properties, and choose the next actions based on this distribution to maximise the information gained. By doing this we improve on the existing methods in several ways. First, by using a physics simulator representation of the object model, we gain a level of generality for the model. A learned model should be able to describe the behaviour of the object in a number of different situations, distinct from the learning environment. Second, we use the physics simulator ~~environment~~ to simulate the potential outcome of different experiments and actions, which allows the robot to choose the most informative experiment to perform. This minimises the number of actions required to learn an accurate model of the object.

Chapter 3

Image Feature Matching for Object Recognition

One of the abilities that a robot must possess to interact with an object in an unstructured environment is perception. A robot cannot effectively interact with an object if it cannot recognise and localise it in the world. There are various types of perception modalities, such as vision, sound, and touch. We are concerned with the vision mo[illegible] Motherhood statements that you don't need [illegible]mera provides the robot with a high-dimensional stream of data in the form of images, each of which is a 2D array of pixels. Each pixel typically contains colour data, but can in some instances also contain distance range data. The problem of object recognition revolves around interpreting this stream of data to determine whether the perceived scene contains a target object and at which location. Many object image recognition algorithms have a common overall structure: training images are used to learn some particular image features of the object (these can be colour, shape, texture, corners, etc), and then these learned features are matched in some way against scene images to recognise the object. The feature matching method can greatly affect the overall accuracy and speed of an object recognition algorithm.

In this chapter we present a method for matching learned local image features (specifically SIFT [34] features) to scene features, for the purpose of object recog-

nition and localisation. Our approach differs from existing methods by taking into account the geometric consistency of matched features concurrently with the description vector similarity. As a result we do not need to over-constrain the description vector matching criteria as is the case with existing methods. The outcome of our approach is a greater number of feature matches which improves the accuracy of object recognition, as well an improvement in matching speed under certain circumstances.

Refer back to discussion in lit. survey to justify

3.1 Introduction

Local interest point detectors [31, 32, 33] are a class of algorithms that can be used for object recognition and localisation in images. Interest point detectors fall into the category of local image features. These use a small neighbourhood of pixels around a point, as opposed to global image features that deal with the entire image. (

You're repeating from chapter 2. Don't need or shorten paragraph.

ansform (SIFT) [34, 35]. The SIFT algorithm generates highly discriminatory scale and rotation invariant description vectors of the pixel neighbourhood around stable points (minima and maxima in image scale space), coupled with the orientation of the image gradient at this point and the scale of the pixel neighbourhood. The SIFT algorithm is described in more detail in Section 3.2.

There are several steps required to recognise and localise an object in ~~a scene~~ an image using SIFT features (these steps also apply to many other interest point descriptors). First a database of object features is learned using training views of the object. The training views are typically images of the object with either a blank or cleanly segmented background. These images can be generated either in a controlled environment (eg: turn table with a clean background [49]), with a human performing background segmentation, or by autonomously separating object features from the background (see Chapter 4).

The next step is to use the properties of each feature (such as the description

vector, gradient orientation, and image position) to match the scene features against learned database features to find the object in the scene. Figure 3.1.1 shows an example match between a reference image and scene image features.

In this chapter we present an algorithm for feature matching that is more accurate, flexible, and (under certain conditions) faster than existing methods. Our algorithm is focused on matching SIFT features, but in future may be applied to other local interest point detectors.

The main contribution of our approach is in combining the feature description vector matching with the geometric consistency filtering stage. Existing methods for SIFT feature matching consist of two distinct ~~and separate~~ redundant stages. First matching individual scene image features to database features using only the description vector, followed by filtering out incorrect matches using a geometric consistency check. Our method combines the two stages, using geometric consistency in parallel with feature description vectors to determine the database to scene feature mapping. This is done by finding a small set of match pairs that have an overall high consistency score, which is a combination of geometric and description vector consistency. This seed set of high confidence matches is then used to find additional matches using a positio

What does consistency mean?

This isn't a good overview because it gives details, without explanation and no real overview.

The advantage of our approach is the relaxed criteria for feature description vector matching, since geometric properties are used in conjunction to find potential feature match pairs. This is in contrast to the existing approach which is over-constrained in this regard, potentially missing valid feature matches. As a result, our algorithm achieves a higher percentage of correct feature matches as compared to the existing matching method. A further benefit of our approach is its structure allows scene knowledge and temporal coherence to be used to greatly speed up the search for feature matches. Under certain conditions our algorithm is much faster than the existing feature matching algorithm.

You said this before.

In this chapter we present the following:

- an overview of the SIFT algorithm (Section 3.2),

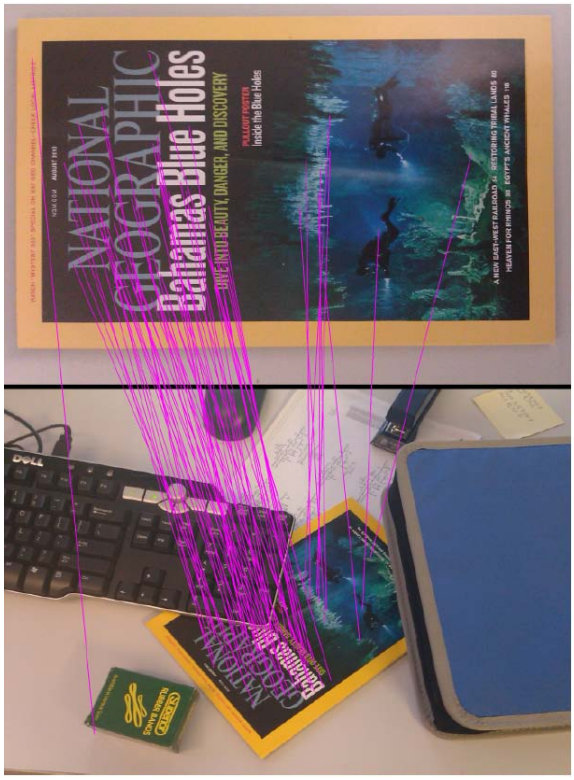

Figure 3.1.1: An example feature mapping from a training image of an object (above) and a cluttered scene containing that object (below). Each line connects a matched SIFT feature pair in the two images.

- current approach to matching features and it's associated shortcomings (Section 3.3),
- a detailed description of our matching algorithm (Section 3.4),
- performance evaluation and experimental results (Section 3.6),
- discussion of results (Section 3.7),
- and potential avenues for future work (Section 3.8).

3.2 SIFT Feature Generation

The SIFT algorithm is described in detail in a paper by D. Lowe [34, 35]. This section presents a brief summary of the algorithm and it's application for object recognition and localisation.

SIFT features are local image features that are invariant to image scale and rotation. They allow robust matching despite changes in illumination, image noise, and small 3D view-point changes. Extracting SIFT features from an image can be divided into several sequential steps:

Repeat of chapter 2

1. **Build a Difference of Gaussians pyramid.** This is computed by taking the difference of adjacent pairs of images from the scale-space pyramid. The scale-space pyramid of an image is built by repeatedly convolving an image with a Gaussian kernel. The resulting image after each convolution forms a layer of the pyramid.

2. **Find the extrema points in the Difference of Gaussians pyramid.** Each point in the pyramid has 26 neighbours, 8 on the same scale and 9 each in the above and below scale. The points that are the minimum or maximum of their 26 neighbours are the extrema points. These are typically found near corners and edges in an image at a given scale.

3. **Rejection of unstable extrema points.** Points with a low absolute value in the Difference of Gaussians pyramid or points that are too *edge-like* are considered to have poor repeatability and unstable image locations in the presence of noise.

4. **Orientation assignment.** Each of the remaining key-points is assigned a principal orientation. This is the most prominent gradient direction of a small neighbourhood of pixels around the key-point.

5. **Generation of key-point description vectors.** For each key-point, a neighbourhood of pixels is used to build an array of histograms of gradients. The histograms are normalised to the principal orientation of the key-point to make the descriptor vector rotation invariant. Additionally, the histogram is normalised on gradient magnitude of the pixel neighbourhood to increase robustness to changes in contrast and lighting. The result is a 128-dimensional description vector for each key-point.

The final output of these series of steps is a list of stable local image features, each characterised by an (x, y) image position, primary gradient orientation angle, scale, and a 128-dimensional description vector. It should be noted that SIFT uses a greyscale image. To extract SIFT features from say an RGB colour image it must first be converted to a single channel greyscale image. Figure 3.2.1 shows a representation of extracted SIFT features for a scene image.

3.3 SIFT Feature Matching

The first step in recognising an object in a scene image is to learn a database of object features from a set of training images of the object. These training images are prototypical views of the object from which SIFT features are extracted and stored in a database. The next step is to match these features against a scene image that may contain a learned object.

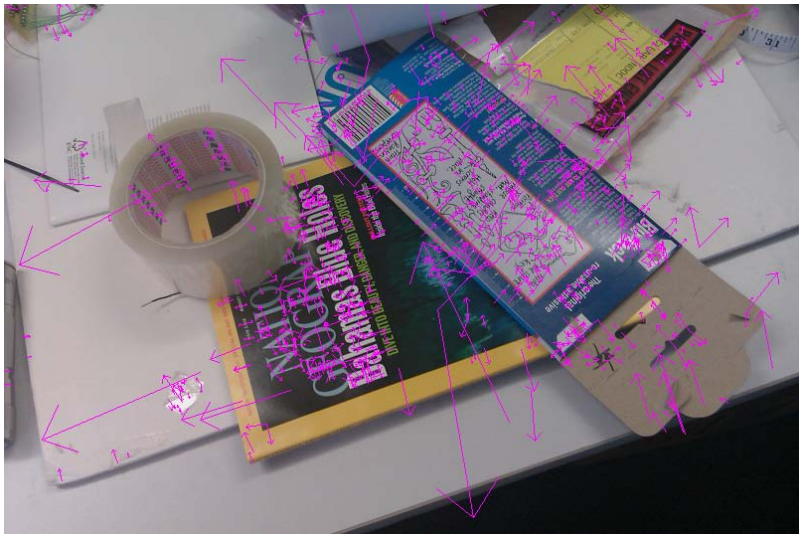

Figure 3.2.1: This image shows the SIFT features extracted from a scene image. Each arrow corresponds to a single SIFT feature at the arrow's origin. The length of the arrow represents the scale of the feature, and the direction of the arrow indicates the feature's primary gradient orientation.

Overview of Lowe's Algorithm (or whoever)

3.3.1 ~~Existing Approach Overview~~

The approach presented with the SIFT algorithm [34, 35] is to match each scene feature independently to the learned database of features, by comparing description vectors. This is followed by a geometric consistency check, such such as the Hough transform [40] or RANSAC [41], to remove inconsistent matches and determine the position and orientation of the object in the scene image.

A nearest neighbour approach is used for matching scene to database feature. For each scene feature the nearest database feature is found. This is done using the Euclidean distance between the 128-dimensional description vectors of the features as a metric. This results in every scene feature having a corresponding matched database feature, including many incorrect matches. To reject spurious feature matches a global description vector distance threshold may be used. However, it was found that not all SIFT features have equally discriminatory description vectors, and therefore a single global threshold performs poorly [34, 35]. Instead, a more effective approach is to compare the distance between the nearest neighbour and the second-nearest neighbour in the database. If the ratio of the distances to the

Algorithm 3.1 Nearest-neighbour feature matching.

input: set of learned database features $\rightarrow D$
input: set of scene feature $\rightarrow S$

$matches \leftarrow \{\}$
forall s **in** S
  $s\_descr\_vec \leftarrow descriptionVector(s)$
  $min\_distance \leftarrow \inf$
  $closest\_feature \leftarrow null$

  forall d **in** D
    $d\_descr\_vec \leftarrow descriptionVector(d)$
    if $|d\_descr\_vec - s\_descr\_vec| < min\_distance$ **then**
      $min\_distance \leftarrow |d\_descr\_vec - s\_descr\_vec|$
      $closest\_feature \leftarrow d$
    endif
  endfor

  $second\_min\_distance \leftarrow \inf$
  forall d **in** D
    if $parentObject(d) \neq parentObject(closest)$ **then**
      $d\_descr\_vec \leftarrow descriptionVector(d)$
      if $|d\_descr\_vec - s\_descr\_vec| < second\_min\_distance$ **then**
        $second\_min\_distance \leftarrow |d\_descr\_vec - s\_descr\_vec|$
      endif
    endif
  endfor

  if $min\_distance \leq 0.8 \cdot second\_min\_distance$ **then**
    $matches \leftarrow matches \cup \{(s, closest\_feature)\}$
  endif
endfor

output: set of feature matches $\leftarrow matches$

nearest and second-nearest neighbours is above a threshold value, the match is said to be spurious. For this threshold, the value of 0.8 eliminates more than 90% of false matches while discarding less than 5% of true matches. If the database of features contains multiples views of the same object, then the second-nearest neighbour feature is redefined to be the nearest feature that belongs to a different object than the first (closest) match. This addresses the problem of matching the same object feature for both the nearest and second-nearest neighbours. This feature matching process is defined in detail in Algorithm 3.1.

3.3.2 Weaknesses of the Existing ~~Approach~~

The existing ~~approach~~, described in the previous section, has several distinct weaknesses, which we seek to improve upon with a new feature matching method. These weaknesses stem from several factors: the first is scene features are matched to database features independent of one another and using only the description vector, the second is constraining each scene feature to only match its nearest neighbour in the database, and third the reliance on the second-nearest neighbour for rejecting spurious matches. We discuss each of these in turn.

The first weakness is matching each feature independently followed by a separate stage in which a geometric consistency check on the resulting feature matches is performed. This results in poor performance if the object to be matched has a regular pattern texture that generates multiple SIFT feature with similar description vectors. In this case, the existing approach is unlikely to generate a geometrically consistent match for the similar features as the initial match is performed independently and purely on the description vectors. An example of this is shown in Figure 3.3.1. The correct and geometrically consistent mapping is show on the left. However, since the features do not have distinct feature vectors, it is instead more likely that the matching stage will produce an incorrect mapping, shown on the right. This incorrect mapping would then be rejected by a geometric consistency check, resulting in the object not being detected in the scene image.

The second weakness refers to the constraint of matching a scene feature only to its description vector nearest neighbour in the database. This severely limits the potential matches for a feature. If the geometric relationship of other feature matches are considered in parallel, the correct match for a scene feature may not necessarily be its nearest neighbour in the database. Figure 3.3.2 shows a scenario in which the geometric relationship of a number of feature matches is contraining the remaining feature match ($F4$). But the suggested feature match is not considered as it is not the description vector nearest neighbour match. The end result of this is

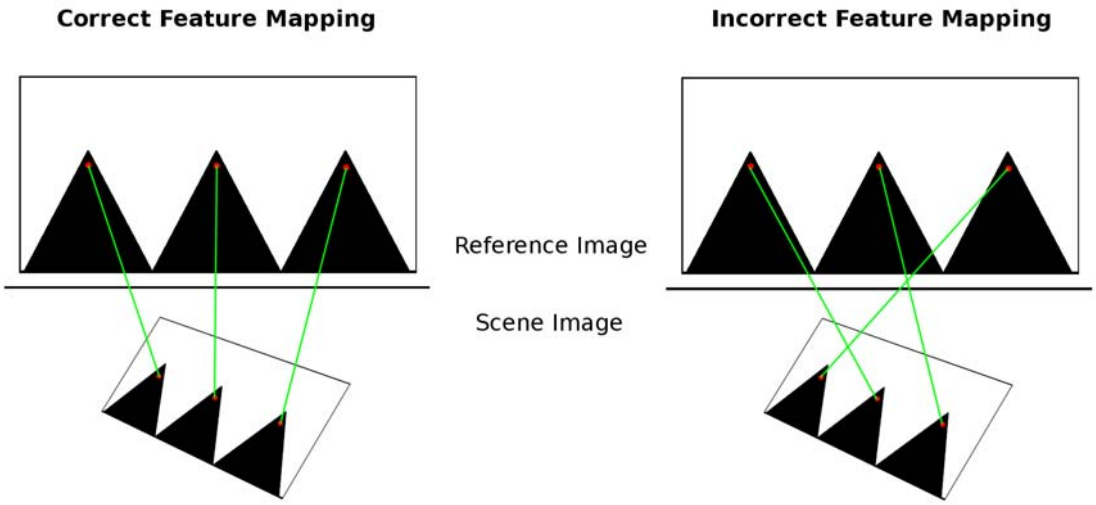

Figure 3.3.1: Consider an object with a regular pattern texture. This regular pattern would give rise to multiple SIFT features (represented by the red dots) with almost identical description vectors. In this case, if the scene features are matched to the database features independently, based only on the description vectors, the correct feature mapping (left) is not guaranteed. Instead an incorrect mapping may result (right), in which case these feature matches would be discarded by a later geometric consistency check.

a lower number of feature matches between the scene and learned database, which could lead to higher errors in the localisation of an object or not recognising an object at all.

Finally, the reliance on the second-nearest neighbour for rejecting spurious matches means that the accuracy and speed of matching an object's features is dependent on the remaining features stored in the database. Consider the scenario of two different feature databases, containing different features of different objects. We then insert into each database features from training images of a new object. Given that the same training data for the new object is used for both databases, we should expect that both databases would have the same matching performance. However, because the remaining features are different and hence the second-nearest neighbour for a scene feature is likely to also be different, the matching performance for the two databases will not necessarily be the same. This is not a desirable outcome.

A further side-effect is the impact on matching speed. The larger the database

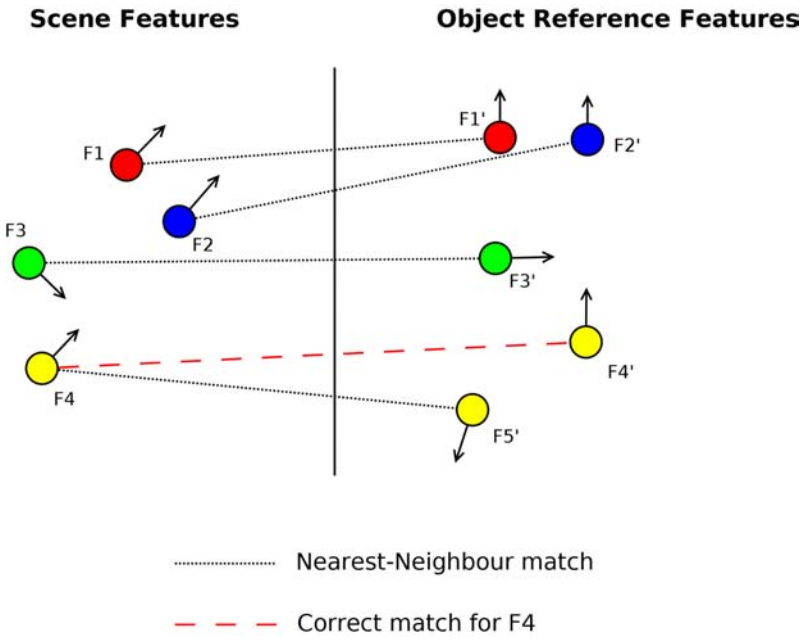

Figure 3.3.2: This diagram demonstrates a weakness of only considering nearest neighbour feature matches. In the above example, the scene features $F1$, $F2$, and $F3$ are successfully matched against the correct corresponding object reference features , $F1'$, $F2'$, and $F3'$. Consider that feature $F4$ and $F4'$ have similar description vectors, but $F5'$ has an even closer description vector to $F4$. In this situation, despite the match ($F4 \leftrightarrow F4'$) being geometrically consistent with the other matches, it will not be considered as they are not description vector nearest neighbours. Instead $F4$ will be matched to $F5'$ and later filtered out by a geometric consistency check.

grows as more objects are learned, the longer feature matching will take. In some cases this is unavoidable, however, in others we may wish to exploit background knowledge of the scene contents to speed up feature matching. Take for example a scenario where a scene is known to contain a specific object. When performing scene feature matching, it would improve performance to consider only the database features belonging to the known object, rather than all of the features in the database, including those belonging to other objects. However, when using the second-nearest neighbour threshold criteria for rejecting spurious matches, this is not possible.

To address these limitations, we present a new approach to matching SIFT features. We call this approach Bipartite Feature Matching, as it resembles building a bipartite graph between scene and database object features.

3.4 Bipartite Feature Matching Algorithm

3.4.1 Overview

The aim of our algorithm is recognise and localised an object by matching features from a set of training images of the object to corresponding features of that object in a scene image, while dealing with clutter, occlusion, orientation, and lighting changes. Let us call the set of object features extracted from a single training image a "snapshot", the feature database consists of multiple snapshots of various objects. It should be noted that we match snapshots from the database to the scene, in contrast to the existing approach described previously in which the opposite is done, scene features are matched to the database.

We consider the problem of matching a snapshot to the scene in terms of forming a bipartite graph between the snapshot's features and the matched scene features (Figure 3.4.1 illustrates this concept). A valid mapping will have each snapshot feature map to a single scene feature or to a null node (signifying no appropriate match for that feature), and each scene feature will have no more than one

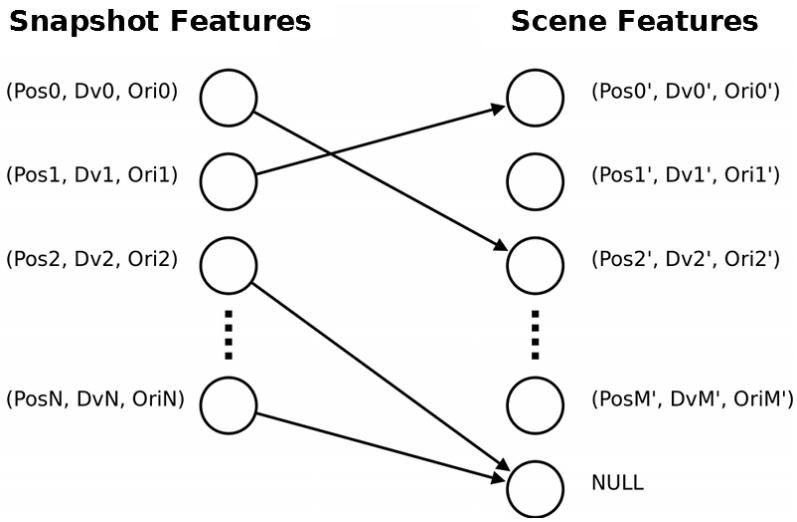

Figure 3.4.1: The task is to map object snapshot features to scene features, each feature has a pixel position (Pos), a description vecto (Dv), and an image gradient orientation (Ori).

snapshot feature mapped to it. There are several other criteria for a valid feature mapping; matching features should exhibit three forms of consistency in relation to one another. These are *Description Vector Consistency*, *Position Consistency*, and *Orientation Consistency*. *Description Vector Consistency* (described in detail in Section 3.4.2) constrains matched features to have similar description vectors. *Position Consistency* (described in detail in Section 3.4.3) constrains matched scene features to have the same relative image positions as the corresponding database features. Similarly, *Orientation Consistency* constrains matched scene features to have the same relative orientations as the corresponding database features.

Emphasise relative in case it appears to be affected by translation and rotation ?

To find a mapping that satisfies the above criteria, we first find a small subset of matches between snapshot and scene features that have a high confidence of being valid (described in detail in Section 3.4.5). This subset is then used to bootstrap the rest of the feature mapping (described in detail in Section 3.4.7). Once a consistent mapping is found between snapshot features and scene features the position and orientation of the object in the scene image can be determined using a least squares model fitting approach. There is no need to remove outliers or inconsistent matches with RANSAC[41] or the Hough Transform[40].

~~Finally,~~ In Section 3.6 we present experimental results demonstrating the effectiveness of this approach for feature matching, followed by discussion of these results and areas for future work.

3.4.2 SIFT Description Vector Consistency

Each SIFT feature has an associated 128-dimensional description vector, which is a scale and rotation invariant description of the pixel neighbourhood around the interest point. We use this description vector as the primary method by which SIFT features are matched between images. SIFT description vectors are highly discriminatory, therefore if two features in two different images of a scene have similar description vectors, they have similar local pixel neighbourhoods and are thus likely to correspond to the same point on an object. A mapping between a snapshot's features and scene features should exhibit *Description Vector Consistency*, which is the requirement that matched features have similar description vectors. The similarty between two SIFT features is measured by the Euclidean distance between their description vectors. Given two feature description vectors A and B, the distance is:

$$Distance(A, B) = \sqrt{\sum_{i=1}^{128} (A_i - B_i)^2} \tag{3.4.1}$$

The lower the descriptor distance between two features the more likely they are to match. We further quantify this relationship by examining the description vector distances between matching and non-matching features. The goal is to determine the function $F(D) \rightarrow p_{match}$ which, given the distance D, returns the probability p_{match}. This is the probability that two features match given the Euclidean distance D between their description vectors.

To determine this function we empirically tabulated the distance between many pairs of matching and non-matching features. For non-matching features, we took images of distinct scenes and recorded the SIFT description vector distance between all pairs of features. We assume that any such pair is a non-matching pair as the

Are the probabilities dependent on the types of objects or background?

scenes did not share common objects. For matching features, we took several objects and placed them in a scene in varying orientations, positions, and lighting conditions. We then manually matched features ~~which~~ that were on the same point on the surface of the same object in different images. For each matched feature pair we recorded the SIFT description vector distance.

Using these two tables (consisting of matching and non-matching feature description vector distances) we can calculate the probability that two features match, given the distance between their description vectors. We do this by taking the percentage of all matching feature pairs with distance less than D, and comparing it to the percentage of all non-matching feature pairs with distance less than D. If we plot the description vector versus match probability points the resulting shape strongly resembles that of a Q-function. What is this? Reference? This is expected, as the distance between matching description vectors should follow a Gaussian distribution. This follows from the Central Limit Theorem and the fact that a description vector is derived from many independent pixel measurements (the pixel neighbourhood around an interest point). We then fit a sigmoid logistic function, which serves as a simple approximation to the Q-function, to these data points. The final result is the following function (also shown in Figure 3.4.2):

Are you putting the data in an appendix What are the errors? How stable are these numbers?

$$F(D) = \frac{1}{1 + e^{\frac{D-397}{48}}} \tag{3.4.2}$$

3.4.3 Feature Position Consistency

Position Consistency refers to the fact that, assuming a rigid object, the mapped object snapshot features should have the same positions relative to one another as the corresponding scene features. Initially position consistency is defined for a triplet of matching feature pairs, and can then be extended to the entire mapping by considering all triplets of matching feature pairs. A matching feature pair is a tuple of features, $(SnapshotFeature, SceneFeature)$, the first is from the set of snapshot

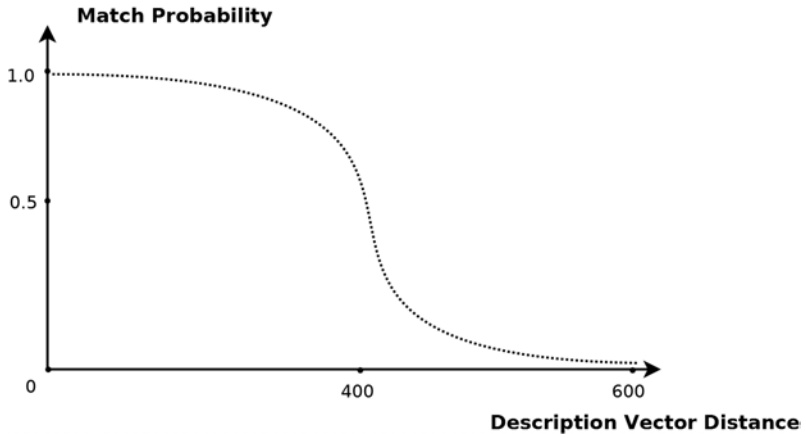

Figure 3.4.2: This function outputs the probability that a pair of SIFT features match, given the Euclidean distance between their description vectors.

features and the second is the matching feature from the set of scene features.

Let the triplet of matching feature pairs be $(F1, F1')$, $(F2, F2')$, and $(F3, F3')$; to determine whether they are *Position Consistent* we need to check whether the relative positions of the snapshot and scene features are similar. This is done by first finding the perimeters of the triangles formed by the snapshot and the scene features' image positions, p and p' respectively. We then calculate the normalized pixel edge lengths of the two triangles by taking the edge distance and scaling by the perimeter of the triangle, eg:

Not clear where triangles come from. Something left out of explanation

$$normalisedLength(F1, F2) = \frac{|imgPos(F1) - imgPos(F2)|}{p} \tag{3.4.3}$$

For a position-consistent match the difference between corresponding normalized edge lengths of the snapshot and scene features will be small. That is,

$$|normalisedLength(F1, F2) - normalisedLength(F1', F2')| < \epsilon \tag{3.4.4}$$

The purpose of normalizing the edge lengths is to account for varying scales and image resolution. Figure 3.4.3 shows a case of a position consistent and inconsistent match. The features in the *Snapshot* are position-consistent with the matching features in *SceneA*, but are inconsistent with matching features in *SceneB*. For

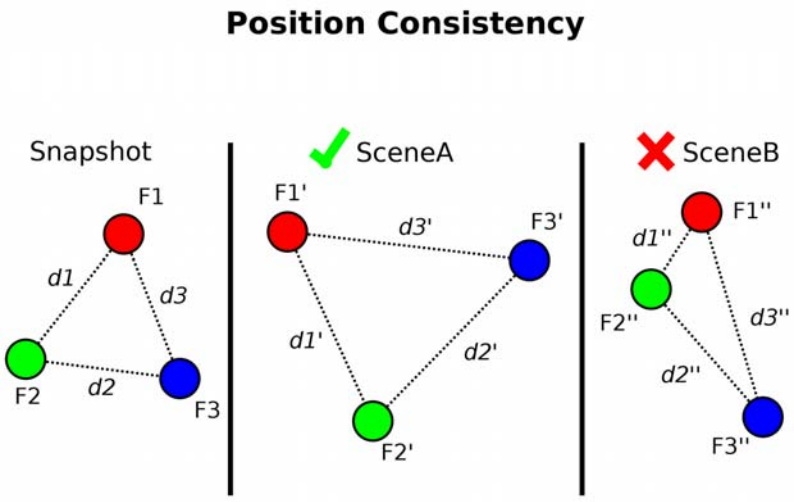

Figure 3.4.3: Features in the *Snapshot* are position-consistent with the matching features in *SceneA*, and not consistent with matching features in *SceneB.*

example in Figure 3.4.3 in the case of *SceneA*,

$$\frac{d1}{d1+d2+d3} \approx \frac{d1'}{d1'+d2'+d3'} \tag{3.4.5}$$

whereas in the case of *SceneB*,

$$\frac{d1}{d1+d2+d3} \not\approx \frac{d1''}{d1''+d2''+d3''} \tag{3.4.6}$$

It is important to note that due to perspective effects position consistency as defined above is not guaranteed to hold between two views of an object. This is because as an object is rotated out of the camera plane, the points on its surface will change their relative positions in the camera image. However, as the object is rotated out of the camera plane, the description vectors of each SIFT feature on its surface will change, since the pixel neighbourhoods around each point change. Because of this, after a few degrees of rotation the SIFT feature will change beyond recognition (or disappear), and as such multiple different snapshots of an object are required to recognize the object from different view angles. As a result, in practice position consistency will hold within some small error bound for the snapshots near the current object viewing direction.

3.4.4 Feature Orientation Consistency

Each SIFT feature has an orientation angle which refers to the direction of the dominant image gradient at the feature point. If an object is rotated the orientation of a feature on that object will also rotate, but will remain constant relative to other features on the same object (since those features are also rotated). A feature mapping has *Orientation Consistency* if the relative feature orientations between snapshot features are equal to the relative orientations between corresponding scene features. Furthermore, if we take a vector between any two snapshot features, and a vector between the corresponding matched scene features, the angle between the snapshot features' orientations and the vector between them should match the angle between the corresponding scene features' orientations and the vector between them.

This concept is shown in Figure 3.4.4. In the case of a match between the Snapshot and SceneA there are matching feature pairs $(F1, F1')$ and $(F2, F2')$; the match between the Snapshot and SceneB has matching feature pairs $(F1, F1'')$ and $(F2, F2'')$. Each feature has an associated orientation vector, for example $ori1$ for the feature $F1$. The relative orientations of the features determine that the former has orientation consistency whereas the latter does not:

$$\begin{aligned} ori1 \cdot ori2 &\approx ori1' \cdot ori2' \\ ori1 \cdot ori2 &\not\approx ori1'' \cdot ori2'' \end{aligned} \tag{3.4.7}$$

A further reason for a lack of *Orientation Consistency* for the match between the Snapshot and SceneB is that the orientation of the features in SceneB relative to the vector between the features is different to that of the Snapshot and SceneA. Let us define the following:

$$\begin{aligned} vec &= \|imgPos(F1) - imgPos(F2)\| \\ vec' &= \|imgPos(F1') - imgPos(F2')\| \\ vec'' &= \|imgPos(F1'') - imgPos(F2'')\| \end{aligned} \tag{3.4.8}$$

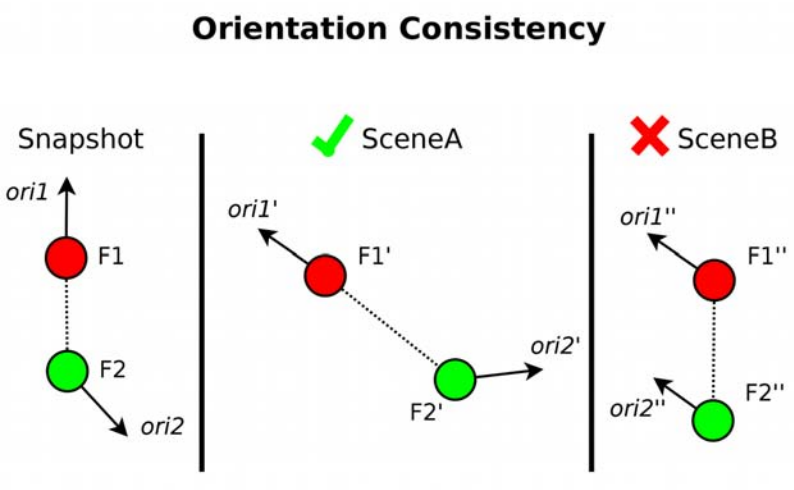

Figure 3.4.4: The matching features between the Snapshot and SceneA, $(F1, F1')$, $(F2, F2')$, exhibit orientation consistency as their feature orientations are similar relative to each other and are also similar relative to the line between them. The features in SceneB, $(F1, F1'')$, $(F2, F2'')$, do not exhibit orientation consistency relative to the Snapshot.

, from this it can be seen that in Figure 3.4.4 the following holds:

$$ori1 \cdot vec \approx ori1' \cdot vec'$$
$$ori1 \cdot vec \not\approx ori1'' \cdot vec'' \tag{3.4.9}$$

3.4.5 Finding a Consistent Feature Mapping

We define a consistent feature mapping is one that exhibits *Description Vector*, *Position*, and *Orientation Consistency*. This section will describe a method for efficiently finding a consistent mapping between snapshot and scene features.

A brute force method for finding a consistent mapping is extremely inefficient as for each snapshot there is in the order of $\frac{m!}{(m-n)!}$ possible mappings to consider for n snapshot features and m scene features. Clearly it is not feasible to simply generate every single possible mapping and iterate through to find the most consistent one.

A more efficient approach is to find a small set of feature matches that exhibit a high degree of consistency and use these as a basis to construct the remainder of

the mapping. This is possible because once a small set of feature matches is fixed, the requirement of *Position Consistency* restricts the remaining feature matches. For example, if there are three snapshot object features and matching scene features have been found for two of them, only features in a small region of the scene image need to be considered to match the last snapshot feature.

Figure 3.4.5 illustrates this concept. If two matched feature pairs $(F1, F1')$ and $(F2, F2')$ are fixed as a basis, this narrows down the search for a mapping for $F3$ to features in a small scene image region (*Search Area*), in this example containing $F3'$. Features outside of this small scene image region, in this case $F4'$, can be ignored. This is because any potential match pair with $F3$, such as $(F3, F4')$, would not exhibit *Position Consistency* relative to the other matched features.

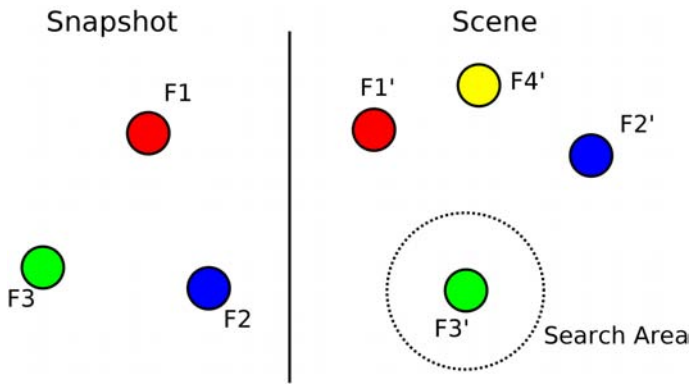

Figure 3.4.5: If $(F1, F1')$ and $(F2, F2')$ match we can narrow down the search for the matching feature for $F3$ to a small *Search Area*, ignoring features such as *F4'* outside this area, as the resulting match would not be *Position Consistent*.

The initial step to generating a consistent feature mapping is to find a small set of matching feature pairs that are highly consistent in all three respects (description vector, position, orientation), indicating a high likelihood they form part of a valid mapping. We call this the basis set of feature matches. To build the basis set, a list of all possible matching feature pairs is generated and sorted according to their SIFT description vector similarity:

$$S = [(SnapshotFeature, SceneFeature, SIFT\_similarity_i)]_{i=0}^{nm}$$
$$SIFT\_similarity_i > SIFT\_similarity_{i+1} \tag{3.4.10}$$

The list S has $n \times m$ elements (for n snapshot features and m scene features), the $SIFT\_similarity$ is the probability the features are a valid match (in the range $[0.0, 1.0]$) based on their SIFT description vector similarity presented in Section 3.4.2. The potential feature matches at the front of the list S have a higher $SIFT\_similarity$ and thus a higher probability of being valid matches.

Don't know why three until you explain it later.

The next step is to select sets of ~~three~~ matching feature pairs from the front of this sequence and calculate the overall *Consistency Score* for each set. We need a minimum of three matching feature pairs for a basis set since this is the minimum number required to determine *Position Consistency*. The overall consistency score for a set of matching feature pairs is defined as the product of the description vector, position, and orientation consistency scores; each is in the range $[0.0, 1.0]$ and is described in the following section.

3.4.6 Calculating Consistency Scores

We determine if a set of three feature matches is a potential basis set by calculating its overall *Consistency Score* and comparing this to a threshold value. The *Consistency Score* is calculated by multiplying the description vector consistency, position consistency, and orientation consistency scores of the set of matches. These are described below.

The description vector consistency score for the three matching feature pairs is their average $SIFT\_similarity$. The $SIFT\_similarity$ of each feature pair is in the range $[0.0, 1.0]$ and is the probability that the features match based purely on their description vector distance (see Section 3.4.2).

The position consistency score is based on the sum of the differences between the corresponding normalized edge lengths of the triangles formed by the three snapshot features and three scene features. Taking Figure 3.4.3 as an example, with the three feature match pairs being $(F1, F1')$, $(F2, F2')$, and $(F3, F3')$. The normalized edge

difference between the *Snapshot* and *SceneA* is equal to:

$$\left|\frac{d1}{p} - \frac{d1'}{p'}\right| + \left|\frac{d2}{p} - \frac{d2'}{p'}\right| + \left|\frac{d3}{p} - \frac{d3'}{p'}\right| \tag{3.4.11}$$

where $p = d1 + d2 + d3$ and $p' = d1' + d2' + d3'$. This sum is normalized to the range $[0.0, 1.0]$ using a Gaussian Radial Basis function, with a mean of 0.0 and standard deviation of 0.2, to give the final position consistency score. These parameters were determined empirically to give good results.

To determine the orientation consistency score of a set of three feature match pairs first consider the orientation score of two feature match pairs. Denote the two feature match pairs as (A, A') and (B, B'), where A and B are snapshot features while A' and B' are scene features. For the snapshot features calculate three values:

- angle difference (in radians) between the orientations of the features,
- angle difference (in radians) between the orientation of A and the vector from A to B,
- angle difference (in radians) between the orientation of B and the vector from B to A.

Similarly these values are calculated for the two corresponding scene features, A' and B'. The sum of the absolute difference between the corresponding values of the snapshot and scene features is the raw orientation consistency score of the two feature match pairs. The raw orientation consistency score of a triplet of matching pairs is defined as the average orientation consistency score across the three possible pairs of matching feature pairs. To calculate the final orientation score, this value is normalized to the range $[0.0, 1.0]$ using a Gaussian Radial Basis function with a mean of 0.0 and standard deviation 0.3 radians, these parameters were determined empirically to give good results.

Having calculated these three separate consistency scores (description vector, position, and orientation), we can then calculate the overall *Consistency Score* of a

particular basis set by multiplying the component scores. As each is in the range of $[0.0, 1.0]$, the resulting overall *Consistency Score* will also be in the range $[0.0, 1.0]$.

3.4.7 Generating the Feature Mapping

Given a basis set of feature match pairs, we can use this to match the remaining snapshot features to scene features. A basis set of feature match pairs contains three pairs of potentially (with high confidence) matching snapshot and scene features. This set of feature matches defines a rigid transform (described in detail below) between snapshot and scene features. We can calculate this transform, apply it to all other snapshot features, and match the transformed features to nearby scene features that have similar description vectors and orientations. In this way a basis set generates a feature mapping between snapshot and scene features. Figure 3.4.6 illustrates this concept.

However, there are many possible basis sets, each of which potentially resulting in a different feature mapping. To determine the single definitive mapping from snapshot to scene features, we consider basis sets of feature matches from the front of the ordered (by description vector similarity) list S (defined in Section 3.4.5), and only those with an overall *Consistency Score* greater than 0.8 (this threshold was found empirically to give good results). For each such basis set we generate the complete feature mapping (described in detail below). We take the definitive feature mapping to be the one with the largest number of feature matches. If the definitive feature mapping has less than a threshold number of feature matches (we use four as a minimum), then no valid matches are said to exist. This would occur for example if the target object is not present in the scene image.

Basis Set Transform A basis set of feature matches implicitly defines a rigid transform between snapshot and scene features. This transform is in the form of a translation, rotation, and scaling. We define the scaling component of this transform as the ratio between the perimeter of the triangle formed by the three snapshot

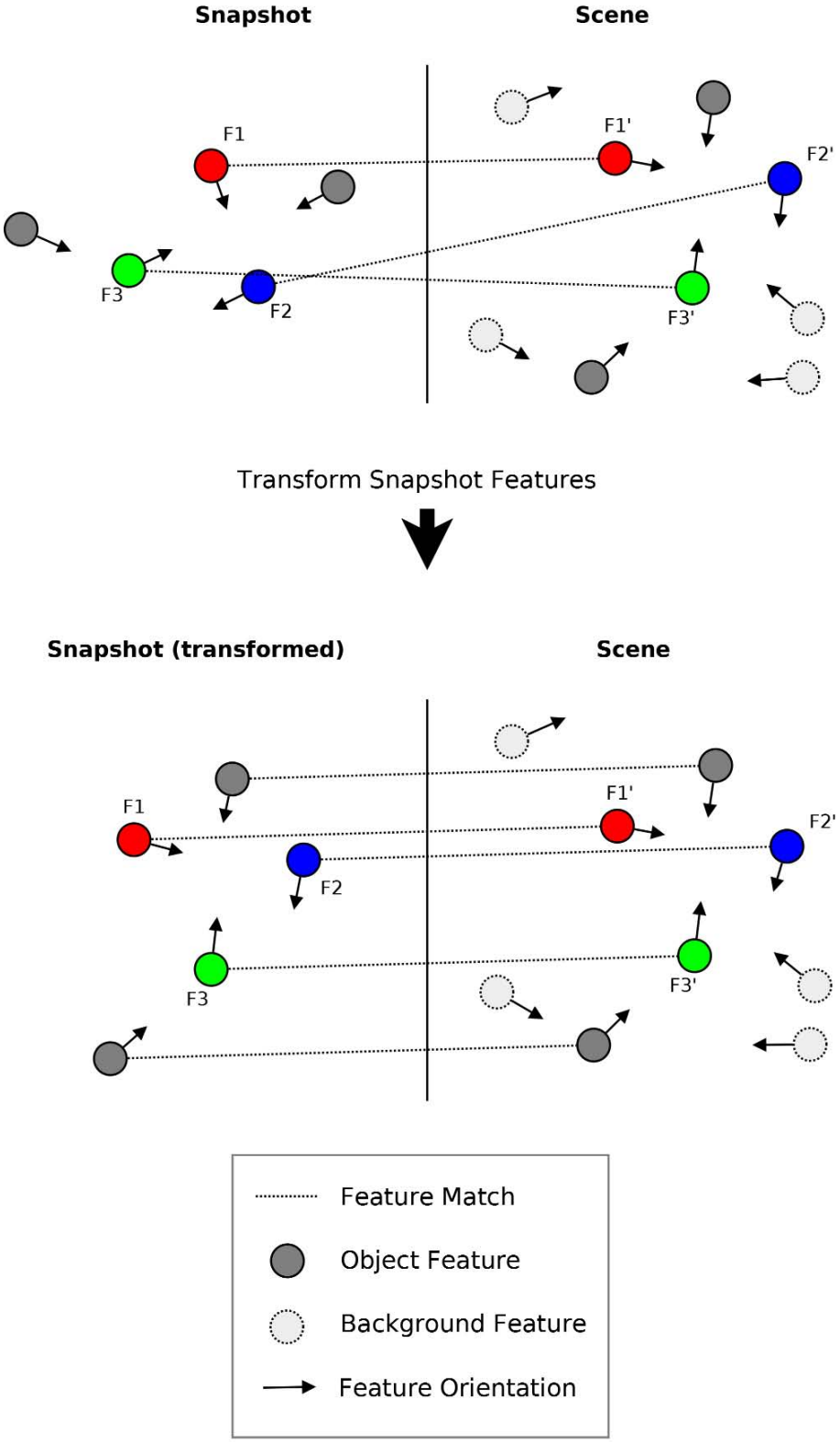

Figure 3.4.6: This diagram shows how a rigid transform can be extracted from the basis set of feature matches $[(F1, F1'), (F2, F2'), (F3, F3')]$ and then used to find the remaining feature matches. The entire snapshot is transformed by a transform derived from the basis feature matches. This then allows the remaining snapshot features to be matches to scene features that are nearby in the image and have similar descriptions and orientations.

features and the perimiter of the triangle formed by the three scene features. The translation component is calculated by taking the difference between the average of the (x, y) image positions of the snapshot and scene basis set features (centre of the triangle). The rotation component is calculated by comparing the orientatations of corresponding snapshot and scene features of the basis set. The average difference in orientation is the rotation amount. We then apply this transform to all of the features of the snapshot, scaling, rotating, and then translating the image positions of each so that they overlay the scene image features. The rotation component is also applied to the SIFT orientation of each snapshot feature.

Matching Features After all the snapshot features have been transformed, each unmatched snapshot feature (not part of the basis set) is compared against nearby scene features. We match a snapshot feature to the most *similar* scene feature within a small radius. The similarity in this case is a function of description vector and feature orientation similarity. The description vector similarity is a value in the range $[0.0, 1.0]$ and was presented in Section 3.4.2. The orientation similarity is the difference in orientation angles of each feature in radians, normalised to the range $[0.0, 1.0]$ using a Gaussian Radial Basis function with mean 0.0 and standard deviation 0.3 radians. The overall similarity is the product of the two values, and is therefore also in the range $[0.0, 1.0]$. We match a snapshot feature only to scene feature that have a similarity score of more than 0.7 (this value was chosen empirically to provide good results). This matching process is detailed in Algorithm 3.2.

3.5 Feature Database Matching Efficiency

So far we have discussed matching a single database snapshot of features to a set of scene features. The run-time complexity of matching a single snapshot to a scene is at least $O(nm)$ for n snapshot features and m scene features. This stems from the need to generate the list of size $n \times m$ of all possible feature match pairs, and

Algorithm 3.2 Generating full feature match from a basis set.

input: set of snapshot features $\rightarrow S$
input: set of scene features $\rightarrow C$
input: transform from snapshot to scene basis features$\rightarrow t$

$matches \leftarrow \{\}$
forall s **in** S
  $s \leftarrow applyTransform(t, s)$
  $near\_features \leftarrow getNearSceneFeatures(C, s)$
  $best\_similarity \leftarrow$ inf
  $best\_match \leftarrow null$
  forall n **in** $near\_features$
    $similarity \leftarrow calculateSimilarity(s, n)$
    if $similariy > 0.7 \wedge similarity > best\_similarity$ **then**
      $best\_similarity \leftarrow similarity$
      $best\_match \leftarrow n$
    endif
  endfor
  if $best\_match \neq null$ **then**
    $matches \leftarrow matches \cup \{(s, best\_match)\}$
  endfor
endfor

output: all match pairs $\leftarrow matches$

Where does this come from?

for each match pair we must compute the SIFT description vector distance. For a database with s snapshots, matching a scene with no *a priori* knowledge of its composition, ~~would have~~ has a computational complexity of $O(nms)$. This compares poorly with the matching performance of the standard nearest neighbour approach. For this approach, if a k-d tree space partitioning structure is used, a complexity of $O(n \log ms)$ is possible. However, our approach matches a single snapshot independently of all others, without impacting matching accuracy. The nearest neighbour approach, on the other hand, relies on the second-nearest neighbour distance to perform rejection of spurious SIFT matches. In this case, the accuracy of feature matching is dependent on the contents of the entire database.

You over use the word "approach" everywhere.

Take a scenario in which the content of the scene is known, or at least it is known that only a small subset of all learned objects may be in the scene. An example of such a scenario is processing a video stream. Due to temporal coherence we may be able to assume that the current frame can only contain the objects present in the

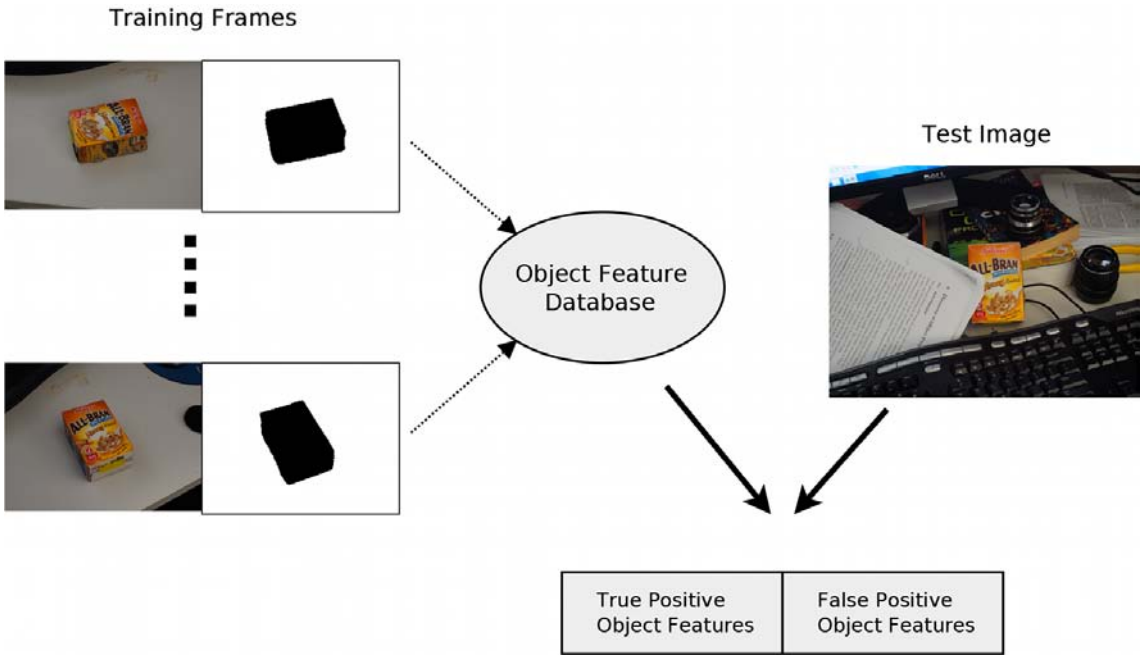

Figure 3.6.1: Schematic representation of the method of testing different feature matching approaches. We add to a feature database training snapshots of the object. Then we use this database to perform object recognition on a test image of the object in a scene, recording the number of correctly and incorrectly identified object features.

previous frame, only checking for the appearance of new objects intermittently. In such a scenario, using our approach, we could match only the snapshots of objects known to be present in the frame, ignoring all others. This is not possible to do using ~~the~~ a nearest neighbour ~~approach~~ algorithm without affecting matching accuracy. In the experimental results section (Section 3.6) we demonstrate the speed-up possible if the contents of the scene can be narrowed down.

3.6 Experiments and Results

We test the effectiveness of the presented feature matching method by comparing it to the nearest neighbour approach. First we test the feature matching accuracy in the context of object recognition. We build a database of object features using a training set of images, followed by using the generated database to match object features in scene images. This process is summarised in Figure 3.6.1.

The reference nearest neighbour feature matching implementation is Hess's[1] Best-Bin First kd-tree implementation for matching SIFT features, with the *KDTREE_BBF_MAX_* parameter set to 5000. When testing the performance of the nearest neighbour approach, we also add to the database 2000 features extracted from random images with no parts in common with the test object images. This is required because the nearest neighbour approach relies on the second-nearest neighbour feature to provide the rejection threshold for spurious matches (discussed in Section 3.3).

The test data set is comprised of images of 17 different objects in a cluttered scene under three different lighting conditions with four images per lighting condition, for a total of 204 images. For each image a mask is available that specifies which areas of the image are object and which are background. This in turn indicates which features in the image are object features and which are background. The objects used were flat faced rectangular prisms with an image on the faces. The camera used was a Bumblebee2 stereo camera (only the left camera image is used) outputting a colour image with resolution 512x384.

For each object 12 images are available, we separate these images into a training set and a test set. In total there are 4094 ways in which 12 images can be split in this way. For each pair of training and testing sets the object features are extracted from the training set of images and inserted into an object feature database. The database is then used to match the test image features to object features, using the nearest neighbour method and the Bipartite Matching method. It should be noted that no further geometric consistency check is performed for either method after the features are matched, as it is not our intention to test the effectiveness of an algorithm such as RANSAC or the Hough Transform.

The percentage of correct and incorrect matches is tabulated according to the number of images in the training set, and averaged over all objects. The true positive match results are shown in Figure 3.6.2, while the false positive match results (background features incorrectly matched to object features) are shown in

[1] http://web.engr.oregonstate.edu/~hess/

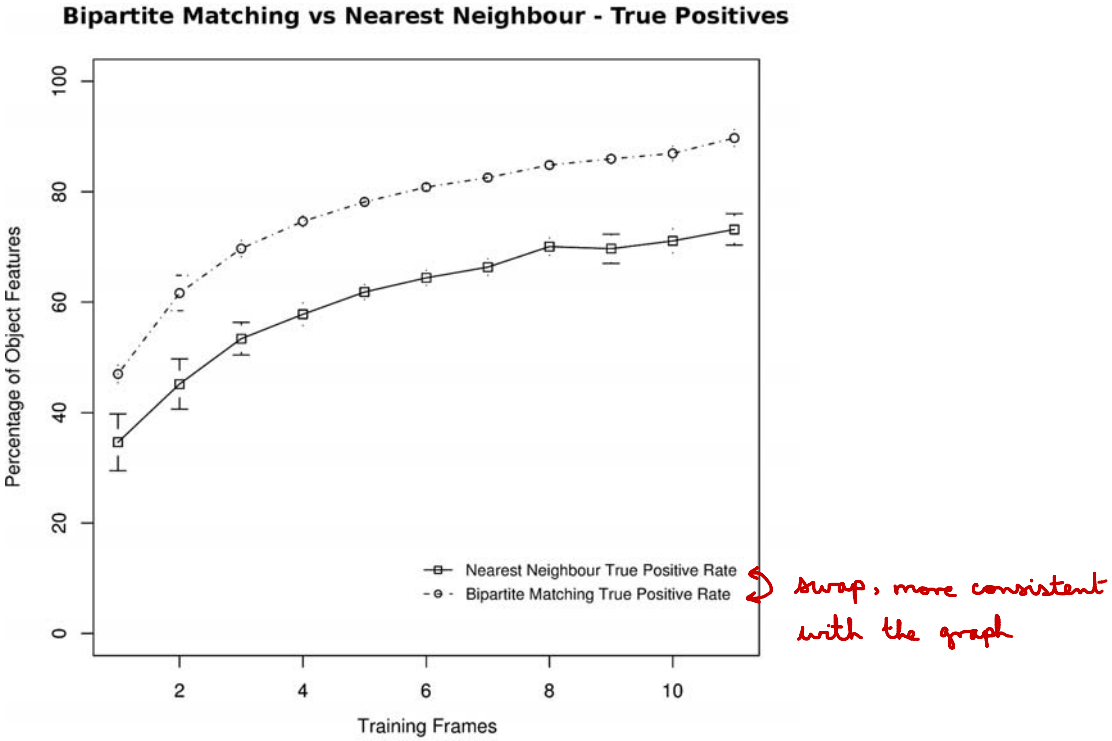

Figure 3.6.2: Percentage of object features correctly detected by the nearest neighbour and Bipartite Matching approaches. The performance of each approach is graphed against the number of object training images used to generate the database of object features. The error bars indicate the Standard Error of the result data.

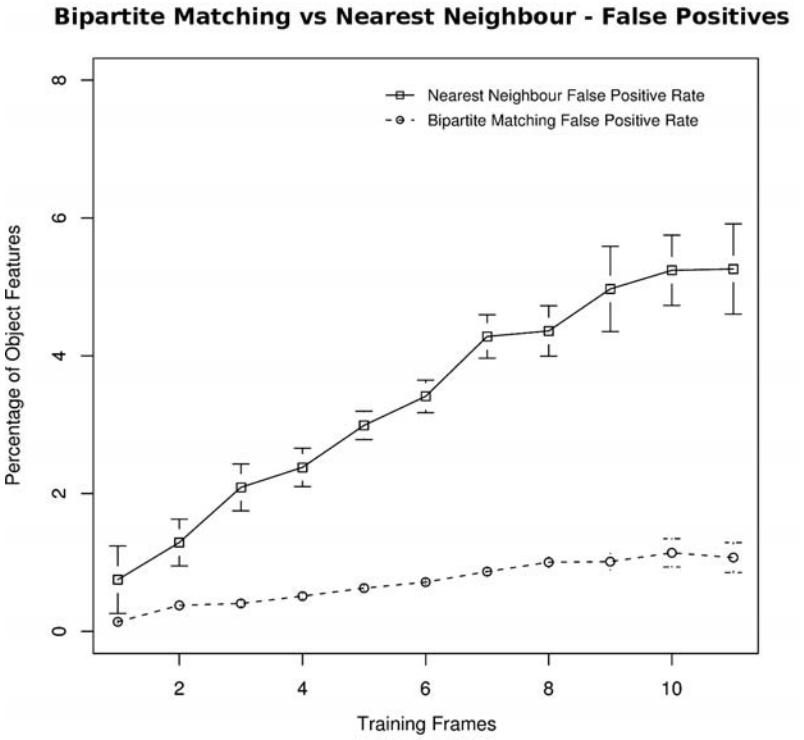

Figure 3.6.3: Number of background features incorrectly classified as object features by the nearest neighbour and Bipartite Matching approaches, expressed as a percentage of object features. The performance of each approach is graphed against the number of object training images used to generate the database of object features. The error bars indicate the Standard Error of the result data.

Figure 3.6.3. These results show that our matching scheme has a higher feature matching rate across all training set sizes, while at the same time having a lower false positive matching rate.

In addition to testing the matching performance, we also compared the matching speed of our algorithm against the Best-Bin-First nearest neighbour method. We recorded the time in milliseconds to perform a feature match for a given feature database size (database size is defined as the total number of features stored). The match time is calculated by taking the time to match a scene image, divided by the number of features in the scene image. The feature database is composed of features from many different objects. We tested two different scenarios, in the first the scene content is unknown and it may contain any of the learned objects, in the second the scene content is known to contain only one of the learned objects. In the latter scenario, using the Bipartite Matching approach we can realise a significant speed-

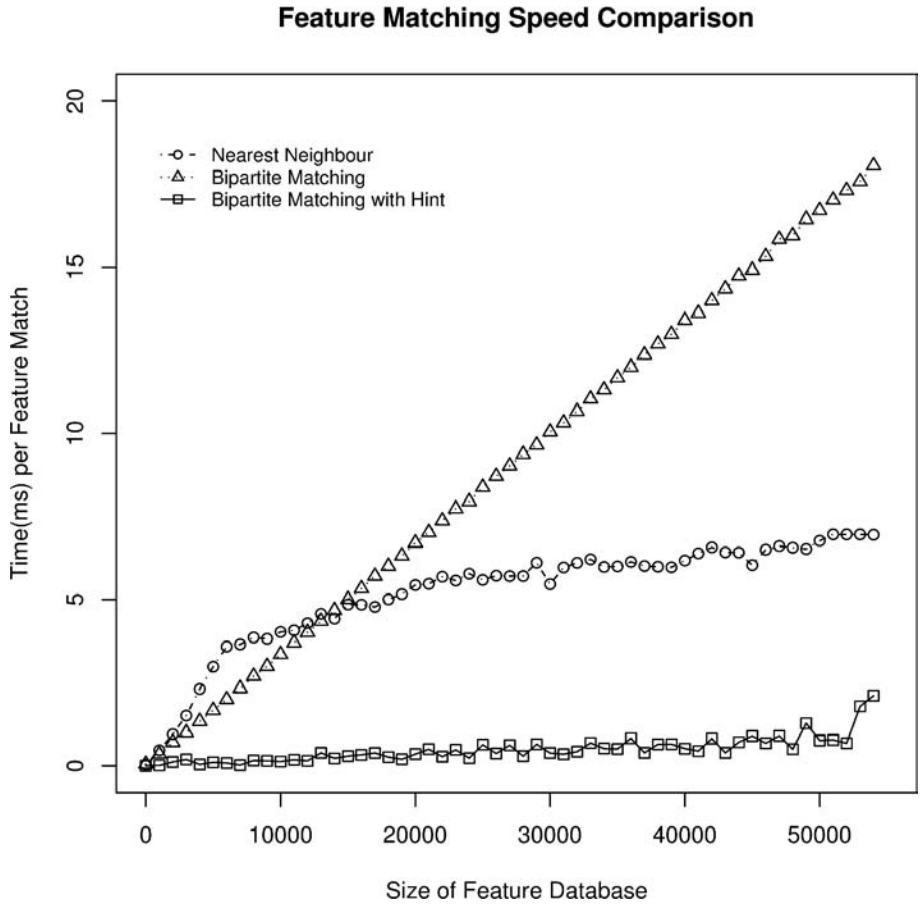

Figure 3.6.4: Comparison of matching speeds between a database using the Nearest neighbour scheme (Best-Bin-First) and one using the Bipartite Matching scheme. The Bipartite Matching scheme is considerably slower when no information about the scene being matched is known. However, if a hint is given regarding the contents of the scene, the Bipartite Matching scheme is extremely efficient, much faster than the Nearest neighbour method.

up by not matching any of the snapshots of objects known to not be in the scene. The results are presented in Figure 3.6.4. It can be seen that the time complexity of our algorithm is linear with the size of the database when no information regarding the scene is available. However, if we specify the object in the scene with a hint, our Bipartite Matching method is much more efficient compared to the nearest neighbour approach.

3.7 Discussion

Our method for feature matching has several demonstrated advantages over existing methods in terms of both accuracy and speed. The accuracy advantage stems from the fact that the Bipartite Matching method takes into account feature position and orientation as well as the description vector at the matching stage. Nearest neighbour and distance threshold methods perform feature matching using only the feature description vector and then as a separate stage remove geometrically inconsistent matches.

In a situation where the object to be recognized has a regular pattern texture, such as a checkerboard pattern for example, the nearest neighbour method may perform very poorly. This is because the object has many features which have very similar description vectors, and when matching scene features to database features either very few features will have a nearest neighbour closer than 80% distance to the second nearest neighbour, or many of the feature matches will be incorrect and removed in the geometric consistency stage.

The presented Bipartite Matching method, on the other hand, does not restrict itself to matching only the nearest neighbour features or features under a certain threshold distance. Instead a holistic approach is used, taking into account multiple properties to determine an appropriate feature match.

The computational complexity of the nearest neighbour method can be a problem, due to the difficulty of nearest neighbour search in a high dimensional space (128 dimensional in the case of SIFT features). Furthermore, the database of features cannot be pruned based on scene knowledge without affecting matching accuracy. This is because we are searching for both a nearest neighbour for a scene feature, as well as its second-nearest neighbour. As a result, the nearest neighbour matching scheme cannot take advantage of scene knowledge which may be available in some situations and applications.

In the case of the presented matching method, the database stores object features

grouped into individual views of the object. The advantage of this is that each set of snapshot features is independent. This means that if there is prior knowledge of the objects in the scene, we can focus only on matching the snapshots of the present objects, ignoring the rest. Furthermore if the views for each object are arranged in an aspect graph, such that it is possible to know if two views of an object are from similar orientations, if we have prior knowledge of the orientation of an object in a scene we can focus our search on the views for that given orientation. ~~If the task is to track~~ For tracking objects in a scene, if an object was present in the scene in a frame it will likely be present in the next few frames. Similarly, if an object was in a particular orientation in a frame it will most likely be in a similar orientation in the next few frames. In such a scenario the Bipartite Feature Matching method can exploit the temporal and spacial coherency and be significantly faster than a nearest neighbour feature matching scheme.

First time you mention this. Define with reference

3.8 Future Work

~~The~~ Our ~~presented~~ Bipartite Feature Matching algorithm has significant scope for development in terms of parameter optimisation. One aspect that can be developed is finding differ ers used. It may be possible to devise a scheme in which different parameters and thresholds are used for different applications. Depending on factors such as scene clutter, noise, and lighting conditions, we may wish to use different values for parameters in match consistency calculation, different thresholds for minimum consistency of basis sets, etc.

For this to make sense, you have to say more about the sensitivity of parameters.

In the current approach we assume a rigid affine transform between the learned snapshot of object features and the scene object features. We extract a potential transform from a basis set of feature matches and apply it to the remaining snapshot features. These transformed features are then matched to nearby scene features to construct the complete match. It may be more accurate to model the transformation

between snapshot and scene features as a perspective transform. To do this, however, we would need to expand the basis set of feature matches to include four match pairs (the minimum required to fit a perspective transform).

The presented algorithm has been developed with SIFT features in mind. However, it should be applicable to other local interest point detectors such as SURF[46] and PCA-SIFT[45]. Adapting this matching method to a different type of local image feature may be relatively straight forward. For SIFT-like detectors, it may be as simple as modeling the given feature type's description vector to be able to convert between a description vector distance between two features to a match probability.

A further extension of the presented method is to apply it to 3D stereo images. We have so far dealt with single-camera images but we could extend this algorithm to stereo-camera images. In the case of a stereo-camera we could match corresponding features between the left and right camera image, resulting in stereo SIFT features which have a 3D position determined by the camera parameters. When learning to recognize an object the features saved in the database would be stereo features which have 3D position information. Then to recognize and localize an object in a scene we would perform Bipartite Matching between stereo features, with an appropriate modification to position consistency to take into account the 3D as opposed to 2D position of each feature. The advantage of this approach would be a possible improvement in matching accuracy since our position consistency would be based on 3D coordinate data and the 3D localization of the object in the scene is a direct result of the feature matching process.

Chapter 4

Feature Segmentation for Object Recognition

4.1 Introduction

In the previous Chapter we presented a method for matching SIFT features from a learned object database to scene image features. This allows a robot to recognise and localise an object in a scene, which is vital for many robotics applications, as it allows the robot to effectively track, grasp, and then manipulate the object.

A requirement for object recognition and localisation is a model of the object's appearance. The nature of the model is dependent on the recognition algorithm used. In the case of the previously described SIFT matching algorithm, the model is in the form of a database of SIFT features. The appearance model is typically extracted from training images of the object. These training images contain a view of the object, with either a plain background or a segmentation mask that defines the regions of the image belonging to the object. Such images can be generated in a controlled environment with a plain background [49] or with the aid of a human operator performing the background segmentation.

There are some applications in which it is not possible to provide training images of an object *a priori*. In the case of an autonomous robot deployed to an

It only needs to be new, not unpredictable

unpredictable environment (for example a household service robot), it is impossible to predict every object that the robot may need to interact with and hence recognise. A solution to this problem is for the robot to be able to autonomously acquire training data of an object in its environment with no human intervention.

In this chapter we present a system for a robot to determine training data for a previously unseen object in the presence of clutter, noise and background motion. Our vision system for object recognition and localisation is based on SIFT features. As a result, the specific problem that we solve is segmenting the SIFT features of an object from the background, in a complex and cluttered environment (an example of this problem is shown in Figure 4.1.1). These features then form snapshots in an object feature database, that can be used for recognition and localisation of the object in a scene at a later time.

Our approach to this problem is focused on obtaining as much information as possible about individual image features, which is then used to determine whether a feature belongs to the object, to the robot arm, or to the background. This is done by having the robot grasp the object and performs a series of moves, while stereo vision and feature tracking are used to gather trajectory data for each feature over multiple frames. By using robot induced object motion and tracking stable image features over many frames, we can effectively separate object and background features in a dynamic, highly cluttered environment. These features can then be used for building an all-aspect object appearance model (discussed in Chapter 5), allowing recognition, localisation, and effective manipulation of the object by the robot.

The outline of this chapter is as follows:

1. a brief overview of related approaches and their corresponding weaknesses (Section 4.2);

2. a detailed description of the feature segmentation method (Section 4.3);

3. experimental results demonstrating the effectiveness of the presented method

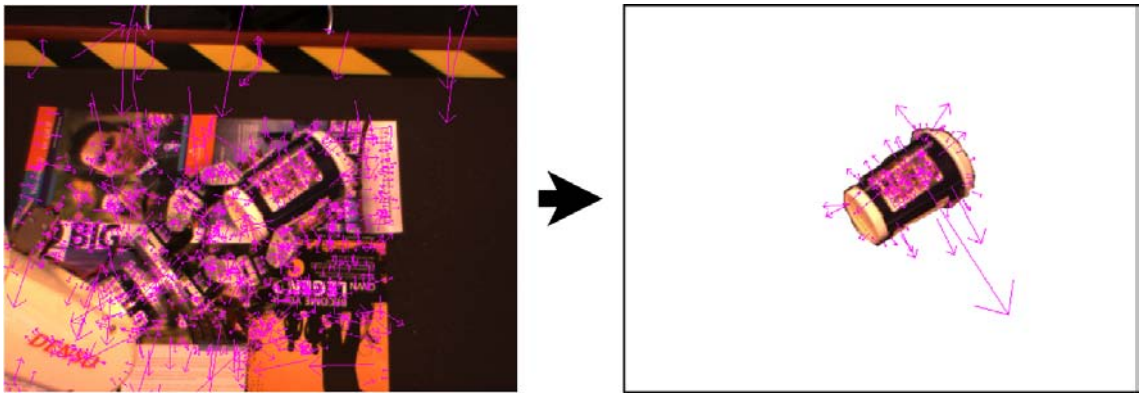

Figure 4.1.1: A robot arm gripping an object in a cluttered environment. The purple arrows represent SIFT features. Our goal is to separate out the object features, which can later be used for recognising and localising the object in a scene.

(Section 4.4);

4. a discussion of the results and of the presented algorithm (Section 4.5);

5. an overview of avenues for future work (Section 4.6).

Current Methods

4.2 Existing Approaches

The problem of learning a new object's appearance model can be viewed as a segmentation problem. Previous approaches to image and feature segmentation ~~are~~ were presented in detail in Chapter 2. In this section we give a brief overview of the most relevant techniques.

In a cluttered and complex environment and target object, static image analysis may not be effective at separating the object image regions or features from the background. An alternate approach is to perform dynamic scene analysis. Background subtraction [60] can be used. This involves having the robot observe the scene without the object in place, and build an image model of the background. The object is then placed in the scene, and to determine the object image regions, the previously learned background model is subtracted from the resulting image. The areas of the image that correspond to the object will not

Did you say this before?

match the learned background model and will be labelled as foreground object regions. This object image region can then be used to learn an appearance model, such as object SIFT feature snapshots.

Another approach is to use object motion to separate its image region from the background. One example [67] involves placing the object in a scene and nudging it with the robot gripper. As the object moves as a result of the contact, there is an increased amount of detected motion in the image stream. This burst of motion can be used to segment the object image region from the background using the min-cut algorithm [55, 69].

These approaches have the advantage of being able to deal with a cluttered background and a complex object. However, in the presence of background motion, these algorithms may not perform well. This is because they only use instantaneous motion rather than long term motion to determine which image regions correspond to the object. To effectively cope with background motion, we propose that robot induced object motion and long term tracking of feature trajectories is required to build a model of each feature and use it to differentiate between background motion and object motion, as well as separating out a cluttered background.

4.3 Feature Segmentation Algorithm

4.3.1 Overview

The purpose of our algorithm is to allow a robot to autonomously find object image features in a dynamic unstructured environment. These features can later be used for object recognition and localisation. We do this by using the robot gripper to grasp and move the object through a scene, recording this motion with a stereo camera, extracting long-term feature trajectories from the resulting video stream, and finally using this data to extract the object features. The challenges we seek to addresses are:

- separating object features from a cluttered background;
- separating object and robot gripper features;
- robustness to background motion;
- robustness to changes in the visual appearance of the robot arm.

The basis for our algorithm is to use motion to separate the features of a target object from background features. We track the motion in 3D world space rather than in 2D image space as this provides more information to determine whether ~~any given~~ feature belongs to the background or the object. The use of SIFT features, that can be reliably correlated from frame to frame, allows a feature to be tracked through multiple frames. This enables long term tracking of features trajectories. This in turn provides more information for segmentation as compared to instantaneous motion methods [67].

The basic outline of our approach is as follows: the robot grasps the object and moves it through the scene in a linear motion (keeping the same aspect of the object facing the camera), during this motion the robot records a video stream from its stereo camera. For every frame we extract SIFT features from the left and right camera images and correlate them to form 3D SIFT features. These features are tracked ~~between frames~~ to form feature trajectories. A single trajectory is a series of corresponding 3D SIFT features that should be located on the same point on a surface in the scene. Every few frames we take a snapshot ~~which involves using~~ the feature trajectories to determine which of the 3D SIFT features belongs to the grasped object and which belong to the background or robot arm. The object features from each snapshot are stored in an object feature database.

~~In this section we describe~~ The ~~different~~ stages of our algorithm ~~these~~ are:

1. extracting and correlating SIFT features from the stereo video stream (Section 4.3.2),
2. tracking the trajectory of each feature during the motion (Section 4.3.3),

3. periodically extract snapshots of object features when sufficient trajectory data is available (Section 4.3.3),

4. separate arm and object features (Section 4.3.4),

5. filter out background motion by ignoring features whose motion differs from the trajectory of the arm (Section 4.3.5),

6. finally we improve the separation of arm and object features by comparing the feature snapshots from different objects and remove matched features (Section 4.3.6).

The platform we used for testing our algorithm consists of a Bumblebee2 (see Figure 4.3.1) stereo camera (resolution of 512×386 pixels at 15 frames per second is used) and a 6 degrees-of-freedom (DOF) industrial arm with a two fingered gripper attached (see Figure 4.3.2). The fingers consist of two servos per finger and silicone pads on the tips to increase friction when grasping objects. A workspace is accessible in front of the robot (see Figure 4.3.1).

Our approach assumes that the robot already has the object in its grasp when it begins to learn the object's appearance model. We do not go into detail on grasping an object with the robot gripper, nor how an object can grasp a new, previously unseen object. The approaches to this problem depend on the specific robot application. For example, in a household service application a person may hand the object to the robot. If the object is placed on a table, the robot could use the height variation from the flat surface as a cue to indicate that there may be an object that can be picked up.

You MUST give details !!!

4.3.2 Stereo Feature Generation

Our overall approach is based on gathering as much data about each feature as possible to determine if it belongs to the held object. The first step is to determine the 3D world position of each feature.

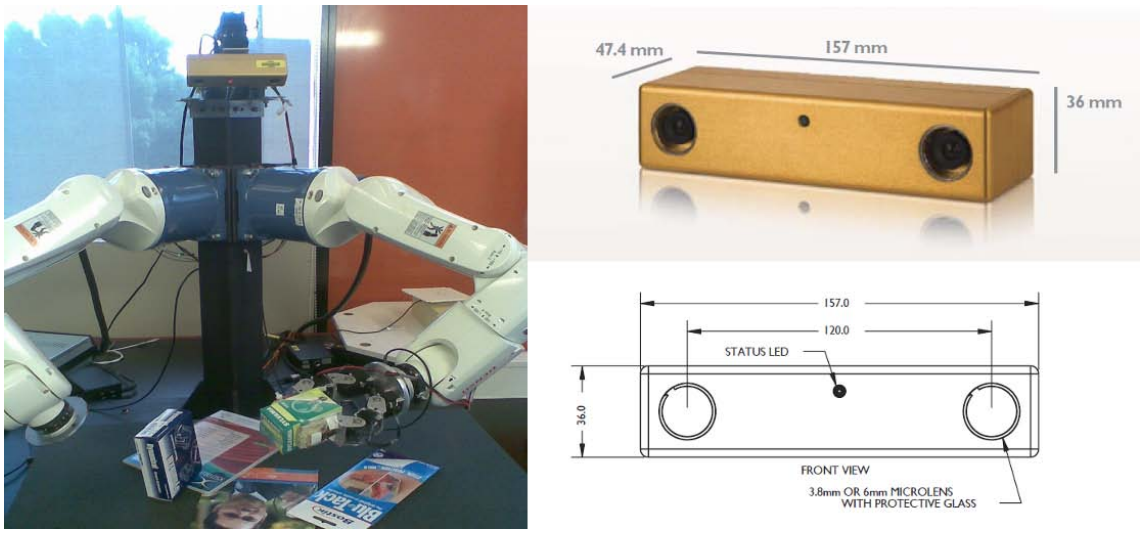

Figure 4.3.1: Robot and workspace setup on the left, the Bumblee2 stereo camera on the right. The robot has a stereo camera on a pan-tilt unit and a 6-DOF arm with a gripper attached. Bumblebee2 schematics from: http://www.ptgrey.com/products/bumblebee2/bumblebee2_xb3_datasheet.pdf

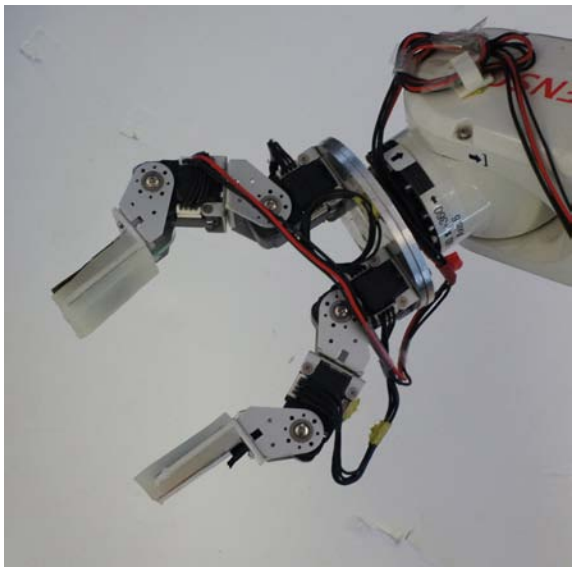

Figure 4.3.2: Two fingered gripper. The gripping pads are constructed from silicone around flat metal plates. Each finger consists of two servo-driven joints.

A single SIFT feature describes a stable interest point in a 2D image, and is characterised by a highly discriminatory 128 dimensional description vector and an image gradient orientation at the feature point. A SIFT feature is a function of a small pixel neighbour[illegible] turn is a function of the appearance of the scene in the corresponding area. This scene area, when viewed from a two slightly different positions, should result in similar image patches, and hence generate similar SIFT features. Using this property, the stereo camera equipped robot, can correlate SIFT features from the left and right scene images and determine the 3D world position of each using the camera parameters (baseline distance, field of view, and image resolution). Each correlated pair of SIFT features from the left and right image is designated a *Stereo Feature* and is a tuple of the form $(LeftFeature, RightFeature, WorldPosition)$.

You've said this many times already.

To find the *Stereo Features* for a camera frame, we extract the SIFT features for both the left and right images and match them by comparing their image positions, description vectors and orientation. For each feature in the left image we search the right image for a feature on the same epipolar line with the closest SIFT description vector by Euclidean distance. In the case of the Bumblebee2 stereo camera geometry, the epipolar contraint simply means that any matching features in the left and right image must have the same y image coordinate. If ~~this pair of~~ the left and right image features have their orientations and description vectors within a threshold distance they are said to match and form a new *Stereo Feature*. The justification for this is that the same point on an object's surface should produce similar SIFT features, in terms of both description vector and orientation, in both the left and right images. Algorithm 4.1 ~~provides~~ gives an overview of the process to match the SIFT features from the left and right image to form the *Stereo Features* for a frame.

The performance of this algorithm is dependent on the choice of threshold for the orientation and description vector difference between the left and right image features (10° and 350 respectively in Algorithm 4.1). Because the left and right images are concurrent views of the scene, with the same lighting conditions and

Algorithm 4.1 Finding stereo features.

input: left image SIFT features $\rightarrow leftFeatures$
input: right image SIFT features $\rightarrow rightFeatures$

$stereoFeatures \leftarrow \{\}$
forall $lFeature$ **in** $leftFeatures$
  $rightEpipolarFeatures \leftarrow epipolarFilter(rightFeatures, lFeature)$
  $rFeature \leftarrow getDescriptionVectorNearest(rightEpipolarFeatures, lFeature)$
  $oriDist \leftarrow orientationDistance(lFeature, rFeature)$

  if $oriDist \leq 10°$ **then**
    $featureDist \leftarrow featureVectorDistance(lFeature, rFeature)$
    if $featureDist \leq 350$ **then**
      $worldPos \leftarrow findPosition(lFeature, rFeature)$
      $newFeature \leftarrow (lFeature, rFeature, worldPos)$
      $stereoFeatures \leftarrow \{newFeature\} \cup stereoFeatures$
    endif
  endif
endfor

output: scene stereo features $\leftarrow stereoFeatures$

perspective, we consider a simple threshold as sufficient for stereo feature matching; as opposed to using a more more complex threshold or matching method as described in Chapter 3. Give the specific section

Again, how do we know if the thresholds only work for your environment?

We determined suitable values for the two thresholds empirically. This is done by finding corresponding SIFT feature pairs in a large number of stereo images and recording the distance between their description vectors and orientation difference. We found the mean orientation difference to be 3.7° with a standard deviation of 3.1°. We set the *orientationThreshold* to be mean plus three standard deivations ($\overline{x} + 3\sigma$), which is equal to 10°.

To determine the description vector threshold we consider the percentage of corresponding stereo feature pairs with a description vector distance less than x : $\{0.0 \leq x \leq 700.0\}$ (we found that none of the feature desciption vector distances exceeded 700.0), as compared to the distance between non-corresponding feature pairs. ~~This~~ These data ~~is~~ are summarised in Figure 4.3.3. We set the feature description vector threshold to 350.0 as we found that 95% of correctly corresponding stereo feature pairs have description vectors with a distance less than this, while only 1.5%

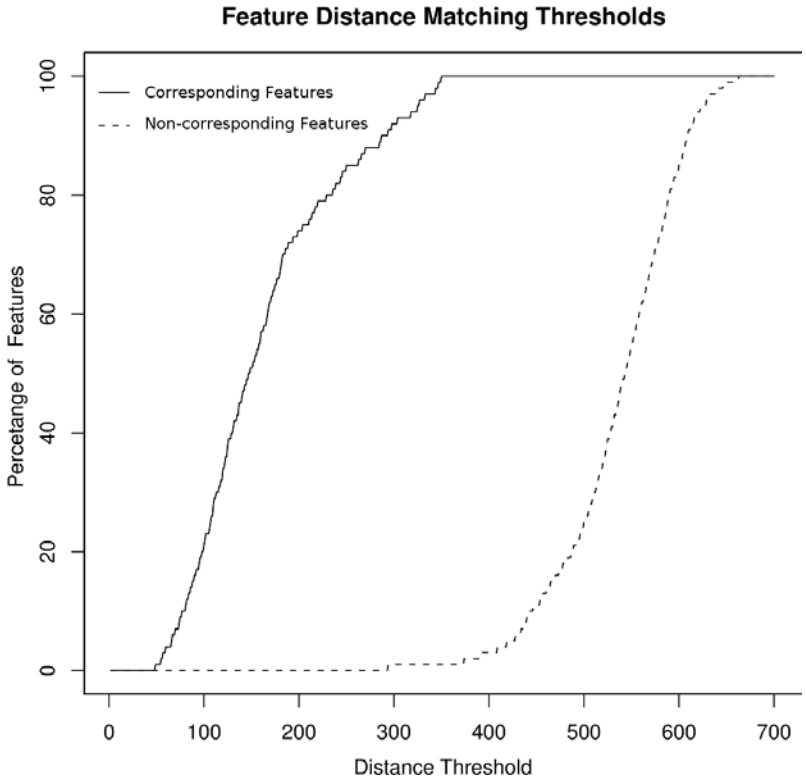

Figure 4.3.3: This shows the percentage of corresponding and non-corresponding SIFT stereo feature pairs with description vectors within a distance threshold.

of non-corresponding feature pairs fall within this threshold.

Evalutating these threshold parameters we found that the rate of incorrect feature correspondence was 1.3%. These thresholds should be applicable to a variety of objects and environments since SIFT features are not dependent on the overall scene composition and are robust to lighting changes. However, cameras of varying quality, and baseline length may require different parameters.

This statement has to be justified

4.3.3 Feature Tracking and Snapshotting

After generating the *Stereo Features* for a frame, the next step is to track the features, dealing with intermittent feature visibility and motion of the features in 3D space. This is done by maintaining a set of feature trajectories, each of which is a series of $(Feature, DetectTime)$ tuples. $DetectTime$ refers to when the given feature was detected and added to the trajectory. The features that make up a trajectory should correspond to the same point on the surface of an object in the scene. The purpose

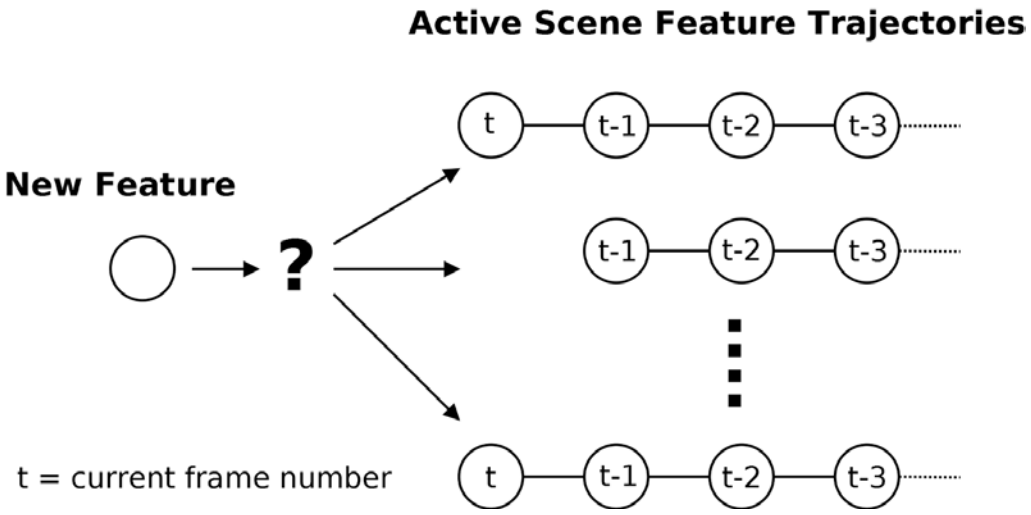

Figure 4.3.4: Every feature from a new image must be inserted into a matching active trajectory, or a new trajectory created if none match. A trajectory is a list of corresponding *Stereo Features* from previous frames.

of keeping track of the feature trajectories is to determine which features are part of the background and which are part of the object held by the gripper.

We maintain a set of active feature trajectories. These are trajectories that have had a new feature added within the last 3 frames (which is 200 milliseconds as we use a video stream of 15fps). Each *Stereo Feature* of a new camera frame is compared to every active trajectory and inserted into the best match (this is illustrated in Figure 4.3.4). If none ~~are~~ is a sufficiently close match, a new trajectory is created containing that feature and is then added to the set of active trajectories.

The reason for maintaining a list of active trajectories (which have been updated in the last 3 frames), rather than a list of all trajectories, is our reliance on spatial locality and descriptor locality for matching a new *Stereo Feature* to a trajectory. We found that in our case, trajectories that have not been updated for more than 3 frames do not contain sufficient locality for reliable feature matching.

To determine if a new feature should be inserted into an existing trajectory, we compare the description vector, orientation and 3D world coordinates of the new feature and the most recent feature of the trajectory. Successive features in a trajectory should have spatial locality in 3D world space, similar feature description vectors and orientations. This is because they should be located on the same point

Algorithm 4.2 Inserting new stereo feature into a matching trajectory.

input: new stereo feature $\rightarrow s$
input: active feature trajectories $\rightarrow Trajectories$

$matchingTrajectory \leftarrow null$
forall t **in** $Trajectories$
  $(tFeature, tDetectTime) \leftarrow latestFeature(t)$
  $worldPosDist \leftarrow getWorldPosDist(s, tFeature)$
  if $worldPosDist \leq (curTime - tDetectTime) \cdot ArmSpeed$ **then**
    $descriptorDist \leftarrow SIFTDist(s, tFeature)$
    $orientationDist \leftarrow oriDist(s, tFeature)$
    if $descriptorDist \leq 250 \wedge orientationDist \leq 5°$ **then**
      $matchingTrajectory \leftarrow t$
    endif
  endif
endfor
if $matchingTrajectory \neq null$ **then**
  $matchingTrajectory \leftarrow (s, curTime) : matchingTrajectory$
else
  $Trajectories \leftarrow [(s, curTime)] \cup Trajectories$
endif

output: updated feature trajectories $\leftarrow allTrajectories$

of an object. For every new *Stereo Feature* we take all trajectories that have their most recent feature within a threshold distance in 3D world space. The distance threshold is:

$$(CurrentTime - TrajectoryTime) \cdot ArmSpeed \tag{4.3.1}$$

$ArmSpeed$ is the maximum arm movement speed, $CurTime$ is the current time, and $TrajectoryTime$ is the time-stamp when the particular trajectory was last updated with a new feature. The justification for this threshold is that no feature on the grasped object can move faster than the arm, and it is not necessary to track features of faster moving objects in the background.

From this set of nearby trajectories we find the one with the closest (by Euclidean distance) [illegible] is less than 250 and the orientation difference between the features is less than 5° then the new

You say more about these parameters later, but when you encounter then here, they are mysterious. Move the explanation up.

feature is inserted into this matching trajectory. The process for matching a single feature to a trajectory is described in detail in Algorithm 4.2.

The description vector and orientation thresholds are determined by examining the variability of the description vector and orientation of features during linear movement (movement such that the underlying object remains in the same orientation in the image). We plot the average description vector distance and orientation difference between a feature and its corresponding match in each of the previous 30 frames; this is shown in Figure 4.3.5. After 3 frames the average distance between description vectors in a trajectory is 100 with a standard deviation of 50, and the mean orientation difference is 1.3° with a standard deviation of 1.2°. We chose the thresholds of 250 and 5° by taking the mean and adding three standard deviations ($\overline{x} + 3\sigma$) for both the feature descriptor and orientation difference.

If for a new *Stereo Feature* no active trajectory is found matching the above criteria, an empty trajectory is created and the feature is inserted into this new trajectory.

The purpose of tracking feature trajectories is to use motion to separate foreground object features from the background, which then form a feature database used for object recognition. While the object is being moved by the robot, we periodically take snapshots of the most recent features of trajectories that have a displacement over a certain threshold. The displacement of a trajectory is defined as the sum of the distance between the 3D world positions of subsequent features of the trajectory. The frequency of feature snapshots is dependent on several factors: the camera frame rate, speed with which the robot moves the object, and the desired level of detail for the learned object feature database. In our case, we perform a snapshot every 5 frames. The snapshotted trajectories are ones that have moved in 3D world space at least $(FramesSinceLastSnapshot - 1) * ArmSpeed$ since the last snapshot. These trajectories should correspond to the robot arm, the held object, and any background motion. Any static background features should belond to trajectories which have a displacement under the threshold, and hence will not be

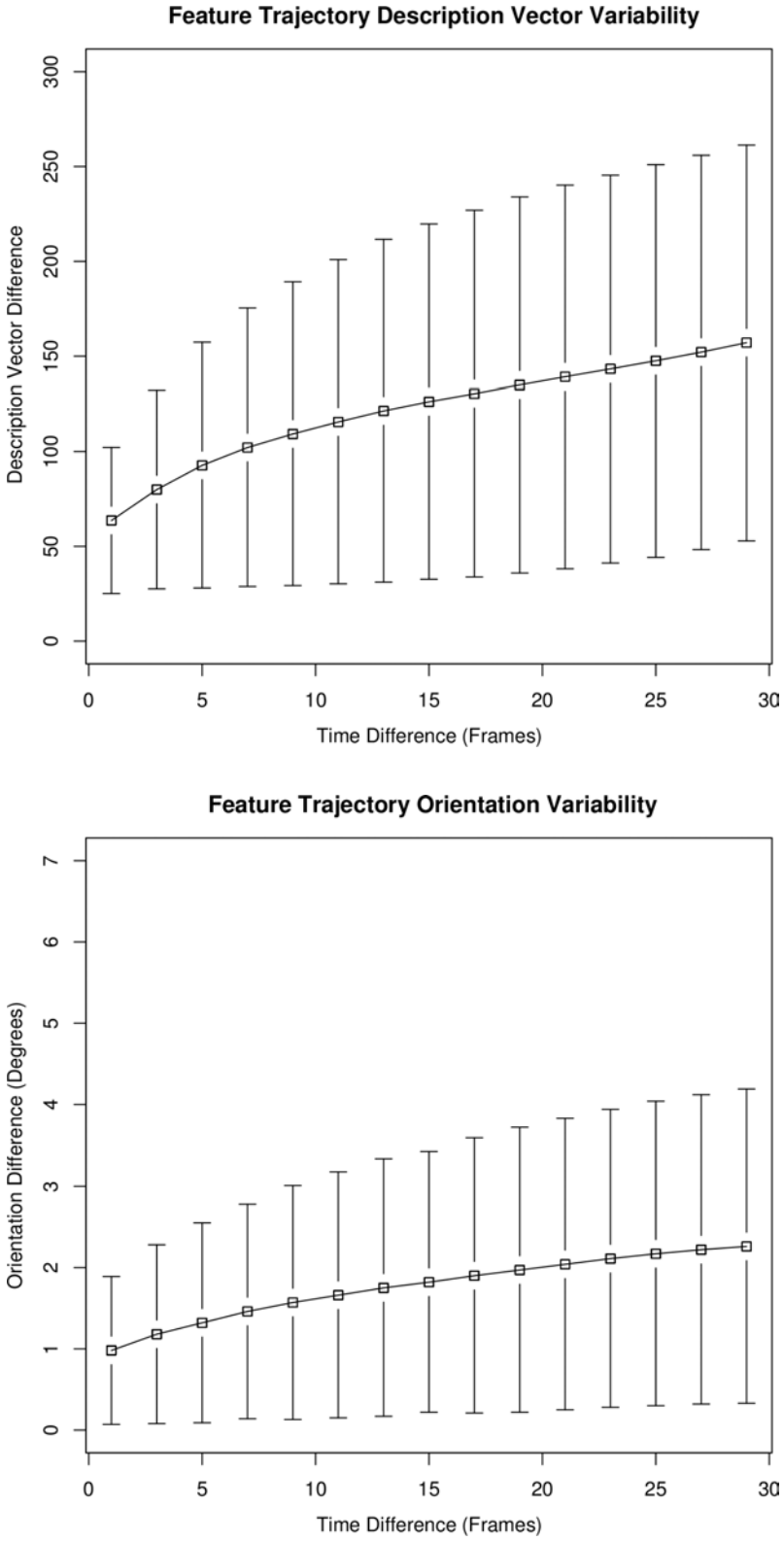

Figure 4.3.5: These graphs show the increasing description vector and orientation difference between the most recent *Stereo Feature* of a trajectory and older features of the same trajectory. The error bars represent the standard deviation.

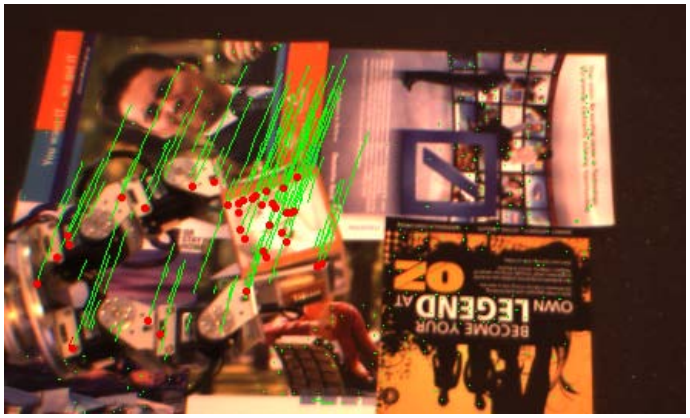

Figure 4.3.6: This image shows a snapshot of features. The *Stereo Feature* trajectory paths are indicated in green. The robot arm and held object features have long trails because they are moving through the scene. The red points indicate *Stereo Features* that form a snapshot due to having moved over a threshold amount. Note that this is before arm features are filtered out.

included in the snapshot. The next step is to separate out the arm features and background motion from the snapshotted trajectories. The remaining snapshotted object features are then inserted into a database, which can later be used for object recognition and localisation. Figure 4.3.6 shows an example of a snapshot.

Sine the source of object motion is movement by the robot's arm and gripper, trajectories produced by the robot itself must be removed.

4.3.4

Need intro

The next step is to filter out the arm features from the snapshots. One way this can be done is if an accurate mapping can be made between arm joint angles and the 3D world space regions occupied by the arm, then any snapshot features that fall in these regions can be labeled as arm features and removed. The disadvantage of this approach is it requires very accurate kinematics, a high degree of synchronisation with the vision system, and an accurate 3D model of the robot arm and gripper.

A different method is to use a database of arm SIFT features to compare against the snapshot and remove any matching features. There are two approaches to generating the arm feature database. The first is to extract SIFT features from segmented and labeled training images of the robot arm and insert them into a database. However, this assumes the availability of appropriate training images, which may not be the case. The second apprach is to have the robot perform the feature snapshotting

steps without holding an object, and instead learn snapshots of arm features. In this way the robot can autonomously generate a database of arm SIFT features as an initialisation procedure. We decided to take this last approach as our aim is to minimise human intervention in the entire process of object learning.

With the generated robot arm feature database, we can filter out arm features from the feature snapshots, leaving behind held object features. For matching snapshot features to the arm database, we use Lowe's method[34, 35] for SIFT feature matching. This involves searching the arm database for the nearest and second-nearest features to the snapshot feature. If the Euclidean distance between the SIFT description vectors of the snapshot feature and the nearest database arm feature is less than 80% of the distance to the second-nearest feature, then the snapshot feature matches the arm feature. We remove from each snapshot all features that match an arm feature, the remaining features are labeled as object features. The justification for this approach is that the density of features in the neighbourhood of a feature indicates how discriminatory that feature's description vector is. The 80% value was determined by Lowe experimentally[34, 35] to give the optimal trade-off between false positives and false negatives.

The reason we use this nearest neighbour method of matching, as opposed to the bipartite matching method described in Chapter 3, is that we do not have a geometric dependency between matched features. The robot gripper can be in many different positions, almost fully closed when gripping small objects, or almost fully open when gripping large objects. As a result, if we were to match using the method described in Chapter 3, we would need to learn the feature appearance model of the gripper for all of the different grip sizes. Instead we take the approach of matching individual snapshot features against the database of arm features based only on the feature description vector.

4.3.5 Filtering Background Motion

The features that comprise a snapshot can come from three sources, the held object, the robot arm and gripper, and any background motion. Static background features are not included as their trajectory displacements are less than the threshold required (described in Subsection 4.3.3). In the previous section we dealt with filtering out the robot arm features. The next step is to filter out background motion features.

Background motion can be filtered out in several ways. First, distant features outside the robot's workspace can be removed by examining their 3D world position. If a feature is further away from the robot that the robot's maximum reach, then it is safe to assume the feature does not belong to the held object. A further refinement is to use arm kinematics to determine the approximate gripper position in each frame and only consider features within a threshold distance of this point. However, it is possible to have background motion that falls within this threshold distance. A further refinement is necessary.

Our solution to this problem is to use the fact that the gripper and the features of the held object will follow similar trajectories in 3D world space. By comparing a feature trajectory to the path of the robot arm, it is possible to determine if it belongs to background motion or the held object. There are two ways the robot arm's path can be determined. The first is by using kinematics and the joint angles. We can track the robot arm joints and for every frame use these to determine the arm position, thus building a trajectory. However, this approach requires accurate arm kinematic feedback synchronised with the vision system. An alternate approach is to use the detected arm feature trajectories (described in the previous section) to determine the robot arm motion.

In the previous section we described how to determine the feature trajectories of a snapshot which belong to the robot arm. Each such trajectory can be interpreted as a series of 3D world space positions, corresponding to the positions of the trajectory's features. This series of positions can used to compare against other trajectories. If

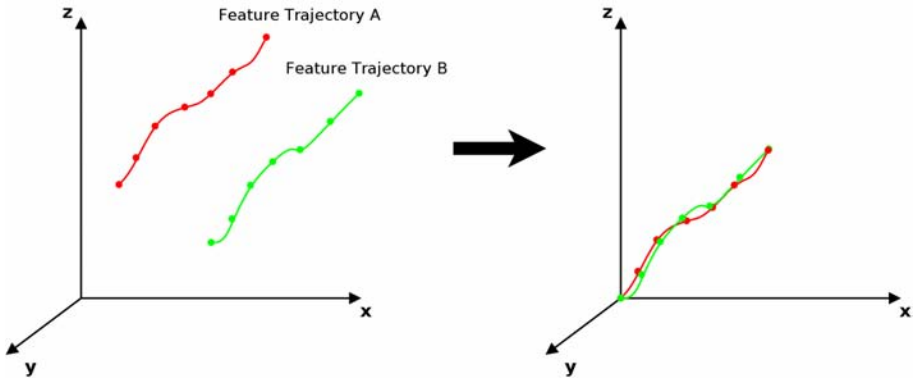

Figure 4.3.7: Feature trajectory position normalisation. When comparing two trajectories, both have their comprising feature world positions shifted so that the trajectory starts at the origin.

more thresholds!

the difference is over a threshold then we conclude that the trajectory cannot belong to the held object as it has not followed a similar path to the arm. To compare two trajectories, a list of feature positions is extracted from each and normalised to the origin. Normalising a series of positions refers to shifting them such that the latest position of the trajectory is at the origin (refer to Figure 4.3.7). The individual feature positions of the trajectory are altered as follows:

$$p_n, p_{n+1}, ..., p_m \rightarrow (p_n - p_m), (p_{n+1} - p_m), ..., (p_m - p_m) \tag{4.3.2}$$

where p_n is the 3D position of a trajectory feature at frame n. To calculate the difference between two trajectories, take the average distance between positions on matching frames across the two normalised position lists P and Q:

$$Dist(P, Q) = \frac{1}{n} \sum |p_a - q_a| \tag{4.3.3}$$

where n is the number of frames in which both trajectories have a recorded feature, p_a is the normalised position of the first trajectory at frame number a, and q_a is the normalised position of the second trajectory at the same frame number.

We remove any feature trajectories that are not within a threshold distance of an arm feature trajectory as determined by the above distance measure. We determined the appropriate threshold by empirically examining the variability of

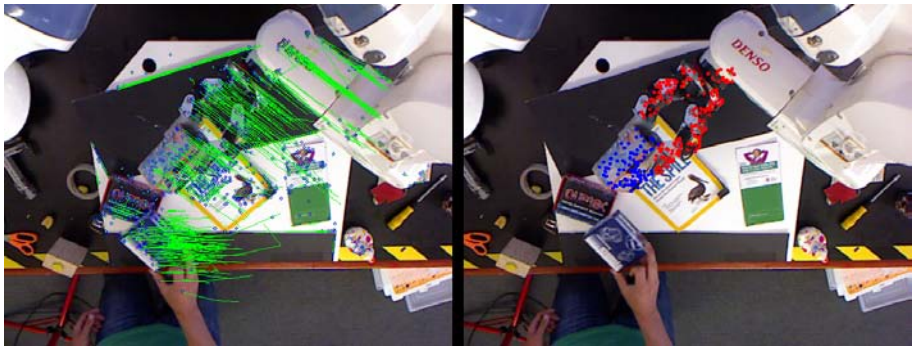

Figure 4.3.8: Example of motion filtering in action. The green trails in the left image represent the feature trajectory paths, the blue points in the right image indicate the snapshotted object features, and the red points indicate the detected arm features. Note that background motion trajectories of the blue box are successfully filtered out and do not contribute to the snapshotted features.

trajectories of a rigid object. We found that the average distance using the above measure of feature trajectories on a rigid object is $0.21cm$ with a standard deviation of $0.25cm$. We set the threshold to be $\bar{x} + 3\sigma$ which is 0.96. Any feature trajectory that is not within this threshold distance from at least one arm feature trajectory is considered background motion and filtered out from a snapshot. Figure 4.3.8 shows the effectiveness of this approach at removing background motion trajectory features from a snapshot.

This approach to trajectory filtering should be equally applicable in the case where accurate and synchronised arm kinematics is available. In this case, rather than using an arm feature trajectory positions, we would use the arm kinematic positions to compare against all other trajectories. We leave this for future work.

It is important to note that this method of background motion filtering can only be applied if [illegible]d held object is a translation rath[illegible]he path of each feature depends on its distance from the axis of rotation, which is not suitable for this approach.

Where do you talk more about rotation and how to get around these limitations?

4.3.6 Refining the Segmentation

In Subsection 4.3.4 we described a method for separating out the features belonging to the robot arm and gripper from the snapshot, with the aim of filtering out all features not belonging to the held object. This is done by learning an arm feature database that can be matched against snapshot features. However, due to noise or lighting variations, some snapshot arm features may fail to match the stored database and instead be incorrectly labelled as object features. Additionally, if the arm appearance changes as compared to its state when the database was learned, some arm features may not match the database. An arm may change appearance due to factors such as accumulated dirt or wear and tear. The result of incorrect matches is arm features incorrectly labelled as object features. This could detrimentally affect the accuracy of the generated object feature database during object recognition and localisation.

We address this problem by comparing the features of snapshots of different objects. Our solution is to use the fact that, due to the highly disciminatory nature of SIFT features, similar features appearing in snapshots of different objects are likely to be misclassified arm or background features. Assuming that the objects the robot learns are visually unique, the only common features between snapshots of different objects must belong to the robot arm or the background.

When a set of feature snapshots has been generated for a number of objects, we consider each object in turn and all of the feature snapshots of that object are compared against the snapshots of the remaining objects. Call the set of snapshots belonging to the current object A, and the set of snapshots belonging to the remaining objects B. Every snapshot $a \in A$ is compared against the snapshots $b \in B$, searching for matching features. Depending on the number of objects learned, it may be computationally prohibitive to compare against all of the snapshots of every object. In this case, a random subset of snapshots $b \in B$ can be used instead, the size depending on the desired accuracy and time constraints.

Algorithm 4.3 Finding mislabeled arm features.

input: current snapshot $\rightarrow a$
input: all snapshots of other objects $\rightarrow B$

$arm\_features \leftarrow \{\}$
forall b **in** B
  forall f_a **in** $objectFeatures(a)$
    $f_b \leftarrow getNearestFeatureIn(b, f_a)$
    $second\_nearest \leftarrow getSecondNearestFeatureIn(b, f_a)$
    if $distance(f_a, f_b) < 0.8 \cdot distance(f_a, second\_nearest)$ **then**
      $arm\_features \leftarrow \{f_a\} \cup arm\_features$
    endif
  endfor
endfor

output: mislabeled arm features $\leftarrow arm\_features$

When comparing a snapshot a to a snapshot b, we search for features that match. The matching criterion is similar to that used for matching arm features: feature f_a from snapshot a matches feature f_b from snapshot b if f_b is the nearest neighbour to f_a of all features occurring in b using the SIFT descriptor distance metric, and this distance is less than 80% of the distance to the second nearest neighbour. All features f_a with a matching feature in b are marked as arm features, which are then removed from the snapshot a. This process for finding mislabeled arm features by correlating features across snapshots of different objects is summarised in Algorithm 4.3.

4.4 Evaluation

To test the effectiveness of ~~the presented~~ our feature segmentation ~~approach~~ method we examine the reliability of the segmentation in different circumstances, as well as the object feature recognition performance of the database that is generated using the segmented features.

To test the accuracy of the feature segmentation, the robot ~~performs~~ executes the algorithm described in this sectoin while moving a grasped object through a cluttered scene. The motion performed by the robot during a single trial is to move the object

Figure 4.4.1: Some of the objects used to test the robot learning of object recognition.

	Features Detected
True Arm Features	328
True Object Features	378
False Arm Features	8
False Object Features	40

Table 4.1: The average number of correctly and incorrectly identified object and arm features per trial, across 10 different objects, with three trials per object. Each trial consists of 20 snapshots.

back and forward in a straight line 50cm in length. During this motion a total of 20 feature snapshots are performed, one every 5 frames. For each snapshot we record the total number of correct object features and arm features detected, as well as the total number of incorrectly labelled object and arm features. We perform this process in different scenarios, three separate times for an object, across 10 different objects, for a total of 30 trials for each scenario. The objects used are simple shapes such as a soft drink cans and cubes of edge length 7cm with an image on the side facing the camera (see Figure 4.4.1). For each trial we sum the total number of detected features from the 20 snapshots which comprise the trial and present the data in table format.

In the first scenario we test the object feature segmentation in a cluttered environment with no background motion. The average number of features detected per trial is shown in Table 4.1. We can see that the number of background and arm features misclassified as object features is low compared to the number of correctly classified object features. All of the *False Object Features* were due to arm features incorrectly classified as object features. No background features were classified as object features.

Motion Filtering	*Disabled*	*Enabled*	*Change*
True Arm Features	289	289	-
True Object Features	312	282	-9%
False Arm Features	3	6	+100%
False Object Features	366	38	-89%

Table 4.2: This table shows the number of features labeled by the feature segmentation algorithm, in the presence of background motion, with motion filtering disabled and enabled. It can be seen that motion filtering greatly reduces false positive object features in the presence of background motion.

The second scenario involved testing the effectiveness of the background motion filtering of our feature segmentation method. This is done by introducing background motion into the scene, in the form of a textured object (the same form and size as the held object) moving in a random path in the vicinity of the robot arm. This motion is performed by a human operator. We then compared the performance of the segmentation algorithm with background motion filtering enabled and disabled. In each case the number of correct and incorrect arm and object features found during the course of the arm movement is counted. The average number of features detected, across 30 trials, is presented in Table 4.2. It can be seen that the effect of background motion filtering is a ten fold reduction in false object features, removing feature trajectories of the object moving in the background. The remaining false object features are due to misclassified arm features. The downside is a small reduction in the number of detected object features.

The next test scenario is to test the effectiveness of the segmentation refinement to account for arm features falsely classified as object features. To test this, we alter the appearance of the arm by attaching a textured marker to the robot gripper (shown in Figure 4.4.2). This marker is not present when the robot is learning the SIFT features of the arm, it is only present when the robot is learning the held object features. This simulates a scenario where the robot arm becomes worn out or dirty through use, altering its appearance as a result. Normally the features of the textured marker would be incorrectly classified as object features, as they will not match the learned arm feature database. However, by cross-matching learned

Figure 4.4.2: The marker attached to the robot gripper is circled in red. This is used to test the effectiveness of the feature correlation filtering.

Cross-Correlation	*Disabled*	*Enabled*	*Change*
True Arm Features	253	581	+129%
True Object Features	446	494	+10%
False Arm Features	1	6	+500%
False Object Features	380	72	-81%

Table 4.3: Feature cross correlation reduces false positive object features when the arm appearance changes.

object features across multiple different objects, we can determine the common features and conclude that they must belong to the arm (assuming objects distinct in appearance).

To evaluate the effectiveness of the cross-matching between object snapshots, the robot performs the feature segmentation and snapshotting for all ~~10~~ ten objects (with ~~20~~ twenty snapshots per object). Following this, the snapshots of each object are iterated through and the features are matching against the ~~9~~ nine other objects. Any matching object features are re-labeled as arm features. The results are presented in Table 4.3. When correlating snapshot features for each object, all ~~9~~ nine other objects are compared against, with ~~20~~ twenty snapshots per object. The result is that the number of false object features is reduced by 81% when correlation filtering is enabled, due to the removal of the texture marker features from the object features set. These results are summarised in Table 4.3.

Finally we tested the object recognition accuracy of the generated object feature database. The goal of the segmentation algorithm is for a robot to learn the SIFT features of an object and use these to later recognise and localise the object in a scene. To test the effectiveness of the segmentation algorithm for this application, we use the snapshotted object features generated by the robot to create an object feature database. We then use this database to match against various scene images containing the learned object, recording the number of object features successfully matched. The performance of the autonomously generated object feature database is compared to an object feature database generated from manually segmented views of the object.

The autonomously generated object feature database is created by having the robot perform the previously described algorithm while moving the object in a linear motion. This process is performed in a cluttered scene, with no background motion. 29 snapshots are performed per object per trial. The object features from each snapshot are inserted into a database, which is then used to match object features in nine different scene images containing that object. Each of these images has a manually generated ground truth feature classification, determining which of the features in the image are object features and which are background or arm features. The number of object features successfully detected as a fraction of all object features in the image is recorded. The number of falsely classified object features is also recorded. This is done for ten different objects, the performance of the database is recorded relative to the number of snapshots that comprise the database. This is done to evaluate the change in matching accuracy as more snapshots are added to the database.

The manually segmented training views of each object, with six image snapshots per object, are used to build a baseline feature database to compare against. For each object, features are extracted from the training images and inserted into the baseline database. This is then used to perform object feature matching in the same way as described for the autonomously generated database above. For each

database, features are matched using Lowe's nearest-neighbour approach [34, 35], with the second nearest neighbour database feature providing the threshold for rejecting spurious feature matches.

The matching performance results of the two databases are shown in Figure 4.4.3. After a total of 29 autonomously generated snapshots are added to the database, it is capable of recognising 71% of an object's features in the test scene images. The baseline database, created from 6 manually segmented training images of the object, is able to detect 62% of object features. The number of false positives of the two databases is 6% and 4% respectively. These results show that the presented feature segmentation method is effective at generating an object feature database that can be used to match object features in a scene image.

The performance of this algorithm is comparable to using manually segmented views of an object, with the advantage that it allows a robot to learn a new object autonomously in a complex and dynamic environment.

4.5 Discussion

We have presented an effective algorithm for active learning of object SIFT features on an autonomous robot platform, using motion based feature segmentation. Stable image features localised in 3D world space, combined with motion, provide sufficient data to perform effective object feature segmentation from background features in the presence of background motion and clutter.

One of the caveats of the described algorithm is the way the target object is moved through the scene. The motion the robot performs while holding the object is chosen to be a translation, allowing one particular aspect of the object's appearance to be learned. The linear motion is required to build up trajectory information about each feature, by tracking it over a sufficient period of time. If features of multiple aspects of the object need to be learned, then a combination of rotation and translation motion must be used. Ideally the robot should rotate the object in

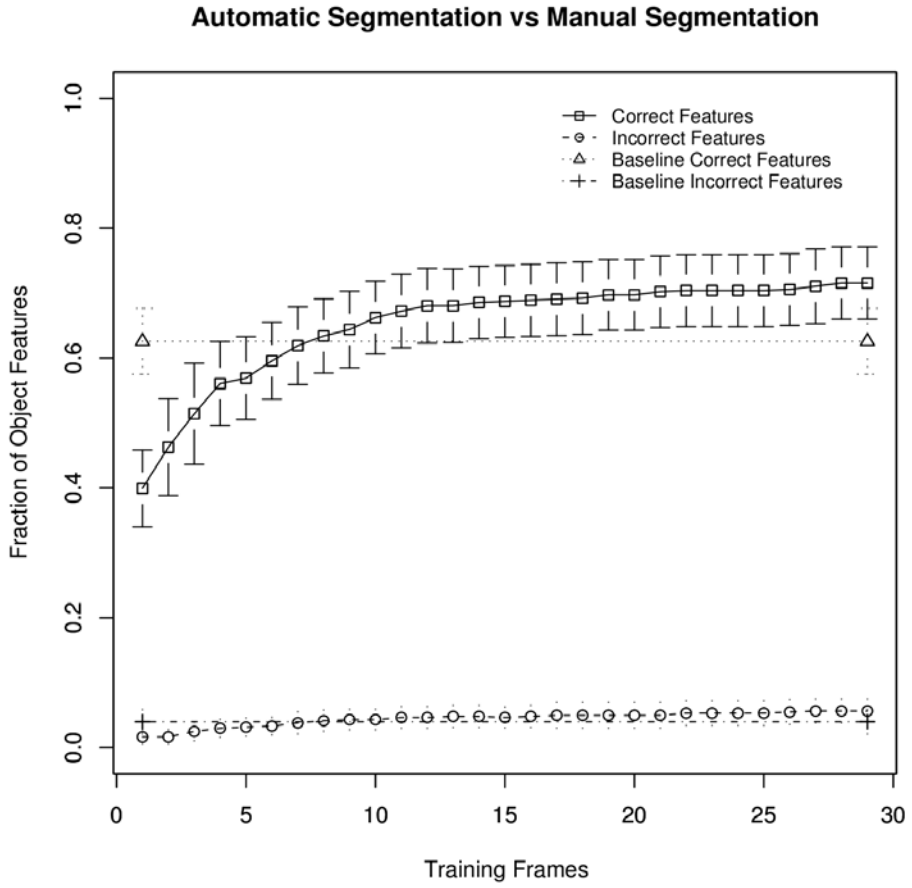

Figure 4.4.3: The fraction of object features, detected by an autonomously generated object feature database, compared to that of a feature database generated from manually labeled training images. The manually labeled database is generated from six segmented training images of each test object. The performance of the autonomously generated database is plotted in relation to the number of snapshots used to build the database. The error bars indicate the Standard Error across the different evaluation iterations.

place so as to view the object from all sides, allowing multiple aspects to be learned. However, we found that tracking features becomes a problem under rotation (pitch or yaw relative to the camera), only being able to track features through approximately 15° of rotation out of the camera plane. This is because under such rotation the pixel neighbourhood around a feature point changes, compared to the original. This reduces the trajectory information per feature as compared to translation motion, making the separation of foreground object and arm features from background features unreliable. Furthermore, background motion filtering based on arm feature trajectories becomes ineffective due to their short length. As a result, to learn the features of an object from multiple viewpoints, we fall back to performing multiple translation motions with a gradual rotation of the robot wrist joint. This allows a feature to be tracked for a sufficient period of time, while still viewing multiple aspects of the object. This is further elaborated on in Chapter 5.

The feature recognition results presented in Figure 4.4.3 raise the question of why the percentage of detected object features does not approach 100% as more snapshot features are added to the database. A potential reason for this is pixels from the background influencing the learned object SIFT features. SIFT features are based on a pixel neighbourhood of a certain size, depending on the scale of the feature. Features near the edges of an object and with large pixel neighbourhoods will have background pixels contributing to the description vector. As a result, when learning the features of an object while it is held by the robot arm, some of the learned features that have a large scale or are close to the object edge will not be recognisable later when the object is placed in a scene. This is because the background pixels will change, which will also change the SIFT feature's description vector. The phenomenon of background pixels influencing an object's SIFT feature is illustrated in Figure 4.5.1. To support this hypothesis, when performing object recognition, we record for each object feature its distance from the object image region and its scale. This data is plotted in Figure 4.5.2 for object features successfully matched by the database, and for object features which were not matched. It can be seen that, as

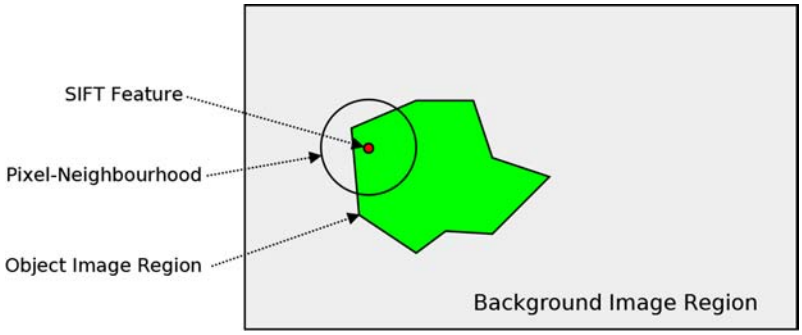

Figure 4.5.1: This diagram demonstrates how a SIFT feature located inside the object image region (green) can be dependent on the background (grey), affecting the resultant description vector and orientation direction. This makes reliable matching of some object features impossible since the background may be different compared to when the feature was learned.

the distance from the object region's edge decreases and the feature scale increases, the number of undetected features grows, supporting our hypothesis.

4.6 Future Work

There are several possible avenues for future work to build upon the general approach presented in this chapter. First, our implementation of the general feature tracking and segmentation method is based on SIFT features. However, there is a wide variety of other local image descriptors (SURF [46], PCA-SIFT [45]) which may be used instead of, or in addition to, SIFT. Second, the feature trajectory tracking aspect of our implementation can be improved by combining the expected direction of arm movement (extracted from the arm kinematics) and Bayesian tracking [114, 115] of features. Third, if a detailed 3D model of the robot gripper and arm can be built autonomously, it may be used to better filter out arm features from snapshots.

This chapter has presented a method of learning the features of one aspect of an object, that is, only a single perspective of the object. However, for object recognition and localisation, the full aspect graph should be learned to be able to recognise the object in a scene from all viewing directions. Furthermore, as each object *Stereo Feature* has a 3D world position, if multiple aspects are learned then they may be joined to form a single coherent 3D point cloud of the object. This can

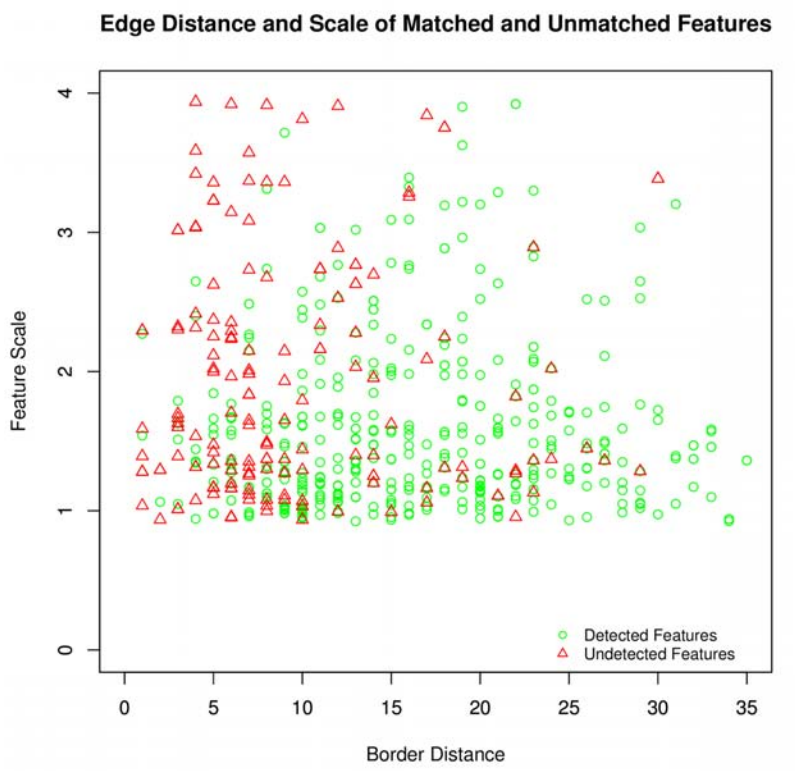

Figure 4.5.2: Plot of detected and undetected features given their pixel distance from the object edge and the feature scale. The border distance is the number of pixels to the nearest non-object pixel from the SIFT feature, the feature scale determines the size of the neighbourhood used to compute the SIFT description vector.

be used to determine the overall shape of the object, which is useful for grasping and manipulation planning. This is presented in the next Chapter.

Chapter 5

Object Reconstruction

5.1 Introduction

In Chapter 3 we presented a method for matching the SIFT features of a single view of an object to a set of scene feature for object recognition. In Chapter 4 we presented a method for a robot to autonomously learn the SIFT features of a single aspect of an object by using motion and feature tracking. In this chapter we combine these methods to build an all-aspect appearance model of the object and reconstruct its 3D shape. This is the next step ~~is~~ in the overall process of a robot autonomously learning to recognise an object, learning it's shape and physical properties, and using this knowledge to accomplish a task.

We present a system that combines the feature segmentation and matching methods (described in the previous two chapters) with ~~existing~~ object reconstruction techniques to extract a complete 3D object model. The model encompasses the 3D shape and the full aspect graph of SIFT features of the object. This model will enable the robot to recognise and localise the object in a scene in different orientations. Additionally, knowing the shape of the object allows the robot to use this model for manipulation and grasp planning.

To build the object appearance and shape model, we use a RGB-D (colour and depth) Kinect camera (hardware platform is described in Section 5.2), combined

with robot induced object motion, to separate the object image region from the background. We then stitch together the different views into a single coherent model. The object segmentation is based on the method described in Chapter 4. The robot grasps an object and moves it through the scene in a linear motion. The object is slowly rotated during the course of the motion so that different aspects of the object are visible to the robot's camera. During the course of this motion, the robot tracks the scene SIFT features, builds a trajectory of each, and periodically performs snapshots of the features that have moved over a threshold amont. ~~This~~ These snapshot data ~~is~~ are used to extract an object-view, consisting of the SIFT features and a surface point cloud of the object as viewed from a single direction (described in Section 5.3). Multiple object-views are extracted as the robot moves and rotates the object, viewing it from multiple directions. The resulting views are stitched together (described in Section 5.4) to form a single coherent model, encompassing all 360° of the object. The model consists of a set of SIFT feature snapshots, covering different viewing directions and a surface point cloud describing the shape of the object. A geometric model is fitted to the point cloud (described in Section 5.6) to provide a compact description of the object's shape. This resulting object appearance model can then be used for for recognising and localising the object in a scene (described in Section 5.6). Figure 5.1.1 summarises the steps involved in the object reconstruction system.

5.2 Hardware Platform

The hardware platform for this system is similar to the one used in the previous chapter. It consists of a six degrees of freedom robot arm mounted on a metal spine on a table. The robot arm is equipped with a two-fingered gripper (see Section 1.3).

One of the aims of the object reconstruction system is to determine the 3D shape of the object. To do this we replaced the Bumblebee2 stereo camera, used in the previous chapter, with a Kinect RGB-D camera system (see Figure 5.2.1). The

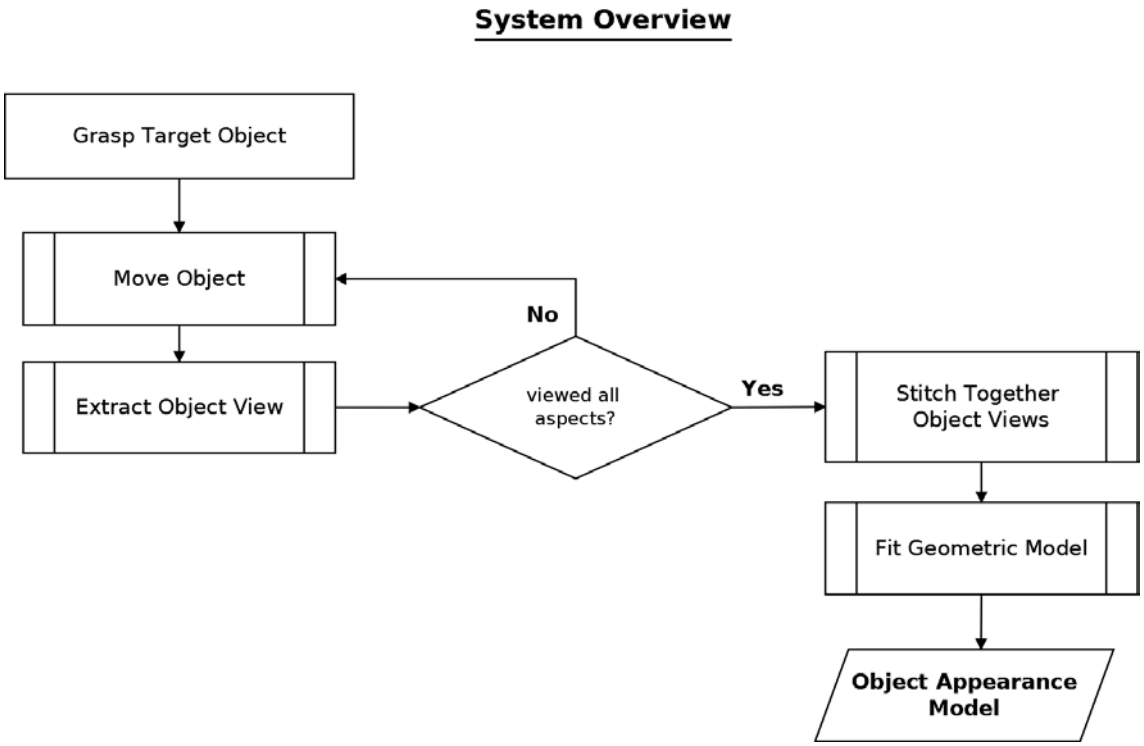

Figure 5.1.1: A top level overview of our object appearance and shape reconstruction system.

Kinect camera features a standard RGB camera that outputs colour images with a resolution of 640 × 480 pixels at a rate of 30 frames per second. Additionally, the Kinect has a depth sensor that outputs a 640 × 480 pixel resolution depth image at a rate of 30 frames per second. This depth image consists of $11 - bit$ values for each pixel, signifying the distance of the scene from the camera at that point. To generate the depth image the Kinect uses an infra-red laser projector to project a structured grid of points into the scene, which is then viewed through an infra-red camera. The pattern of infra-red light is used to determine the depth of the scene at each pixel (see Figure 5.2.2).

The advantage of using the Kinect over a stereo camera is that it provides accurate real-time depth information for each pixel without needing to compute a dense correspondence between left and right stereo images. We use the per-pixel depth data to construct a dense point cloud of the surface of the object.

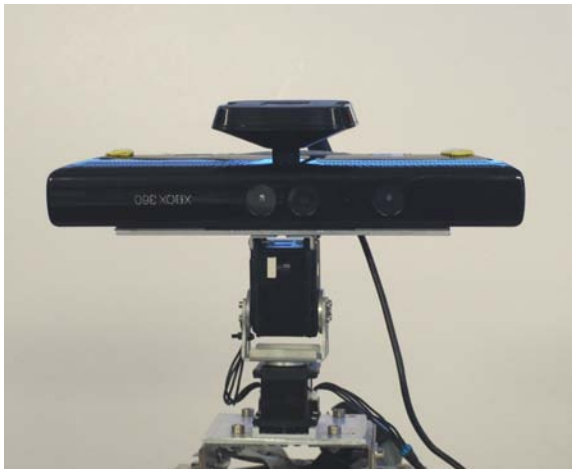

Figure 5.2.1: Kinect RGB-D camera mounted on a pan-tilt servo pair. This is located on the "neck" of the robot, allowing the Kinect sensor to look down to observe the workspace, robot gripper, and object.

Figure 5.2.2: Top: the Kinect projects a structured pattern onto the scene in infrared (image courtesy of http://graphics.stanford.edu/~mdfisher/Kinect.html). Bottom: the resulting per-pixel depth information is expressed in grey scale, darker pixels representing areas closer to the camera. Pixels that are black have no valid depth information (image courtesy of http://www.brekel.com/wp-content/uploads/2010/12/kinect-3D-scanner-capture-depth.jpg).

5.3 Generating Object-Views

In this section we extend the concept of object feature snapshots (introduced in the previous chapter) to form an object-view. In addition to SIFT features, an object-view also includes a dense surface point cloud of the object as viewed from a given direction.

In Chapter 4 we presented a method for generating object SIFT feature snapshots consisting of the features visible from a single aspect. Object features are separated from the background by having the robot move the object through the scene and track the resulting feature motion. The SIFT features are tracked in 3D world space using a stereo camera. Those that follow a similar trajectory to the robot gripper are labeled as object features.

A single snapshot provides both appearance and shape information of a single aspect of the object. The SIFT features encapsulate the appearance information, allowing the object to be recognised and localised in a scene by using the feature matching approach presented in Chapter 3. The 3D positions of the snapshot SIFT features also provide information about the shape of the object, forming a point cloud of its surface.

The main weakness of using only the SIFT features for shape information is the potentially sparse nature of the point cloud. Plain untextured image regions will not produce many SIFT features [34, 35]. This results in little shape information in untextured areas of the object. To address this problem, we replace the stereo camera with a Kinect depth camera. The Kinect camera, in addition to a standard RGB image, outputs a per pixel depth value. This allows the robot to construct a dense surface point cloud of the object, independent of the object's appearance. This process is represented in Figure 5.3.1. In this section we describe the modifications to the object snapshotting method introduced in Chapter 4. First we describe the changes required to move from a stereo camera to a Kinect depth camera. We then outline the method used to extract the object and arm surface point cloud, in

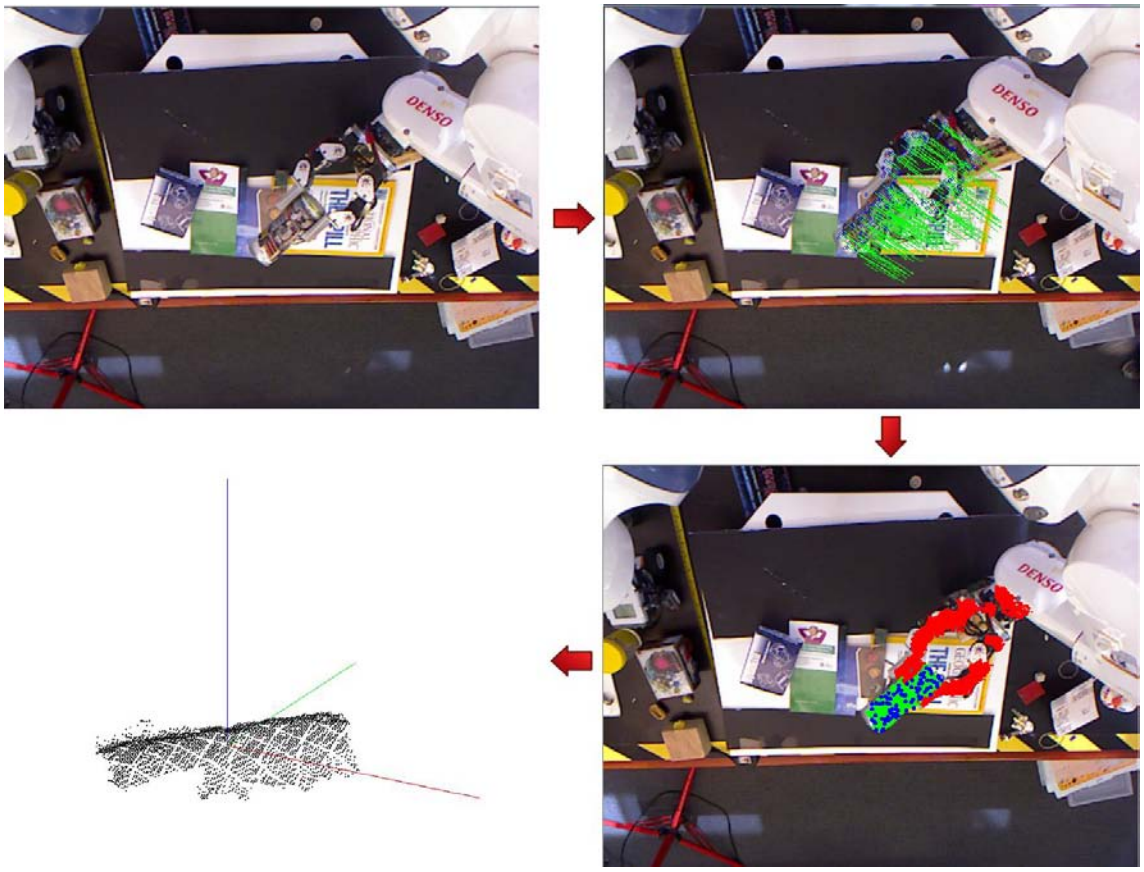

Figure 5.3.1: The process of extracting an object-view. The object is moved through the scene and its SIFT features are tracked in 3D world space (top left, and top right). We then take a snapshot, separating the object and arm features from the background. The image regions that correspond to the object and arm are extracted by performing a flood fill over the depth image (bottom right). The resulting object-view consist of the object's SIFT features (blue dots in the bottom right), and the object's dense surface point cloud (bottom left).

addition to the SIFT features, and how to separate the arm and object regions of the point cloud. In later sections we describe how the individual object-views are stitched together to form a complete all-aspect object appearance and shape model.

5.3.1 ~~Kinect Camera~~ Image

In Chapter 4 we presented a method using a stereo camera to determine the 3D position of each SIFT feature by correlating features in the left and right images. In the case of a ~~Kinect~~ camera, each pixel has an associated depth value, along with the colour RGB values. We extract the SIFT features using the RGB colour image,

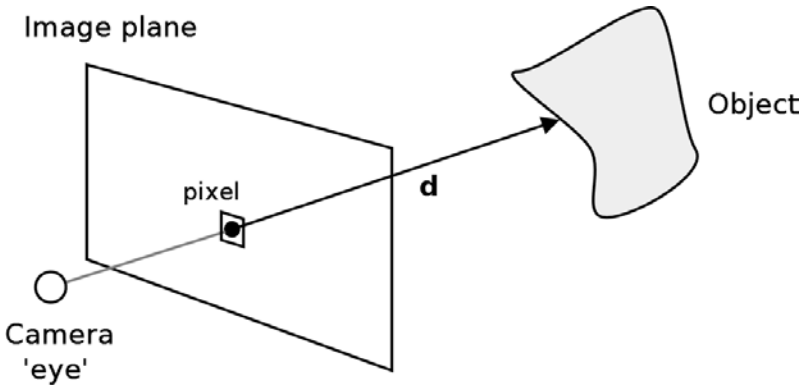

Figure 5.3.2: To find the world space position of a feature or a point on the surface of an object, we take the distance **d** associated with the feature's pixel and project a ray out of the camera image plane into the scene. The end-point of the ray corresponds to the 3D world space position of the feature.

and to find the 3D positions we use the depth values of the pixels on which the features are centered. We project a ray out of the camera's image plane a distance indicated by the pixel's depth value. The resulting point is the 3D world position of the feature. This is illustrated in Figure 5.3.2.

5.3.2 Arm and Object Surface Point Cloud

As the robot moves the object through the scene, it periodically performs feature snapshots. This consists of features that have moved more than a threshold distance and follow a similar trajectory to the arm end point. These features correspond to the held object and robot arm. This process ~~is~~ was presented in detail in the previous chapter. We now use the dense depth information provided by ~~the Kinect~~ an RGB-D sensor to perform further processing and extract the arm and object surface point clouds for each snapshot.

When a snapshot is ~~performed~~ obtained, we use the SIFT features as seed points for region growing segmentation to extract the arm and object surface point cloud. To do this we make use of the following assumptions:

- the snapshot SIFT features are located on the robot arm or target object's surface;

Algorithm 5.1 Arm and Object Region Growing.

input: snapshot features $\rightarrow S$
input: snapshot depth image $\rightarrow D$

$labeled\_pixels \leftarrow \{\}$
forall s **in** S
  $p \leftarrow pixelPosition(s)$
  $floodFill(p)$
endfor

function floodFill (**input:** pixel p)
  $labeled\_pixels \leftarrow labeled\_pixels \cup \{p\}$
  $N \leftarrow getEightNeighbourPixels(p, D)$
  $p\_depth \leftarrow getDepth(p)$

  forall n **in** N
    $n\_depth \leftarrow getDepth(p)$
    if $|p\_depth - n\_depth| \leq 0.5cm$ **then**
      $floodFill(n)$
    endif
  endfor
endfunction

output: arm and object image region pixels $\leftarrow labeled\_pixels$

- the surface of the viewed aspect of the robot arm and target object is mostly continuous and smooth;

- no background objects or surfaces are in contact with the target object or arm.

We use the snapshot scene depth image to segment the object and arm image regions from the background. This is done by performing an eight-neighbour flood fill starting at every snapshot SIFT feature. The flood fill is performed over the depth image, with a neighbouring pixel being filled if its depth value is within a small threshold distance of the current pixel (we used a threshold of $0.5cm$). All pixels that are filled are then labeled as arm/object pixels. These pixels, with their associated depth values, form a surface point cloud of the arm and object. This process is summarised in Algorithm 5.1 and Figure 5.3.3. The result of this process is a set of SIFT features and a surface point cloud of the robot arm and the held object. The next step is to separate the object and arm features and surface points.

All these arbitrary fixed thresholds really worry me. Can this be done any other way?

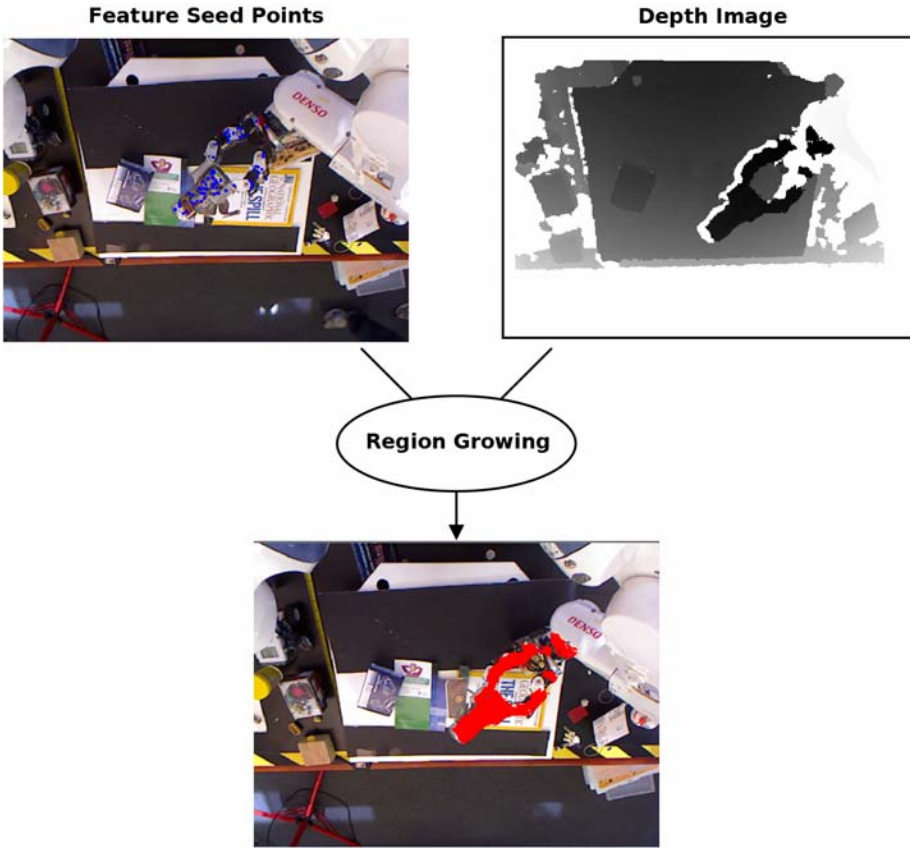

Figure 5.3.3: To find the robot arm and held object image regions (bottom in red) and associated surface point cloud, we perform seeded region growing. The snapshot SIFT features (top left in blue) are the seeds, and the region growing is performed over the depth image (top right).

5.3.3 Arm and Object Segmentation

The next step is to separate the arm and object point cloud regions, as well as the SIFT features. This is done in a similar way ~~as~~ to separating snapshotted arm and object SIFT features in Chapter 4. First, the appearance and shape model of the robot arm and gripper is learned as an initialisation stage. This is done by having the robot move its arm, without holding an object, through the scene. The gripper is slowly rotated 360° through the course of the linear movement. Periodically, snapshots of arm features are taken. In addition to snapshotting the arm features, the arm surface point cloud is also extracted for each snapshot (using the approach described in the previous section). Since the robot is not holding an object, the filled region corresponds to the robot arm. The full set of these views, generated through the course of the entire movement routine as the gripper is rotated through the full 360°, forms the arm appearance and shape model.

Once the arm model is learned we can use it to separate the arm and object image regions and SIFT features in a snapshot. First, we find the pose of the arm in the snapshot image scene. To do this we match every learned arm snapshot to the features of an object-view using the method presented in Chapter 3. For the best matching arm snapshot (with the highest number of feature matches) we find the aligning transform between the snapshot and the object-view. The method to find this transform is presented in the next section (Section 5.3.4). We use the aligning transform to overlay the learned arm surface points of the arm snapshot over the object-view. All object-view surface points and SIFT features that are within a small threshold distance of an overlayed arm surface point are labeled as belonging to the arm.

The final result is a set of views of the different aspects of the object. Each object-view is composed of the object SIFT features, the object surface point cloud, the robot arm SIFT features, and the robot arm surface point cloud. The next step is to stitch these views together into a single coherent model. This is presented in

Section 5.4.

5.3.4 Aligning Corresponding 3D Points

A common task that is performed at several stages of object reconstruction, as well as recognition and localisation, is finding a transform to align two sets of corresponding points. For example, when arm snapshot SIFT features are matched to the object-view features we need to align the arm snapshot features with the matching snapshot features to determine the arm's pose.

where a_i and b_i are triples <x,y,z> or polar or what?

First let us define the specific problem. Let there be a set of corresponding 3D point pairs $P = \{(a_0, b_0), \ldots, (a_n, b_n)\}$. The goal is to find a rigid transform that aligns the points a with the corresponding points b. A rigid transform is defined in this case as a rotation followed by a translation (a 3×3 matrix R, and $3D$ vector t, respectively). An aligning transform should minimise the error function over the corresponding point pairs. The error function is defined as the sum of square distances between transformed points a and their corresponding points b:

$$Transform = \underset{R,t}{argmin}\ error(R, t) \tag{5.3.1}$$

$$error(R, t) = \sum_{i=0}^{n} |(Ra_i + t) - b_i|^2 \tag{5.3.2}$$

Finding the transform to minimises this error function is a least-squares fitting problem and can be solved in several ways [88]. We use the Singular Value Decomposition (SVD) [Give full details of how SVD is applied. Pseudo code?] outcome is a rigid transform that aligns the two sets of points by minimising the distance between corresponding points. You said this already

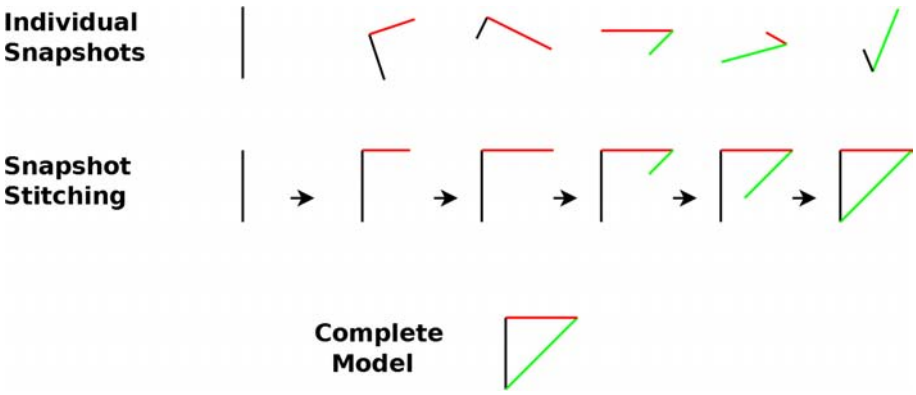

Figure 5.4.1: A simple 2D example of stitching together multiple snapshots (top) of an object. The snapshots are aligned one by one (middle) to finally reconstruct the complete model (bottom).

5.4 Object-View Stitching

To form a complete object model we need to align the separate object-views such that their relative poses match their alignment on the physical object. There has been a lot of work in this area, stitching together views for object reconstruction [86, 87] as well as for building environment maps for robot navigation [117]. In our approach we use both the SIFT features and surface point cloud information of each object-view to determine its relative pose. We use the SIFT features to find an approximate transform for an object-view to align it with the already processed views. This is followed by using the surface point cloud and the Iterative Closest Point (ICP) method (described in Section 5.4.1) to further refine the transform. Figure 5.4.1 shows a simple example of stitching together multiple independent snapshots to reconstruct the overall object shape.

Let the list of extracted object-views be defined as $V = \{v_0, v_1, \ldots, v_n\}$. Each view $v \in V$ is composed of a list of object SIFT features, arm SIFT features, object surface points, and arm surface points. The aim is to find a list of rigid transform $T = \{t_0, t_1, \ldots, t_n\}$, one per view, such that the transformed views are aligned relative to each other according to the shape of the object.

First, we initialise the list of aligned object-views to contain only the first view (v_0). We will align the remainder of the views relative to the first view. We then

Algorithm 5.2 Aligning Views Algorithm Overview

input: list of object-views $\rightarrow V$

$alignedViews \leftarrow \{v_0\}$
$allPoints \leftarrow objectPoints(v_0) \cup armPoints(v_0)$
forall v **in** V
  $bestMatch \leftarrow findBestSIFTMatch(alignedViews)$
  $approxTransform \leftarrow findTransform(bestMatch)$
  $v \leftarrow applyTransform(v, approxTransform)$
  $viewPoints \leftarrow objectPoints(v) \cup armPoints(v)$
  $refinedTransform \leftarrow performICP(viewPoints, allPoints)$
  $a \leftarrow applyTransform(v, refinedTransform)$
  $alignedViews \leftarrow alignedViews \cup \{a\}$
  $allPoints \leftarrow allPoints \cup objectPoints(v) \cup armPoints(v)$
endfor

output: list of aligned object-views $\leftarrow alignedViews$

initialise a point cloud to contain the arm and object surface points of the first view. The next step is to iterate through the remaining object-views and align each one relative to the previously aligned views.

To align an object-view, we match its SIFT features to the SIFT features of every aligned object-view, using the method presented in Chapter 3. We take the match with the highest number of feature matches and use it to generate an alignment transform (using the method presented in Section 5.3.4). This transform is then applied to the current object-view by transforming all of its surface points (arm and object) and all of its SIFT features (arm and object).

The next step is to use the object-view's surface points to refine the alignment. This is done by using ICP to line up the object-view's surface points with the surface points of all of the already aligned object-views. This refined transform is then applied to the object-view and it is inserted into the list of aligned views. These steps are summarised in Algorithm 5.2.

The set of aligned object-views forms the basis for the object model. The remaining step is to fit a geometric model to the surface point cloud. This is presented in Section 5.5.

5.4.1 Iterative Closest Point

Iterative closest point (ICP) is a method to align two point clouds that have a subset of points describing a common surface. It is used to find the transformation of a point cloud such that the distance to the corresponding points of the other point cloud are minimised. Example applications of this method include reconstructing object shapes from several individual scans [89, 86] and motion estimation for mobile robots [118].

Let us define two points clouds, A and B. The ICP algorithm involves the following sequence of steps:

1. determine a correspondence between points in A and B using a nearest neighbour criteria;
2. find a rigid transformation (rotation and translation) for the points in A to minimise the square distance to the corresponding points in B;
3. apply the transformation to the points in A;
4. repeat the process (by finding new correspondences, etc).

A simple representation of the ICP process is shown in Figure 5.4.2. The ICP algorithm works best when the approximate relative pose between the two point clouds is known. In our case this is achieved by first aligning the object-views based on the SIFT features, which provides a good initial alignment. We then use the PCL point cloud library[1] to implement the ICP method. This library takes as input two point clouds, in the form of a list of 3D points, and outputs a rigid transform that best aligns the first point cloud with the second. This library uses a point to point nearest neighbour approach for building a correspondence and uses SVD for finding the alignment transform for a given correspondence. When using the ICP library method, we set the maximum iterations to 20 and the RANSAC outlier threshold to $2cm$. These parameters were chosen ad hoc, but were experimentally verified to give

[1] http://pointclouds.org/

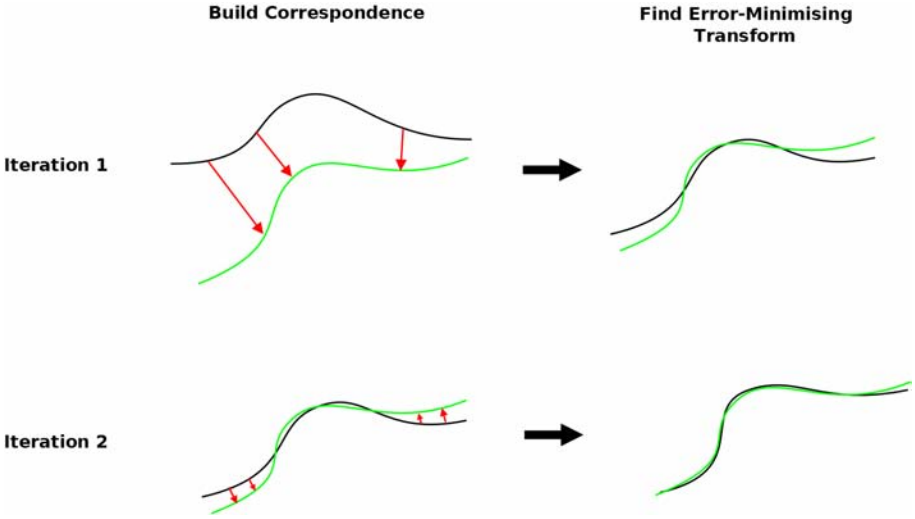

Figure 5.4.2: An example of two iterations of the ICP method on point clouds represented by a green and black curve. First, a correspondence is found between points in the two clouds, then an error minimising transform is found using these point pairs. This process is repeated multiple times, finding new correspondences and transforms, and converges to the best alignment transform between the point clouds.

good results with the objects used. The final output of the ICP stage is a refined transform between the object-view point cloud and the current object point cloud.

5.4.2 Loop Closure and Error Accumulation

In the previous section we presented a method for stitching together individual views of an object to align them relative to each other. Each view is aligned relative to the already aligned views. However, each alignment will inevitably result in some error due to sensor noise and incorrectly classified surface points. These errors will potentially accumulate over time, until the final object-views are not aligned properly relative to the initial object-views.

A related problem of loop closure arises when the robot observes an object view that was observed a long time ago. This arisises when the robot has rotated the object in its hand the full 360°. The problem is deciding whether the robot has seen the object-view previously, and in correlating features between the object views separated by a large time difference. This problem is of particular importance in the field of mobile robotics and Simultaneous Localisation and Mapping (SLAM)

[119], but is also important in 3D object reconstruction [89].

In our system we do not specifically address these issues. We have found that by aligning each object-view relative to the point cloud of all of the already aligned views (as opposed to only the single previously aligned view) reduced the accumulated stitching errors to a negligible amount for our purposes. However, future versions of this system should address this issue by incorporating existing methods. We discuss this further in the conclusion (Section 5.7).

5.4.3 Handling Gripper Occlusion

The robot extracts object-views by moving the target object through the scene in a linear motion while rotating it along one axis (using the wrist joint). As a result, the robot is not able to observe all areas of the object's surface. Some areas are hidden by the robot gripper, while some aspects are not observed due to the single-axis rotation. To address this problem, the robot performs two separate passes of reconstructing the object, holding the object in a different grap each time. We perform the entire object reconstruction procedure, described in this chapter, twice on the same object with different grasps. Figure 5.4.3 shows the robot grasping the object differently in the two reconstruction passes. This generates two separate object-view sets and surface point clouds. We then take these two independent object models and combine them into one by finding a transform that aligns the two object models together. We do this by first using SIFT features to match every object-view from one model to the other, this is done using the method described in Chapter 3. The match that has the largest number of feature match pairs is then used to generate an approximate transform using the method described in Section 5.3.4. After the two object models are aligned using SIFT features, we perform an ICP pass to align the two separate surface point clouds of the object models. At the conclusion of this, the two object models are merged by combining the lists of object-views and the surface point clouds. In this way we are able to build a model

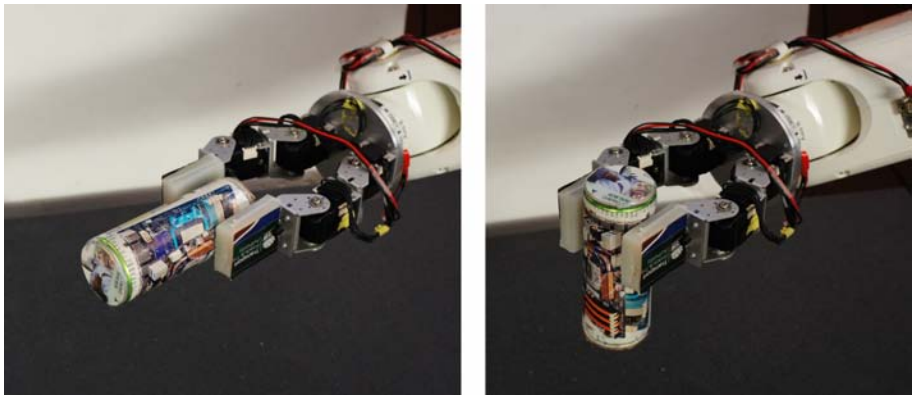

Figure 5.4.3: The robot holding the cylinder object in two different orientations and positions. The robot performs the object-view extraction and aligning process with both, to create two models which are then combined into one. By doing this the robot can compensate for blind spots in the individual model caused by gripper occlusion.

of an object that includes all of the object's surface and aspects.

5.4.4 Results

To test the object-view stitching method, we used three different objects. The first object is a box with highly textured sides and dimensions $5.5cm \times 8.4cm \times 11.9cm$. The second object is a cylinder, also with highly textured sides, of height $13cm$ and diameter $6cm$. The final object is a toy truck $25cm$ long, $12cm$ wide, and with a maximum height of $12cm$. The trucks has a small box attached on the back to enable the robot gripper to securely grasp the object. The majority of the body of the truck is not textured and composed of only two colours.

The robot performed the object reconstruction method described in this chapter on each of these objects, extracting the object-views and stitching them together to form a coherent model. The surface point clouds of the reconstructed models of the different objects are shown in Figure 5.4.4 (box), Figure 5.4.5 (cylinder), and Figure 5.4.6 (truck). It can be seen that the point clouds for all three objects closely match the shape of each object, indicating that the object-views were stitched together correctly.

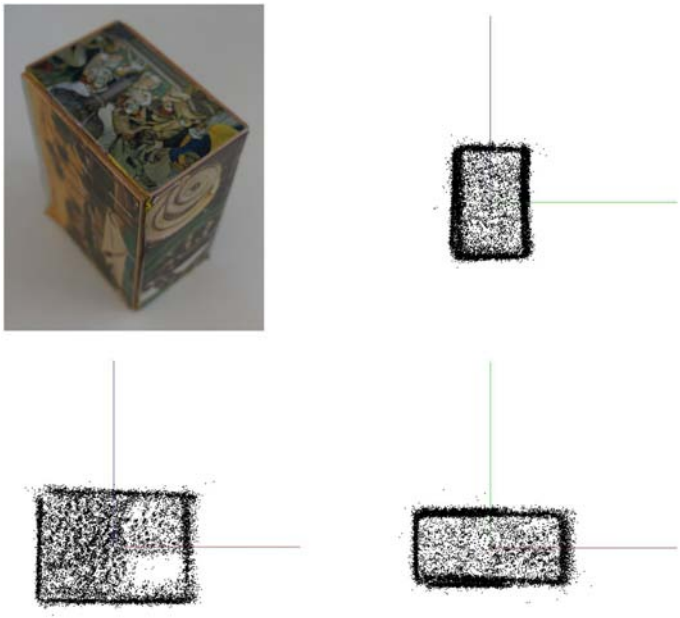

Figure 5.4.4: The box object (top left) and different views of the reconstructed surface point cloud.

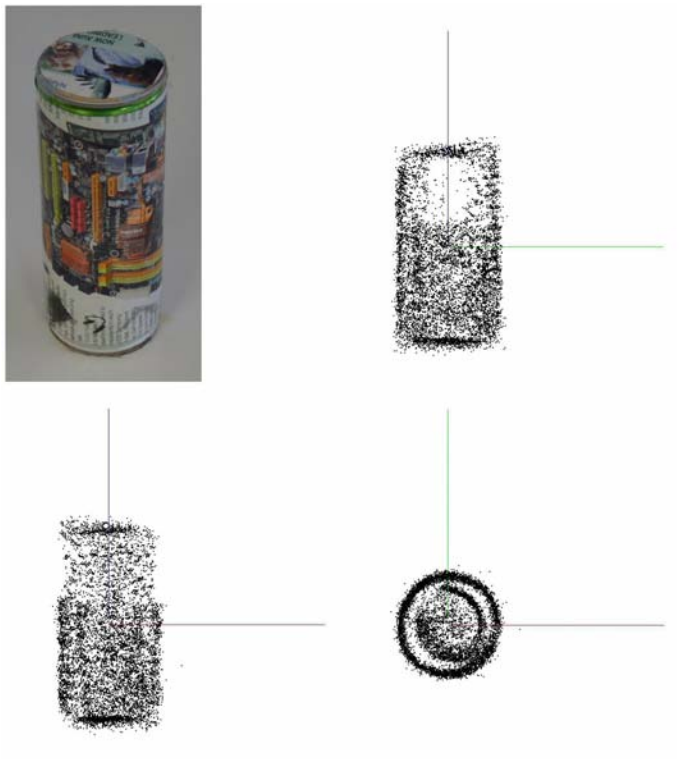

Figure 5.4.5: The cylinder object (top left) and different views of the reconstructed surface point cloud.

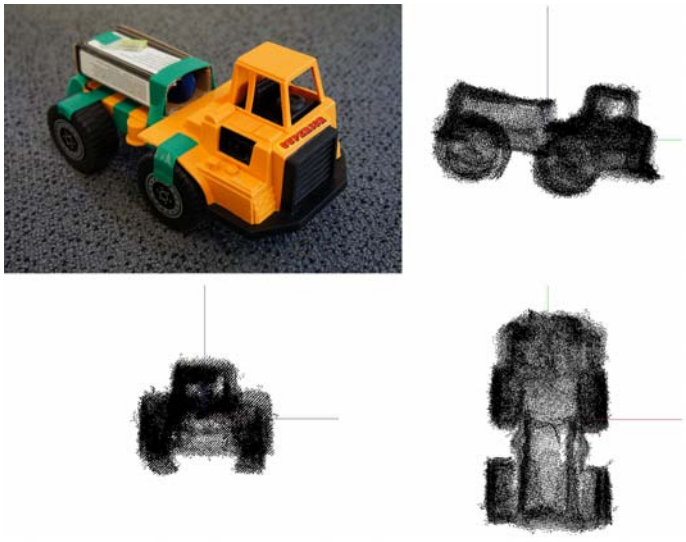

Figure 5.4.6: The truck object (top left) and different views of the reconstructed surface point cloud.

5.5 Shape Fitting

A dense surface point cloud contains a large amount of information about the shape of an object. However, a more compact shape representation is more suitable for some applications (eg: collision detection). For this reason we fit a geometric model to the point cloud. The complexity and type of geometric model can vary greatly, depending on the class of shapes that need to be represented and the application domain. For example, a very simple geometric model is a sphere. However, this is a very restrictive model that is only able to accurately represent a single shape. At the other end of the spectrum, a triangular mesh can be fitted to a point cloud. A triangular mesh can represent arbitrary 3D shapes, but has the disadvantage that fitting a mesh to a point cloud is more complex than fitting a simpler model (eg: a sphere). Other tasks, such as collision detection, are also more complex to perform on a triangular mesh.

Our approach uses a super-ellipsoid to model an object's shape. Super-ellipsoids are a subclass of super-quadrics, and are a popular method for representing basic shapes in computer graphics [120]. The surface of a super-ellipsoid is defined as all points (x, y, z) that satisfy the following equation:

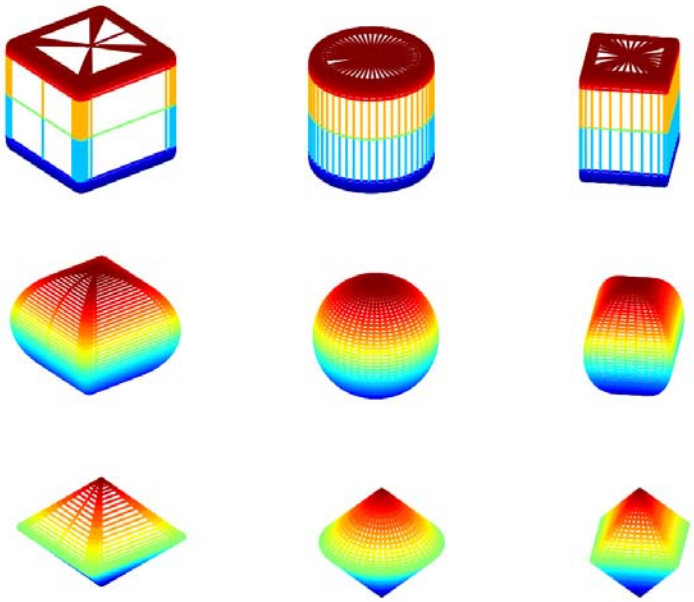

Figure 5.5.1: Example shapes that can be represented by the parametric super-ellipse geometric model.

$$F(x,y,z) \equiv \left[\left(\frac{x}{A}\right)^{\frac{2}{e_2}} + \left(\frac{y}{B}\right)^{\frac{2}{e_2}}\right]^{\frac{e_2}{e_1}} + \left(\frac{z}{C}\right)^{\frac{2}{e_1}} = 1 \tag{5.5.1}$$

The internal volume of the super-ellipsoid is defined as all points such that $F(x,y,z) \leq 1$. There are five parameters ~~which~~ that define the shape and size of a super-ellipsoid. These are A, B, C, e_1, and e_2. The parameters A, B and C define the extents of the shape in the x, y, and z dimensions, respectively. The parameters e_1 and e_2 define the general shape of the surface, Figure 5.5.1 shows several example super-ellipsoid shapes.

5.5.1 Model Fitting Method

To fit a super-ellipse model to the object's surface point cloud we find a set of parameters that minimise~~s~~ (set is singular) the distance between the model surface and the point cloud's points. The total number of parameters of the geometric model is 12. Five of these parameters ~~encapsulate~~ determine the super-ellipsoid shape, and seven ~~encapsulate~~ determine the point cloud alignment transform. The transform parameters consist of four

describing a rotation transform (in the form of quaternions) and three describing a translation. The purpose of the transform is to align the point cloud with the super-ellipsoid.

The next step is to formulate the objective function. Let the point cloud points be the set $\{p_0, p_1, \ldots, p_n\}$, where each point is a 3D world position $p_i = (x_i, y_i, z_i)$. The objective function is defined as the average radial distance between the transformed point cloud and the surface of the super-ellipsoid. For calculating the distance between the super-ellipsoid surface and a point we use the following radial distance function [121, 122]:

$$RadialDist(x, y, z) = \sqrt{x^2 + y^2 + z^2} \cdot \left| 1 - F^{-e_1/2}(x, y, z) \right|, \tag{5.5.2}$$

where the function F is defined in the equation 5.5.1. The overall objective error function of a super-ellipsoid shape and transform is defined as:

$$error = \frac{1}{n} \sum_{i=0}^{n} RadialDist\left(M(p_i + T)\right), \tag{5.5.3}$$

where M is the rotation matrix (derived from the quaternion parameters of the transform) and T is the translation vector component of the point cloud transform. The next step is to find the 12 parameters (which determine transform and super-ellipsoid shape) that minimise this error function. There are many different methods [123, 124] that can be used to do this. For ease of implementation we used a Random Restart Nelder-Mead optimisation method, performing ten restarts of five hundred iterations each. These parameters were chosen *ad hoc*. On completion, the twelve parameters that yield the lowest objective error function across the point cloud is taken as the final super-ellipsoid geometric model of the object.

say more about how you chose and evaluated them

5.5.2 Results

We performed the super-ellipsoid fitting method on the box and cylinder objects. The box object point cloud resulted in a superellipsoid with the following parameters:

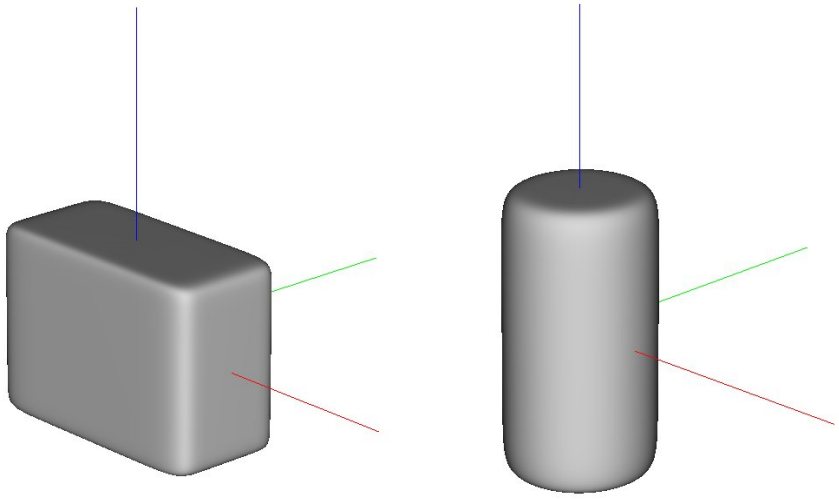

Figure 5.5.2: The superquadric shapes fitted to the box (left) and cylinder (right) surface point clouds.

e_1 : 0.11, e_2 : 0.22, A : 6.09, B : 4.41, C : 2.93. The super-ellipsoid fitted to the cylinder object had the following parameters: e_1 : 0.19, e_2 : 0.96, A : 6.64, B : 3.46, C : 3.38. The fitted shapes are shown in Figure 5.5.2. These geometric models closely correspond to the true shapes of the corresponding objects.

5.6 Object Recognition and Localisation

One of the main ~~applications~~ uses of the reconstructed object model is to allow the robot to recognise and localise the object in a scene. The model consits of a set of object-views, a surface point cloud, and a fitted geometric model. The scene, as observed by the Kinect sensor, is taken to consist of a set of SIFT features (each with an associated 3D world space position) and a point cloud. We use the SIFT feature and point cloud data of the model to recognise the target object and, if it is present in the scene, determine its position and orientation.

Our approach is to first use the SIFT features of the object-views to recognise the object and find its approximate pose, followed by using the object's surface point cloud to refine the pose estimate using ICP. The object transform calculated from the SIFT feature matches provides a good initial guess for the ICP alignment stage.

The ICP refinement stage is then able to use a greater amount of surface information (from the dense surface point cloud) to calculate a more accurate object pose.

First, we try to match every object-view to the scene based on SIFT features, using the matching method presented in Chapter 3. We take the object-view with the highest number of features that match the scene features as the best candidate view. If the best candidate view has less than four matching features, we use that a signal that the object is not present in the scene.

The result of the feature match between the best candidate object-view and the scene is a set of feature match pairs, each pair consisting of an object feature and a corresponding scene feature. Each object feature has a 3D position in object space, and each scene feature has a 3D position in world space. The next step is to find an object transform T (consisting of a 3×3 rotation matrix and a translation vector) such that the distance between the transformed object feature and the corresponding scene feature is minimised. This is done using the Singular Value Decomposition method described in Section 5.3.4.

Next, we use this transform as an initial approximation for the object's pose for a refinement stage using ICP. We apply the transform to all of the surface points of the object, and perform an ICP alignment alignment between the scene point cloud and the surface point cloud. The final output is the transform describing the pose of the object in the scene.

5.6.1 Results

To test the recognition and localisation performance of the reconstructed object models we carried out a series of tests. Each test involved placing the target object in the scene in one of six positions and in one of three orientations (each of which corresponds to a different aspect facing the camera). Each test is carried out three times, for a total of 54 iterations per object. Furthermore, these tests are carried out both with and without the ICP refinement stage. For each test we record the position

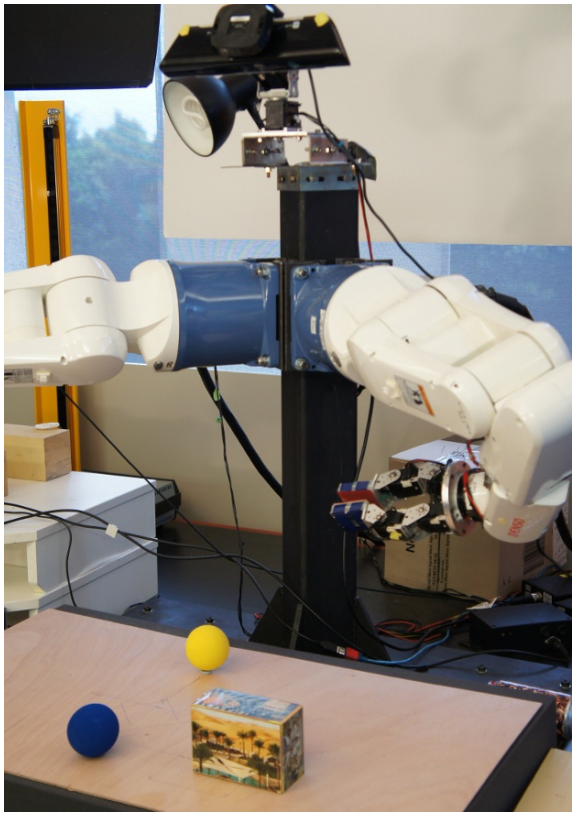

Figure 5.6.1: The test object (in this case the box object) is placed in the scene at a predefined position and orientation. The robot localises the object using SIFT feature matching and ICP refinement.

error of the object and the angular error of the orientation. The average errors are presented for each aspect facing the camera. Figure 5.6.1 shows an example experimental set up for the box object.

The results for the box and cylinder objects are presented in Tables 5.1 and 5.2 respectively. With the truck ~~object~~ we only used two orientations for testings, with wheels on the table, and with the truck on its back. These results are presented in Table 5.3.

It can be seen that the accuracy of the object localisation is high for the box and cylinder objects, with a clear improvement when ICP is used for refinement. With the truck object, however, accuracy is very poor. We discuss the reasons for this in the next section.

Box Pose Estimate Errors

	SIFT Only		SIFT+ICP	
	Position Err.	Orientation Err.	Position Err.	Orientation Err.
Aspect 1	1.35cm (0.08)	1.32° (0.43)	0.82cm (0.19)	0.99° (0.38)
Aspect 2	2.09cm (0.31)	1.39° (0.56)	1.00cm (0.18)	1.58° (0.44)
Aspect 3	1.96cm (0.38)	7.66° (2.58)	0.87cm (0.14)	1.44° (0.39)

Table 5.1: The average position and orientation errors of localising the box object in a scene. This is done three different aspects of the box facing the camera. Localisation is performed using only SIFT features, and using SIFT features followed by Iterative Closest Point (ICP) refinement. The 95% confidence interval is given in parentheses in the same units as the error value.

Cylinder Pose Estimate Errors

	SIFT Only		SIFT+ICP	
	Position Err.	Orientation Err.	Position Err.	Orientation Err.
Aspect 1	2.32cm (0.54)	7.44° (6.70)	0.83cm (0.22)	2.57° (2.46)
Aspect 2	1.09cm (0.35)	5.50° (1.77)	0.86cm (0.21)	2.11° (0.80)
Aspect 3	2.63cm (0.34)	7.73° (4.58)	0.69cm (0.19)	3.09° (1.14)

Table 5.2: The average position and orientation errors of localising the cylinder object in a scene. This is done three different aspects of the cylinder facing the camera. Localisation is performed using only SIFT features, and using SIFT features followed by Iterative Closest Point (ICP) refinement. The 95% confidence interval is given in parentheses in the same units as the error value.

Truck Pose Estimate Errors

	SIFT Only		SIFT+ICP	
	Position Err.	Orientation Err.	Position Err.	Orientation Err.
Aspect 1	7.11cm (3.38)	17.87° (11.00)	4.72cm (2.55)	14.61° (8.99)
Aspect 2	11.11cm (2.77)	30.65° (12.94)	6.82cm (2.32)	30.88° (12.71)

Table 5.3: The average position and orientation errors of localising the truck object in a scene. This is done two different aspects of the box facing the camera. Localisation is performed using only SIFT features, and using SIFT features followed by Iterative Closest Point (ICP) refinement. The 95% confidence interval is given in parentheses in the same units as the error value.

5.7 Conclusion and Future Work

We have presented an object reconstruction system building on the feature matching and segmentation methods introduced in Chapter 3 and Chapter 4. This system enables a robot to autonomously learn an all aspect appearance and shape model of an object in a complex environment. This model is then used to recognise and localise the object in a scene.

However, there are some issues with the presented approach and areas that can be improved in future work. First, the object recognition and localisation performance for the truck object was very poor. This is in contrast to the good performance for the box and cylinder objects. This poor performance is due to the way light interacts with the truck object. The truck is largely composed of flat, single coloured facets that produce specular reflections. This means that the apparent colour of a given facet is strongly dependent on the intensity and angle of incidence of the light (example shown in Figure 5.7.1). The relative colour/brightness of adjacent facets is also strongly affected by the relative light position. As a result the appearance of the truck in the camera image changes significantly depending on the lighting conditions. This in turn means that the SIFT features (which are calculated based on small image neighbourhoods) of the learned model can differ greatly compared to the features of the truck placed in the scene, resulting in poor recognition and localisation performance. A further problem is caused by the fact that a single image of the truck object has many similar SIFT features. This is because there is very little difference in the local appearance of the different corners and edges of the flat facets of the truck. The box and cylinder, on the other hand, have diffuse surfaces that have coloured patterns and textures. In this case, the appearance of the object in the camera image is less dependent on the relative light position, which results in similar SIFT features between the learned model and a test scene image. This problem can potentially may be overcome using a different local image feature algorithms (eg: SURF [46]), a mixture of different types of features, or by

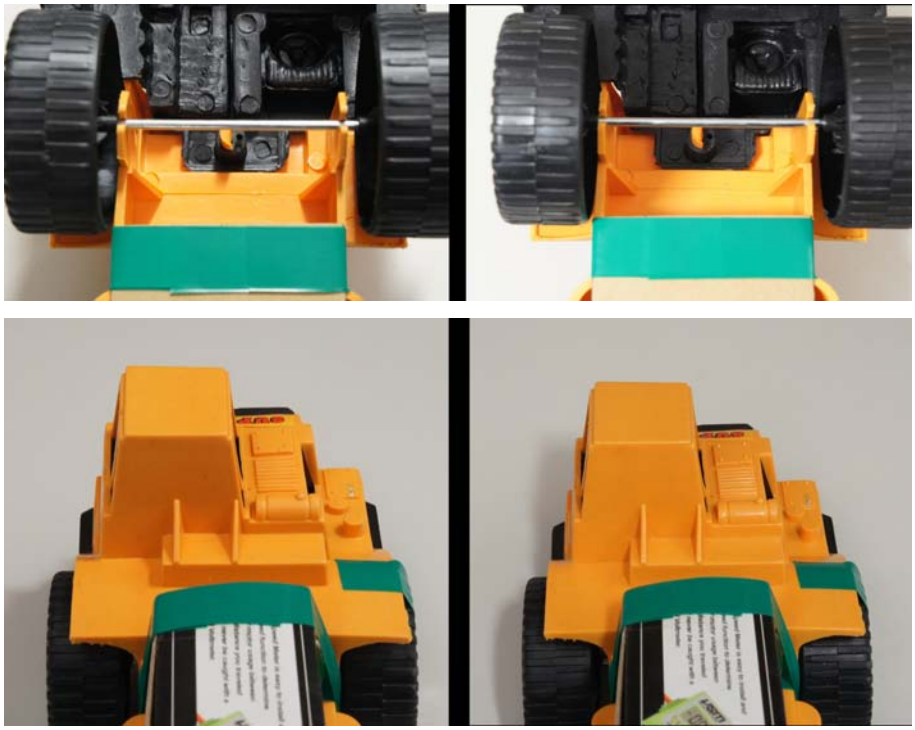

Figure 5.7.1: With flat coloured specular surfaces, the image colour depends strongly on the scene lighting. As the lighting changes, the object's local appearance can change significantly. This is shown in the above pair of images, with different light positions in the left and right images.

learning the appearance of the objects in several different lighting conditions.

A further area for improvement is how the robot observes the different aspects of the object during object-view acquisition. Currently, the object is moved in a linear motion while being rotated around a single axis. This motion is repeated with the object grasped in an orthogonal grasp for the robot to observe the areas of the object occluded by the gripper. The two separate models are then merged. One of the problems of this approach is that the directions from which the object is observed are not uniformly distributed, with some aspects of the object having less coverage than others. This potentially reduces the feature matching accuracy, and subsequently the object localisation accuracy, if the object is observed from an angle that was not covered during learning. A solution to this problem is to keep track of the aspects of the object which have been learned and to dynamically choose which aspects to observe, also taking into account occlusions by the robot arm [89, 125].

The third issue is our use a relatively simple geometric model for fitting to the

object's surface point cloud. We use a super-ellipsoid parametric model, which can represent a narrow class of shapes. In future this should be improved by fitting a more general geometric model, such as a triangular mesh, to the point cloud. This would allow a wider variety of shapes to be represented.

Finally, we have not rigorously addressed the issue of loop closure (Section 5.4.2) when stitching together the different object-views. Loop closure refers to the problem of handling accumulated errors as the adjacent object-views are stitched together, which results in a potentially large final error. We found that in our case the accumulated errors were not large enough to be noticeable. However, it is possible that with different objects it can be a problem. There are many existing approaches for loop closure for stitching together different views and point clouds [126, 127, 128], involving for example global graph optimisation methods, and should be considered for future versions of this system.

In this chapter we have demonstrated that the methods introduced in Chapter 3 and Chapter 4 can be used to build a system for a robot to autonomously learn an all-aspect appearance and shape model of a previously unkown object. The next step is for the robot to learn the physical properties of an object. Knowing shape and appearance is not sufficient to reliably manipulate and use the object to complete a task. The object may have some physical or internal properties that affect its behaviour, properties which cannot be determined by passive observation (eg: centre of mass). In this case, the robot must actively interact with the object, performing experiments to build an object model of these properties. This problem is addressed in the next chapter.

Chapter 6

Discovery of Object Properties

6.1 Introduction

The previous chapters dealt with the task of a robot learning to recognise a new object and determining its 3D shape. However, to effectively manipulate and use an object, knowing the shape and appearance is not always sufficient. The robot needs an internal model of the object which, in addition to encapsulating the visual appearance and shape of the object, also encapsulates other physical properties. Examples of physical properties include the coefficient of friction of the object's surface, the centre of mass and the weight of the object, whether the object is rigid or articulated, etc. Having an accurate model of such properties can allow the robot to effectively manipulate and use the object to accomplish a task.

To determine some physical properties, passive observation with a robot's sensors (such as a camera) may not be possible. Instead, we present an approach in which the robot performs some experiments on an object, and the outcome of these interactions provides information as to the underlying physical properties.

For example, take the task of finding the centre of mass of an object. The centre of mass is dependent on the internal composition of the object and cannot be determined by a robot using vision alone. Instead the robot can take the object and drop it onto a flat surface from various starting orientations. The resulting orientation of

the object, after each drop, provides the robot with information about the location of the centre of mass ~~of the object~~.

Our approach involves using a physics simulator to generate hypotheses about the object's properties and predictions of the outcome of an action on a simulated model of the object. We can then match the outcome of a real world action to the simulated outcomes to determine which hypothesis ~~model~~ most accurately describes the real world object. The simulated outcomes can also be used to determine the most informative action ~~to be performed, minimising the total~~ that will minimise number of actions the robot ~~needs to~~ must perform to reliably determine the object's physical properties. Figure 6.1.1 shows a simple representation of the hypothesise-simulate-perform loop of our ~~approach~~ method. The ~~end~~ final result is a simulated model that accurately describes ~~and encapsulates~~ the physical properties of the real ~~world~~ object. This simulated model can then be used to ~~applications such as predicting~~ predict the outcome of an action on the object or plan~~ning~~ tool use tasks using the object.

The ~~method~~ idea (it's not the same method) of internally generating hypotheses that are then checked for consistency with real world experiments has previously been applied in biology to discover metabolic pathways in yeast [129]. The use of internal simulation to make predictions of action consequences and hypotheses about the world has parallels with cognitive theory of the mind [130].

We present a description of our general approach (Section 6.2), followed by ~~presenting~~ several ~~concrete~~ experiments in which the robot learns the physical properties of an object by performing various actions (Section 6.3). Next we demonstrate a tool use task in which the robot uses the learned physical properties to plan and ~~accomplish~~ achieve a set goal (Section 6.4). The chapter concludes with a discussion of the results (Section 6.5) and possible avenues for future work (Section 6.6).

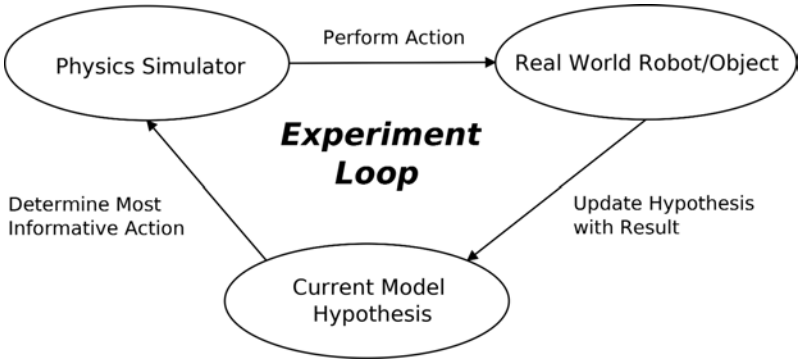

Figure 6.1.1: The experiment loop view of our system. The robot uses a physics simulator to make hypotheses about the outcome of actions on an object and uses this to determine the most informative action. This action is then carried out by the robot on the real world object, the result of which is then used to update the probability distribution over hypothesis models. This process is repeated several times to determine the most accurate model of the object.

6.2 Active Robot Learning Framework

6.2.1 Problem Definition

The problem we solve is how can a robot best determine the underlying physical properties of an object by performing some actions and observing the outcomes. First we define the problem. Let there be an object that the robot can localise in a scene using the approach from Chapter 5. The 3D shape of this object has also been determined using the previously presented method. The goal is to find a model, M, that accurately describes the physical properties of this object.

Let us define the following:

- an *a priori* defined list of possible models $H = \{h_1, h_2, \ldots, h_n\}$ that can describe the object;
- discrete probability distribution $C = \{c_1, c_2, \ldots, c_n\} \mid c_i = P(M = h_i)$ representing the confidence of a corresponding model matching the object;
- set of actions $A = \{a_1, a_2, \ldots, a_m\}$ that can be carried out by the robot;
- set of possible action outcomes $R = \{r_1, r_2, \ldots, r_l\}$, where the result of each action $a \in A$ is some $r \in R$.

The goal is to determine the model $h \in H$ that most accurately describes the object, such that $M = h$. To do this, the robot repeatedly performs actions $a \in A$, observes the results $r \in R$, and updates the confidence distribution C. After several iterations, the model h_x with the highest corresponding confidence c_x is taken to be the model that best describes the object ($M = h_x$).

6.2.2 Approach Overview

The challenges involved include: representing the object models, determining the best actions to perform, and updating the probability distribution over the possible models using the outcome result of an action. In this section we describe our general object model learning approach which addresses these challenges. Further on in this chapter we present some concrete implementations, demonstrating the effectiveness and suitability of our approach to a variety of problems.

The learning algorithm consists of several distinct steps:

1. First a result probability model of all of the available actions is built (described in Subsection 6.2.4).
2. This is followed by generating a discrete set of possible result labels R (described in Subsection 6.2.5).
3. After these initialisation steps, the robot performs the experimentation loop. In this loop the robot:
   - 3.1 determines the most informative action based on the current confidence distribution and action probability models (described in Subsection 6.2.6),
   - 3.2 performs the best action,
   - and updates the confidence distribution using the result.
4. After a desired number of iterations, the model with the highest confidence is returned as the best model M.

Algorithm 6.1 Object model learning algorithm.

input: set of actions $\rightarrow A$
input: set of possible models $\rightarrow H$

$C_{current} \leftarrow \left\{ \frac{1}{|H|} \right\}$
$buildActionModels(A, H)$
$R \leftarrow buildResultLabels(A, H)$
repeat
  $a_{best} \leftarrow mostInformativeFrom(A, C_{current})$
  $world\_state \leftarrow performAction(a_{best})$
  $r \leftarrow classifyToResultLabel(world\_state, R)$
  $C_{new} \leftarrow updateModelConfidence(C_{current}, r)$
  $C_{current} \leftarrow C_{new}$
until $max\_iterations$

$highest\_confidence \leftarrow 0$
forall h_i **in** H
  if $c_i > highest\_confidence$ **then**
    $highest\_confidence \leftarrow c_i$
    $M \leftarrow h_i$
  endif
endfor

output: most accurate object model $\leftarrow M$

These steps are outlined in a flowchart in Figure 6.2.1 and in further detail in Algorithm 6.1

6.2.3 Object Model Representation

The object model refers to the way in which the robot internally represents an object. A particular model determines the different ~~object~~ properties that can be represented, as well as being used to predict the ~~result~~ outcomes of the robot actions. There are many different representations that can be used for the object model, depending on the particular application domain in which the model will be used, and the properties of the object ~~that need to be encapsulated~~ are required in the model. The nature of the object model representation will also affect how it can be used for planning and prediction.

The representation can be very low level. For example, a mapping between motor input commands and the resulting change in pixel values in the camera image

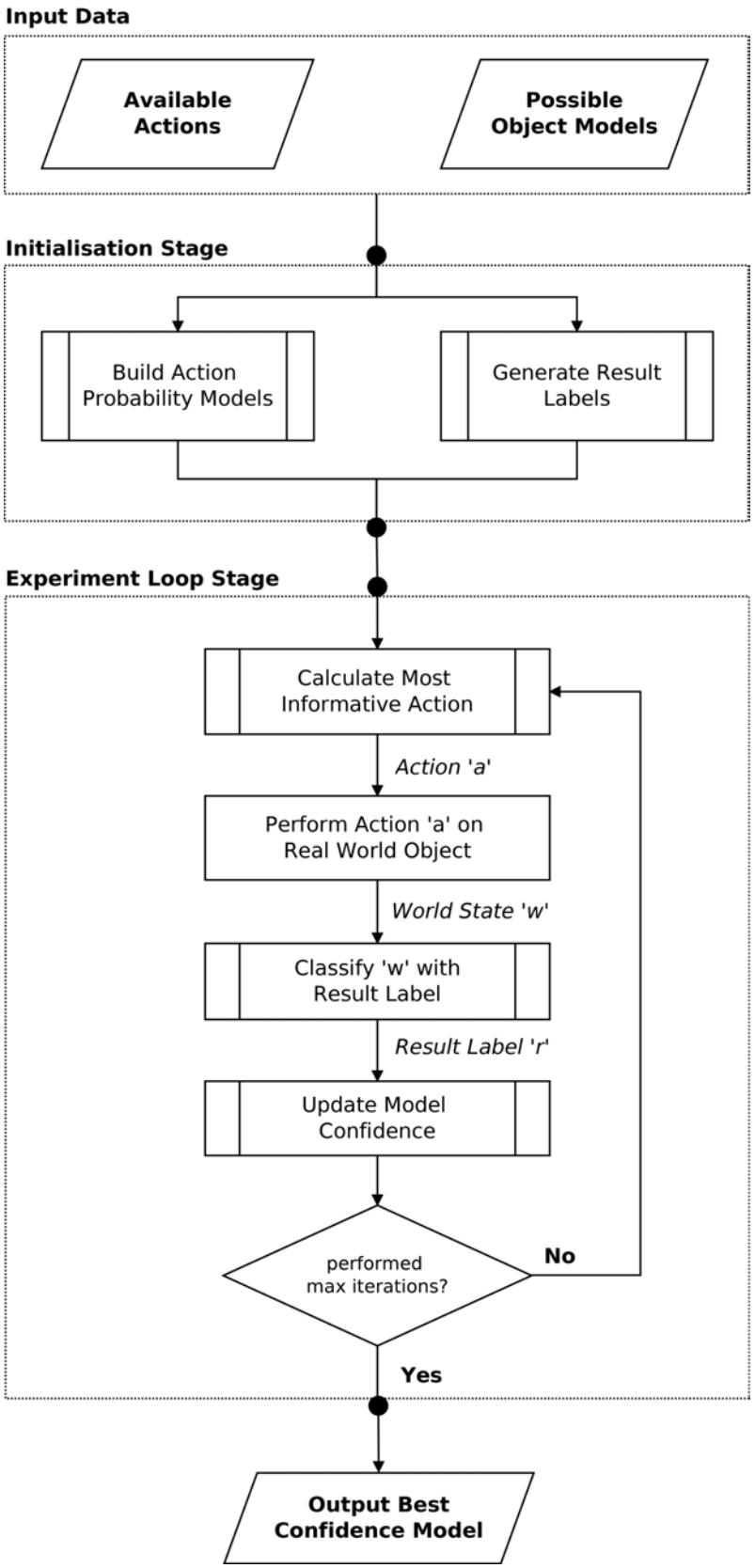

Figure 6.2.1: Flowchart ~~representing the work flow~~ of the object model learning system.

due to the object being moved [131]. An example of a higher level ~~object model representation is to learn~~ feature is the motion vector of an object when it is bumped by the robot manipulator from ~~a~~ some direction [101]. However, this type of low level explicit representation ~~would~~ generalise/s poorly ~~to~~ for different circumstances. For example, in the case of learning the movement vector after a bump from ~~a~~ some direction, if the model is learned on a flat surface it would perform poorly if the object ~~was~~ were located on a sloped surface. To overcome these limitations, our ~~approach~~ method uses a physics simulator to model an object.

~~We use a simulated entity in a 3D physics engine as the object model representation~~ repetitions. There are several advantages ~~of~~ to this approach. First, many laws of physics are encoded in the simulated environment. The robot does not need to learn from scratch concepts such as gravity or friction.

Second, a ~~wide~~ variety of object properties can be represented, depending on the sophistication of the physics simulator ~~used.~~ This includes properties such as weight, centre of mass, coefficient of friction, hinges and axles, etc.

Third, the physics engine ~~environment~~ can be used to carry out simulated actions on an object model to build a posterior distribution over results for each action, to build hypotheses about the object's underlying properties, and decide on which actions it should carry out in the real world to efficiently determine an accurate model for the object. We elaborate on this in Subsection 6.2.4.

Finally, a ~~learned~~ physics engine model can be used ~~for~~ to predicting the outcome of an action in ~~a wide variety of~~ many different scenarios, even if the ~~scenario~~ environment differs ~~to~~ from the one in which the object model was learned. For example, the robot may learn the centre of mass of a box ~~object~~ by dropping it onto a flat surface and observing the results. This learned model can later be used for accurately predicting how the box will land if it is dropped onto a sloped surface.

6.2.4 Action Probability Model

The action probability model refers to the likelihood of observing a particular result when performing an action on an object with properties matching a specific model.

To determine the model that most accurately describes the object, the robot performs a series of actions. The type of motion and manipulation involved in each action will depend largely on the nature of the object, the physical properties of interest, and the methods of manipulation available to the robot. However, the overall structure of the action probability model and how it fits into the overall framework remains the same.

When the robot performs an action $a \in A$, the outcome is some world state that can be labeled $r \in R$, where R is a discrete and finite set of possible results (how a result is labeled is described in the following section). Using this outcome, we can use Bayesian inference to update the probability distribution, C, over the possible models. Bayesian inference [132] is the application of Bayes rule to calculate the change in a belief distribution due to newly acquired evidence. This is expressed ~~in~~ as: ~~the following equation:~~

$$P(B|E) = \frac{P(E|B) \cdot P(B)}{P(E)} \tag{6.2.1}$$

where $P(B|E)$ is the posterior probability (the confidence of belief B being true given the new evidence E), $P(B)$ is the prior probability (the certainty of B being true before the observation was made), $P(E)$ is the probability of observing evidence E independent of B, and $P(E|B)$ is the probability of observing evidence E, if B is known to be true. In the context of our system, after performing action a and observing result r, using this formula, the robot can update the probability distribution C as follows:

$$\forall c_i \in C : c_i^{new} \leftarrow \frac{P\left(r|h_i, a\right) \cdot c_i^{old}}{P\left(r|a\right)} \tag{6.2.2}$$

where $P\left(r|h_i, a\right)$ is the probability of observing the result r when performing action a on an object whose model matches h_i, $P\left(r|a\right)$ is the probability of observing result r independent of the object model when performing action a, c_i^{old} is the prior confidence level $P\left(h_i\right)$, and c_i^{new} is the updated posterior confidence. This update follows from the fact that the probability distribution C is our confidence that a model corresponds to the object. That is, $c_i^{old} = P_{prior}\left(h_i\right)$ and $c_i^{new} = P_{posterior}(h_i|r, a)$.

To perform this update, we must first determine the values of the terms $P\left(r|h_i, a\right)$ and $P\left(r|a\right)$ for an action a. We ~~need to~~ must calculate the probability $P\left(r|h_i, a\right)$ of observing a result r when an action is performed on an object described by a particular model, and the probability $P\left(r|a\right)$ of a result when the action is performed independent of the object's model.

The naïve method is to have the robot determine the result probabilties by performing every action $a \in A$ on every possible object with model $h \in H$ multiple times. However, this would be time consuming depending on the size of A and H. Due to the noise and errors ~~of~~ in manipulation and perception, each action must be performed ~~a number of~~ several times to determine the underlying result probability distribution. Furthermore, this method requires the availability of a reference object for every model $h \in H$ for the robot to perform the actions on a known model. For these reasons, this approach is not feasible.

We take an alternate approach, using the physics engine to simulate the outcome of actions on all of the possible object models. These simulated results are used to build the probability model of each action. This requires that each of the actions $a \in A$ can be simulated with acceptable accuracy, such that the outcome of a simulated action is representative of the expected outcome when performed by the robot on the actual object. Additionally, the simulated action should account for

the errors and noise present in the system when the robot carries out the action in the real world. We present some specific examples of action simulation in Section 6.3.

Calculating the probability model for each action can be done as an initialisation step, prior to the robot interacting with the object. To do, this each action is performed on each possible object model in simulation. This is done ~~multiple~~ several (how many?) times ~~such~~ so? that the noise of the action is taken into account by the probability distribution over results. When a simulated action outputs a particular result label, a corresponding counter is incremented for both the object model dependent probability $P(r|h_i, a)$ and independent probability $P(r|a)$. At the end these counters are normalised according to the number of simulations performed, and become the action's model probability distributions. The process is summarised in Algorithm 6.2.

6.2.5 Result Labels

We calculate and query Eh? the discrete result probability distributions $P(r|a)$ and $P(r|h_i, a)$. To do this, we represent the outcome of an action as a discrete result label $r \in R$, were R is a discrete set of all possible outcomes. However, the world state at the conclusion of an action is typically represented as a multi-dimensional ~~and~~ continuous vector describing various aspects of the world (object pose, robot arm joints, etc). This world state may contain a large amount of information not related to the performed action and the object model. To convert the world state into a result label r, we extract the relevant information (for example the object pose) and match it to a set of result bins, each of which has a corresponding label $r \in R$. This label is the output result of an action, which can then be used for updating the confidence distribution over possible object models.

The result bins are predefined as an initialisation step of the overall system. The specific format of the result bins is heavily dependent on the application domain. A

Algorithm 6.2 Calculating action result probabilities.

input: set of actions $\rightarrow A$
input: set of possible models $\rightarrow H$
input: set of possible action results $\rightarrow R$

forall a **in** A
  $independent\_probabilities[a] \leftarrow \{0\}$
  forall h **in** H
    $model\_probabilities[a][h] \leftarrow \{0\}$
    repeat
      $world\_state \leftarrow performSimulatedAction(a, h)$
      $r \leftarrow classifyToResultLabel(world\_state, R)$
      $independent\_probabilities[a][r] \leftarrow independent\_probabilities[a][r] + 1$
      $model\_probabilities[a][h][r] \leftarrow model\_probabilities[a][h][r] + 1$
    until $num\_iterations$

    forall x **in** $model\_probabilities[e][h]$
      $x \leftarrow \frac{x}{num\_iterations}$
    endfor
  endfor

  forall x **in** $independent\_probabilities[a]$
    $x \leftarrow \frac{x}{|H| \cdot max\_iterations}$
  endfor
endfor

output: result probabilities $\forall a \in A : P(r|a) \leftarrow independent\_probabilities$
output: result probabilities $\forall a \in A, h \in H : P(r|h, a) \leftarrow model\_probabilities$

simple example of generating the result bins is by discretising the space of possible (x, y, z) object positions into uniform cells. After an action is carried out, we match the object pose to the closest discretised cell and return the corresponding label. A more complex example is performing many simulations and clustering the results using an algorithm such as K-means [133] to generate the result bins. Different methods of generating discrete result labels are presented in detail in Section 6.3.

6.2.6 Choosing an Action

simulation or real?

The robot performs experiments on an object and observes the results to determine some underlying properties of that object. Howevere, not all actions that the robot may perform are equally informative. Figure 6.2.2 illustrates this concept. The worst case scenario is if an action produces the same result regardless of the properties of the object. In this case the information gained is zero. Our goal is to choose the most informative action, given the current level of uncertainty over the possible object models. This reduces the number of actions that must be performed by the robot to determine the object's properties, minimising wear and tear on the hardware and reducing learning time.

To determine the information gain of an action we calculate its expected Kullback-Leibler divergence (KL divergence) [134]. KL divergence is a distance measure between two probability distributions P and Q, denoted as $D_{KL}(P, Q)$. For discrete probability distributions P and Q the KL divergence is defined as:

$$D_{KL}(P, Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)} \tag{6.2.3}$$

Formally, it is

~~It is formally~~ a measure of the expected number of bits ~~that are~~ needed to encode samples from one distribution when using a code based on the other. In Bayesian statistics the KL divergence between two distributions can be used as measure of the information gained moving from one to the other. Furthermore, in the domain

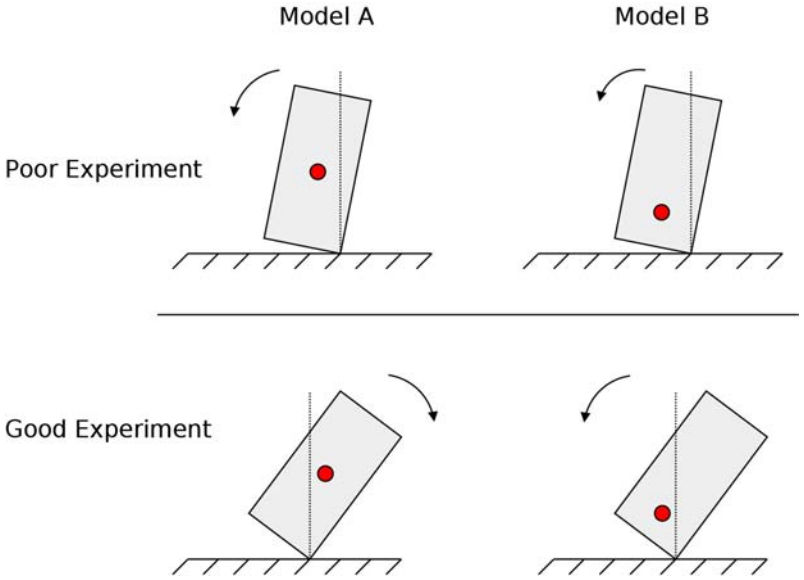

Figure 6.2.2: Not all actions that a robot can perform to determine the physical model of an object are equally informative. In the above example, if the robot needs to determine whether the centre of mass of a box (indicated by the red circles) is located in the middle or at one of the ends, dropping the box in the top orientation will not provide much information, as the outcome will be the same in both instances (in both cases the centre of mass is on the same side of the vertical line drawn from the pivot point). The bottom action is more informative as the outcome will depend on the centre of mass of the box.

of Bayesian optimal experimental design [135] a common aim is to maximise the KL divergence between the prior and posterior. In our system, the robot chooses to perform the experiment with the highest expected KL divergence from the current object model confidence distribution, thus maximising the expected information gain.

To calculate the expected KL divergence of an action we use its probability model (presented in Subsection 6.2.4) and the current confidence distribution over object models. The outcome of an action is some $r \in R$, and for every possible outcome there is a corresponding new confidence distribution (computed using Equation 6.2.2). To calculate the expected KL divergence, we take the weighted sum of the KL divergences between the current confidence distribution and every possible resulting distribution, weighted by the probability of the corresponding result. Figure 6.2.3 illustrates this concept for a simple example with only two possible outcomes for an action. Algorithm 6.3 ~~provides~~ gives a step by step description of calculating the

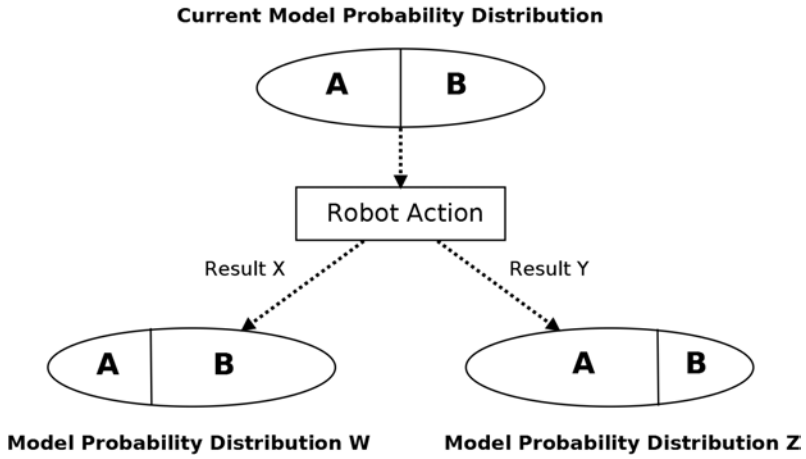

Figure 6.2.3: To determine the usefullness of an action, we find the expected KL-divergence between the prior probability distribution (Current Probability Distribution) and the resulting posterior probability distribution after the action is performed and result known. In the simple example above (with two possible outcomes X and Y, and two hypothesis models A and B) this is equal to $P\left(ResultX\right) \cdot D_{KL}\left(CPD, PDW\right) + P\left(ResultY\right) \cdot D_{KL}\left(CPD, PDZ\right)$, where CPD is the current probability distribution, PDW is the resulting distribution if the outcome is X, and PDZ is the resulting distribution if the outcome is Y.

expected information gain.

6.2.7 Relation to Active Machine Learning

Say what it is. Don't ask the reader to skip to the references and back

Our system can be viewed as a modification of a well-known active machine learning approach [136]. Active learning is a form of supervised machine learning where the algorithm actively queries the supervisor for labels of specific data points, to reduce the training time and improve accuracy. It is motivated by problems where unlabelled data is abundant, but labeling each training instance is expensive and/or time consuming. With active learning the algorithm queries the supervisor to label the training instances which are most informative.

what is a query strategy?

While there are various query strategies [137], in our case the most relevant is *query-by-committee* (QBC)[138]. The QBC framework features a committee of competing classifiers, a labelled set of training data points on which they are trained, and an unlabeled set of data points. The aim is to determine the most accurate classifier. The competing classifiers are used to generate hypothetical labels for the

Algorithm 6.3 Calculating the expected KL divergence of an action.

input: action $\rightarrow a$
input: set of possible models $\rightarrow H$
input: set of possible results $\rightarrow R$
input: current confidence distribution $\rightarrow C_{current}$

$expected\_information \leftarrow 0$
forall r **in** R
  $weight \leftarrow 0$
  $C_{new} = \{c_1^{new}, c_2^{new}, \ldots, c_n^{new}\}$
  forall h_i **in** H
    $weight \leftarrow weight + c_i P\left(r|M = h_i, a\right)$
    $c_i^{new} \leftarrow \dfrac{P\left(r|M = h_i, a\right) c_i^{current}}{P\left(r|a\right)}$
  endfor
  $expected\_information \leftarrow expected\_information + weight \cdot D_{KL}(C_{current}, C_{new})$
endfor

output: expected KL divergence of the action $\leftarrow expected\_information$

unlabeled data points. The data point on which they disagree is considered to be the most informative and is then queried to be labeled by the supervisor. The outcome of this query is used to update the accuracy of each classifier. Kullback-Leibler divergence is one particular method used as a measure of disagreement between classifier models, for example used for active learning of document class classification [139].

Our system can be viewed as a QBC active learning task if we consider each of the simulated object physics models as a classifier, each of the actions that can be performed as an unlabeled training data point, and the robot carrying out a particular action on the actual object as the supervisor. First, we generate hypothetical labels for each of the actions (unlabeled training data points) by performing them in simulation with each of the different models (competing classifiers). The result of each is a probability distribution over result labels. We use the KL divergence measure to choose the action that produces the most disagreement. We then query the supervisor to obtain a label, which in our case is the robot carrying out the chosen action on the actual object. This label is then used to update the model likelihood distribution, indicating the most accurate object model.

6.3 Experimental Results

To test the effectiveness of the object modeling framework described in this chapter, we perform three experiments. In the first experiment (Subsection 6.3.1) we show that the method can be used to learn the centre of mass of an object by dropping it from different orientations onto a flat surface and observing the results. The second experiment (Subsection 6.3.2) investigates the application of the framework to the task of learning the wheel configuration of a box-cart object. This is done by releasing the box-cart on a sloped ramp in a particular orientation and observing the trajectory of the cart. The third experiment (Subsection 6.3.3) demonstrates the general applicability of the learning method by showing that it can be used to learn object properties not limited to physical attributes. In this case, the object is a Lego Mindstorms[1] robot programmed to perform a particular movement behaviour in response to some stimulation. The stimulation is in the form of activating a light sensor of the Lego robot. The goal of the learning robot is to determine the particular behaviour model by using a flashlight to illuminate the light sensors and observing the resulting movement of the Lego robot.

The final experiment demonstrates the benefits of learning a predictive model of an object. This is done by having the robot plan and perform a tool use task using an object and a learned physics engine model of that object. The learned model is used to plan an appropriate action in simulation and then carry it out to accomplish a goal.

6.3.1 Centre of Mass Experiment

6.3.1.1 Overview

In this experiment we apply the presented active robot learning method to the task of finding the centre of mass of an object. This is done with two different objects: a box and a cylinder. The robot must determine the centre of mass of

[1] http://mindstorms.lego.com

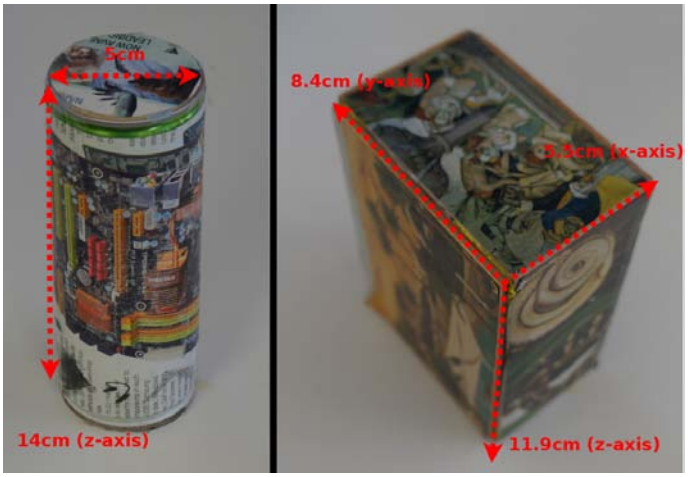

Figure 6.3.1: The cylinder (left) and box (right) objects used in the *Centre of Mass Experiment*. Each can be unweighted, in which case it has a centre of mass in the centre, or can have weights added internally to offset the centre of mass.

each by dropping them onto a flat surface. The dimensions of the box object are $5.5cm \times 8.4cm \times 11.9cm$, the cylinder object is $13cm$ in height (along the z-axis) and $6cm$ in diametre (see Figure 6.3.1). Both the cylinder and the box may have weights added internally to change the location of their centre-of-mass. The drop orientations and heights are chosen to have the highest expected information gain. This is done using the previously described method of finding the most informative action from a set of available actions. A physics ~~engine~~ simulator is used to simulate the various possible object models and actions.

To find the centre of mass of the box and cylinder objects, the robot performs the following steps:

1. localise the target object in the scene;
2. grasp the object with the robot gripper;
3. calculate the information gain of each action given the current model likelihood distribution;
4. carry out the most informative action by dropping the object from the appropriate height and orientation;
5. determine the resulting pose of the object;

Who says what is appropriate?

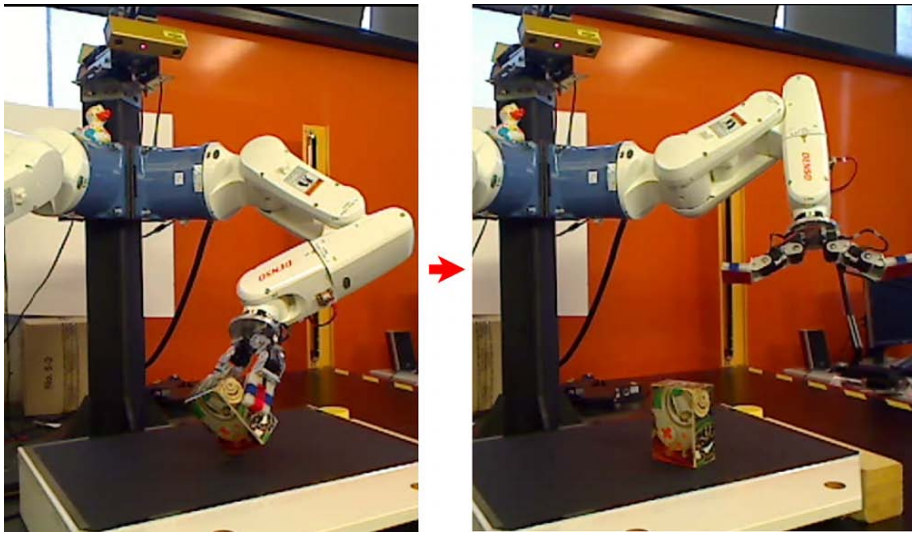

Figure 6.3.2: The before (left) and after (right) state of a single iteration of the *Centre of Mass Experiment*. The robot positions the box object in an appropriate orientation and height above the flat table-top, and then releases it. The resulting orientation of the object provides information on the location of its centre of mass.

6. classify the resulting pose by matching it to a result label;

7. update the model likelihood distribution using the outcome probabilities of the performed action and the result label.

The object is localised in a scene using the method described in ~~previous~~ chapter using a complete aspect graph of SIFT [34] features and a depth camera. In the case of a box and cylinder objects, grasping is performed along the longest axis of each object with the vector between the two gripper pads parallel to the ground plane. Figure 6.3.2 shows a before and after image of the robot performing a drop experiment on the box object.

~~In this section~~ we present the following: the possible models that can describe the objects and the actions that the robot can perform (Section 6.3.1.2), the simulation method used to determine the action probability models (Section 6.3.1.3), the method of labelling the result of each drop action with a discrete label (Section 6.3.1.4), and the performance results of applying the object model learning method to determining the centre of mass of a box and cylinder object (Section 6.3.3.4).

6.3.1.2 Possible Object Models and Actions

The goal for the robot is to determine which of the models, from an *a priori* defined set of possible models, best fits the object in question. This is done by performing actions from a pre-defined pool. For this experiment, we define three possible centre-of-mass models for both the box and cylinder objects (shown in Figure 6.3.3). In the case of the box object, the possible models have their centre-of-mass either in the middle of the box, or offset half-way along the main axis. The coordinates (in centimetres) of the centre-of-mass of the three possible box models are: $(0, 0, 0)$, $(0, 0, 3)$, and $(0, 0, -3)$. The coordinate $(0, 0, 0)$ corresponds to the centre of the box. In the case of the cylinder object, the possible models also have their centre-of-mass either in the middle, or offset half-way along the main axis. The coordinates (in centimetres) of the centre-of-mass of the three possible cylinder models are: $(0, 0, 0)$, $(0, 0, 3.5)$, and $(0, 0, -3.5)$. The coordinate $(0, 0, 0)$ corresponds to the centre of the cylinder.

The actions that the robot can perform on the box and cylinder is dropping each from a specified orientation and height. The drop height is defined as the distance between the flat tablet-top and the nearest point on the object's surface. The drop orientation is defined as the upward facing direction of the object at the time of release, as dropping an object onto a flat surface has rotational symmetry around the vertical axis. The pool of available actions is specified *ad hoc* to consist of 100 instances, each with a randomly generated drop height (in the range $0.5cm$ to $2cm$) and drop orientation. The drop height is generated by choosing a random value in the specified range $[0.5, 2.0]$, the drop orientation is generated by choosing a random unit vector.

The robot performs actions drawn from this pool to determine which of the pre-defined models best matches the box and cylinder object.

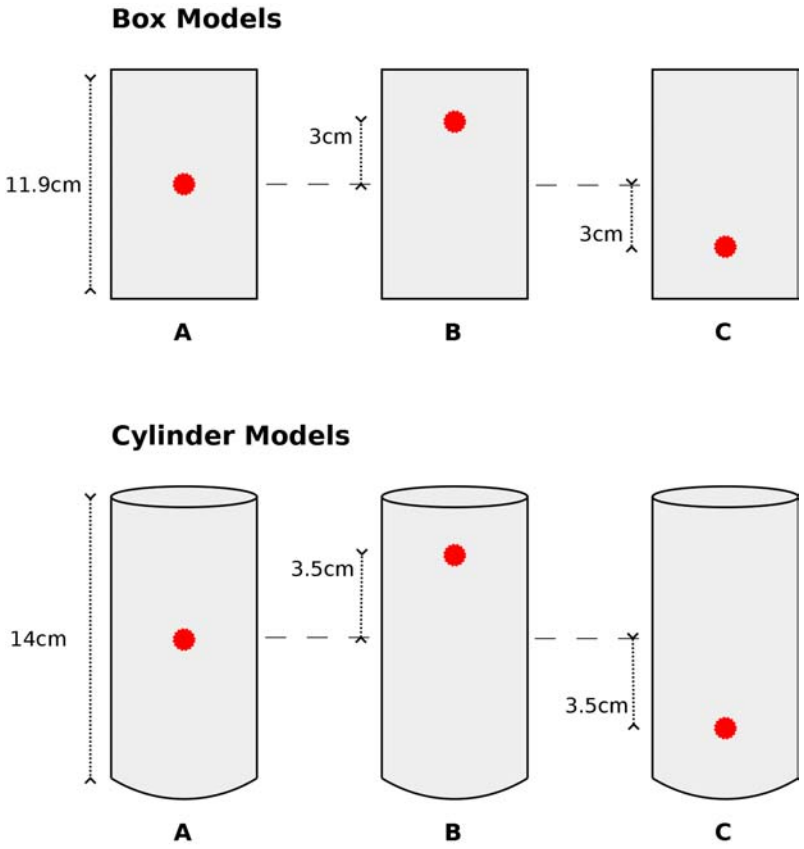

Figure 6.3.3: The potential models for the box and cylinder objects. Each can be described by one of 3 models, with the centre of mass (denoted by the red circle) in the middle (model A), or with the centre of mass offset half way to the end along the longest axis of the object (models B and C).

6.3.1.3 Simulation Method

Our method ~~to~~ for learning the properties of an object involves simulating the outcome of various actions performed on the different possible object models. This is done to calculate the action probability model (ie: the probabilities $P(r|h,a)$ and $P(r|a)$ discussed in Section 6.2.4). This is then used to update the confidence distribution after an action is performed and a result observed, as well as to calculate the expected KL divergence of the different actions to perform the most informative one.

To simulate the actions and find their conditional probability distributions we use the Bullet Physics Engine[2] (version 2.76), ~~The Bullet Physics Engine~~ which is capable of simulating the interactions between various bodies. This includes collision detection between bodies, soft-body and rigid-body dynamics, as well as being able to simulate various types of joints, hinges, and axles.

For the *Centre of Mass Experiment*, the flat workspace surface is simulated by an infinite plane with friction set to 1.0, and the gravity vector set to $(0, 0, -9.8)$ m/s^2. The box and cylinder are simulated by a *ConvexHull* rigid body, with friction set to 0.7, restitution parameter set to 0.01, linear damping parameter set to 0.05, and angular damping set to 0.5. These parameters were chosen ~~*ad hoc* and result in~~ to give visually realistic simulations. The mass of the simulated objects does not affect the outcome of a simulated actions and is set to 1.0.

For simulation, we simplify the action significantly. We do not simulate the robot arm grasping the object, moving it into position, and releasing the object. Instead, the simulated object is set to the appropriate orientation and height above the ground plane (as specified by the drop parameters) and released. To model the noise and errors resulting from the robot manipulating an object in the real world, the simulated object's position and orientation are perturbed from the values specified by the simulated action. Gaussian noise is added to the drop height with a mean of $0cm$ and standard deviation of $0.5cm$, while the orientation is rotated around a random vector by an amount specified by a Gaussian noise variable with

[2] http://www.bulletphysics.com

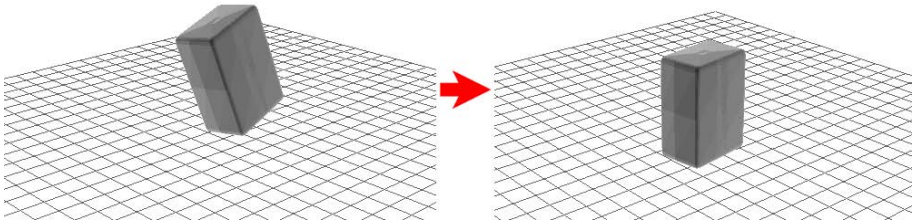

Figure 6.3.4: Before (left) and after (right) screenshots of a simulated action, dropping a box object in a physics engine from a particular orientation and height. The image on the left shows the world state when the box is released, the right image shows the final pose of the box after it has come to a stop.

too many ad hoc choices

mean 0° and standard deviation of 10°. These values are also chosen *ad hoc*, but are conservative over-estimates of the errors of the robot manipulation of the objects.

After setting the simulated object to the appropriate pose, the physics simulator is run for 300 frames, each frame corresponding to $\frac{1}{30}$ of a second. Figure 6.3.4 shows a before and after screenshot of dropping a box object in a physics engine simulation. At the conclusion of a simulation, the resting pose of the simulated object is converted to a discrete result label. This is discussed in the following section.

6.3.1.4 Result Classification

Our method requires the outcome of an action to be expressed as a discrete result label $r \in R$, where R is a finite set of all possible result labels (see Section 6.2.5). This is due to the fact that the action model is represented as a discrete probability distribution over the result labels (see Section 6.2.4). For a robot action, the outcome is generically some world state described by a continous variable. By discretising the relevant information, this world state can be converted into a result label.

In this experiment, the relevant part of the world state is the resting pose of the object. For effectively classifying the results of a drop, the full 6 degrees of freedom pose of the object is not required. Only the upward facing direction of the object is needed. The upward direction vector is calculated using the following formula:

$$up\_vector = M^{-1}\,(0,0,1)^T,$$

where M is the object's world space orientation matrix and $(0,0,1)^T$ is the world "up" direction. The next step is to discretise the outcome up-vector of the object to form a result label. This is done by matching the up-vector to a predetermined set of result bins.

There are several ways in which these result bins can be defined. The most straight forward is to subdivide the unit sphere of up-vectors into uniform sections, with each section corresponding to a result bin. This approach, however, is not optimal for objects such as a box and cylinder. In the case of a six-sided box, there are only six possible up-vectors that can be the outcome of a drop action (corresponding to one of the sides facing upward). Similarly with a cylinder, only a subset of all up-vectors is possible in the outcome. In this case, the cylinder can have either of its flat ends facing up, or the rounded side.

To generate the appropriate result bins for the *Centre of Mass Experiment* we use a clustering approach. The reasoning behind this is to group together up-vectors that are logically similar into the same buckets. This clustering stage is performed at initialisation, prior to the experiment loop or the building of the action models. This involves running many simulations of actions on all of the possible object models, recording the outcome result, and finally clustering all of the recorded results into bins. This scheme is outlined in Figure 6.3.5.

In the case of this experiment, this is done by taking the outcome up-vectors from many simulated runs and grouping together vectors that are within 45° of one another. Each up-vector is viewed as a graph node, and any two vectors within 45° of each other are joined by a graph edge. This is done because the box and cylinder objects have their faces separated by 90° intervals. The resulting graph of up-vector nodes will have multiple connected components, with each component corresponding to a result bucket. In the case of the box, the result is six connected components, corresponding to the six sides. In the case of the cylinder, the result

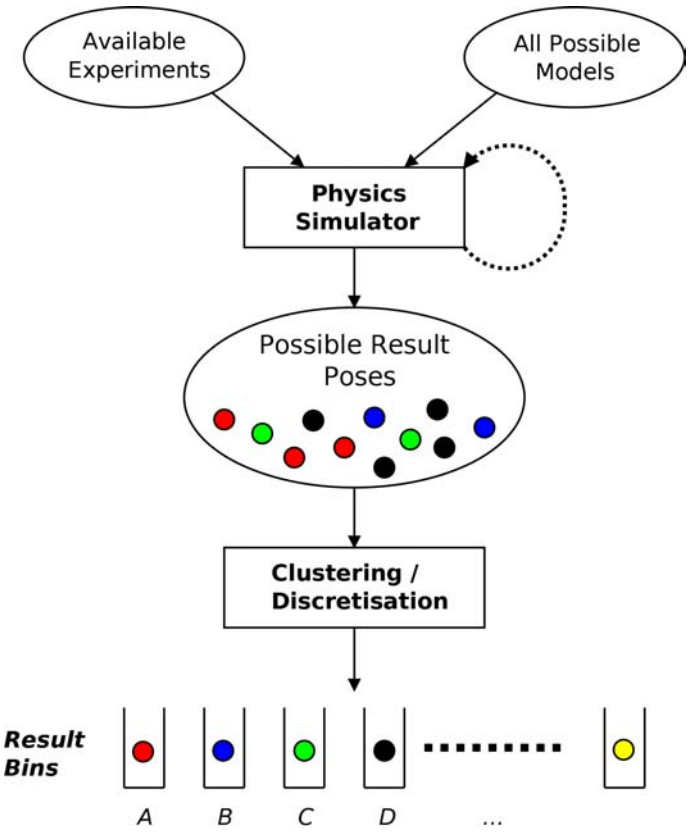

Figure 6.3.5: This diagram outlines how result bins are generated. The available actions are run multiple times on all of the available object models in simulation. The set of output results is then clustered or discretised to create a smaller set of possible result bins. These are later used for result labeling.

is three connected components, two for the flat ends and one for the round side. Each of these groups of up-vectors are stored as a result bucket. When an action is performed (or simulated when building the action models), the resulting object up-vector is taken and compared against these result buckets. The result bucket which contains the closest up-vector is taken to be the matching result label.

6.3.1.5 Performance Results

We test the performance of the learning system by having the robot carry out multiple runs of learning the object model. In each run the robot performs a series of actions, updating the confidence distribution over the possible models using the result of each action. By performing the most informative action at every stage and using the learned action models, the robot's confidence distribution should have the

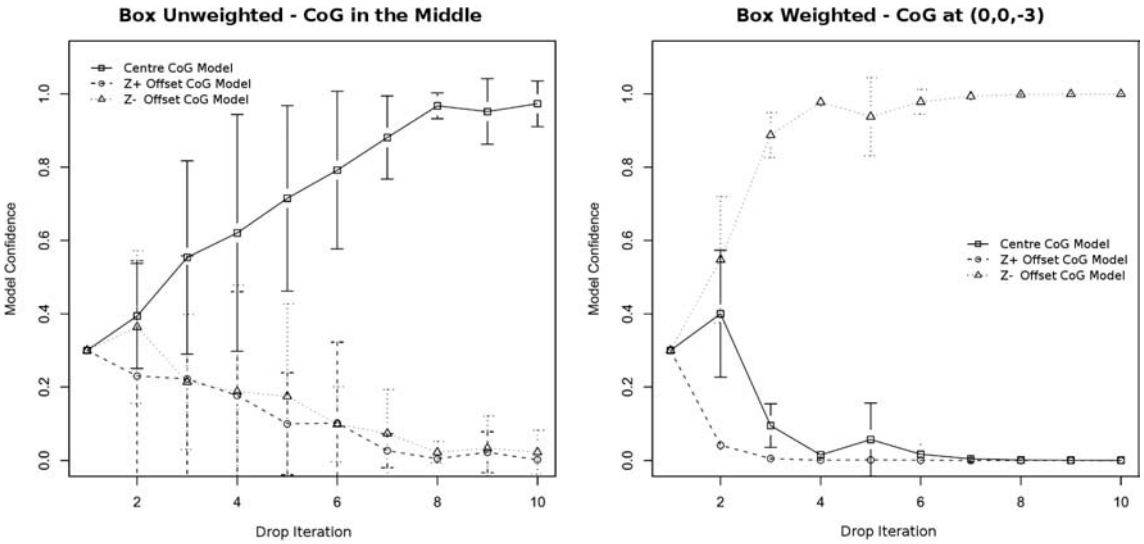

Figure 6.3.6: The results of the *Centre of Mass Experiment* on the box object, showing the change in the confidence distribution after a number of drop iterations. The robot performs multiple drops of the box and updates its model hypothesis based on the results. The left graph corresponds to the trials with an unweighted box with the centre of mass at $(0, 0, 0)$, with the matching object model indicated as *Central CoG Model*. The right graph corresponds to the trials with a weighted box with the centre of mass at $(0, 0, 3)$, with the matching object model indicated as *Z+ CoG Model*. The error bars represent the 95% confidence interval of across 8 independant runs.

matching object model's confidence rise to 1.0 and the remaining models' confidence fall to 0.0 as more actions are performed.

The robot performed eight separate trials of determining the object model of the box. In each trial the robot performed nine drops. This procedure was done with the box unweighted (centre-of-mass in the middle), and with the box having weights added internally (centre of mass offset $-3cm$ from the middle along the z-axis). Figure 6.3.6 shows how the robot's model confidence distribution changed during the nine drops. It can be seen that after even a small number of drops, the confidence of the object's correct corresponding model rises above the remaining incorrect models.

With the cylinder object, the robot performed eight seperate trials, with each trial consisting of six drops. This is done with the cylinder unweighted (centre of mass located in the middle) and with weights added internally (centre of mass

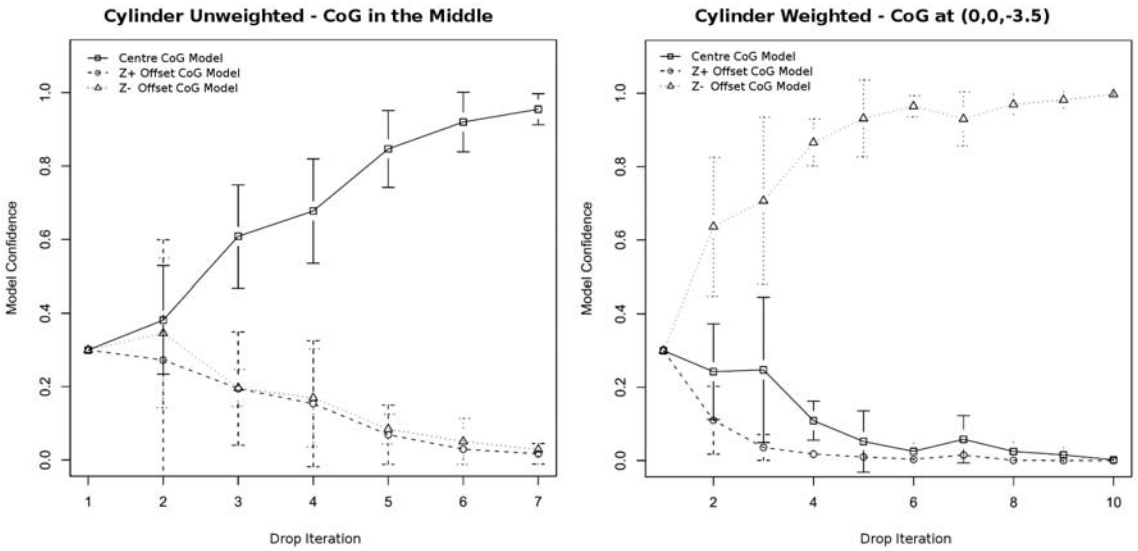

Figure 6.3.7: The results of the *Centre of Mass Experiment* on the cylinder object, showing the change in the confidence distribution after a number of drop iterations. The robot performs multiple drops of the cylinder and updates its model hypothesis based on the results. The left graph corresponds to an unweighted cylinder with centre of mass at $(0, 0, 0)$, the right graph to a weighted cylinder with the centre of mass offset to $(0, 0, -3.5)$ along the main axis. The error bars represent the standard deviation of the hypothesis across 8 independant runs.

offset $-3.5cm$ from the middle along the z-axis). Figure 6.3.7 shows the change in the confidence distribution over the possible models during the trials. We can see from the results that the robot is able to quickly and accurately determine the correct model of the object by performing the action with the highest information gain.

In additional to evaluating the performance of the active learning framework when applied to determining the centre of mass of an object, we investigated the effect of choosing the action with highest expected information gain on the learning rate. A key part of the learning framework is choosing to perform the most informative action, as measured by expected KL divergence. We compared in simulation a robot learning the centre of mass of a box and cylinder objects using the most informative action at every step, and using a random action at every step. It is important to note that this is carried out purely in simulation, the actions are simulated in the same manner as when building the action model. For the best action and random action selection policies, six simulated runs are performed, each with

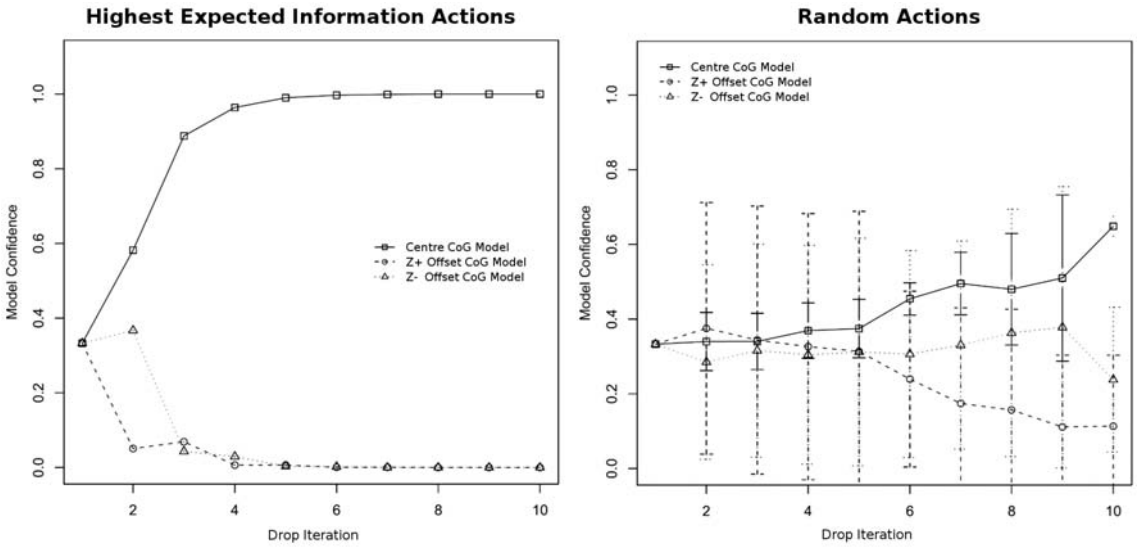

Figure 6.3.8: Learning the box centre of mass using the action with highest expected information gain (left), and using a random action (right) at every iteration. This is performed in simulation, and simulated box has its centre of mass in the centre (corresponding to the Centre CoG Model). It can be seen that by performing the most informative action the confidence distribution quickly converges on the correct object model. When performing a random action, the confidence distribution on average does not converge. The error bars indicate the 95% confidence interval.

nine drop actions performed. The progression of the confidence distribution over the possible object models is show in Figure 6.3.8 for the box object, and in Figure 6.3.9 for the cylinder object. It can be seen that by choosing the most informative action leads to a much faster convergence of the confidence distribution to the correct object model, whereas choosing a random action leads to much worse performance.

6.3.2 Wheel Configuration Experiment

6.3.2.1 Overview

In this experiment we apply the active robot learning ~~framework~~ method to the task of learning the configuration of the wheels on a box-cart (pictured in Figure 6.3.10). The box-cart consists of a rectangular prism body, two wheels attached at one end, and a wooden block at the other. The configuration of the two wheels is unknown to the robot, specifically how they are oriented and whether they are able to rotate. For this experiment the workspace consists of a flat surface and a

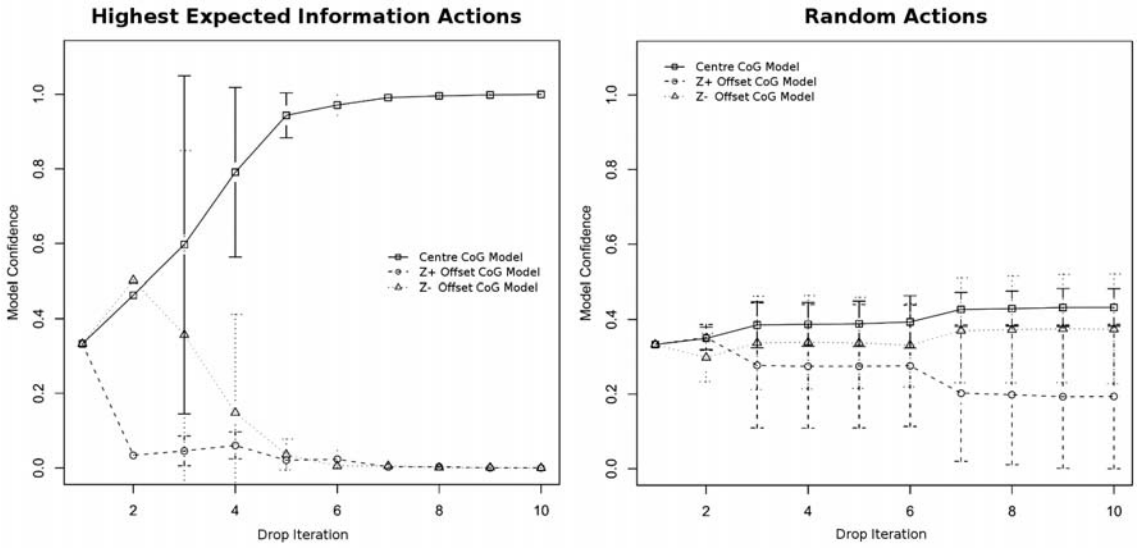

Figure 6.3.9: Learning the cylinder centre of mass using the action with highest expected information gain (left), and using a random action (right) at every iteration. This is performed in simulation, and the simulated cylinder has its centre of mass in the centre (corresponding to the Centre CoG Model). It can be seen that by performing the most informative action the confidence distribution quickly converges on the correct object model, as compared to performing a random action. The error bars indicate the 95% confidence interval.

25° ramp at one end (see Figure 6.3.11). The robot determines the configuration of the box-cart by releasing it on the sloped ramp from various orientations and observing the resting pose of the object. By choosing the release pose with the highest expected information gain, the robot is able to efficiently determine the object model that describes the box-cart. To choose an optimal action and to update the model confidence distribution, a physics engine is used to simulate the outcome of the different actions on the possible object models.

The procedure followed by the robot in this experiment is similar to the *Centre of Mass Experiment*. To find the wheel configuration of the box-cart, the robot repeats the following steps:

1. localise the target box-cart in the scene;

2. grasp the cart with the robot gripper;

3. calculate the information gain of each action given the current model likelihood distribution;

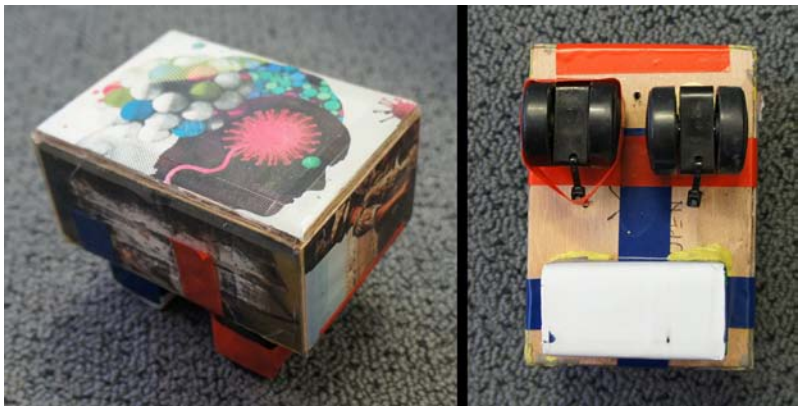

Figure 6.3.10: The box cart object used for the *Wheel Configuration Experiment*. The cart consits of a box, two wheels on one end, and a wooden block on the opposite end. The wheels can be set to point straight ahead or 90° to the side. The wheels may also be prevented from rotating by the application of sticky-tape.

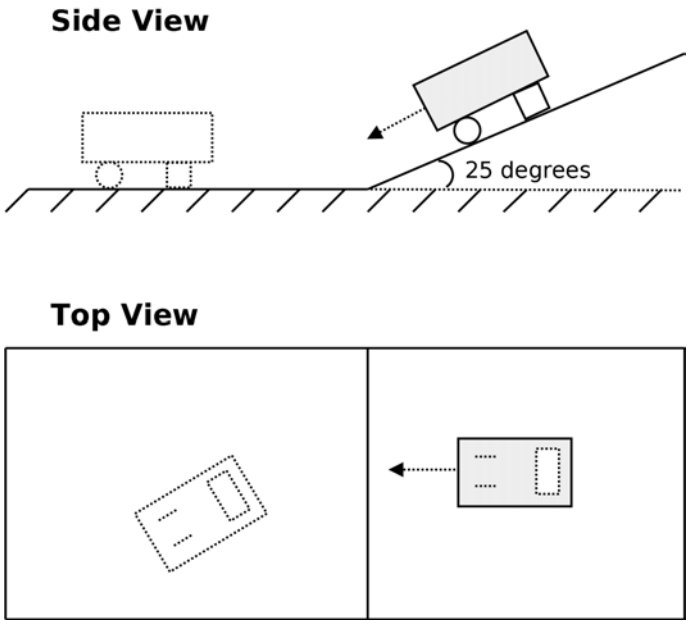

Figure 6.3.11: The workspace layout for the *Wheel Configuration Experiment*. The robot places the box-cart on a 25° ramp and releases it, allowing it to roll down. The final pose of the cart provides the robot with information regarding the configuration of the wheels.

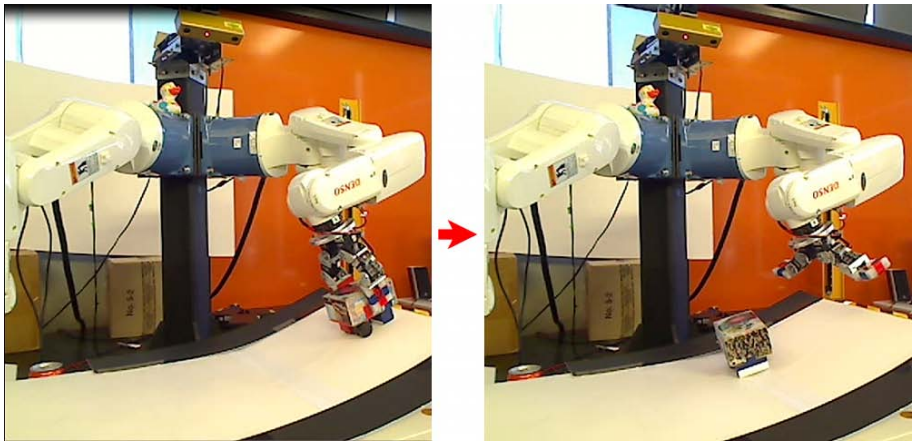

Figure 6.3.12: The before (left) and after (right) state of an iteration of the *Wheel Configuration Experiment*. The robot positions the box-cart object in a particular orientation and position on the sloped ramp, and then releases it. The resulting pose of the cart provides information on the configuration of its wheels.

4. carry out the most informative action by releasing the cart from the appropriate pose on the ramp;

5. determine the resulting pose of the cart;

6. classify the resulting pose by matching it to a result label;

7. update the model likelihood distribution using the outcome probabilities of the performed action and the result label.

The cart is localised in a scene using the method described in Chapter 5, ~~using~~ with a complete aspect graph of SIFT features and a depth camera. Grasping is performed along the longest axis of the top of the cart, with the vector between the two gripper pads parallel to the ground plane. Figure 6.3.12 shows a before and after image of the robot performing a drop experiment on the box object.

~~In this section~~ Next, we will present ~~the following:~~ the possible models that can describe the box-cart and the actions that the robot can perform (Section 6.3.2.2), the simulation method used to determine the action probability models (Section 6.3.2.3), the method of labeling the result of each action with a discrete label (Section 6.3.2.4), and the performance results of using the object model learning method to determining the wheel configuration of a cart object (Section 6.3.2.5).

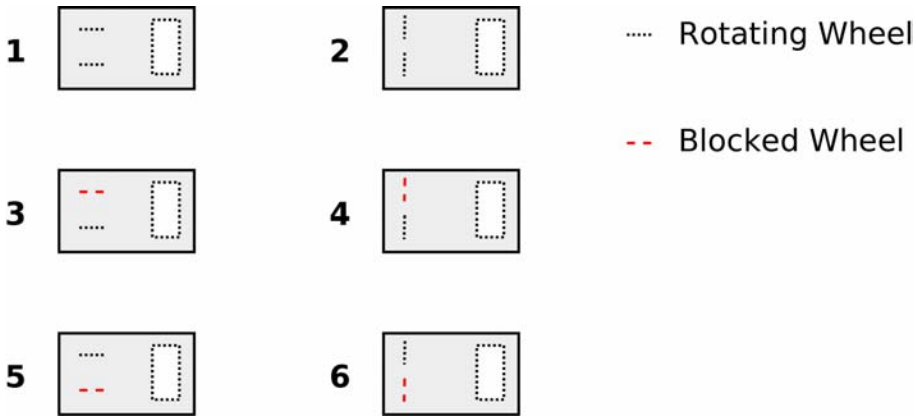

Figure 6.3.13: The above diagram depicts the 6 different potential models of the box-cart. Models 1, 3, and 5 have wheels oriented in the direction of the main axis, models 2, 4, and 6 have wheels oriented at 90° to the main axis. Models 1 and 2 have freely rotating wheels, models 3 and 4 have the right wheel blocked (prevented from rotating), models 5 and 6 have the left wheel blocked.

6.3.2.2 Possible Object Models and Actions

The robot's goal is to determine which model from a predefined set of models best fits the box-cart. For this experiment we define the set of possible models to consists of six different wheel configurations (see Figure 6.3.13). The two wheels of the box-cart may both be pointing straight ahead or both 90° to the side, and they may freely rotate or either of the two wheels may be taped shut (but not both).

The actions that the robot can perform are releasing the cart on the ramp at a particular vertical height and orientation. The cart's wheels and rear block are in contact with the ramp surface at the time of release. The vertical height of the release is defined as the vertical distance of the centre of the cart from the horizontal ground plane. The orientation of the cart is defined as the angle it is facing in the horizontal plane at the time of release. The pool of available actions is specified *ad hoc* to be 210 actions. These uniformly consist of 7 different release heights between $2cm$ and $8cm$ (in $1cm$ increments), and at each height 30 possible orientations in 12° increments. We judged this pool of experiments to provide sufficient coverage of the experiment parameter space, given the workspace layout.

6.3.2.3 Simultion Method

As with the *Centre of Mass Experiment*, to determine the probability models (ie: the probabilities $P(r|h,a)$ and $P(r|a)$ discussed in Section 6.2.4) for each action, we use the Bullet Physics Engine to simulate the outcome for each possible action on each possible object model.

The ramp and flat table surface are simulated with two planes with friction coefficient set to 0.7, the gravity vector set to $(0, 0, -9.8)$ m/s^2. The box-cart object is simulated by two box rigid bodies for the main body and rear block, and two cylinder objects for the wheels. The rear block object is joined to the main box object by a rigid six degrees of freedom constraint, forcing them to behave as a single rigid body. The two cylinders are joined to the box ~~object~~ by hinge constraints. The direction of the hinge constraint is defined by the particular model being simulated (whether the wheels of the model are pointing ahead or to the side). The rear block and wheel friction coefficient is set to 0.7. The mass of the main box object is set to 150 grams and the mass of each wheel and the rear block is set to 20 grams; this matches the measured weight of the real world box-cart components. The angular damping parameter of each wheel is set to either 0.15 or 1.0, depending on whether the wheel in the particular model being simulated is free to rotate or not.

As with the *Centre of Mass Experiment*, we simplify the action during simulation. Instead of simulating the robot arm grasping the object and moving it into position, the simulated object is directly set to the appropriate orientation and position on the ramp, as specified by the action parameters. When the robot performs an action in the real world, there are several sources of noise and error that affect the outcome. This includes errors in the positioning of the box-cart on the ramp, slight variations in the ramp and tabletop surface, variations in the wheel axle's turning resistance, etc. To account for these sources of noise when performing a simulated action, we perturb the release height and orientation as well as perturbing the axle damping parameters for each wheel. The release height and orientation are

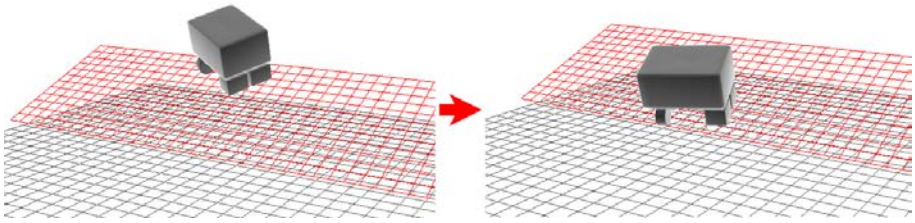

Figure 6.3.14: Before (left) and after (right) screenshots of a simulated action, releasing a box-cart object on a ramp in a the physics engine. The image on the left shows the world state when the box-cart is released, the right image shows the final pose of the box-cart after it has come to a stop.

perturbed by Gaussian noise with mean 0 and standard deviation set to $0.5cm$ and $10°$ respectively. The wheel damping parameters, wheel friction, and rear block friction are each perturbed by uniform noise in the range $[-0.1, 0.1]$.

After settings the simulated box-cart to the appropriate pose, the physics simulation is run for 300 frames, each frame corresponding to $\frac{1}{30}$ of a second. Figure 6.3.14 shows a before and after screenshot of a release action carried out in physics engine simulation. At the conclusion of a simulation, the resting pose of the simulated object is converted to a discrete result label. This is discussed in the next section.

6.3.2.4 Result Classification

Our method requires that the outcome of an action is a discrete result label $r \in R$. This is done by discretising the relevant information from the outcome world state of an action. In the case of the *Wheel Configuration Experiment*, the discretisation is performed using the pose of the box-cart after it has rolled down the ramp and come to rest. The resting pose is constrained to be on the surface of the flat tabletop or ramp, and we assume the box-cart remains upright. Therefore we can simplify the full six degrees of freedom cart pose by expressing the result as the (x, y) position and an angle θ for the orientation of the cart on the tabletop plane. For labeling the simplified pose we use a uniform discretisation scheme. The workspace is divided into cells, $3cm \times 3cm$ in size, and orientations are divided into $10°$ increments. Each

of these cells corresponds to a single result bin. An (x, y, θ) result pose can be mapped to a result bin by finding the particular cell which contains the result pose. The matching cell for the box-cart pose is taken to be the result label.

6.3.2.5 Performance Results

We test the performance of the learning system by having the robot carry out multiple runs learning the box-cart model. In each run the robot performs a series of actions, updating the confidence distribution over the possible models using the result of each action.

The robot performed six separate trials of determining the object model of the box. In each trial, the robot rolled the cart down the ramp six times. This was repeated with the box-cart's wheels set to all six of the possible configurations (see Figure 6.3.13). The results of these trials are presented in Figure 6.3.15, showing how the robot's model confidence distribution changed during the course of the trials. Each of the separate graphs corresponds to a different configuration of the box-cart object. It can be seen that very quickly the confidence of the object's corresponding model rises above the remaining incorrect models.

In additional to evaluating the learning performance, we investigated the effect of choosing the action with highest expected information gain on the learning rate. We compare in simulation a robot learning the box-cart's wheel configuration using the most informative action at every step, and using a random action at every step. It is important to note that this is carried out purely in simulation. The actions are simulated in the same manner as when building the action model. For the best action and random action selection policies, six simulated runs are performed, each with nine actions performed. The progression of the confidence distribution over the possible object models is shown in Figure 6.3.16. It can be seen that by choosing the most informative action results in a much faster convergence of the confidence distribution to the correct object model, whereas choosing a random action leads to much worse performance.

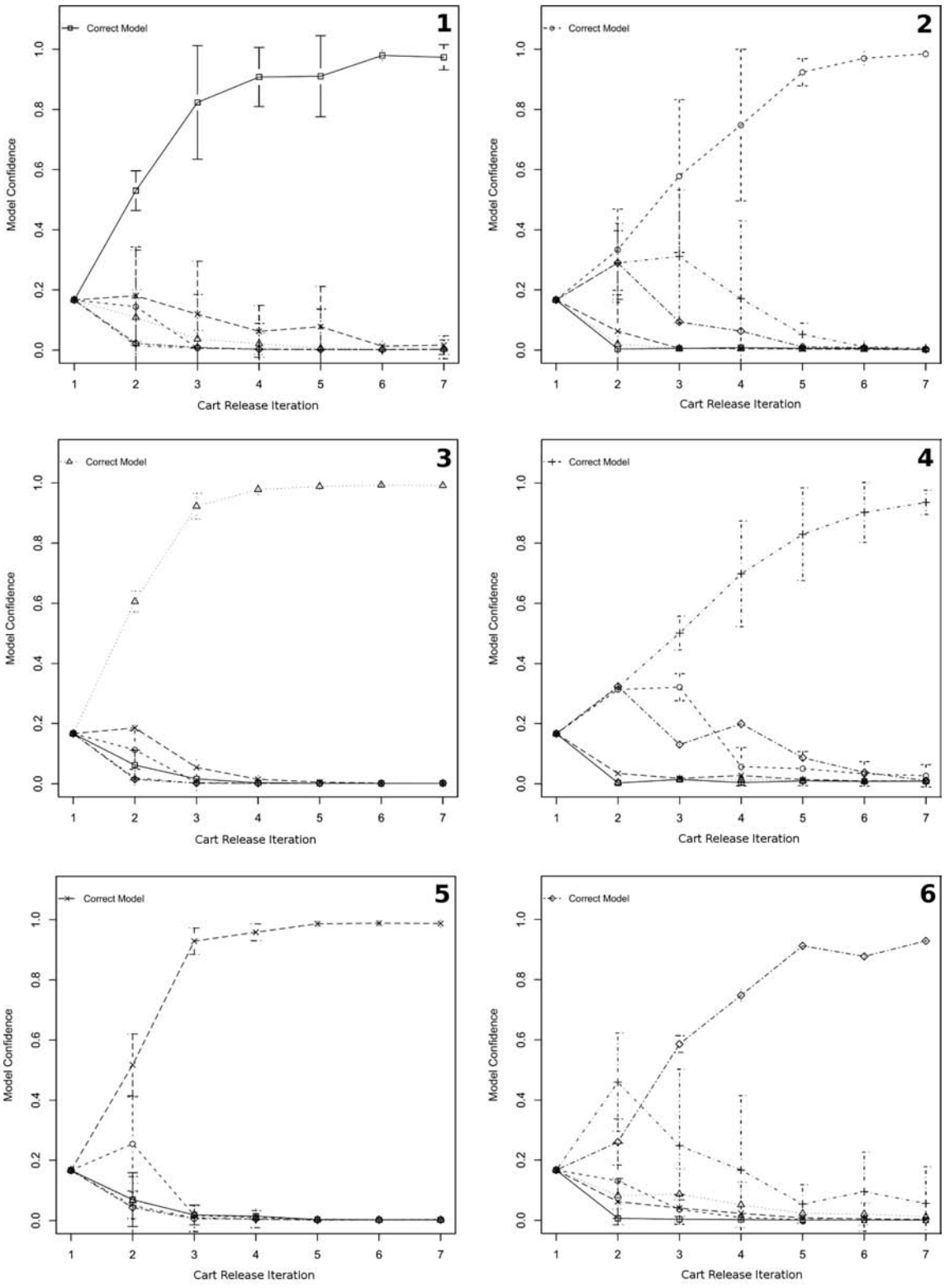

Figure 6.3.15: The results of the *Wheel Configuration Experiment* in terms of the model confidence distribution after a number of release iterations. The underlying configuration of the physical box-cart in each of the above runs matches the corresponding configuration shown in Figure 6.3.13. The error bars signify the 95% confidence interval. We label only the correct object model confidence in each graph. This is the model corresponding to the true underlying configuration of the box cart.

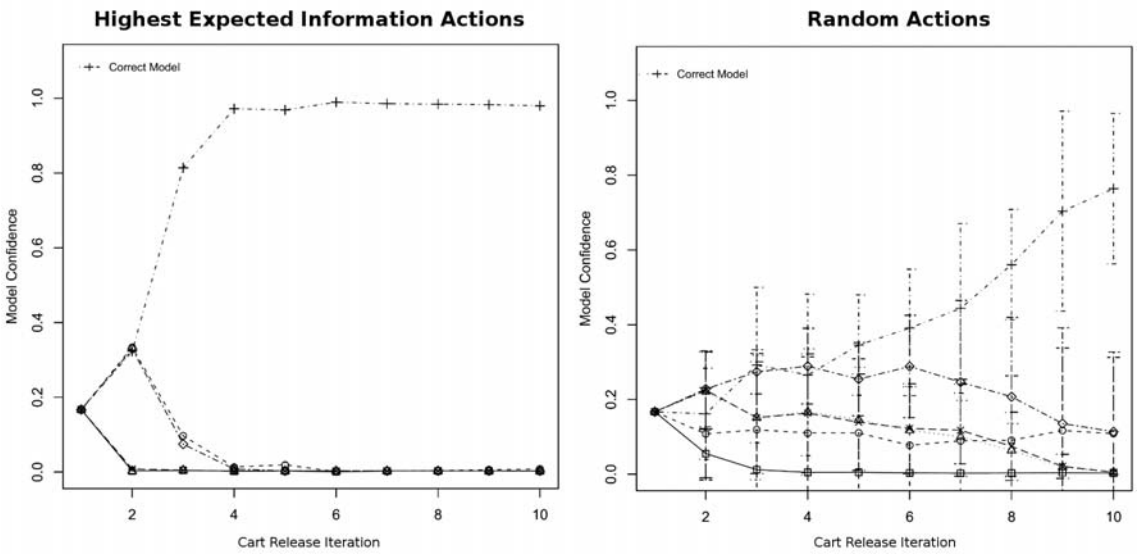

Figure 6.3.16: Learning the box-cart wheel configuration using the action with highest expected information gain (left), and using a random action (right) at every iteration. This is performed in simulation with the simulated box-cart having its wheels set to configuration 4 as seen in Figure 6.3.13. By performing the most informative action the confidence distribution quickly converges on the correct object model. When performing a random action, the confidence distribution on average does not reliably converge. The error bars indicate the 95% confidence interval.

6.3.3 Stimulus Response Behaviour Experiment

Overview

In this experiment we demonstrate that the general active learning framework is applicable to a wide variety of object properties. In this case the robot determines a behavioural model of an object, rather than a physical property. The object in question is a Lego Mindstorms[3] bot which is programmed to respond to light stimuli. The Lego-bot is composed of two motor driven wheels, and two light sensors (see Figure 6.3.17). It is programmed to perform a certain movement behaviour if one of its sensors is stimulated by light. A sensor is said to be stimulated if it detects a high light level as compared to the other sensor. An example movement behaviour is turn left 45° and move forward 15cm if the left light sensor is stimulated.

The task of the main robot is to perform some actions to determine a model that describes the Lego-bot's movement behaviour. For this purpose, a flashlight is

[3]http://mindstorms.lego.com

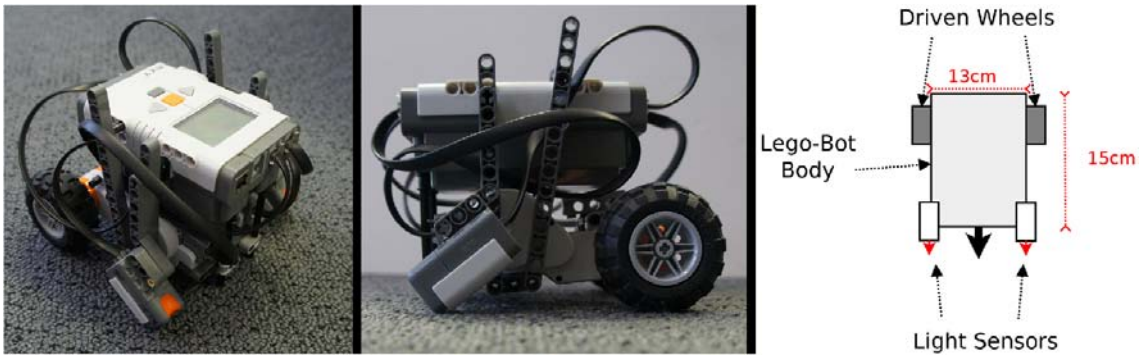

Figure 6.3.17: Lego Mindstorms robot. It is composed of the NXT Intelligent brick, two motor driven rear wheels, and two light sensors. The light sensorrs are facing forward and downward at a 45° angle.

attached to the gripper which the robot can use to project light onto the workspace. This is illustrated in Figure 6.3.18. The robot can perform actions in the form of shining the flashlight onto specific areas of the workspace and observing the resultant Lego bot's motion. This motion provides information as to the bot's underlying programmed behaviour. A simulator is used to predict the outcome of the various possible actions on the possible object models.

The procedure followed by the robot in this experiment differs from the previous two experiments in that the robot does not directly manipulate the target object. Instead the robot uses a flashlight attached to its gripper to illuminate the workspace and trigger the Lego bot's movement behaviour. To learn the object model the robot performs the following steps:

1. localise the Lego bot in the scene;

2. calculate the information gain of each action given the current model likelihood distribution;

3. carry out the most informative action by illuminating the appropriate part of the workspace with the flashlight;

4. determine the resulting pose of the Lego bot;

5. classify the resulting pose by matching it to a result label;

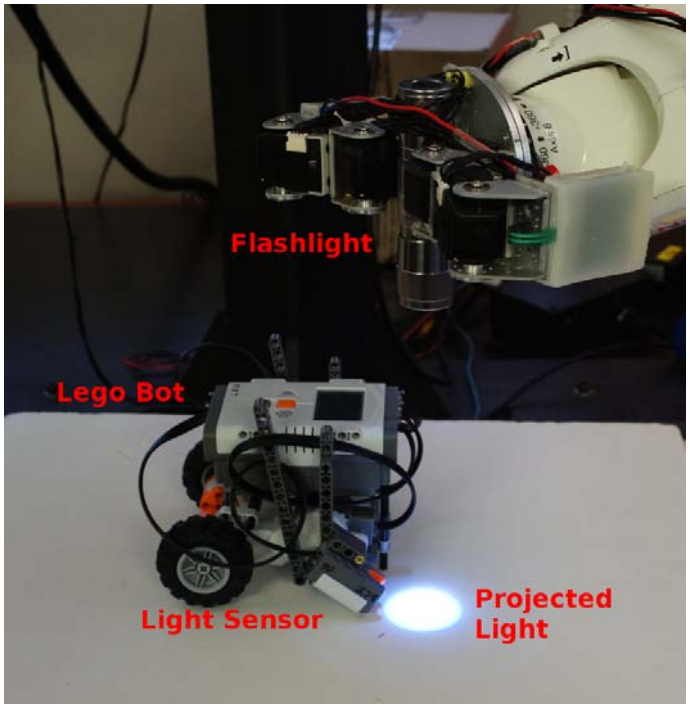

Figure 6.3.18: The robot uses a flashlight attached to the gripper to perform illumination actions to determine the light stimulation response behaviour of the Lego Mindstorms bot. An action is in the form of shining the flashlight onto the workspace to stimulate the light sensors on the Lego bot.

6. update the model likelihood distribution using the outcome probabilities of the performed action and the result label.

The cart is localised in a scene using SIFT features and a textured marker placed on top of the Lego bot.

Next, ~~In this section~~ we present ~~the following:~~ the possible models that can describe the movement behaviour of the Lego bot and the actions that the robot can perform (Section 6.3.3.1), the simulation method used to determine the action probability models (Section 6.3.3.2), the method of labeling the result of each action with a discrete label (Section 6.3.3.3), and the performance results of determining the behaviour model of the Lego cart using the active object model learning approach (Section 6.3.3.4).

Where do these come from?
Could they be learned in future work?

6.3.3.1 Possible Object Models and Actions

The robot's task is to determine the underlying behaviour of the Lego bot. This is done by finding a model from a predefined set of possible behaviour models that best fits the observed results of performed actions. For this experiment, we define the set of potential models that can describe the Lego bot's behaviour to be of size 16. If the left sensor is stimulated, the bot may turn 0° or 45° left, followed by moving forward $0cm$ or $15cm$. If the right sensor is stimulated, the bot may turn 0° or 45° right, followed by moving forward $0cm$ or $15cm$. A sensor is said to be stimulated when its sensed light level exceeds the sensed light level of the opposing sensor by 50%.

The robot determines which of the potential models most accurately matches the behaviour of the Lego bot by performing actions from an available pool and observing the results. The robot gripper has a flashlight attached that emits a focused light cone measured to have a 15.3° spread. The robot moves the gripper into a particular position to shine the light onto the Lego bot. Each illumination action can be defined by three parameters, the (x, y) position of the light cone relative to the Lego bot, and the height of the light above the table surface. The height determines the size of the projected light circle on the table surface. We define an *ad hoc* pool of 250 possible actions, generated by randomly choosing 50 (x, y) positions falling inside a circle of radius $15cm$ from the centre of the Lego bot, and for each position the height of the light above the table top may be one of the following 5 values $\{20cm, 25cm, 30cm, 35cm, 40cm\}$.

parameters again

6.3.3.2 Simulation Method

To determine the probability distribution over results of each action on the potential models we use a simulation method. However, unlike with the *Centre of Mass* and *Wheel Configuration* experiments, there are no convenient physics engine primitives for simulating a Lego Mindstorms robot programmed with a certain behaviour.

Instead we ~~explicitly~~ approximate the amount of light sensed by a light sensor when the flashlight is placed in a particular position relative to the Lego bot. This is followed by simulating the movement performed by the Lego bot if the light response behaviour is triggered.

We simulate the flashlight by computing the circle of light it projects onto the table surface from a particular position. The flashlight is held vertical in all cases, and the beam spread is set to 15.3°. This is used to compute the light circle's position and size for any given illumination action, which is specified by the relative position and hei te how much of this projected li sor. We do this by projecting a sensor. These rays are projected in a cone of angle 45° with the axis oriented in the direction of the light sensor (at a 45° angle looking at the ground). Each of these sample rays are intersected with the ground plane The percentage of all of the sampling rays of a light sensor that have their ground intersection point fall within the projected light circle is said to be the light value sensed by the sensor. Figure 6.3.19 illustrates the ray casting method. We do this for both the left and right light sensors, and compare the computed values. If the computed value of one of the sensors is more than 1.5× that of the other, the model movement behaviour is triggered. To simulate a movement behaviour, we rotate and translate the simulated Lego bot object as dictated by the particular behaviour model.

Although Bullet doesn't simulate the lights, other things, e.g. jME, which includes Bullet, do. So you did it yourself because it was easier than doing everything with jME, right?

To account for various sources of error and noise when the illumination action is performed on the robot, we apply a noise model to each simulated run. This is done by perturbing the light circle's size and position, as well as the amount the simulated Lego-bot turns and drives forward, using Gaussian noise. The light circle position is shifted in a random direction on the xy plane by noise with standard deviation $0.5cm$ and mean $0cm$. The light circle size perturbation has a standard deviation of $0.5cm$ and mean $0cm$. When the simulated Lego-bot turns, the turn amount perturbation has a standard deviation of 10° and mean $0cm$, when it drives

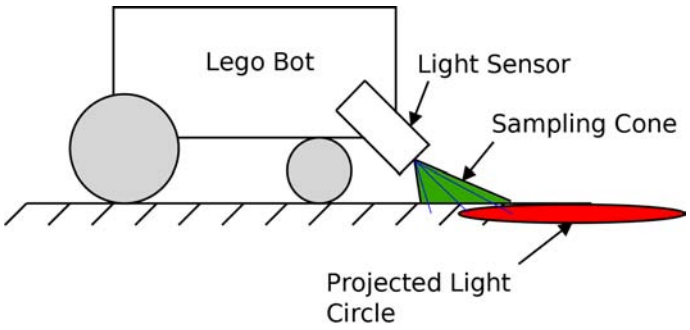

Figure 6.3.19: This diagram shows how we simulate the amount of light sensed by a light sensor when the flashlight casts a light circle on the ground. We project multiple rays (denoted by blue lines inside the sampling cone) from the light sensor's position, in a sampling cone of angle 45° in the direction the sensor is pointing. The percentage of these light rays that intersect the projected light circle is taken as the light sensed by the sensor. This value can then be compared to the light sensed by the opposing sensor to determine if the Lego bot's movement behaviour is triggered.

forward the perturbation for the distance driven has a standard deviation of $1cm$ and mean $0cm$. These perturbations are chosen *ad hoc*, but are set to overestimate the error in the Lego bot's movement.

6.3.3.3 Result Classification

The result labels are generated by discretising the workspace into uniform cells, followed by matching the Lego-bot's resulting pose at the conclusion of an action iteration to one of these cells. The discretisation is achieved by dividing a square $50cm \times 50cm$ centered on the Lego-bot into 100 $5cm \times 5cm$ cells. To discretise the orientation, each of these is further divided into 24 orientation cells, dividing the 360° circle into 15° increments. The end result is 2400 possible result labels.

When an illumination action is performed, to determine the outcome result label, we first calculate the relative motion of the Lego-bot to its starting pose. From this relative pose we then calculate the (x, y, θ) values, which are the relative amount the robot moved on the (x, y) plane and the angle amount it has rotated around the vertical axis. This relative pose is then matched to one of the 2400 possible result label cells.

6.3.3.4 Performance Results

We test the performance of the learning system by having the robot carry out multiple runs of learning the behaviour model of the Lego bot. In each run the robot performs a series of actions, updating the confidence distribution over the possible models using the result of each action.

To test our approach we used three different scenarios (the Lego bot was programmed with three different behaviours). For each scenario the robot performed three independent learning runs, each run involved a series of 6 illumination actions to determine the underlying model of the Lego bot's behaviour. In the first scenario the Lego bot is programmed to not move at all, that is, to not respond to stimuli. In the second scenario the Lego bot is programmed to turn left when the left sensor is stimulated, otherwise to remain stationary. In the third scenario the Lego bot is programmed to turn left and move forward when the left sensor is stimulated, and move forward without rotating when the right sensor is stimulated. The results are presented in Figure 6.3.20 and show how the robot's confidence distribution over the 16 possible behaviour models changed during the course of each run. We can see that the robot's confidence distribution quickly tends toward the correct model describing the Lego bot's behaviour in all 3 instances.

In additional to evaluating the performance of the learning method, we investigated the effect of choosing the action with highest expected information gain on the learning rate. We compare in simulation a robot learning the Lego bot's behaviour using the most informative action at every step, and using a random action at every step. The actions are simulated in the same manner as when building the action model. For the best and random action selection policies, three simulated runs are performed, each with six actions performed. The progression of the confidence distribution over the possible Lego bot behaviour models is show in Figure 6.3.21. It can be seen that by choosing the most informative action leads to a convergence of the confidence distribution to the correct object model, whereas choosing a random

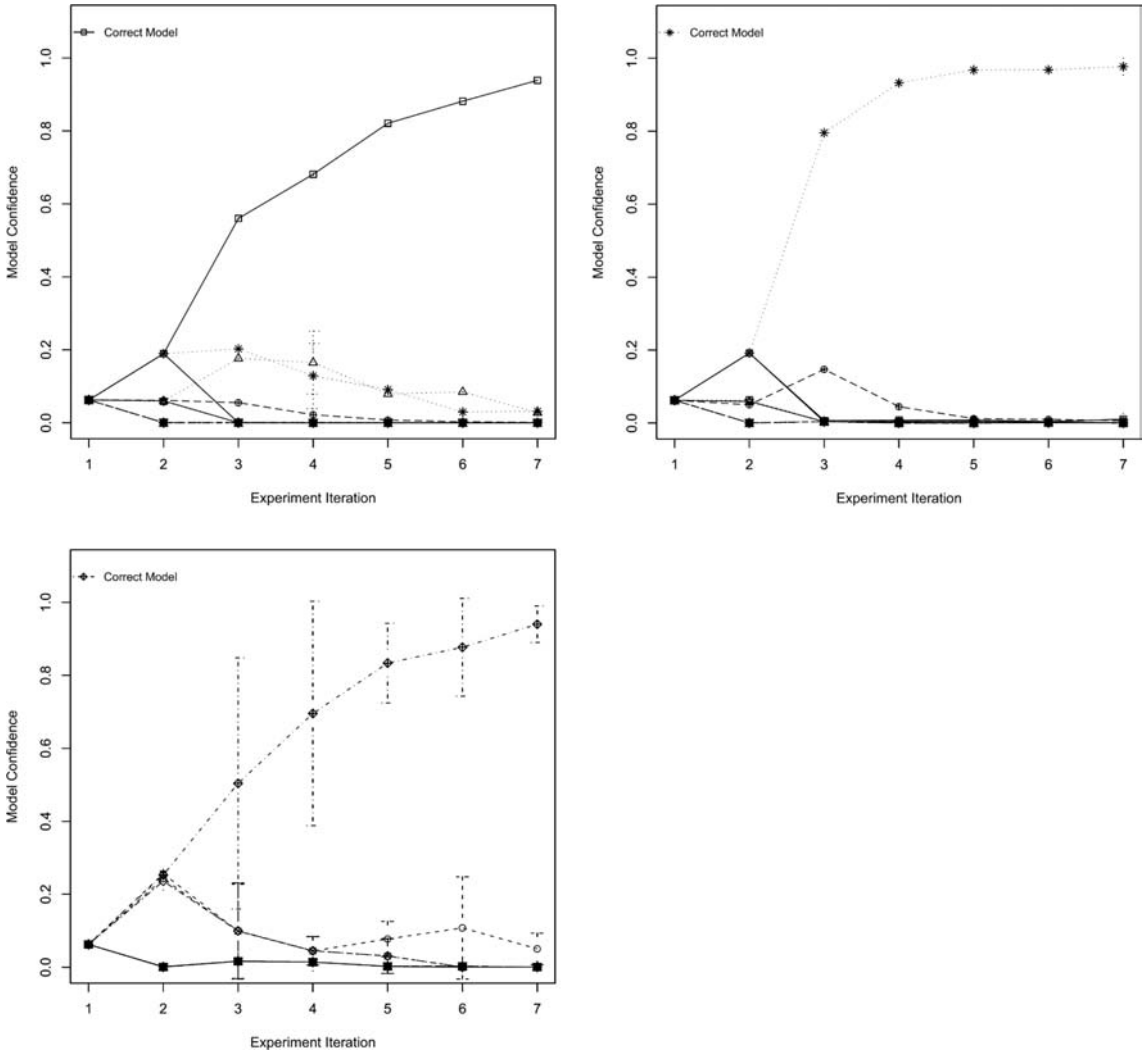

Figure 6.3.20: The results of the *Stimulus Response Experiment* showing the progression of the model confidence distribution after a number of illumination action iterations. There are 16 possible models that can describe the Lego bot behaviour, the above results show how the robot's confidence distribution over the potential models evolved when performing actions in 3 different scenarios. Top Left corresponds to a scenario in which the Lego bot does not respond in any way to light. Top Right corresponds to the scenario in which the Lego bot is programmed to turn left when the left sensor is stimulated. The Bottom Left corresponds to the scenario in which the Lego bot is programmed to turn left and move forward when the left sensor is stimulated, and move forward when the right sensor is stimulated. The error bars correspond to the standard deviation across 3 trials for each scenario. We omit individually labeling each line in the graphs as there are 16 different models. We only provide a label for the correct model confidence in each instance.

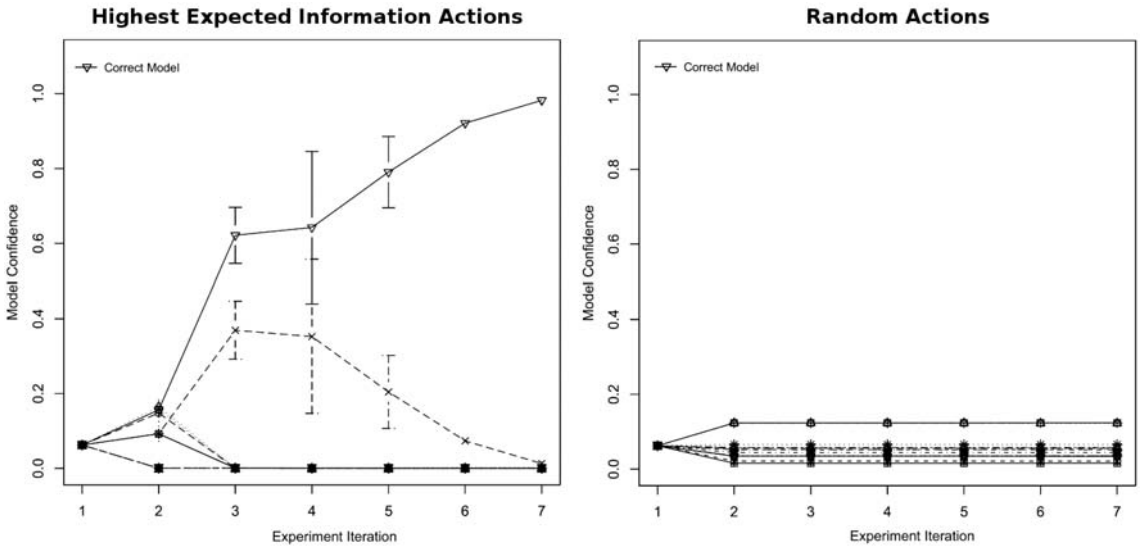

Figure 6.3.21: Learning the Lego bot's behaviour using the action with highest expected information gain (left), and using a random action (right) at every iteration. This is performed in simulation with the simulated Lego bot programmed to drive forward if its left sensor is stimulated, and turn left and drive forward if its right sensor is stimulated. By performing the most informative action the confidence distribution quickly converges on the correct object model. When performing a random action, the confidence distribution on average does not converge. The error bars indicate the 95% confidence interval.

action does not result in a convergence to the correct model. This is due to the fact that many of the possible actions do not result in any response by the Lego bot. If neither of the light sensors are illuminated by the flashlight, then the Lego bot's behaviour will not be triggered and no information will be gained as a result.

6.4 Learned Model Exploitation

Overview

The final experiment involves the robot performing a basic tool use task using an object for which it has previously learned a physics model. This learned model allows the robot to plan a solution to a task using the predictive qualities of the model, and then carry out the plan to complete the task.

The objective for the robot is to knock over a cylinder standing vertically on a flat surface. This cylinder is out of reach of the robot arm, instead the robot must

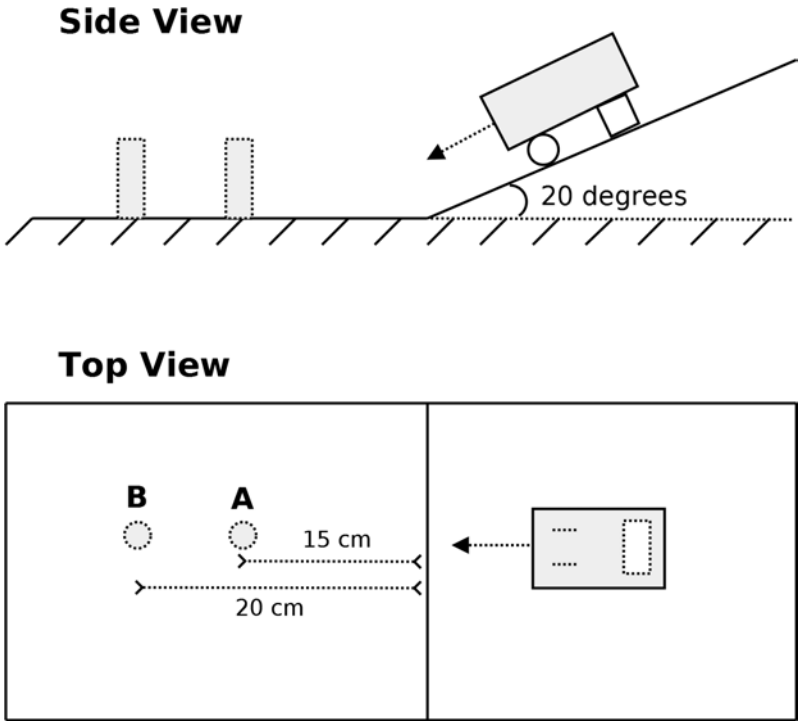

Figure 6.4.1: The above figure represents the workspace layout for the *Tool Use* task. In this task the robot has a workspace similar to the *Wheels Experiment*, but with the addition of a cylindrical peg placed at one of two possible positions, A and B ($15cm$ and $20cm$ from the ramp), the ramp angle is set to 20°. The robot's task is to use the internal simulated model of the box-cart object to release the cart from an appropriate position and orientation on the ramp to knock over the peg.

use a ramp and a box-cart to knock down the cylinder by placing and releasing the cart on the ramp in the appropriate orientation such that its trajectory intersects the position of the cylinder. The workspace consists of a flat table top and a 20° ramp. The cylinder is placed upright on the flat table top in one of two positions, 15cm and 20cm from the end of the ramp. This configuration is shown in Figure 6.4.1.

We use two different box-cart configurations, the wheels can either be directed forward, or at a 45° angle to the left, both wheels are free to rotate on the axle. These two cart configurations are illustrated in Figure 6.4.2. The robot has a learned physics engine model of the box-cart, which is determined using the method from Subsection 6.3.2 (note that the ramp angle during learning is 25°, whereas during this task it is 20°). The robot uses this model to simulate the outcome of releasing

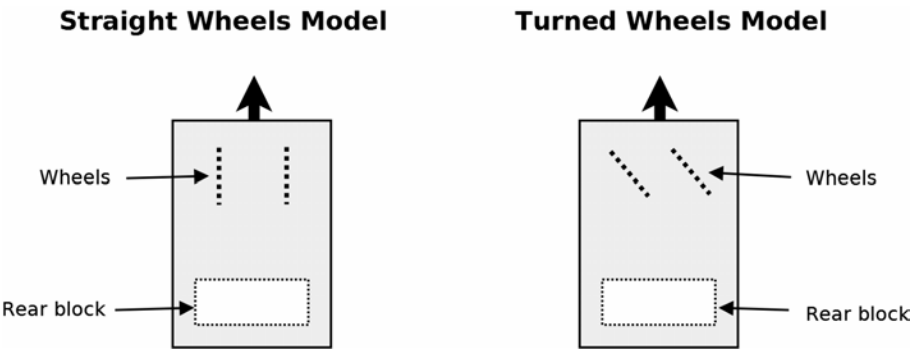

Figure 6.4.2: Possible box-cart wheel configurations models. The wheels of the box-cart can either be oriented straight ahead, or at a 45° angle to the left. Box wheels are free to rotate.

the box-cart from many different poses on the ramp. It chooses the best release pose, and performs the corresponding action on the real world box-cart to knock down the cylinder. The purpose of this experiment is to demonstrate the feasibility of using a predictive object model in the form of a physics engine to plan a solution to a task, demonstrate the importance of learning the correct object model. We also test the ability to use a learned physics model to make predictions in an environment different to that during learning.

To find the release pose for the box-cart with the best chance of knocking over the cylinder, the robot performs a non-linear numerical optimisation over the release pose parameter space. The release pose parameters specify the (x, y) position of the box-cart on the ramp, and its orientation angle θ. We use the Nelder Mead [140] optimisation method, as it is simple to implement and this particular problem has a low dimension and smooth parameter space. In principle other optimisation methods, such as Simulated Annealing [141], may be used instead. The value of the release pose objective function to be optimised is determined by performing a number of simulations where the simulated box-cart model is released in the specified pose. The simulation is run for 600 frames at a step rate of 30 frames per second. For each simulated frame the position of the box-cart is recorded. The output of a single simulation run is the smallest distance between the box-cart and the position of the cylinder during the run. This is performed multiple times to account for the

Algorithm 6.4 Box-cart positioning objective function.

input: box cart release pose→ *pose*
input: box cart physics model → *model*
input: cylinder position → *target_position*

error_sum ← 0
for *iter* **in** {1 . . . 50}
  initialiseSimulationEnvironment()
  noisy_object_model ← *perturbModel*(*model*)
  noisy_release_pose ← *perturbPose*(*pose*)
  positionBoxCart(*noisy_release_pose*, *noisy_object_model*)
  min_distance ← inf
  for *frame* **in** {1 . . . 600}
    simulationStep()
    cart_position ← *getBoxCartPosition*()
    if ‖*cart_position* − *target_position*‖ < *min_distance* **then**
      min_distance ← ‖*cart_position* − *target_position*‖
    endif
  endfor
  error_sum ← *error_sum* + *min_distance*
endfor
average_error ← $\frac{error\_sum}{50}$

output: value of the object function ← *average_error*

noise in the simulated action. The noise model is the same as the one used for the *Wheels Experiment* (Section 6.3.2). The final output of the objective function is the average smallest distance between the box-cart and the cylinder. This objective function is summarised in Algorithm 6.4. Using the Nelder Mead algorithm, we find the box-cart release pose that minimises this objective function. Finally, the robot picks up the box-cart and releases it from the optimal pose to knock down the cylinder.

Results

To test the above approach, as well as to determine the importance of learning the correct object model, the robot used the described method to release the box-cart from the appropriate position and orientation on the ramp to knock down the cylinder. We performed this experiments under the following conditions: with the cylinder placed 15cm and 20cm (position A and B respectively in Figure 6.4.1) from

	Correct Model	Incorrect Model
Position A - Straight Wheels	8/8	0/8
Position A - Turned Wheels	8/8	0/8
Position B - Straight Wheels	4/8	0/8
Position B - Turned Wheels	5/8	0/8

Table 6.1: Success rate of knocking down the cylinder with the box-cart. The above table shows the importance of learning the correct object model. The lower success rate when the cylinder is in position B is due to a longer distance from the ramp, increasing the chance of the box-cart running off course.

the end of the ramp, with the box-cart wheels pointing straight ahead and turned 45° to the left, and with the correct and incorrect object model. "Correct object" model refers to the physics model of the box cart matching the actual configuration of the box cart. "Incorrect object" model refers to the physics model being the opposite of the actual configuration of the box cart. That is, the robot thinks the box cart wheels are turned while in reality they are straight, and vice versa. This is done to demonstrate the importance of the robot having an accurate internal model of an object to successfully complete the task. In total this results in 8 different test scenarios.

For each test scenario the task is performed 8 times. Table 6.1 shows the success rate of knocking down the cylinder for the different scenarios. Additionally, for each iteration we recorded the predicted distance of the box-cart from the target peg, calculated using the simulated object model. We also recorded the distance between the box-cart and the cylinder after it has rolled down the ramp and come to a stop. A distance of 0 was recorded if the cylinder was knocked down by the box-cart. This data demonstrates the importance of an accurate model to describe the object to plan and complete the task. These results are summarised in Figure 6.4.3.

These results clearly demonstrate the importance of learning the correct object physics model, which allows accurate prediction of action outcomes and task planning. When the robot had an incorrect model of the object it was unable to accurately predict the action outcomes and successfully complete the task even once. It should be noted that when the cylinder is in position B, the lower success rate

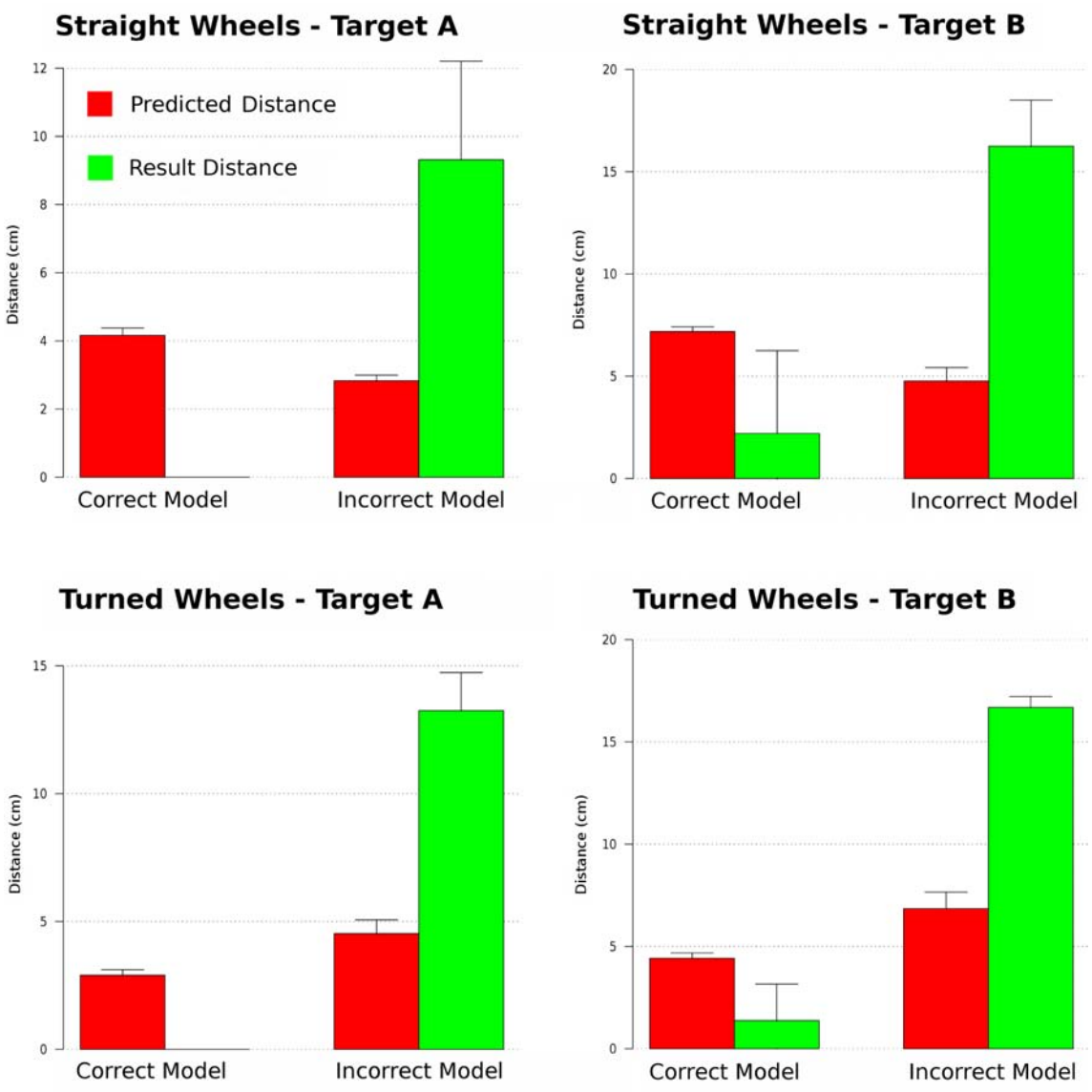

Figure 6.4.3: Predicted and actual results of using a box-cart to knock down an out of reach cylinder, expressed as a distance between the box-cart and the target cylinder. The result distance is the smallest distance between the cylinder and the box-cart after it comes to rest. The errors bars represent the standard deviation across eight repeated iterations of each scenario.

is due to the longer distance between the cylinder and the end of the ramp. The further the distance the larger the drift of the box-cart trajectory resulting from positioning errors and small variations in the table and ramp surface.

6.5 Discussion

We have presented a general method for a robot to determine hidden properties of an object by building a predictive model of the object using experimentation and simulation. A simulator is used to calculate the outcome probabilities of the actions that the robot can perform on an object, as well as to determine which is the most informative. The expected KL divergence is used to determine the information gain of an action. By choosing the most informative action, the robot can learn the model of the object in a minimum number of experiments.

We have presented both a generalised algorithm as well as several concrete applications of the method. We have demonstrated the robot determining the centre of mass of an object, as well as learning the wheel configuration of a box-cart. We have demonstrated the generality of our method by using it to model the behaviour of another robot responding to outside stimuli. Finally, a learned object model is applied to a simple tool use task in which the robot uses the object to plan and accomplish a goal.

It must be noted that some of the results of the concrete system implementations showed a higher than expected variance in terms of the robot's confidence distribution over the potential object models. This is likely due to the simplified noise model used during action simulation, as compared to the real world noise model. When building the action probability models we typically used a simple linear Gaussian noise model. However, in the real world the noise is in many instances highly non-linear and non-Gaussian. For example, one of the most common error modes during the *Centre of Mass Experiment* was for the robot to place the object too low, resulting in the bottom of the object striking the table surface. This would in

turn shift the object in the gripper, changing its orientation significantly. This type of noise is not accounted for during simulation and would require a more detailed simulation, including simulating the robot arm and its grasp of the object. We leave this for future work.

6.6 Future Work

The work presented in this chapter sets out a general framework for determining an object's model, which can then be used for further task planning and prediction. However, there are several areas for improvement that can be addressed in future work.

First, the presented approach is limited to classifying an object into one of several pre-determined models. These models are discrete, finite, and fixed. However, many of the properties that define the nature of an object are inherently continuous values. For example, the coefficient of friction of a surface, or the location of the centre of gravity. We may discretise the values as an approximation, but this may result in a very large number of potential models, which would in turn result in a large computational load to simulate all of the actions on all of the models. Instead, it may be possible to adapt the presented method to perform an optimisation search through the continuous parameter space to determine the most accurate object model.

Likewise, the set of possible actions that the robot can perform is predefined and fixed. A possible improvement is to allow the robot to dynamically add new actions to choose from. It may be able to use information about the quality of existing actions to generate new instances of increased quality, while rejecting ones with a low expected information gain. This may be increasingly important for action models with a large parameter space. The models used in the presented experiments were of low dimensionality, described by only 2 or 3 parameters. However, for more complicated models with many parameters, providing sufficient coverage of

the parameter space may not be feasible with a fixed predetermined set of actions.

Third, the result space of each action is restricted to being finite and discrete in the form of a fixed number of result labels. This is essentially a method of function approximation of the underlying probability distribution over the resulting world state after an action is performed. The discrete and finite nature of the function approximation simplifies the update of the confidence over the potential models distribution, as well as simplifying the calculation of the expected information gain. However, discretisation leads to a potential loss of information. A future improvement is to move to some form of sampling method for representing the probability distribution over the result world states.

Finally, we have only demonstrated a very simple planning and tool use task using the learned object model. In this task the action performed by the robot was predetermined ahead of time, the only planning involved was to determine the parameters for the action (where to release the box-cart). This is sufficient to demonstrate the predictive qualities of the learned model, but for more complex tasks and actions a more comprehensive planner (eg: STRIPS [142]) should be used. Additionally, a potential future direction is to use the learned quantitative model of the object to learn a higher level qualitative model, this would improve model generalisation and planning of complex actions [143].

Chapter 7

Conclusion and Future Work

In this thesis we have presented methods for a robot to learn the various object properties required to interact with an object effectively. With these methods, the robot is able to learn the object's appearance in a complex environment, reconstruct its shape, and use the appearance model to recognise and localise the object in a scene. These skills allow to robot to then manipulate and experiment with the object to determine a predictive model of the properties of the object. The combination of these skills enables the robot to progress from encountering a previously unknown object, to being able to use the object as a tool to accomplish a task.

We have presented a method for effectively segmenting object image features from the background. Long term tracking of object SIFT features, combined with robot induced object motion, enables the robot to use the feature trajectory data to separate the object's features from a cluttered and dynamic background. In this way the robot can autonomously learn an appearance model of a novel object, which can then be used to recognise and localise the object in a scene.

To regonise and localise an object, we have presented a method of matching SIFT features from a learned database of object features to scene features. We use feature geometric properties and feature description vectors concurrently to perform the match, as opposed to existing methods that perform two separate feature matching stages. By doing this we attain a higher number of feature matches, as well improved

(higher than what?)

efficiency in some circumstances.

We then combined the object feature segmentation and matching methods with existing 3D reconstruction techniques to build a system for a robot to autonomously learn the full 3D shape and appearance model of the object. This is done by stitching together many separate object views, each of which consisting of the object SIFT features and a dense surface point cloud.

The learned object appearance and shape model allows a robot to effectively recognise and localise the object in a scene from all aspects, as well as to plan grasping and manipulation actions. The robot uses these skills to perform experiments to build a model of the object's physical and internal properties. The robot plans and rehearses actions internally using a simulator, and then carries out the most informative experiments on the object. The learned model is then used to plan and complete tool use tasks using the object.

Future Work

There is a wide scope for future work in the topics covered in this thesis. We have detailed in the individual chapters potential areas for future work directly related to each topic.

The feature matching algorithm can be improved by extending it to other local image feature and interest point detectors (eg: SURF [46]). Additionally, there is scope for optimising the various parameters and thresholds used in the feature matching process.

The object feature segmentation method can also be extended to other types of features, as well as improving the feature tracking and arm feature filtering methods. For the feature segmentation, we track scene features while moving the target object with the robot arm. We use a lightweight approach for tracking a feature between frames. We can improve this by using a Bayesian tracking approach, taking into account feature velocity, position, and description vector to correlate features between frames.

The object shape reconstruction can be improved by incorporating a loop closure technique for stitching together the individual aspects of the object's shape. In addition to this, the directions from which the robot observes the object can be designed to optimise the amount of surface shape information gained.

There are numerous avenues for future work in the area of robot active learning of object properties. First, the discrete and pre-defined object models and available robot actions can be improved to be dynamically generated from the continuous parameter space. The descrete experiment result function approximation, in the form of result labels, can be replaced with a more accurate sample based function approximation. We can improve the noise model, used in simulation, to take into account non-Gaussian and non-linear sources of experiment uncertainty.

In addition to these areas of possible improvement, other avenues for future work include how a robot can initially detect a new object in a scene, and how the final learned object model can be used for planning of tool-use tasks. In our overall system, we do not address the issue of how the robot can initially detect and grasp a new object in a scene. We assume that the robot starts off with the object in its grasp. There are various possible approaches to this problem, for example the robot can detect areas of irregular displacement on a flat tabletop surface and use this as a hint that it may be an object. Finally, the learned model, encapsulating the physical and internal properties of the object, should be incorporated into a higher level planner, such as STRIPS [142]. This would allow for effectively planning complex actions, using the object to accomplish a wide variety of tasks.

Bibliography

[1] The gentle rise of the machines. *The Economist*, March 2004.

[2] T. Arai H. Makino. New developments in assembly systems. *CIRP Annals - Manufacturing Technology*, 43(2):501 – 512, 1994.

[3] C. DiSalvo J. Forlizzi. Service robots in the domestic environment: a study of the roomba vacuum in the home. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 258 – 265, 2006.

[4] Household robot not so futuristic. *The Columbus Dispatch*, September 2011.

[5] M. Pollack N. Roy S. Thrun J. Pineau, M. Montemerlo. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3-4):271 – 281, 2003.

[6] E. Torres-Jara C. Kemp, A. Edsinger. Challenges for robot manipulation in human environments. *Robotics Automation Magazine, IEEE*, 14(1):20 – 29, march 2007.

[7] A frosh view on the internet and society. `http://cs47n.blogspot.com.au/2011/10/article-on-willow-garage-where-well.html`.

[8] G. Hirzinger C. Borst, M. Fischer. A fast and robust grasp planner for arbitrary 3d objects. *Robotics and Automation, 1999. Proceedings. IEEE International Conference on*, 3:1890 – 1896, 1999.

[9] C. Sammut S. Brown. An architecture for tool use and learning in robots. In *Australian Conference on Robotics and Automation*, 2007.

[10] K. Roberts P. Allen. Haptic object recognition using a multi-fingered dextrous hand. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, volume 1, pages 342 – 347, May 1989.

[11] J. Aggarwal F. Arman. Model-based object recognition in dense-range images - a review. *ACM Comput. Surv.*, 25(1):5 – 43, March 1993.

[12] A. State M. Livingston. Magnetic tracker calibration for improved augmented reality registration. *Presence*, 6(5):532 – 546, 1997.

[13] A. Stoytchev J. Sinapov, M. Wiemer. Interactive learning of the acoustic properties of household objects. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 2518 – 2524, May 2009.

[14] J. Ponce D. Forsyth. *Computer Vision: A Modern Approach*. Prentice Hall, August 2002.

[15] G. Stockman L. Shapiro. *Computer Vision*. Prentice Hall, 2001.

[16] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 8(6):679 – 698, November 1986.

[17] R. Cipolla T. Drummond. Real-time visual tracking of complex structures. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):932 – 946, July 2002.

[18] I. Kweon S. Kim. Automatic model-based 3d object recognition by combining feature matching with tracking. *Machine Vision and Applications*, 16:267 – 272, 2005.

[19] R. Hanek-T. Schmitt T. Bandlow, M. Klupsch. Fast image segmentation, object recognition and localization in a robocup scenario. In *RoboCup-99: Robot Soccer World Cup III*, volume 1856, pages 111 – 128. 2000.

[20] V. Vapnik O. Chapelle, P. Haffner. Support vector machines for histogram-based image classification. *Neural Networks, IEEE Transactions on*, 10(5):1055 – 1064, September 1999.

[21] D. Ballard M. Swain. Color indexing. *International Journal of Computer Vision*, 7:11 – 32, 1991.

[22] G. Metta.

[23] I. Patras A. Diplaros, T. Gevers. Combining color and shape information for illumination-viewpoint invariant object recognition. *Image Processing, IEEE Transactions on*, 15(1):1 – 11, January 2006.

[24] G. Lu D. Zhang. Review of shape representation and description techniques. *Pattern Recognition*, 37(1):1 – 19, 2004.

[25] J. Puzicha S. Belongie, J. Malik. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(4):509 – 522, April 2002.

[26] L. Cole L. Cole, D. Austin. Visual object recognition using template matching. In *Proceedings of Australian Conference on Robotics and Automation*, 2004.

[27] I.T. Jolliffe. *Principal component analysis*. 1986.

[28] H. Murase S. Nayar, S. Nene. Real-time 100 object recognition system. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2321 – 2325, April 1996.

[29] A. Frangi-Jing-yu Yang Jian Yang, D. Zhang. Two-dimensional pca: a new approach to appearance-based face representation and recognition. *Pattern*

Analysis and Machine Intelligence, IEEE Transactions on, 26(1):131 – 137, January 2004.

[30] M. Stephens C. Harris. *A combined corner and edge detector*, volume 15, pages 147 – 151. 1988.

[31] K. Mikolajczyk T. Tuytelaars. Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177 – 280, July 2008.

[32] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *Int. J. Comput. Vision*, 37:151 – 172, June 2000.

[33] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615 – 1630, October 2005.

[34] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60:91 – 110, November 2004.

[35] D. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150 – 1157, 1999.

He's the potential examiner

[36] S. Srinivasa-D. Ferguson A. Collet, D. Berenson. Object recognition and full pose registration from a single image for robotic manipulation. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 48 – 55, May 2009.

[37] D. Lowe I. Gordon. What and where: 3d object recognition with accurate pose. In *Toward Category-Level Object Recognition*, volume 4170, pages 67 – 82. 2006.

[38] J. Little S. Se, D. Lowe. Vision-based mobile robot localization and mapping using scale-invariant features. In *Robotics and Automation, 2001. Proceedings*

2001 ICRA. IEEE International Conference on, volume 2, pages 2051 – 2058, 2001.

[39] J. Little S. Stephen, D. Lowe. Global localization using distinctive visual features. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 226 – 231, 2002.

[40] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15:11 – 15, January 1972.

[41] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381 – 395, June 1981.

[42] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209 – 226, September 1977.

[43] J.S. Beis and D.G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1000 – 1006, June 1997.

[44] A. Farag A. Abdel-Hakim. Csift: A sift descriptor with color invariant characteristics. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1978 – 1983, 2006.

[45] Yan Ke and R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages 506 – 513, June 2004.

[46] Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool. Surf: Speeded up robust features. In *ECCV (1)*, pages 404 – 417, 2006.

[47] K. Sankar R. Nikhil. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277 – 1294, 1993.

[48] R. Gonzalez A. Perez. An iterative thresholding algorithm for image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):742 – 751, November 1987.

[49] J. Denzler-H. Niemann M. Reinhold, M. Grzegorzek. Appearance-based recognition of 3-d objects by cluttered background and occlusions. *Pattern Recognition*, 38(5):739 – 753, 2005.

[50] G. Peters. A vision system for interactive object learning. In *Computer Vision Systems, 2006 ICVS '06. IEEE International Conference on*, page 32, January 2006.

[51] L. Bischof R. Adams. Seeded region growing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(6):641 – 647, June 1994.

[52] H. Talbot Z. Lin, J. Jin. Unseeded region growing for 3d image segmentation. In *Selected papers from the Pan-Sydney workshop on Visualisation*, volume 2, pages 31 – 37, 2001.

[53] D. Huttenlocher P. Felzenszwalb. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:167 – 181, 2004.

[54] R. Leahy Z. Wu. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(11):1101 – 1113, November 1993.

[55] V. Kolmogorov Y. Boykov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):112 – 1137, September 2004.

[56] J. Malik S. Jianbo. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888 – 905, August 2000.

[57] P. Soille L. Vincent. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):583 – 598, June 1991.

[58] L. Najman-M. Couprie J. Cousty, G. Bertrand. Watershed cuts: Minimum spanning forests and the drop of water principle. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(8):1362 – 1374, August 2009.

[59] M. Piccardi. Background subtraction techniques: a review. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 4, pages 3099 – 3104, October 2004.

[60] T. Darrell-A. Pentland C. Wren, A. Azarbayejani. Pfinder: real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):780 – 785, July 1997.

[61] W. Grimson C. Stauffer. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, pages 637 – 663, 1999.

[62] F. Monay J. Yao, J.M. Odobez. Idiap human detection code. `http://exoplanet.eu/catalog.php`.

[63] A. Blake-V. Kolmogorov A. Criminisi, G. Cross. Bilayer segmentation of live video. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 53 – 60, June 2006.

[64] M. Shah Jiangjian Xiao. Motion layer extraction in the presence of occlusion using graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1644 – 1659, October 2005.

[65] J.J. Little T. Southey. Object discovery through motion, appearance and shape. In *AAAI Workshop on Cognitive Robotics*, 2006.

[66] L. Kleeman Wai Ho Li. Autonomous segmentation of near-symmetric objects through vision and robotic nudging. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3604 – 3609, September 2008.

[67] P. Fitzpatrick. First contact: an active vision approach to segmentation. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2161 – 2166, October 2003.

[68] P. Fitzpatrick G. Metta. Better vision through manipulation. *Adaptive Behavior*, 11(2):109 – 128, 2003.

[69] F. Wagner M. Stoer. A simple min-cut algorithm. *J. ACM*, 44(4):585 – 591, July 1997.

[70] O. Brock J. Kenney, T. Buckley. Interactive segmentation for manipulation in unstructured environments. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1377 – 1382, May 2009.

[71] H. Christensen-P. Allen A. Miller, S. Knoop. Automatic grasp planning using shape primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2003*, volume 2, pages 1824 – 1829, 2003.

[72] S. Kagami-M. Inaba H. Inoue J. Kuffner, K. Nishiwaki. Motion planning for humanoid robots. In *Robotics Research*, volume 15, pages 365 – 374. Springer Berlin / Heidelberg, 2005.

[73] M. Levoy G. Turk. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311 – 318, 1994.

[74] M. Levoy B. Curless. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303 – 312, 1996.

[75] J. Aggarwal W. Martin. Volumetric descriptions of objects from multiple views. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 5(2):150 – 158, March 1983.

[76] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(2):150 – 162, February 1994.

[77] A. Zisserman R. Hartley. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2003.

[78] A. van Doorn J. Koenderink. Affine structure from motion. *J. Opt. Soc. Am. A*, 8(2):377 – 385, February 1991.

[79] R. Barea-E. Lopez M. Ocana J. Nuevo D. Schleicher, L. Bergasa. Real-time wide-angle stereo visual slam on large environments using sift features correction. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3878 – 3883, November 2007.

[80] D. Lowe I. Skrypnyk. Scene modelling, recognition and tracking with invariant image features. In *Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on*, pages 110 – 119, November 2004.

[81] D. Lowe M. Brown. Unsupervised 3d object recognition and reconstruction in unordered datasets. In *3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on*, pages 56 – 63, June 2005.

[82] R. Keriven P. Labatut, J. Pons. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In

Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, pages 1 – 8, October 2007.

[83] T. Tsubouchi-S. Yuta K. Yamazaki, M. Tomono. Object shape reconstruction and pose estimation by a camera mounted on a mobile robot. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 4, pages 4019 – 4025, September 2004.

[84] T. Kanade B. Lucas. An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674 – 679, 1981.

[85] T. Kanade C. Tomasi. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

[86] M. Levoy S. Rusinkiewicz, O. Hall-Holt. Real-time 3d model acquisition. *ACM Trans. Graph.*, 21(3):438 – 446, July 2002.

[87] O. Hilliges D. Molyneaux R. Newcombe P. Kohli J. Shotton S. Hodges D. Freeman A. Davison A. Fitzgibbon S. Izadi, D. Kim. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559 – 568, 2011.

[88] M. Levoy S. Rusinkiewicz. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145 – 152, 2001.

[89] X. Ren D. Fox M. Krainin, P. Henry. Manipulator and object tracking for in-hand 3d object modeling. *The International Journal of Robotics Research*, 30(11):1311 – 1327, 2011.

[90] D. Fox M. Krainin, B. Curless. Autonomous generation of complete 3d object models using next best view manipulation planning. In *Robotics and Automa-*

tion (ICRA), 2011 IEEE International Conference on, pages 5031 – 5037, May 2011.

[91] C. Kemp A. Edsinger. Manipulation in human environments. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 102 – 109, December 2006.

[92] J.C. Latombe. *Robot Motion Planning*. 1991.

[93] J.J. Gibson. *The Theory of Affordances*. 1977.

[94] M. Miller A. Stoytchev S. Griffith, J. Sinapov. Toward interactive learning of object categories by a robot: A case study with container and non-container objects. In *Development and Learning, 2009. ICDL 2009. IEEE 8th International Conference on*, pages 1 – 6, June 2009.

[95] I. Walker B. Willimon, S. Birchfield. Rigid and non-rigid classification using interactive perception. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1728 –1733, October 2010.

[96] O. Brock D. Katz. Manipulating articulated objects with interactive perception. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 272 – 277, May 2008.

[97] N. Pugeault E. Baseski F. Guerin J.H. Piater N. Kruger D. Kraft, R. Detry. Development of object and grasping knowledge by robot exploration. *Autonomous Mental Development, IEEE Transactions on*, 2(4):368 – 383, December 2010.

[98] N. Pugeault E. Baseski J. Piater N. Kruger D. Kraft, R. Detry. Learning objects and grasp affordances through autonomous exploration. In *Computer Vision Systems*, volume 5815, pages 235 – 244. 2009.

[99] A. Stoytchev. Behavior-grounded representation of tool affordances. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3060 – 3065, April 2005.

[100] A. Stoytchev. Toward learning the binding affordances of objects: A behavior-grounded approach. In *AAAI Symposium on Developmental Robotics*.

[101] L. Natale S. Rao G. Sandini P. Fitzpatrick, G. Metta. Learning about objects through action - initial steps towards artificial cognition. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 3140 – 3145, September 2003.

[102] P. Fitzpatrick. From first contact to close encounters: A developmentally deep perceptual system for a humanoid robot. 2003.

[103] R. Yokoya Jun Tani K. Komatani H. Okuno S. Nishide, T. Ogata. Object dynamics prediction and motion generation based on reliable predictability. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1608 – 1614, May 2008.

[104] A. Bernardino J. Santos-Victor L. Montesano, M. Lopes. Modeling affordances using bayesian networks. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 4102 – 4107, November 2007.

[105] G. Sandini S. Sakka R. Saegusa, G. Metta. Active motor babbling for sensorimotor learning. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 794 – 799, February 2009.

[106] A. Stoytchev. Learning the affordances of tools using a behavior-grounded approach. *Towards Affordance Based Robot Control*, 4760:140 – 158, 2008.

[107] A. Bernardino J. Santos-Victor L. Montesano, M. Lopes. Affordances, development and imitation. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 270 – 275, July 2007.

[108] L. Ljung, editor. *System identification: theory for the user*. Prentice Hall, 1987.

[109] G. C. Goodwin and R. L. Payne. *Dynamic System Identification: Experiment Design and Data Analysis*. 1977.

[110] D. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4):986 – 1005, 1956.

[111] V. Fedorov. *Theory of Optimal Experiments*. Academic Press, 1972.

[112] T. Kanade B. Mettler, M. Tischler. System identification of small-size unmanned helicopter dynamics. In *In American Helicopter Society, 55th Forum*, 1999.

[113] S. Omkar V. Mani-P. Sampath S. Suresh, M. Kumar. Neural networks based identification of helicopter dynamics using flight data. In *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, volume 1, pages 10 – 14, November 2002.

[114] N. Gordon T. Clapp M. Arulampalam, S. Maskell. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174 – 188, February 2002.

[115] Eh?

[116] C. Reinsch G. Golub. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14:403 – 420, 1970.

[117] J. Hertzberg H. Surmann. A. Nuchter. An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45(3 - 4):181 – 198, 2003.

[118] R. Siegwart A. Milella. Stereo-based ego-motion estimation using pixel tracking and iterative closest point. In *Computer Vision Systems, 2006 ICVS '06. IEEE International Conference on*, January 2006.

[119] S. Thrun. Simultaneous localization and mapping. In *Robotics and Cognitive Approaches to Spatial Mapping*, volume 38, pages 13 – 41. 2008.

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

[121] A. Koschan M. Abidi Y. Zhang, J. Paik. 3-d object representation from multiview range data applying deformable superquadrics. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 611 – 614, 2002.

[122] Y. Zhang. Experimental comparison of superquadric fitting objective functions. *Pattern Recognition Letters*, 24(14):2185 – 2193, 2003.

[123] D. Bertsekas D. Bertsekas. *Nonlinear Programming.* Athena Scientific, September 1999.

[124] C. Kelley. Iterative methods for optimization. *Frontiers in Applied Mathematics*, 1999.

[125] J. Rivest W. Scott, G. Roth. View planning for automated three-dimensional object reconstruction and inspection. *ACM Comput. Surv.*, 35(1):64 – 96, March 2003.

[126] K. Pulli. Multiview registration for large data sets. In *3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on*, pages 160 – 168, 1999.

[127] B. Leibe L. Van Gool T. Weise, T. Wismer. In-hand scanning with online loop closure. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1630 – 1637, October 2009.

[128] M. Hebert D. Huber. Fully automatic registration of multiple 3d data sets. *Image and Vision Computing*, 21(7):637 – 650, 2003.

[129] F. Jones P. Reiser C. Bryant S. Muggleton D. Kell S. Oliver R. King, K. Whelan. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971):247 – 252, January 2004.

[130] Germund Hesslow. Conscious thought as simulation of behaviour and perception. *Trends in Cognitive Sciences*, 6(6):242 – 247, 2002.

[131] A. Stoytchev J. Sinapov. Learning and generalization of behavior-grounded tool affordances. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 19 – 24, July 2007.

[132] G. Tiao G. Box. Bayesian inference in statistical analysis. 1973.

[133] M. Wong J. Hartigan. A k-means clustering algorithm. *Applied Statistics*, 28(1):100 – 108, 1979.

[134] R. Leibler S. Kullback. On information and sufficiency. *Ann. Math. Statist.*, 22:79 – 86, 1951.

[135] I. Verdinelli K. Chaloner. Bayesian experimental design: A review. *Statistical Science*, 10:273 – 304, 1995.

[136] R. Ladner D. Cohn, L. Atlas. Improving generalization with active learning. *Machine Learning*, 15:201 – 221, 1994.

[137] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin - Madison, 2009.

[138] H. Sompolinsky H. Seung, M. Opper. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287 – 294, 1992.

[139] K. Nigam A. McCallum. Employing em and pool-based active learning for text classification. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 350 – 358, 1998.

[140] R. Mead J. Nelder. A simplex method for function minimization. *The Computer Journal*, 7(4):308 – 313, January 1965.

[141] M. Vecchi S. Kirkpatrick, C. Gelatt. Optimization by simulated annealing. *Science*, 220(4598):671 – 680, 1983.

[142] N. Nilsson R. Fikes. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189 – 208, 1971.

[143] D. Bobrow. Qualitative reasoning about physical systems: An introduction. *Artificial Intelligence*, 24:1 – 5, 1984.

Chapter 8

Appendix

This thesis will concentrate on showing how I'm the best and awesome.

What's supposed to go here?