3D-Tracking of A Priori Unknown Objects in Cluttered Dynamic Environments

by

Hans de Ruiter

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Copyright © 2008 by Hans de Ruiter
3D-Tracking of A Priori Unknown Objects in Cluttered Dynamic Environments

Hans de Ruiter
Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2008

Abstract

Tracking of an object’s full six degree-of-freedom (6-dof) position and orientation (pose) would allow a robotic system to autonomously perform a variety of complex tasks, such as docking from any preferred angle, surveillance of moving subjects, etc. Computer vision has been commonly advocated as an effective tool for 3D (i.e., 6-dof) tracking Objects of Interest (OIs). However, the vast majority of vision-based 6-dof pose trackers reported in the literature require a model of the OI to be provided a priori. Finding/selecting the OI to track is also essential to autonomous operation. A problem that has often been neglected.

This Thesis proposes a novel, real-time object-tracking system that solves all of the aforementioned problems. The tracking procedure begins with OI selection. Since what constitutes an OI is application dependent, selection is achieved via a customizable framework of Interest Filters (IFs) that highlight regions of interest within an image. The region of greatest interest becomes the selected OI. Next, an approximate visual 3D model of the selected OI is built online by a real-time modeller. Unlike previously proposed techniques, this modeller can build the model of the OI even in the presence of background clutter; an essential task for tracking one object amongst many. Once a model is built, a real-time 6-dof tracker (i.e., the third sub-component) performs the actual 6-dof object tracking via 3D model projection and optical flow.
Performing simultaneous modelling and tracking presents several challenges requiring novel solutions. For example, a novel data-reduction scheme based on colour-gradient redundancy is proposed herein that facilitates using colour input images whilst still maintaining real-time performance on current computer hardware. Likewise, a per-pixel occlusion-rejection scheme is proposed which enables tracking in the presence of partial occlusions. Various other techniques have also been developed within the framework of this Thesis in order to achieve real-time efficiency, robustness to lighting variations, ability to cope with high OI speeds, etc.

Extensive experiments with both synthetic and real-world motion sequences have demonstrated the ability of the proposed object-tracking system to track a priori unknown objects. The proposed algorithm has also been tested within two target applications: autonomous convoying, and dynamic camera reconfiguration.
To my parents, Annie and Conradus
Acknowledgements

This Thesis is the culmination of years of research that would not have been possible without the support of others. I wish to acknowledge and thank all those whose help and support has been so valuable over the duration of my time at the University of Toronto.

First, I would like to thank my supervisor, Professor Beno Benhabib, whose patience and guidance has kept my research focussed on the overall goal of object tracking for autonomous systems. His ability to ask the pertinent questions about experimental results ensured that every aspect was researched thoroughly whilst helping me to develop the critical thinking required. My Ph.D. advisory committee Professors R. Ben Mrad and F. Ben Amara also gave valuable suggestions, whilst monitoring my progress. Their collective guidance is greatly appreciated.

My colleagues in the Computer Integrated Manufacturing Laboratory provided great company and a sounding board for ideas. They are: Ardevan Bakhtari, Usman Ghumman, Hamid Golmakani, Faraz Kunwar, Imtehaz Heerah, Matthew Mackay, Goldie Nejat, Bill Owen, Felix Wong, Joseph Wong, Leslie Wong, Hu Yongbiao, and Michael Zheng. In particular Michael Zheng implemented a depth-map extractor that my research required, as part of his Master of Engineering degree. The research presented in this Thesis was also aided by Ling Xiao’s work as a summer student.

I am also extremely grateful for the support from my parents Annie and Conradus, and my siblings, Anton, Ingrid, and Niels. It was my eldest brother Anton who first alerted me to the University of Toronto’s research programs in robotics and had an apartment ready for me before I had even arrived in Canada. My parents gave continual advice and support despite the distance between Toronto, Canada, and Wellington, New Zealand. Ingrid’s visits to Canada were always enjoyable, as were brief discussions with my youngest brother Niels, via email and web-cam.

Life is much more than just university research. Fr. Patrick O’Dea and the community at the
Newman Centre (the Catholic Chaplaincy at the University of Toronto) served both my social and spiritual needs. Whilst too numerous to name, being amongst friends in this community made life half way around the world from home enjoyable.

Finally, I would like to gratefully acknowledge the financial support of the University of Toronto Graduate Fellowship, and the Natural Science and Engineering Research Council (NSERC).
List of Publications Generated from this Thesis

So far, the research detailed in this Thesis has produced nine publications: three journal papers and six conference papers.

Journal Papers:


Conference Papers:


Contents

Abstract ii

Acknowledgements v

List of Tables xiv

List of Figures xv

Nomenclature and Acronyms xxi

1 Introduction 1

1.1 Motivation ......................................................... 1

1.2 Problem Statement ............................................... 3

1.3 Literature Review ................................................. 5

1.3.1 OI Selection ....................................................... 5

1.3.2 Object Modelling ............................................... 6

1.3.3 Visual Object Tracking ........................................ 7

1.3.4 Object Tracking for Autonomous Navigation and other Real-Time Applications ........................................ 9
1.4 Research Objectives ......................................................... 10
1.5 Thesis Overview .............................................................. 12

2 Tracking Initialization ......................................................... 15
  2.1 Overview ................................................................. 15
  2.2 Object of Interest Selection and Tracking Initialization .......... 19
    2.2.1 OI Selection Framework .......................................... 20
    2.2.2 Initial-Pose Estimation .......................................... 28
    2.2.3 Experiments ...................................................... 30
  2.3 Summary ................................................................. 37

3 On-line Modelling of A Priori Unknown Objects ......................... 39
  3.1 Depth-Map Extraction .................................................. 41
  3.2 Building the Model Surface .......................................... 43
  3.3 Extracting Texture Maps .............................................. 49
  3.4 Model Updating ......................................................... 51
  3.5 Segmenting the Object of Interest from the Background .......... 54
    3.5.1 Initial Object Segmentation .................................... 56
    3.5.2 Segmentation for Model Rebuilding ............................ 57
  3.6 Experiments ............................................................. 58
    3.6.1 The Implemented Depth-Map Extractor ....................... 58
    3.6.2 Experimental Set-up ............................................ 59
    3.6.3 Results .......................................................... 59
3.7 Discussion and Recommendations ........................................... 68
3.8 Summary ................................................................. 69

4 3D Visual-Model-Based Object Tracking ........................................ 71

4.1 The Core Object Tracker ....................................................... 71
  4.1.1 6-dof Optical Flow ...................................................... 72
  4.1.2 Projecting the Model .................................................... 74
  4.1.3 Estimating Motion ...................................................... 76

4.2 Improving Robustness Through Extensions ................................. 78
  4.2.1 Local Illumination Normalization .................................... 78
  4.2.2 Multi-Scale Optical Flow .............................................. 80
  4.2.3 Occlusion Rejection .................................................... 81

4.3 Improving Speed Through Extensions ....................................... 85
  4.3.1 Using a Bounding Sphere .............................................. 85
  4.3.2 Using the GPU and OpenGL ......................................... 87
  4.3.3 Colour-Gradient Redundancy ........................................ 87

4.4 Tracking Experiments ......................................................... 93
  4.4.1 Expected Accuracies .................................................... 94
  4.4.2 Robustness to Lighting ............................................... 95
  4.4.3 Robustness to Modelling Errors ..................................... 101
  4.4.4 Robustness to Large Motions ....................................... 102
  4.4.5 Robustness to Occlusions ............................................. 105
  4.4.6 Colour-Gradient Redundancy – Speed .............................. 111
4.5 Example Application – Convoying .............................................. 112
  4.5.1 Autonomous Convoy Controller ......................................... 115
  4.5.2 Overview of Complete Convoy-Control Algorithm .................. 116
  4.5.3 Experimental Set-up ...................................................... 116
  4.5.4 Results ........................................................................ 118
4.6 Summary ............................................................................. 122

5 System Implementation – Simulations and Experiments .................. 125
  5.1 Implementation .................................................................. 125
  5.2 Complete System Test ....................................................... 128
    5.2.1 Tracking Results .......................................................... 129
  5.3 An Example Application – Tracking for Dynamic Camera Reconfiguration Systems ......................................................... 136
    5.3.1 The Camera-Reconfiguration Algorithm ............................ 139
    5.3.2 Experimental Set-up ...................................................... 142
    5.3.3 Results ...................................................................... 144
  5.4 Summary ............................................................................. 147

6 Conclusions and Recommendations ............................................. 150
  6.1 Summary and Conclusions .................................................... 150
    6.1.1 OI Selection ............................................................... 151
    6.1.2 Real-Time Modelling ................................................... 152
    6.1.3 Tracking ................................................................. 154
  6.2 Recommendations ............................................................. 156
6.2.1 OI Selection ...................................................... 156
6.2.2 Real-Time Modelling ........................................... 157
6.2.3 Tracking .......................................................... 158

References ............................................................. 159

Appendices ............................................................. 173

A Principal Component Analysis ...................................... 174
  A.1 Procedure .......................................................... 175
    A.1.1 Database Training ............................................ 175
    A.1.2 Recognition ..................................................... 176

B Harris Feature Points .............................................. 178

C Delaunay Triangulation ............................................ 181
List of Tables

4.1 Image gradient statistics. ............................................ 91
4.2 Tracking error as a function of modeling error. ............... 103
4.3 Tracking success/failure for a range of different velocities. . 104
List of Figures

1.1 An example control system using a real-time 6-dof object-tracker. . . . . . . . 12

2.1 An overview of the complete object-tracking system. . . . . . . . . . . . . . . 17

2.2 3D points are projected onto a 2D image plane. . . . . . . . . . . . . . . . . . 19

2.3 An example OI selector containing multiple IFs. . . . . . . . . . . . . . . . . . 21

2.4 A restricted room requires OI selection for an active face-recognition system. . 22

2.5 A colour IF highlights humans (a); a positional IF highlights regions close to the door (b); a velocity IF highlights regions moving toward the restricted room (c); thus, isolating the OI to track (d). . . . . . . . . . . . . . . . . . . . . . . 22

2.6 The common structure of all IFs. . . . . . . . . . . . . . . . . . . . . . . . . . 23

2.7 An input image (a) results in a sparse IM (b) with a particular IF bank. . . . . 25

2.8 The motion-segmentation map for a sequence containing two objects. . . . . 33

2.9 Using the motion-segmentation map in Figure 2.8, the IF framework has been tuned to the cube, resulting in the IM (a) and model (b). . . . . . . . . . . . . . . 33

2.10 Using the motion-segmentation map in Figure 2.8, the IF framework has been tuned to the globe, resulting in the IM (a) and model (b). . . . . . . . . . . . . . . 33

2.11 The motion-segmentation map (a), IM (b), and resulting model for a real-world motion sequence (c). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
2.12 Frames #0 (a), #59 (b), and #119 (c) of a real-object sequence. 35

2.13 The positional (a) and orientational (b) tracking errors for the motion sequence shown in Figure 2.12. 36

3.1 Overview of the on-line modelling subsystem. 41

3.2 Point \( p \) can be found via triangulation of two or more 2D projections: \( p_i' \) and \( p_r' \). 42

3.3 An input image (a) and its DM (b). 45

3.4 The depth-map in Figure 3.3 (b) after median filtering. 46

3.5 An input image (a) and the resulting mesh (b). 49

3.6 Projective texture mapping on a plane. 50

3.7 An input image (a), corresponding IM (b) and the resulting model segmented from the background (c). 55

3.8 The motion-segmentation map (a), IM (b), and resulting model (c) for a real-world motion sequence. 56

3.9 Frames #0 (a), #120 (b), and #240 (c) of a motion sequence with no background. 60

3.10 Frames #0 (a), #120 (b), and #240 (c) of a motion sequence with a cluttered background. 60

3.11 Frames #0 (a), #120 (b), and #240 (c) of a motion sequence with a cluttered background. 60

3.12 The path taken by the OI in synthetic sequences. 61

3.13 The positional (a) and orientational (b) tracking errors for the sequence in Figure 3.9. 62

3.14 The positional (a) and orientational (b) tracking errors for the sequence in Figure 3.11. 63
3.15 The path taken by the OI in real-world motion sequences. .......................... 65

3.16 The tracked model overlaid over darkened frames #5 (a), #125 (b), and #240 (c) of a real-world motion sequence. ................................................................. 66

3.17 The positional (a) and orientational (b) tracking errors for the sequence in Figure 3.16. ................................................................. 67

4.1 A virtual-image (a), depth information (b) in the z-buffer, and (c) a mask in the stencil buffer. ................................................................. 75

4.2 The near- and far-clipping planes. ................................................................. 76

4.3 A photo of Convocation Hall Building at the University of Toronto shown in its original form (a), and after it has been filtered by a LINF (b). ................. 80

4.4 Green marks pixels used in optical flow calculations without occlusion rejection. 82

4.5 The 2D geometry for projecting a bounding-sphere onto an image plane. .... 86

4.6 The cube image (a) and corresponding gradient regions (b). ......................... 92

4.7 The path taken by the OI. .............................................................................. 96

4.8 Frames from motion sequences with diffuse lighting (a), directional lighting from the side (b), low light conditions (c), a complex background (d), a globe as OI (e), and a cardboard box and complex background (f). ..................... 97

4.9 The positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (a). ................................................................. 98

4.10 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (b). ................................................................. 99

4.11 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (c). ................................................................. 99
4.12 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (d) ................................................................. 100

4.13 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (e) ................................................................. 100

4.14 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (c) ................................................................. 101

4.15 The positional tracking errors in the presence of shading without (a) and with (b) using the LINF. ................................................................. 102

4.16 A white bar partially occludes the Olimex box, which is the OI. .............. 105

4.17 Without occlusion rejection, the tracker incorrectly uses pixels belonging to the white bar for tracking (a); the occlusion-rejection algorithm filters out the white bar (b). ................................................................. 106

4.18 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.17, when tracking without the occlusion-rejection algorithm. .... 107

4.19 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.17, when tracking with the occlusion-rejection algorithm. ......... 107

4.20 Without occlusion rejection, the tracker incorrectly uses pixels belonging to the white bar for tracking (a); the occlusion-rejection algorithm filters out the cube (b). ................................................................. 108

4.21 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.20, when tracking without the occlusion-rejection algorithm. .... 109

4.22 Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.20, when tracking with the occlusion-rejection algorithm. ......... 109

4.23 Unreliable pixels used by the original algorithm (a) are filtered out by the occlusion-rejection algorithm (b). ................................................................. 110
4.24 Positional (a) and orientational (b) tracking errors for the motion sequence shown in Figure 2.12 obtained using the colour-gradient redundancy algorithm. 113

4.25 Positional (a) and orientational (b) tracking errors for the motion sequence shown in Figure 2.12 obtained when the raw-colour data was used for tracking. 113

4.26 A lead vehicle image as captured by the following-vehicle’s camera. 117

4.27 The lead vehicle’s $x$ (a), $y$ (b), and $\theta$ (c) trajectories. 119

4.28 The $x$ (a), $y$ (b), and $\theta$ (c) tracking errors. 120

4.29 The following distance as a function of time. 121

4.30 The perpendicular trajectory error. 121

5.1 An overview of the complete object-tracking system. 126

5.2 Frame 21 from a synthetic sequence taken by the central camera. 129

5.3 The path taken by the OI in synthetic motion sequences. 130

5.4 The segmentation-map (a) and interest-map (b) for Frame 21, Figure 5.2. 130

5.5 The generated model and reference frames at Frame #21 (a), #120 (b), and #230 (c), respectively, for the sequence containing Figure 5.2. 131

5.6 Positional (a) and orientational (b) tracking errors for the motion of the object in Figure 5.2, using the approximate OI model built on-line. 132

5.7 Positional (a) and orientational (b) tracking errors for the motion of the object in Figure 5.2, using an $a$ priori provided model. 133

5.8 Frame 10 from a real-world sequence taken by the central camera. 134

5.9 The segmentation map (a) and IM (b) for Frame 10, Figure 5.8. 135

5.10 The generated model and reference frames at Frame #10 (a), #120 (b), and #230 (c), respectively, for the motion of the object shown in Figure 5.8. 135
5.11 The path taken by the OI in real-world motion sequences. .................. 136

5.12 Positional (a) and orientational (b) tracking errors for the object in Figure 5.8, using the approximate OI model built on-line. .................. 137

5.13 Positional (a) and orientational (b) tracking errors for the object in Figure 5.8, using an \textit{a priori} provided model. .................. 138

5.14 The experimental set-up with all cameras at their default poses. ............. 143

5.15 The camera configuration at Frame # 45 (a) and (e), #81 (b) and (f), #117 (c) and (g), and #153 (d) and (h) for the ideal (a) to (d), and tracked (e) to (h) cases. 145

5.16 The OI at Frame 153, as viewed by Camera #1 (a) and (b), Camera #2 (c) and (d), and, Camera #3 (e) and (f) for the ideal (a), (c), and (e), and tracked (b), (d), and (f) cases. .................. 146

5.17 The visibility metric for Cameras #1 (a), #2 (b), and #3 (c). ............. 148

C.1 A triangle and its circum-circle. .................. 182

C.2 A pair of triangles is “flipped” so that no circum-circle encloses extra vertices. . 182
Nomenclature and Acronyms

Mathematical Symbols

* The convolution operator

\( \angle (a, b) \) the angle between vector \( a \) and \( b \)

\( A \) A label for a generic matrix

\( a_i \) Column-vector \( i \) of matrix \( A \)

\( \bar{a} \) The mean column-vector of \( A \) (i.e., each element is the mean value for that row in \( A \))

\( I(x, y) \) or \( I \) An image (usually RGB)

\( I_v \) A virtual (projected) image

\( I_i \) An input image

\( I_{i,j} \) Colour channel \( j \) of image \( I_i \)

\( D(x, y) \) or \( D \) A depth-map

\( M_{int} \) A 3×4 matrix containing a camera’s internal parameters

\( M_{ext} \) A 4×4 matrix containing a camera’s external parameters

\( P \) The 6 dof pose of an object (i.e., a reference frame)

\( p \) A point in 2D/3D space

\( p' \) The 2D projection of 3D point \( p \) in homogeneous coordinates

\( q \) The 2D projection of 3D point \( p \) in image coordinates

\( R \) A rotation matrix
$R$  
A selected 2D region

$R_i$  
Region $i$ within a set of regions

$r$  
The radius of a circle/sphere

$T$  
The texturedness image

$w$  
The width of an object/shape

$\epsilon_n$  
Noise reduction constant

$\sigma$  
A constant in the Gaussian blurring filter related to blur radius

$\tau$  
Minimum gradient magnitude for an optical-flow constraint

$\tau_A$  
Threshold for the proportion of a region covered by interest pixels

$\tau_{gm}$  
The minimum acceptable gradient magnitude for optical-flow (for occlusion rejection)

$\tau_{md}$  
The maximum acceptable difference for gradient magnitude difference (for occlusion rejection)

$\tau_S$  
Minimum interest level threshold

$\tau_T$  
Minimum texturedness threshold

$\tau_{vmax}$  
The maximum motion that can be reliably estimated by optical-flow

$W_i$  
A sub-region of $Z_i$ in which the texturedness is greater than a threshold, i.e., $T(x, y) > \tau_T$

$Z_i$  
A sub-region of $R_i$ in which the interest level is above a threshold, i.e., $S(x, y) > \tau_S$

**Acronyms**

2D  
2 dimensional

3D  
3 dimensional

AGV  
Automated Guided Vehicle
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV</td>
<td>Autonomous Vehicle</td>
</tr>
<tr>
<td>DM</td>
<td>Depth Map</td>
</tr>
<tr>
<td>DME</td>
<td>Depth-Map Extractor/Extraction</td>
</tr>
<tr>
<td>dof</td>
<td>degrees of freedom</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman Filter</td>
</tr>
<tr>
<td>EEKF</td>
<td>Extended-Extended Kalman Filter</td>
</tr>
<tr>
<td>fps</td>
<td>frames-per-second</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>IF</td>
<td>Interest Filter</td>
</tr>
<tr>
<td>IM</td>
<td>Interest Map</td>
</tr>
<tr>
<td>KF</td>
<td>Kalman Filter</td>
</tr>
<tr>
<td>LINF</td>
<td>Local Illumination Normalization Filter</td>
</tr>
<tr>
<td>MG</td>
<td>Mesh Generator</td>
</tr>
<tr>
<td>OI</td>
<td>Object of Interest</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>RGB</td>
<td>Red, Green, and Blue (a colour image format)</td>
</tr>
<tr>
<td>SVD</td>
<td>Singular Value Decomposition</td>
</tr>
<tr>
<td>TE</td>
<td>Texture Extractor</td>
</tr>
<tr>
<td>TM</td>
<td>Texture Map</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

A major goal for robotics is to develop systems with greater autonomy and interactivity. Autonomy reduces the need for human intervention by allowing the robot to make decisions itself, whilst interactivity entails being able to respond to and interacting with changing environments. For example, if a component in a production line fell over and/or was out of place, an autonomous system would be able to adapt to such an unexpected event. A non-autonomous robot, however, would fail under these circumstances as it would only be able to follow predefined motions. Thus, greater autonomy would increase productivity by reducing downtime, and decrease costs by reducing the need for human monitoring. Similarly, greater autonomy would also be beneficial for mobile robots that are operated over links with long lag times (e.g., the Mars rovers) as waiting for new instructions when something unexpected happens would leave the robot idle for significant periods of time (over quarter of an hour in the case of the Mars rovers). It is also important for mobile robots operating in dynamic environments that they be able to sense the presence of other objects in order to avoid or interact with them effectively.

Apart from new future applications, there are two primary motivators for increasing auton-
omy for current robotic systems. The first is safety; systems that can detect possible emergencies and eliminate or minimize their impact would improve their overall safety. This would become more and more important as robots migrate from the manufacturing floor into houses. In industry, robots can be placed in areas designed for machinery only; in homes, robots must coexist with humans, in an environment tailored to humans. Much greater safety will be required, particularly when considering that the people using such robots will probably not be skilled technicians. We are already seeing the introduction of vacuum cleaning and lawn mowing robots; this will most likely expand to other applications as well. Risk of serious injury must be minimized before this can become mainstream.

The other primary motivator for increased autonomy is reducing the need for human monitoring. It would be desirable to be able to give a robot a task and walk away, similar to the way washing machines and other such appliances are operated today. At home, this is more a matter of convenience as it frees us up to do more interesting things. In industry, however, it provides several advantages. The obvious advantage is reduction of cost. Requiring constant monitoring by human operators is costly. This is particularly important to manufacturing competitiveness. Requiring human monitoring brings another problem, namely operator fatigue and attention. Monitoring monotonous repetitive machines for failure can quickly cause people to lose focus, resulting in problems being detected late, or worse, not being detected at all.

Achieving autonomy and interactivity requires knowledge of the location and motion of objects in the surrounding environment, that is, objects need to be tracked. Once the locations of these objects are known, decisions can be made. For example, interception of or rendezvous with a maneuvering Object of Interest (OI) are important navigation tasks for mobile robots (e.g., [1], [2]).

Visual object tracking is a possible method to sense the pose (position and orientation) of objects. Vision is an integral part of the human sensory system. This provides a passive sensing mechanism capable of extracting very detailed information about the environment that we live in. We use this to effortlessly navigate within our world, recognize and interact with objects;
precisely the tasks that we wish to make autonomous systems perform. It is clear that computer
vision could potentially provide the required object tracking capability in order to achieve the
goals listed above.

To date, numerous visual-tracking systems have been designed using various different tech-
niques. However, most techniques have various limitations: Some are too slow for real-time
use, others are unable to cope with occlusions, and most tracking systems provide only the 2D
on-screen location of an object (e.g., [3]-[7]). Finally, none are currently able to track a generic
object’s orientation as well as its position. Jurie and Dhome [8] were able to track orientation
but their technique requires a learning process beforehand and is, thus, not suitable for tracking
an object that is unknown \textit{a priori}.

In certain applications, tracking orientation can be important as important as tracking posi-
tion. For example, an autonomous robot that needs to rendezvous with another robot may need
to approach it from a specific direction. Likewise, a robot arm may need to pick up a moving
object from a particular direction. Another example is surveillance systems incorporating ac-
tive facial recognition. In such situations, cameras may need to be aligned to obtain a direct
frontal view of facial features. Additionally, most environments are cluttered with many static
and moving objects, all of which can occlude an OI from view. Thus, there is a need for an
object tracking system that can track the full 6-dof (degree-of-freedom) pose of a generic OI
without requiring a model of that OI \textit{a priori}.

\section*{1.2 Problem Statement}

The objective of this thesis is to develop a multi-camera object tracking system capable of
tracking the full 6-dof pose of a generic OI subject to partial occlusions caused by static and
dynamic obstacles. Because the tracking system is intended to be used as part of a robot’s
control system (or some other real-time system), it is essential that it operate in real-time.
Corke and Good [9] mention that a digital control system’s sampling rate should be 4 to 20
times the desired closed-loop bandwidth. Thus, a frame-rate of 10 frames-per-second (fps) could provide a bandwidth of up to 2.5 Hz. This provides approximately a 400 ms response time to adverse situations, enough for autonomous/robotic systems moving at typical speeds. Therefore, 10 fps is considered to be a realistic minimum frame rate.

An automated system needs to be able to select an OI and initialize the tracker with an initial pose. This problem is often avoided or ignored as it is challenging and is also separate problem from pure tracking. However, if an OI’s initial pose cannot be obtained, then, the tracker cannot track it. Thus, it is an essential part of a complete tracking system. What constitutes an OI is application-dependent so the system must be adaptable. For some systems, an OI may be a human subject that needs to be identified via face recognition; for others, an object that could potentially collide with the robot may be of greatest interest; for others still, the OI may be a particular type of object that is located within a particular region (such as a part on a production line). It is preferable to have a common framework that can be customized to the application at hand.

Tracking orientation as well as position requires some kind of a 3D model. This poses a problem when an object is encountered for which no model is provided a priori. In order to track such an object, a model must first be obtained on-line. Not only must the model be built on-line, the modeller must separate the OI from its environment.

Given the issues above, three sub-problems were identified:

- Selection of an OI,
- Building of a model of the OI on-line, and
- Tracking the full 6 dof pose of the selected OI using either a provided model, or one built on-line.

The primary goal of this thesis is, thus, to develop a generic method for real-time 6-dof autonomous object tracking, where OI selection and modelling are sub-components required
to achieve this goal. As far as pure object-tracking is concerned, the proposed core object tracker does not require either of the other problems to be solved. It is functionally complete and could track any object given a suitable model and approximate initial pose. Nevertheless, OI selection and on-line modelling are required for operating in real-world environments.

### 1.3 Literature Review

#### 1.3.1 OI Selection

The specific literature on OI selection is limited. As was mentioned earlier, it appears that most researchers have avoided or ignored tracking initialization and focused purely on tracking, assuming that the object’s approximate initial pose is somehow provided. Nevertheless, an autonomous robotic system requires some method of selecting objects to track.

OI selection bears some similarity to visual attention. Visual-attention algorithms try to highlight interesting parts of an image so that subsequent algorithms only operate on a subset of the entire image, reducing the total computational requirement. Ahrns and Neumann [10], for example, use dynamic neural fields in order to highlight interesting regions. Sun and Fisher [11] use the visual saliency of feature groupings in order to highlight interesting regions. Ouerhani and Hügli [12] compute a map of interest – saliency map – based on both static and dynamic features.

Despite the similarity to visual-attention algorithms, OI selection has its own primary-objective: the goal is to select objects that are of interest specifically to the system (i.e., to the end application), as opposed to searching with more general criteria. What constitutes an OI is application dependent. No literature found so far has addressed the OI-selection problem directly.
1.3.2 Object Modelling

Constructing object models from multiple images has been researched extensively. Various techniques have been proposed, including space-carving algorithms (e.g., [13]), silhouette-based methods (e.g., [14]), space-curve-fitting algorithms (e.g., [15]), and shape-from-motion algorithms (e.g., [16]). However, the vast majority of existing methods are unsuitable for use in a real-time object tracking system for the simple reason of lack of speed.

In order to be used for tracking, 3D geometry must be extracted and a model built at near real-time rates (i.e., models should be built in a fraction of a second). This is a difficult proposition. The problem is that extracting 3D geometry and then converting this data into a 3D model is a computationally expensive task. Additionally, the object-tracker itself may require significant processing power, leaving even less computational resources for the modelling task.

There have been a few real-time modellers proposed in the literature; Simard and Ferrie have presented an incremental modeler [16] and Franco et al. [17] use parallel processing on multiple computers in order to achieve real-time performance. However, these are still not adequate for use in an object-tracking system. There is yet another problem that neither address; namely, object segmentation. Both modelers assume that only one object is visible, whereas in many applications, the target object would be one of many in the scene. This is particularly true in cluttered environments, in which it is almost always true that there will be multiple objects viewed at some point in time. No real-time modeller found in the literature so far has addressed the segmentation problem directly.

Given all the problems mentioned above, one might ask how to achieve on-line modelling of an OI. The key lies in realizing that, unlike in the animation-based movie industry, the object-tracker does not require a precise model. For typical applications of 3D model building from input images, such as for movies, an accurate representation of the target object is required. Speed is not a problem, as movies are built over years and they can leave a computer running for days, if necessary. In our application, the model is required as soon as possible; a delay
of even a few seconds could render the model useless, as the OI has already disappeared from view.

Thus, the challenge is not building an accurate model, but rather, it is building an approximate visual model in real-time. There is a trade-off between the accuracy that is obtainable with off-line modelers and speed. In [18], Lee et al. propose a feature-based real-time modeler that comes fairly close to what our object-tracking system requires. Their approach builds a point-cloud model and extracts planes. However, they assume that any OI has a corresponding model in a database, thus, precluding the possibility of tracking a priori unknown objects. Additionally, our object-tracking system requires a visual 3D model built out of texture-mapped surfaces. Therefore, a point-cloud is inadequate.

1.3.3 Visual Object Tracking

There exists a wide range of object-tracking techniques of varying complexity. At the simple end, there are techniques that use background-subtraction and object-matching techniques, such as has been presented by Collins et al. [3], McKenna et al. [4], and Han et al. [5]. Their simplicity reduces the computational requirements at the expense of robustness and versatility. Background subtraction highlights which regions of the image have moved. These regions can, then, be processed to determine which one contains the tracked object. If, however, a region contains independently moving objects, the target object may be lost.

An alternative, but computationally more expensive, approach is to use optical flow to track the motion of an image region. Optical flow estimates the motion from one image to the next. Hager and Belhumeur [6] as well as Fun and Balasuriya [19] have used this method for tracking rectangular regions. Yet another alternative is to track contours (e.g., Isard and Blake [7] and Wu et al. [20]). Using contours rather than a window increases robustness to background clutter.

One difficulty with the above methods is that an object’s visual appearance may change
Chapter 1. Introduction

over time due to lighting or orientation changes. Adaptive models\(^1\) have been proposed to cope with this issue. For example, Jepson et al. [21] use mixture models to represent changing 2D appearance models while Mathews et al. [22] use an adaptive template, but keep the original template to reduce the possibility of drift.

All of the aforementioned techniques track the 2D on-screen position. Whilst this has a number of applications, for many robotic systems, the 3D pose is required. In the literature, a number of 3D tracking algorithms have been proposed (e.g., [8], and [23]-[34]). Of these, [8], [27], and [29]-[34] track the target object’s orientation as well as its position. These can be divided broadly into feature-based and template/visual-model based methods. Feature-based methods are more common (e.g., [27]-[34]) due to their lower computational requirements. They track local features such as corners and lines in order to track a whole object. This does limit the tracker to a small subset of the available data (e.g., corners and/or line-segments). Moreover, it is not possible to model complex objects and patterns in this manner. They are also sensitive to feature detection failure.

Template/visual-model based methods use a complete template/projected-model in the tracking process, i.e., they are global tracking methods. These methods are less common because they are more computationally intensive than their feature-based counterparts. However, they use more of the available data than feature-based methods and are able to model complex objects and patterns. As such, template/visual-model methods have great potential. Examples of such methods have been presented by Jurie and Dhome [8] and Cascia et al. [23]. Jurie and Dhome’s method in [8] requires off-line training that is specific to a particular object, precluding the possibility of extending their method to tracking \textit{a priori} unknown objects. The method presented by Cascia et al. is specific to tracking heads. The method presented in this thesis is also visual-model based, but is able to track \textit{a priori} unknown objects as well as objects whose models are provided in advance.

\(^1\)Adaptive models change over time in order to adapt to changes in the object being modeled.
Despite the advantages of visual-model based object trackers, there are a number of barriers to their practical use. Being a global algorithm, robustness to occlusions is difficult to achieve. Proponents of feature-based methods point to this problem as a reason to use feature-based methods instead, which often are able to cope with partial occlusions simply by allowing tracking by catching only a subset of the visible features (e.g., [20], [30], [31], and [34]-[38]). Several techniques do exist for occlusion-robust template-based tracking (e.g., [21], and [39]-[41]). However, these are for 2D object tracking, and not full 6-dof pose tracking.

There has been some progress with robustness to partial occlusions for 6-dof visual-model based trackers. Mittraoiyanuruk et al. [42] use robust estimation to achieve occlusion robustness. Liebelt and Schertler [43] take a different tact; they use swarming particles, an optimization technique. Unfortunately, neither is real-time. For example, despite off-loading processing to a GPU (Graphics Processing Unit), Liebelt and Schertler’s method [43] operates at 0.5 fps. This is far below the 10 fps minimum needed for robotics purposes.

1.3.4 Object Tracking for Autonomous Navigation and other Real-Time Applications

Real-time applications for object tracking, such as autonomous navigation, have specific requirements, such as:

- Tracking of orientation as well as position,
- Obtaining pose estimates in world or Autonomous Vehicle (AV) coordinates,
- Coping with cluttered environments, and
- Operating in real-time (i.e., >10 fps, as mentioned above).

In [44], on visual servoing, Hutchison et al. report that vision for use in robotic systems has been in development since the early 1980s and that numerous 2D pose-tracking algorithms
have been successfully applied to robot navigation. For example, Sen Gupta et al. [45] and Wong and Benhabib [46] use an overhead vision system to provide feedback to their respective Autonomous Vehicle (AV) navigation systems. These, typically, impose a ground-plane assumption. Whilst this may be acceptable when the motion is indeed on a flat plane, such systems cannot be used on uneven terrain or for flying objects. For object tracking to be successfully applied to autonomous navigation under any motion, robust, full 6-dof pose tracking is required.

Recently, some 6-dof pose trackers have also been applied to robotic applications. Both Kyrki and Kragic [33] and Vincze et al. [34] track by matching a 3D model of features such as points/lines. Almost all trackers used in real-time control systems have been feature-based, primarily because their lower computational requirements makes achieving real-time operation much easier.

1.4 Research Objectives

The primary objective of this thesis is the development of a 6-dof object-tracking methodology suitable for use in autonomous and robotic systems. This requires real-time performance. Most existing 6-dof object trackers operate by tracking features such as corners and line-segments. The chief disadvantage, however, is that they cannot model complex objects effectively and use only a small subset of the available data for tracking. Instead, this thesis presents a visual 3D model-based method, which overcomes this issue. 3D pose is defined by

$$\mathbf{P}(t) = \begin{bmatrix} \mathbf{R}(t) & \mathbf{p}(t) \\ \mathbf{0}^T & 1 \end{bmatrix},$$

(1.1)

where $\mathbf{R}(t)$ is a $(3 \times 3)$ rotation matrix representing the target’s orientation at time $t$ and $\mathbf{p}(t)$ is its position. $\mathbf{P}(t)$ is essentially the pose of a reference frame anchored on the OI.
Tracking an OI in real-time is a computationally intensive task; even more so with visual-model-based trackers. While this could be addressed by implementing the algorithms in custom hardware inside Field Programmable Gate Arrays (FPGAs), such an approach is very time consuming as it involves designing at the individual logic-gate level. Instead, in this Thesis, a Graphics Processing Unit (GPU) will be used in tandem with the main CPU of a standard PC. This allows decades of research in real-time graphics hardware design to be harnessed for computer vision. GPUs found on modern graphics cards have programmable rendering pipelines and are well suited to vector/image processing, often exceeding the capabilities of CPUs. The Radeon X800 from ATI [47], for example, can perform up to 200 Giga-Floating-Point Operations per Second (GFLOPS), whereas a Pentium 4 running at 3 GHz performs 12 GFLOPS [48].

Another major aspect of this thesis is the ability to track \textit{a priori} unknown objects. All 6-dof pose trackers found in the literature require a model to be provided beforehand. A 3D model is essential as orientation can only be extracted based on appearance. As a result, all existing 6-dof object-trackers can only track objects whose models are given beforehand, thus, limiting their usefulness. For this reason, one of the objectives is to overcome this limitation via the development of an on-line modeller. An on-line modeller must be able to rapidly separate an OI from its surroundings, and build a model. This is in itself a complex problem which has not yet been addressed in the current literature.

Before an object can be modelled and tracked, an OI must be selected. Even if known objects are to be tracked, the object must still be identified. Because what constitutes an OI is application dependent, the goal is to develop an OI selection framework that can be tailored to specific applications. Such a system would contain a set of adaptable modules that identify \textit{interesting} regions within an image based on set criteria. Once the regions are highlighted, an OI selector would select the region that is of greatest interest.

The final issue is partial occlusions. Partial occlusions occur when another object (static or dynamic) comes between the camera and the OI, resulting in partial occlusion. In order to
cope with this problem, object trackers must have some method of tracking with only a partial view of the OI. Whilst this thesis will not attempt to fully solve the partial occlusion problem for generic objects, partial occlusions will be addressed for the *a priori* known case.

As an example, Figure 1.1 depicts how the proposed object tracker would fit into a control system. Herein, in the autonomous-convoying example, the object tracker is being used for the rear vehicle to follow the lead vehicle’s trajectory with a constant separation distance. The OI is the lead vehicle and the convoy controller uses the lead vehicle’s pose relative to the camera mounted on the rear vehicle in order to control the rear vehicle’s motion. In this case, the OI (lead vehicle) may be known beforehand, thus, an object modeller would not be required. An OI selector would still be needed in order to start the tracker in preparation for convoying.

1.5 Thesis Overview

This Thesis presents a novel object tracking system capable of tracking the full 6-dof pose of an *a priori* unknown object. Object tracking is achieved by matching a visual 3D model to input data via optical flow. An modeller builds models of *a priori* unknown objects on-line,
so that they can be tracked. Also addressed is the issue of tracking initialization. A modular and expandable framework automatically selects objects that are of interest to the system. The framework can be customized in order to match whatever target application the tracking system is being applied to. Each of the sub-problems is presented in isolation first, before the entire system is assembled and integrated into a complete object-tracking system.

The proposed methodology is presented as follows:

Chapter 1 has presented an overall introduction to the tracking problem.

Chapter 2 presents an overview of the proposed object tracking methodology, and details the OI selection framework and tracking initialization. Implementation details of two example Interest Filters (IFs) are detailed in order to demonstrate the OI selection framework’s operation. This chapter also includes initial pose estimation, which is an alternative to OI selection that estimates the initial pose of an object for which one has the model \(a priori\). Experimental results are presented to confirm the operation of the OI selector and initial pose estimator.

Chapter 3 details the on-line modeller. This includes the sub-components of the modeller such as the Depth-Map Extractor (DME) and projective-texture extractor. As the on-line modeller is dependent on the OI selector, interactions with the OI selector are described.

Chapter 4 details the core object-tracking methodology. Additionally, methods of improving robustness to large motions, lighting variations, and partial occlusions are presented. Achieving real-time performance is also addressed via exploiting GPUs and an observed property of images, i.e., colour-gradient redundancy. In experiments, the colour-gradient redundancy algorithm provided a fivefold performance increase. Finally, experimental results are presented demonstrating the object-tracker’s robustness and performance when operating standalone (i.e., without the OI selector and modeller in the loop).

Chapter 5 describes the full system implementation, combining the OI selector, modeller and object tracker into one object-tracking system. Experimental results are also presented to confirm the viability of the proposed methodology.
Chapter 6 highlights this work’s contributions and presents some recommendations for future work.
Chapter 2

Tracking Initialization

2.1 Overview

Herein, tracking is defined as maintaining an estimate of the 6-dof pose (Equation (1.1)) of an Object of Interest (OI) over time. This is the pose of a reference frame anchored to a fixed point on/in the OI. The reference frame could be placed anywhere; though, it would be preferable to place it at a practical place if possible, such as the centroid of the object.

A robotic system would typically require more information than just pose in order to fully interact with an OI. Properties such as shape and object type are also important. However, characterizing the OI is beyond the scope of object tracking, and this Thesis. Henceforth, it will be assumed that some external system is responsible for characterizing the object.

The object tracker provides a stream of estimated poses of the chosen reference frame over time. Characterization would occur on-line, but would not need to work in real-time as the object tracker provides information about motion during and after the characterization process. For further clarity, let us imagine that the object tracker starts tracking at Frame $n$. An object-recognition algorithm takes the input images at Frame $n$ and proceeds to recognize the object and determine where the chosen reference frame that is being tracked is attached to the object.
This process takes \( m \) frames. Thus, at Frame \( n + m \), the robotic system has identified what type of object it is, and how it is currently oriented (by using the tracker’s pose estimate at Frame \( n + m \)). It is now fully prepared to interact with the object; no more characterization is required as the pose of the reference frame is all that is required for future interaction.

A broad overview of the object-tracking system is given in Figure 2.1, where the object tracker is only part of the whole system. The relevant Chapter/Section for each module is listed in the figure. OI selection is for tracking initialization only; hence, it is displayed using dashed lines. The modeller and tracker are the other two major modules. The Interest Filter (IF) framework is part of the OI selector. However, it is also used by the modeller.

The OI selector’s task is (as mentioned earlier) to identify and select an object that is of interest to whatever end application the object-tracking system is being used for. The key to achieving this is the modular IF framework (shown separately in Figure 2.1). This framework uses a set of connected IFs that each highlights interesting regions based on separate criteria, such as colour or position. The output is an Interest Map (IM) that highlights “interesting” regions. Different combinations of filters with different settings can highlight different objects. The OI selector then searches for the region of greatest interest, which becomes the OI.

Once an OI has been selected, a model must be built for it. Without a model, the 6-dof pose cannot be tracked. The modeller takes the given OI-region, input images, and IM, and proceeds to build a tessellated model. The IM is used to separate the OI from background clutter. As the OI moves and rotates over time, the modeller would need to update the OI model. This is necessary as only the visible side of the OI can be modelled, i.e., the model is incomplete. In order to maintain tracking as the OI rotates, the model must be updated to include parts that are now visible. The estimated pose is, thus, fed back to the modeller when updating the model (once every \( m \) frames).

With the model built, the object can finally be tracked. At this point, the object tracker takes over, and theoretically does not need any of the other modules any more. However, for
the reasons given above, the modeller is periodically passed the current estimated pose so that it can update the model. This is the only interaction between the tracker and the rest of the system after initialization. The object tracker projects the model onto the image plane at its current estimated pose. Optical flow is used to compare this projection to the actual image, and estimate the 6-dof disparity between the predicted and actual poses. The pose estimate is obtained by combining the predicted pose with the estimated disparity. This is repeated for every frame, producing a stream of pose estimates for the OI.

The pose has already been defined in Equation (1.1), as both the position and orientation
of a reference frame in 3D space. Images, on the other hand, are 2D projections of 3D space. Thus, the relationship between 3D position and its 2D projection is fundamental to all work in this Thesis, Figure 2.2. In Figure 2.2, the 3D point, \( p \), is projected onto the image plane, resulting in the 2D point \( p' \). Every point, \( p \), on an object’s surface has a corresponding point, \( p' \), on the image plane. The projection equation for an ideal camera is:

\[
p' = M_{int}M_{ext}p,
\]

where \( p' \) is a \((3 \times 1)\) vector denoting the 2D position of the projection of point \( p \) in homogeneous coordinates\(^1\). The 2D image coordinates are found by normalizing \( p' \) such that \( p'_z \) is equal to one, i.e.:

\[
q = \begin{bmatrix}
p'_x/p'_z \\
p'_y/p'_z
\end{bmatrix}
\]

where \( q \) is a \((2 \times 1)\) vector giving the 2D projection in image coordinates. The \((3 \times 4)\) matrix \( M_{int} \) is the camera’s internal parameters:

\[
M_{int} = \begin{bmatrix}
\frac{f}{s_x} & 0 & -o_x & 0 \\
0 & \frac{f}{s_y} & -o_y & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}, \tag{2.3}
\]

where \( f \) is the camera’s focal length, \( s_x \) and \( s_y \) are the \( x \)- and \( y \)-axis scaling factors, and \( o_x \) and \( o_y \) are the coordinates for the point at which the optical axis of the camera pierces the image plane. \( M_{ext} \) is a \((4 \times 4)\) matrix that transforms 3D coordinates between the world frame to the camera’s coordinate frame (i.e., it is the inverse of the camera’s pose, which is given by (1.1)).

The camera’s 3D reference frame origin is located at the focal-point, Figure 2.2, with the \( z \)-axis pointing along the optical axis and the \( x \)- and \( y \)-axes matching the 2D axes of the image on the

---

\(^1\)Homogeneous coordinates are used in projective geometry and are defined up to a scale-factor. Thus, a 2D point is given in homogeneous coordinates as a three-element vector, in which the third element is set to one.
image plane. The 2D \( x \) and \( y \) coordinates for a point can be found by normalizing \( p' \) such that \( p'_z \) is equal to one (i.e., divide \( p' \) by \( p'_z \)).

The ideal camera model (2.1) assumes perfect projection, such as is obtained with a pinhole camera. Real cameras use lenses that may cause radial distortion. It is assumed that the radial distortion for the camera is sufficiently small as to be negligible. For cameras with larger distortion (e.g., web-cams), input images should be corrected for distortion using established techniques before being passed to the tracking system (e.g., [49, 50], and [51]). In particular, Fung [51] proposes a graphics-card-based distortion-correction algorithm that operates in real-time.

### 2.2 Object of Interest Selection and Tracking Initialization

Before examining the OI selection/tracking initialization process in detail, it is necessary to define what an OI is: *an OI is an object or subject that is of interest to an external system and,*
hence, needs to be tracked. A mobile robot, for example, might be most interested in the object with the least time to impact. Another robotic system might be more interested in a robot it needs to dock with. A manufacturing robot might be interested in a range of components/tools. By the definition above, there are no defining visual attributes that can be used for recognizing a generic OI.

### 2.2.1 OI Selection Framework

The proposed OI-selection methodology overcomes the lack of generic visual attributes by providing a framework that can be customized to suit the target application. A set of serially connected Interest Filters (IFs) generate an IM that highlights regions that are “of-interest”. This, coupled with motion segmentation, is used by the OI selector to select a region with the highest level of interest. Different applications can use different filters and/or customize filter settings to suit what is deemed to be “interesting”. Thus, the same framework can be used to identify and track different objects.

An example OI selector is shown in Figure 2.3. A series of IFs are connected to a region of interest selector. The region of interest selector includes a motion segmentor which is not shown separately for simplicity. Each IF is based on different criteria. The colour IF highlights regions that match the colour of OIs; this alone will probably not be enough to identify objects that are of interest. Next, a positional IF is used that highlights objects that occur within a specified 3D region. The next IF filters based on motion. Finally, if particular objects are sought, an object-recognition IF could be used. Each IF takes the interest-map from the previous and reduces the total region of interest. The final IM is used by the region of interest selector to select the region that is of greatest interest to the system.

The working of the OI selection framework could be better understood via an example: Let us consider a room that Automated Guided Vehicles (AGVs) require free access to, but only humans with an appropriate security clearance have access to, Figure 2.4. An active face/iris-
recognition system would enable such individuals to simply walk up to the room and enter, whilst others would walk up and be barred. This would require tracking people in order for the recognition camera to be correctly aligned, and in order to ensure that the person entering is the person recognized earlier. An IF framework could select the OI as follows: a colour-based IF filters out floor and AGVs based on their colour, resulting in Figure 2.5(a); a positional IF focuses attention on objects close to the restricted-room entrance, Figure 2.5(b); a velocity-based IF highlights regions moving toward the entrance, Figure 2.5(c). Finally, the OI selector selects the region of greatest interest and, for example, a face-recognition system determines whether or not the individual has access rights, Figure 2.5(d).

**Interest Filters**

The core of the OI-selection system is the IF framework, in which individual IFs are serially connected, forming a filter bank (i.e., the top half of Figure 2.3). Each IF takes the previous filter’s IM as input and outputs a new IM with a reduced total area of interest. The first filter in the system receives a blank IM filled with ones. For efficiency, fast IFs should be placed first in the chain. Subsequent IFs, then, only need to operate on regions still marked as of-interest. Thus, the object-recognition IF in Figure 2.3 would not have to operate on the entire image.
Figure 2.4: A restricted room requires OI selection for an active face-recognition system.

Figure 2.5: A colour IF highlights humans (a); a positional IF highlights regions close to the door (b); a velocity IF highlights regions moving toward the restricted room (c); thus, isolating the OI to track (d).

which would be prohibitively expensive computationally.

An interest-map is an image in which each pixel contains a level of interest for that pixel, instead of a colour value. An interest value of one signifies full interest; a value of zero signifies no interest at all. Naturally, if this is being stored in 8-bit integers, the range would be [0-255]. Mathematically, however, [0-1] is used.

Different IFs require different input data. A colour IF requires only the input images. Positional IFs, on the other hand, would require a DM too. If an IF is filtering based on
motion, it might require a motion map, or the current and previous input images. For 3D motion, DMs from multiple time steps may be required. Nevertheless, there is a common structure, Figure 2.6; all IFs take a previous filter’s output IM as input and output a new one. All IFs also have one or more IF specific inputs, as outlined above.

![Diagram of IF structure](image)

Figure 2.6: The common structure of all IFs.

Given the wide range of different IFs that could be implemented, and the wide range of different inputs, it is impossible to define how an IM should be generated. What is possible, however, would be to define how IMs should be combined. If an IF generates a local IM, this can be combined with the IM using a common transform. For Filter $n$, the output IM would be given by:

$$S_n = S_{n-1} \cdot S_{n,calc}, \quad (2.4)$$

where $S_n$ is the interest value output by IF $n$, and $S_{n,calc}$ is the interest value calculated by the filter operation for IF $n$. This is a per-pixel operation. Thus, a new IF only needs to specify a local interest value, $S_{n,calc}$; its interaction with the rest of the filter bank is decided by (2.4).

Let us now examine a possible IF implementation. If one is searching for objects that have particular colours on its surface, a good place to start would be to reject image regions
containing other colours. Such a colour IF could be defined by the following formula:

\[
S_{n,\text{calc}} = \text{clamp} \left( \sum_{i=0}^{M} \text{clamp} \left( 1 - \frac{\|I - I_i\|^2}{I_{i,\text{md}}} , 0, 1 \right), 0, 1 \right),
\]

(2.5)

where \( I \) is the RGB colour value for the current pixel, \( I_i \) is the Reference Colour \( i \) to which it is being compared, and \( I_{i,\text{md}} \) is the maximum Euclidean distance allowed between the pixel’s colour and the reference Colour \( i \). There are \( M \) reference colours to which the input image is compared. The function clamp clamps values to the range 0 to 1, i.e.,

\[
\text{clamp} (x, a, b) = \begin{cases} 
  x, & x \in [a, b] \\
  a, & x < a \\
  b, & x > b 
\end{cases},
\]

(2.6)

with \( a \) and \( b \) set to 0 and 1, respectively. The result of (2.5) is an interest value that is high when a pixel’s colour is close to one of the desired colour values and low elsewhere. \( I_{i,\text{md}} \) limits the radius around a desired colour that produces an interest value above zero.

**Motion Segmentation**

The IM itself is insufficient to select an OI. It highlights regions of interest, but these regions are not well defined. A single object may have only certain fragments highlighted, Figure 2.7. The IM in Figure 2.7(b) resulted from an IF bank containing a colour-filter, and a background-subtraction-based motion IF. In this case, there is only one moving object, so the OI could still be identified. However, with multiple moving objects, different objects would be hard to distinguish.

Some form of image segmentation is required in addition to the IM. Image-segmentation algorithms exist that can divide an image into regions. However, they tend to be non-real-time and segment into regions of uniform colour. Real objects, however, rarely are composed of a single colour. Thus, an image segmentor would over-segment the image, and hence, would be unsuitable for that task.
The proposed approach is to use a motion segmentor. Motion segmentation divides an image up into regions moving under similar transforms. Assuming that objects are rigid, regions within an image that move under different transformations belong to different objects. Therefore, each region output by a motion segmentor can be treated as a separate object.

Motion segmentation would not work if all objects are stationary. If searching for static objects is required, an alternative segmentation algorithm would be required. However, given that they are static, this would not require real-time performance, so a slower algorithm could be used. In this Thesis, it is assumed that the OIs are moving and real-time selection is required.

Any motion-segmentation algorithm can be used, provided that it operates in real-time: The more accurate the segmentation, the better the resulting model and tracking accuracy would be. As segmentation must be performed simultaneously with various other tasks (e.g., interest filtering, modelling, and tracking), computational requirements are very stringent. Whilst 3D motion segmentation would be preferable, such algorithms are non-real-time at present. 2D motion segmentation can still provide usable results, as discussed in Section 2.2.3.

**Region-of-Interest Selection**

Following IM highlighting and motion segmentation, the region that is of greatest interest must be selected. The proposed OI-selection algorithm essentially constructs a list of properties for each region by scanning through the segmentation map, IM, and DM (including its texturedness image, $T$). For every region (or segment), $R_i$, where $i \in [1, N]$ and $N$ is the number of regions,
the properties are:

\[
TotalArea_i = \sum_{(x,y) \in R_i} 1, \quad (2.7)
\]

\[
InterestArea_i = \sum_{(x,y) \notin Z_i} 1, \quad (2.8)
\]

\[
TotalInterest_i = \sum_{(x,y) \in Z_i} S(x,y), \quad (2.9)
\]

\[
MeanInterest_i = \frac{TotalInterest_i}{TotalArea_i}, \quad (2.10)
\]

\[
DepthPixelArea_i = \sum_{(x,y) \in W_i} 1, \quad \text{and} \quad (2.11)
\]

\[
MeanDepth_i = \sum_{(x,y) \in W_i} D(x,y). \quad (2.12)
\]

Above, \(S(x,y)\) and \(D(x,y)\) are the IM and DM (i.e., distance to the surface along the camera’s optical axis, \(z\)) values at pixel \((x,y)\), respectively. \(Z_i\) is a sub-region to \(R_i\) in which \(S(x,y) > \tau_S\), i.e., the interest level is greater than a threshold, and \(W_i\) is a sub-region of \(Z_i\) containing all pixels belonging to \(Z_i\) for which the texturedness is greater than a threshold, i.e., \(T(x,y) > \tau_T\) (\(\tau_T\) is set to the same value as used by the depth-map extractor). This additional restriction for the pixels that are used to calculate the mean depth ensures that only pixels with reliable depth values are used.\(^2\) Texturedness is covered in more detail in Chapter 3. Another item that needs to be calculated is the region’s bounding box. This is combined with the mean depth for that region in order to estimate the region’s physical width and height (as viewed by the camera).

The physical width and height of region \(R_i\) are obtained by projecting the top-left and bottom-right corners of the bounding box out into 3D space by \(MeanDepth_i\) in camera coordinates. This is the reverse of Equations (2.1) and (2.2) with \(q\) set to the desired corner’s 2D image coordinates and \(p'_z\) set to \(p'_z = MeanDepth_i\). The width and height are the difference between the 3D projections of the bottom-right and top-left corners along the \(x\)- and \(y\)-axes, respectively. This provides us with an approximate 3D size. If the size of a region

\(^2\)Depth can only be measured in image regions with non-zero image gradients (i.e., textured regions).
is approximately the same as that of our desired OI, then, it is a potential OI; otherwise, it is discarded.

Once all the above properties have been calculated, the OI region selector must decide whether an OI exists and, if so, which OI is of greatest interest. For an OI to exist, the following conditions must be met:

- its physical width and height are within a predetermined range,
- its mean and total interest levels are above their respective thresholds,
- its total area is greater than a threshold, and
- \( \frac{Z_i}{R_i} > \tau_A \) (i.e., the proportion of the OI region covered by interest pixels is high enough).

The selected OI is the region within \( R_i, i \in [1 - N] \), that satisfies the above conditions and has the highest mean-interest level. This selected region region is labeled \( R \). This is the region that is passed on to the modeller and the object tracker.

**Initial-Pose Generation**

There is one final task required before the modelling process can begin: an initial pose must be generated. The modeller builds a model around a reference frame; it does not select one itself. An OI’s reference frame can, theoretically, be placed anywhere relative to the OI. For a given application there may be a preferred reference frame. In some cases, the geometric centre may be most appropriate; for others, the centre of mass, or even some other point within the object. When the target OI’s model is not known \textit{a priori}, it could be difficult to find such reference frame. Despite this, a suitable reference frame must be selected based on the available data.

The specific initial-pose-generation algorithm is not critical to the operation of the reference frame. As a general rule, the reference frame should be placed within the object, or close to
the visible surface. The algorithm presented below performs the desired task. It is an example of how such algorithms can quickly generate a reference frame using limited data.

The implemented initial-pose generator places the reference frame at the 2D centroid of the object’s bounding box, just behind the mean depth of the surface. The reference frame’s pose is generated as follows:

- The 2D mid-point of region $R$’s bounding box is calculated,
- The reference frame’s 3D position is obtained by projecting the 2D region mid-point back into the scene by the mean depth plus an offset, and
- The orientation is set to match the world coordinate system’s orientation (i.e., it is set to the identity rotation matrix).

This procedure places the reference frame approximately in the middle of the visible object, just behind the 3D surface that will be generated by the modeller. The reference frame’s orientation can be set to anything as this does not affect tracking. It is the responsibility of an external system to characterize the object and determine how to interact with it (e.g., pick a reference frame with the $x$-axis aligned with a vehicle’s main direction of motion).

### 2.2.2 Initial-Pose Estimation

There may be situations in which, instead of building a model, an approximate initial-pose estimate may be needed for an existing model. This is particularly true if single-camera tracking is being performed, as 3D geometry cannot be extracted without multiple views. In the single camera case, the object tracker would need to be given an approximate initial pose (including orientation). This is required because the object tracker operates by comparing a projection of the OI’s model at its current estimated pose with the actual OI in the input image. Without an estimated pose, this comparison cannot take place. The estimated initial pose need not be
accurate as the object tracker can cope with large errors in pose estimation. Thus, the goal is to rapidly provide an approximate initial pose for the OI.

A number of different 3D-pose-estimation techniques have been proposed in the literature. For example, Chen and Sockman [52], Lee et al. [53], Lepetit et al. [54], and Johanson and Moe [55] all use feature-based object recognition for pose estimation. Dunker et al. [56] propose using neural-networks for pose estimation. Ekvall et al. [57] present an alternative method using color co-occurrence histograms. Another possible technique is to adapt Principal Component Analysis (PCA) for pose estimation [58].

While initial-pose estimation is largely beyond the scope of this Thesis, a proof-of-concept initial-pose estimator has been implemented and tested with the object tracker. An object-tracker is not useful if it cannot be initialized, thus, it was deemed necessary to confirm that initial-pose estimation was feasible. The implemented initial-pose estimation algorithm uses PCA, and is based on Zhao et al.’s algorithm [58]. PCA was chosen as it is an appearance based technique that was easy to implement with the available code-base. An overview of PCA is presented in Appendix A.

In the implemented algorithm, pose estimation is a two-step sequential process: position estimation and orientation estimation. 3D position is estimated from the size and position of the target object’s on-screen bounding box. Orientation, on the other hand, is based on appearance. PCA is used for orientation estimation.

The region-of-interest selector (or some other external system) provides a region \( R \). If this is a single-camera system, the depth will not be calculated. Instead, the region \( R \) is a bounding-box, containing the OI. However, depth is not required, as it can be estimated based on the size of \( R \) on-screen; the size of an object is part of its model. Region \( R \) is converted to a bounding-square that has the same centre, and encloses the original bounding box. A bounding square is used as this is what the PCA-based orientation estimator requires. This square has a centre at \( p \) and a width of \( w \). The target object’s model includes a bounding sphere of radius \( r \). Using
this information, the target object’s 3D position in camera coordinates is:

\[
p_{\text{target}} = \begin{bmatrix}
    (p'_x - o_x)s_x / f \\
    (p'_y - o_y)s_y / f \\
    f_x / s_x w / 2
\end{bmatrix},
\]  

(2.13)

where \( f \) is the camera’s focal length, and \( s_x \) and \( s_y \) are the camera’s \( x \) and \( y \) scale factors, respectively. The parameters \( o_x \) and \( o_y \) give the image centre’s offsets relative to the point at which the optical axis pierces the image plane. One can note that (2.13) assumes \( s_x \approx s_y \). This still needs to be converted to world coordinates, effectively performing the inverse of the projection equation, (2.1).

Orientation is estimated by comparing the square input region to a reference database via PCA. The square region is down-sampled to match the PCA database’s image size. It must be filled by the target OI, or orientation estimation would fail. The orientation in the database that best matches the input image is the estimated initial orientation. The initial pose is obtained by combining the orientation estimate with the position calculated in (2.13).

As a PCA database is used for orientation extraction, this database must be constructed \textit{a priori}. It is achieved by projecting the target’s (known) model onto a square image of set size at various orientations. The orientations within the database should be spread out uniformly across the range that can occur. The more orientations, the more accurate the orientation estimate can be; but, increasing the number of orientations also increases the database size and decreases the speed of orientation estimation.

### 2.2.3 Experiments

Operation of both the OI selector and the initial-pose estimator was confirmed experimentally. In both cases, motion sequences were captured using a Sony SNC-RZ30N camera. An OI was moved along a predefined trajectory using a precision \( x \)-\( y \) table. Synthetic sequences were also generated and used where appropriate. Synthetic sequences have the advantage
that all parameters are known precisely, whereas real sequences are limited in precision by measurement accuracies.

**OI Selection**

Operation of the OI selector was tested using a series of motion sequences. In all of the experiments, two IFs were used: a colour IF (defined by (2.5)) and a motion IF based on background subtraction. The colour IF was adjusted to match the OI. The tests were performed within the complete object-tracking system, including the modeller and tracker. This tests the OI selector within its target application. Five cameras were used for DM Extraction (DME).

The motion IF was designed to detect regions. It does this by subtracting a reference image of the background from the current input image (i.e., using background subtraction). All differences above a certain threshold are marked as moving. The background subtraction algorithm in this IF is based on the adaptive algorithm by Collins and Dennis [3] minus the region-filling algorithm used to fill in solid colour regions. Region filling was not required for our IF. Moving pixels were given an interest value of one; static pixels were given a value of zero.

The motion segmentor that was used is also based upon an existing algorithm; the algorithm presented by Chung et al. in [59]. It is block based and operates in real-time. The image is divided into \((8 \times 8)\) blocks, for which optical flow is calculated. Adjacent blocks with similar motions are grouped into regions. Precise details of the inner workings are not required for understanding the experiments and will, therefore, not be covered here.

Another module that the OI selector uses is the Depth-Map Extractor (DME). A DM is required by Equation (2.12) and is used to estimate the physical width and height of the object. This is the same DME that is used by the modeller in the next chapter.

A two-object test sequence is shown in Figure 2.8. Red lines in this figure outline the regions; small line segments or dots indicate the direction and magnitude of motion of each
segment. The colour IF was tuned first to the cube (Figure 2.9) and, then, to the globe (Figure 2.10). The OI selector identified and isolated the object of the desired type, built a model, and proceeded to track it. In these examples, tracking proceeded despite incomplete OI models. This is particularly evident in Figure 2.10 (b), where incomplete motion segmentation highlighted only a small part of the globe, resulting in a partial model of the OI. The motion segmentor that was used has difficulty with image regions containing mostly image gradients perpendicular to the motion. This problem is due to segmentation being based purely on motion; motion can only be determined along the direction of image gradients. Despite this issue, an appropriate model is still selected and a partial model built, Figure 2.10 (b). The proposed object tracker was able to successfully track the OI based on the partial model that was given.

Figure 2.11 shows a real-world test. In this test, the OI was a cardboard box. Once again, the colour IF was tuned to the box. The OI selector separated the box from the background, resulting in successful tracking initialization.

Examining the results above, there is room for improvement in the motion segmentor. The current algorithm segments based purely on optical flow and is 2D only. Using additional information such as colour (e.g., Cucchiara et al. [60]) could improve object separation. Using higher-order flow models, such as affine flow [61], could yield better segmentation when objects rotate. Achieving real-time operation with higher-order models could be challenging. Nevertheless, the overall OI-selection concept has been validated.

Initial-Pose Estimation

The initial-pose estimator detailed above is a proof-of-concept. The goal was to show that such a system could be used to initialize the object tracker. Hence, what is being sought is the successful initiation of object tracking. As such, the results presented are tracking errors for the object tracker, not direct results for PCA-based pose estimation. While the object tracker has not yet been discussed, it is not necessary to understand its inner workings in order to interpret
Figure 2.8: The motion-segmentation map for a sequence containing two objects.

Figure 2.9: Using the motion-segmentation map in Figure 2.8, the IF framework has been tuned to the cube, resulting in the IM (a) and model (b).

Figure 2.10: Using the motion-segmentation map in Figure 2.8, the IF framework has been tuned to the globe, resulting in the IM (a) and model (b).
the results below.

Figure 2.12 shows an example input sequence, in which the OI is a cube with letters on the side. In this experiment, the initial-pose estimator was passed a bounding box generated by dragging a box with the mouse. This provides a simple method of introducing variability into the experiment as the box dimensions and position can be changed at will.

Dragging a box, which approximately enclosed only the cube (i.e., the OI), resulted in a tracking response such as in Figure 2.13. In Figure 2.13, the tracker successfully took the rough initial estimate from the pose estimator and proceeded to track the OI. When examining the first few frames, one can note large initial errors. This is due to the coarse initial-pose estimate which is quickly corrected by the object tracker. It shows the object tracker taking the approximate initial pose, and successfully tracking to high accuracy. The object tracker incorporates algorithms to cope with large disparities between the estimated pose, and actual pose. In this experiment, the PCA database for orientation estimation contained 288 images with a resolution of (64×64) pixels that were taken from different orientations, yielding a resolution of 1.25° about one axis. These images were generated with the same 3D graphics engine used by the object tracker.

It is important to note that the initial-pose estimator would also, on occasion, generate erroneous initial poses that would cause tracking failure. In some situations, the initial pose was correct, but the orientation was off far enough that a different side of the cube was visible. This resulted in tracking with a static offset in the orientation. The object tracker has some robustness to modelling errors, allowing it to track the object in situations such as this. Nevertheless, in most situations, the object tracker was successfully initialized, and tracking proceeded successfully.

One difficulty evident in the results is that orientation can only be extracted based on appearance. The implemented algorithm is essentially an adapted object-recognition algorithm. Instead of recognizing different objects, it recognizes different views of the same object. Ob-
Figure 2.11: The motion-segmentation map (a), IM (b), and resulting model for a real-world motion sequence (c).

Figure 2.12: Frames #0 (a), #59 (b), and #119 (c) of a real-object sequence.
Figure 2.13: The positional (a) and orientational (b) tracking errors for the motion sequence shown in Figure 2.12.
ject recognition is still an active field of research, and performance is measured in true-/false-positives and true-/false-negatives. This implies that recognition algorithms will fail to correctly recognize objects some of the time. For initial-pose estimation, this manifests itself as incorrect poses. Because a cube bears some similarity when rotated by $90^\circ$, it can be difficult for the PCA-based algorithm to distinguish different sides. This results in some of the failures noted above.

As object-recognition technology improves, initial-pose estimation will also improve. In particular, real-time or near real-time recognition algorithms are required. The results presented above show that this could be a viable tracking initialization step. Further research into orientation estimation would make this technique more versatile.

### 2.3 Summary

This Chapter has presented a broad overview of the complete object tracking system; from object tracking itself, to OI selection, and to modelling. It also discussed OI selection and initial-pose estimation, both of which are tracking initialization tasks.

The object-tracking system comprises three major systems: the OI selector, modeller, and the actual object tracker. The OI selector and modeller are support modules for the object tracker. They are essential to successful tracking initialization, particularly when the OI is *a priori* unknown. The OI selector searches for and selects an OI to track, providing its on-screen region. The modeller rapidly builds a model of the chosen OI. Once this is done, the model and the OI’s initial pose are passed on to the object tracker.

Once running, the object tracker can operate independently. It has a model of the OI and can use this in order to track it. However, because the modeller can only see one side of the OI, the model provided to the object tracker is incomplete. For this reason, the modeller must periodically update the model.
At the core of the OI-selection system is the IF framework. Also used by the modeller for segmentation, IFs are connected serially to form a filter bank. Each IF highlights regions based on specific criteria, such as colour. The combined result of these filters is an IM that highlights regions of interest. The IF bank can be customized to suit any target application via appropriate selection of IFs and their parameters.

OI selection also requires some form of image segmentation. A motion segmentor is used for this task as regions moving under the same transformation can, generally, be assumed to belong to a single object. The region of interest selector takes the IM and examines the regions (segments) given by the motion segmentor. The region with the highest level of interest is chosen as the OI.

The OI selector's operation was confirmed experimentally. Experiments containing multiple objects were run with the IF bank tuned to be “interested” in different objects. The OI selector successfully highlighted the desired OI and initiated tracking of that OI.

In some cases, an OI's model may be provided beforehand. This is essential if only a single camera is used for tracking. In such situations, the modeller is not used, but an approximate initial pose is needed in order to initialize the object tracking. Without an approximate initial pose, tracking cannot commence, rendering the object tracker ineffective. A proof-of-concept initial-pose estimator was presented, demonstrating the feasibility of tracking initialization with a rough initial-pose estimate. Successful tracking initialization was demonstrated experimentally.
Chapter 3

On-line Modelling of A Priori Unknown Objects

This chapter presents an on-line modelling methodology for the tracking of objects whose 3D geometric models are not known. Although, 2D trackers require the models of the objects that they need to track, such models are typically easier to generate on-line from input images (e.g., Wu et al. [20]). Furthermore, of the myriad of available different 3D modelling techniques (Section 1.3.2), only a few can operate in real-time and none of these are capable of modelling a single object separated from background clutter.

One must note that not only must modelling be carried out on-line, it must also occur concurrently with object tracking, which is computationally intensive in its own right. This problem bears some similarity to Simultaneous Localization and Mapping (SLAM) algorithms (e.g., [62]) in the sense that, the tracking system must simultaneously model the target Object of Interest (OI) and use the model to track the OI. There are, however, some distinct dissimilarities. For example, SLAM algorithms, typically, treat their environment as static (moving objects may be treated as outliers). This would be acceptable as the goal is to localize the camera/robot relative to its environment. The proposed system, on the other hand, must separate (segment) the OI from its surroundings in order to track it.
Given the above problems, and that 3D reconstruction algorithms can take excessive time to complete, one might erroneously conclude that on-line segmentation and modelling may be unfeasible. However, unlike modelling for the movie industry, tracking of real objects requires only their approximate model. It is possible, therefore, to sacrifice accuracy for speed.

A block diagram of the on-line modeller is shown in Figure 3.1. The Depth-Map-Extraction (DME) algorithm is responsible for extracting 3D information from the scene. It constructs a Depth Map (DM) containing the z-axis (optical-axis) distance from the central (or reference) camera’s focal point to the visible surface at every pixel. The Texture Extractor (TE) is responsible for building a Texture Map (TM) that will contain the fine surface colorations/features. The Mesh Generator (MG) builds a 3D tessellated model (mesh), representing the surface of the OI.

The meshed model is built by incrementally adding vertices (points) to the tessellated model. Locations for these vertices in 3D space are calculated by solving the inverse of the camera-projection Equation (2.1), i.e., solving for \( p \). Solving (2.1) is possible since the DM provides the value of \( p'_{z} \) for every 2D \( q' \).

As was mentioned earlier, separating the OI from its environment is a challenging task in real-time modelling. In the proposed modelling methodology, segmentation is achieved using the Interest Map (IM) and the OI region passed to the modeller by the OI selector. The modeller limits the mesh to the region covered by the OI region. Vertices in the mesh are, in turn, restricted to pixels within the OI region marked as “of interest” in the IM.

The modelling process can be summarized as follows:

- The DME extracts 3D information,
- The TE builds a TM containing surface features,
- The OI region and IM are used for separating the OI from its environment, and
- The MG builds a tessellated model approximating the 3D surface.
The combined result of all of the above is a TM triangle mesh of just the OI; the exact same type of model that is used by real-time graphics software and games. This model can be passed on to the object tracker in order to track the object’s 6-dof pose over time.

### 3.1 Depth-Map Extraction

The DME’s purpose is to extract the visible 3D surface in real-time. It produces a DM which contains the $z$-axis (optical-axis) distance to the visible surface for each pixel, that is, it contains the value of $p_z$ for every pixel. As will be discussed below, depth can only be measured in regions containing image gradients (i.e., textured regions). Obtaining a DM is the first step in
building a 3D model.

Another possible method to extract 3D information would be to match feature points in multiple images. However, one of the reasons for not using a feature-based object tracker would be to enable tracking of more complex shapes that do not have strong feature points. Such objects could be, for example, those with smooth curved edges.

DME is a type of stereo-vision algorithm. A stereo-vision algorithm extracts 3D structure by comparing two different views of the scene from two different, but close, viewpoints. Both cameras view the object from the same side, but have a separation distance. More than two cameras can be used in the 3D structure calculations. This is known as a multi-view stereo algorithm. All stereo algorithms operate on the same principle: they match points on one image with points on the other image (or images in multi-view stereo), and use triangulation to obtain the 3D position in space, Figure 3.1.

![Diagram showing triangulation of points](image)

Figure 3.2: Point $p$ can be found via triangulation of two or more 2D projections: $p'_l$ and $p'_r$.

In theory, any real-time DME algorithm could be used in our modelling sub-system. There are no properties of the currently implemented DME algorithm that are critical for the modeller’s operation. Nevertheless, the behaviour of the DME algorithm does affect the resulting model and, hence, tracking accuracy. For example, most DME algorithms assume that all surfaces exhibit Lambertian reflectance. Any algorithm with the Lambertian assumption would
likely have trouble extracting the depth of any object whose surface violates this assumption. In particular, highly specular objects, such as metals, are likely to cause errors because specular highlights depend on both the position of the light source and the camera. As a result, when using a typical DME algorithm, one can expect tracking of objects with large specular highlights to be less accurate than for Lambertian ones. Care should be taken, therefore, in the selection of the DME algorithm used. The more accurate the DME algorithm is, the better the resulting model would be and the more accurate the object tracking. Wang et al. [63] discuss the performance of various such real-time stereo algorithms.

Another issue with DME algorithms is that depth cannot be measured in textureless-regions (i.e., regions of uniform colour). This is due to there being no depth information in textureless regions; there are no features in flat areas to compare between multiple images, thus, the 3D position cannot be triangulated. As a result, stereo-vision algorithms must guess what the depth is in these regions based on interpolation or smoothness constraints. This is a problem that plagues all stereo-vision algorithms and is computationally expensive to solve. Rather than attempting to fix the textureless region problem, the propose mesh-generator/modeller can address this itself. Thus, interpolation in textureless regions is not necessary.

For use in the proposed modeller, a DME algorithm must operate in real-time and build a DM from the viewpoint of a specific camera. Real-time operation would preferably be at a higher frame-rate than the desired tracking frame-rate as it is operating simultaneously with the rest of the system, and as much computational power free for tracking as possible is needed herein. The modeller requires the DM to be referenced to a physical camera as it needs both a DM, and an image from the same view. This will be discussed in more detail below.

### 3.2 Building the Model Surface

A tessellated model is generated by selecting points on the DM and adding them to a mesh as vertices. The mesh is built out of triangles. Successfully approximating a surface’s shape
requires a certain amount of care in selecting vertices. A regular grid could be used. However, it would have to be a fine mesh with numerous triangles in order to cope with complex shapes. Instead, the proposed method builds an irregular mesh of non-uniform triangles.

DMs built by current state of the art DME algorithms suffer from spikes/pits and other errors. These are most troublesome because they represent unacceptably large errors in depth, which could be problematic if they were to be included in the model. The proposed object tracker can cope with inaccurate models to a certain degree. Pits/spike would cause unacceptably large errors in appearance (and geometry), and would very likely cause tracking failure. DME is still a field of active research, thus, DM quality should improve in future. Nevertheless, it is desirable for the modeller to be able to cope with such errors; if current state of the art DME algorithms are to be used, this is essential. Some method of removing these spikes/pits is necessary. This must all be done within the constraints of real-time operation:

**Step 1: Filtering Noisy Depth Maps:** The first step is to clean up the DM as quickly as one can. Figure 3.3 (b) shows a DM containing spikes (circled in red). These spikes/pits are typically one pixel in size. An averaging (low-pass) filter would still be sensitive to these errors due to their unacceptably large magnitudes. Hence, the proposed system filters the DMs using a $(3 \times 3)$ median filter. After filtering, Figure 3.3 (b) becomes Figure 3.4.

A few spikes/pits remain even after median filtering. Removing these via more filtering would be computationally prohibitive. These remaining spikes are dealt with by the vertex-selection algorithm discussed below.

**Step 2: Selecting Mesh Vertices:** Now that the DM has been “cleaned-up”, it can be used to build a 3D mesh. The key to forming an approximate model is appropriate vertex selection. Poor selection could cause the resulting model to be a poor approximation of the actual OI.

Ideally, a set of principal points would be chosen that minimizes the number of triangles in the mesh for a desired accuracy. Principal points are points on an object’s surface that strongly define the object’s shape. For example, a cube’s principal points are its corners. Principal
points are usually sharp corners or regions in which the surface normals vary quickly. Flat regions do not contain principal points (except at their edges) because they can be perfectly modelled by triangles/polygons.

Quickly finding such suitable principal points in a DM poses a few challenges. Little computation time is available to carefully select principal points within the DM that provide a good surface approximation whilst minimizing the triangle count in the mesh. Moreover, the DM provided will, generally, still contain some spikes/pits or other errors. This prevents the use of the DM itself for principal point selection using techniques such as the one proposed by Pedrini [64] for range images. Added to this, algorithms such as Pedrini’s [64] expect a range-image produced by laser range-finding or other active-sensing systems that can accurately find the depth regardless of surface colorations. In essence, they expect accurate DMs. A DME algorithm, on the other hand, provides much lower quality DMs, and cannot measure depth in regions of uniform colour (i.e., textureless regions). As a result, the vertices cannot be selected reliably by analysing the DM directly.

The technique proposed in this Thesis selects points based on the input image associated with the DM, rather than the DM itself; that is, the image captured by the camera from whose
viewpoint the DM was generated. It is based on two principles:

- Sharp edges, or feature-points, in an image often coincide with principal points of 3D objects; and,
- Depth can only be measured in textured-image regions (i.e., regions with non-zero intensity gradients).

Smooth image regions have depth values estimated based on smoothness constraints and adjacent textured regions. There exists no way to extract the actual depth in smooth regions based on image data.

The first principle listed above is that feature points should make good principal points. However, relying on them once again restricts the models, and hence our tracking ability, to objects with well defined feature points. Smooth curved edges have no definable feature-points. Also, sparsely-spaced feature points can cause large errors at locations between points. Thus, a method of inserting additional feature points is required.

In [64], Pedrini refines a mesh iteratively by progressively inserting additional points based on the error between the DM and the mesh. However, such computations may be costly and make real-time performance difficult to achieve. Additionally, it has already been established
that DMs from current DME algorithms are not reliable enough for such procedures. Hence, rather than adding points based on an error measure, points are added in a grid-like fashion. This ensures that enough points are available to produce a reasonable approximation of the target object. The grid enforces a minimum density of vertices in textured regions. Meshes resulting from this procedure would have more vertices than could be obtained by more complicated techniques. However, the number of triangles in the mesh would not be excessive (i.e., any slow down in tracking speed would be insignificant) and the modeller operates with minimal computational effort.

Prior to discussing the grid-based point-insertion scheme, it would be beneficial to discuss the concept of texturedness. Texturedness has been mentioned above, but not defined. It is a measure of intensity variations within the image, i.e.:

\[
T(x, y) = \sum_{i \in \{R, G, B\}} |\nabla I_i(x, y)|^2,
\]

(3.1)

where \(\nabla I_i(x, y)\) denotes the gradient of Image \(I_i(x, y)\) for Colour Channel \(i\). Regions of uniform colour have zero texturedness, and the texturedness value for a pixel increases as the intensity-gradients increase.

Whilst the proposed vertex-insertion scheme is grid-based, it is not a regular grid. This is due to all the issues described above related to DME algorithms, accuracy and textureless regions. Instead, the image is divided up into a grid of boxes. If a box in the grid contains one or more feature points, those feature points are added to the mesh. Otherwise, the pixel closest to the box’s centre that has a texturedness value over threshold \(\tau_T\) (i.e., \(T(x, y) > \tau_T\)) is selected as the point to add for that box in the grid. If the box has no pixels with \(T(x, y) > \tau_T\), then, no point is added. The principal-point selection algorithm is, thus, as follows:

- Find all feature points using a Harris feature-point detector [65] (Appendix B),
- Calculate the texturedness image, \(T(x, y)\), (note, this can be combined with the Harris feature detector into one step),
Perform non-maximal suppression on $T(x, y)$ using a $(3 \times 3)$ window, and

For each box in an $m \times n$ grid:

- If there are any feature-points within this box, add those points to the mesh,
- Else, if the box contains pixels for which $T(x, y) > \tau_T$, then:

  - Find the closest point to the box’s centre with $T(x, y) > \tau_T$, and
  - Add this point to the mesh.

- Else, proceed to the next box in the grid.

Despite vertex selection being grid based, the resulting vertices are not on a uniform grid. This precludes the use of grid-based meshing procedures. Instead, Delaunay triangulation is used. Delaunay triangulation ensures that the generated mesh is “optimal” [66] in the sense that the error between the mesh and the DM is minimized for the given set of vertices. It achieves this minimal error by minimizing the number of “sliver” triangles in the mesh (i.e., very thin triangles that could cause large depth errors along their longest side). More information on Delaunay triangulation can be found in Appendix C.

Non-maximal suppression sets pixels to zero if they do not have the maximum value within a window $(3 \times 3$, in this case). It forces the selection of points with the largest local $T(x, y)$. Such local maxima are likely to have the most reliable depth values. An example mesh is shown in Figure 3.5 (b). It was generated from images captured by five cameras, such as the image in Figure 3.5 (a). This novel technique generates an approximate model of an OI at several frames per second, fast enough to use in the proposed real-time object-tracking system.

Earlier, it was noted that not all the spikes and pits in the DM could be eliminated with a median filter. Some large errors in depth may remain. These must not be included in the final model, or tracking accuracy will be degraded, or worse, tracking could fail. As a final guard against including such extraneous points, any points that are too far from the target object’s origin are discarded. A maximum bounding radius is defined outside which none of
the vertices may exist. Discarding such points ensures that the model has no large outliers capable of significantly affecting the object tracker’s motion calculations.

### 3.3 Extracting Texture Maps

A tessellated mesh is only half the object model: it defines the 3D surface geometry, but not its surface appearance. The target-object’s surface features, such as coloured markings, are just as important as the object’s surface geometry. Without both the 3D geometry and the surface features tracking cannot proceed. The surface features are modelled herein as a TM.

TMs are images that are mapped on to a 3D surface. This is an effective and compact method of modelling complex surface features. It is a technique used extensively in both the real-time and photo-realistic computer-graphics industries.

Traditionally, TMs are mapped onto a 3D surface in a manner similar to the way wallpaper is applied to a surface, or the way that a rubber sheet is stretched and wrapped around an object. This poses a problem when extracting this texture information from input images. First, the mapping of the TM around the mesh must be established. Next, the input image
must be warped in order to map onto the TM. Furthermore, the resolution of the TM must be established beforehand based on the detail in the surface markings. The resolution should be high enough to capture all the details in the available image, but not too high so as to waste valuable memory. Selecting the appropriate TM orientation, mapping and resolution would be both very difficult and computationally expensive.

Instead of attempting to solve the above problem, a simpler technique can be employed: projective texture mapping [67]. Projective texture mapping effectively projects a texture onto a 3D surface in the same manner that a real slide/film projector projects an image onto a screen, Figure 3.6. The easiest way to visualize projective texture mapping is to imagine that a projector is placed relative to the OI, at the exact same location as the camera that captured the texture (image) being projected. Each projector projects the image captured by the camera back onto the object’s surface. As the OI moves and rotates, the projectors move and rotate with it. Thus, the projected features remain static on the OI model’s surface. This concept can be extended to multiple projectors from multiple viewpoints.

Figure 3.6: Projective texture mapping on a plane.

The advantage of using projective texture mapping is that the input images can be used as the TM directly, requiring no calculations at all. All the issues associated with extracting regular TMs are avoided. If memory is scarce, the image can be cropped to the region covered by the OI. The texture-extraction procedure is as follows:

- Copy the image from the reference camera into the object’s TM; and,
• Record the reference camera’s parameters and pose relative to the OI’s reference frame.

The recorded camera parameters and relative pose are the texture projector’s internal parameters and pose.

Using the TM and projector parameters obtained above, the object tracker can project the OI’s model onto the image plane including its surface texture. The texture is projected onto the model surface from the viewpoint of the camera that the input image was captured from, i.e., from the texture projector. A texture-projector’s pose is fixed relative to the OI’s reference frame.

3.4 Model Updating

The modeller can only build a model of what it can “see.” This results in a model covering only one (visible) side of the OI. As the OI moves and rotates over time, parts of it that were modelled earlier would “disappear” from view and new parts become visible. If the OI rotates far enough, the modelled side of the object could completely disappear and tracking would fail. Hence, the model needs to be updated in a timely manner, before the original model rotates away from view.

There are two possible schemes for updating the meshed model: periodic (complete) replacement and gradual improvement. In periodic replacement, the entire model is discarded and rebuilt every \( n \) frames. Gradual model improvement, on the other hand, entails improving and augmenting the model over time. As time progresses, new data is added to the model as it becomes available. The merits and challenges of both methods are discussed below:

**Periodic Replacement:** Of the two model-updating schemes, this is the simplest. Every \( n \) frames, the model is discarded and a new one built using exactly the same procedure as for the initial model. The primary advantages are speed and simplicity. A disadvantage is that it discards information that may still be of use. In doing so, tracking accuracy could decrease.
There is also no guarantee that the model will not drift relative to its own reference frame. Every time a new model is built, it uses an estimated pose, not the actual pose, of the original reference frame. Thus, at every update, a static offset is added to the pose of the tracked reference frame. As a result, the pose of the reference frame being tracked can drift relative to the actual OI.

**Gradual Improvement:** This scheme adds to and improves a model over time. In doing so it collects data from multiple frames and incorporates it into the OI model. Potentially, this method could provide a more accurate and complete model than would periodic replacement. Tracking accuracy could, therefore, be improved. As old model data is retained, tracking could be more robust to fast and erratic rotations. If the OI rotates, such that all sides have been viewed by the cameras, a complete model could be built. Such a complete model could be saved for future use and the modeller switched off. Additionally, model drift relative to the target’s reference frame could be eliminated as the new data must always be aligned and merged with the existing model.

A few incremental modelers exist in the literature. Cadman and Tjahjadi [68] and Moyung and Fieguth [69] both present incremental modelers for feature-point-based models. However, the tracker proposed herein requires solid texture-mapped surfaces. Also, both techniques assume that only one object is present, thus, precluding the possibility of tracking an object in a cluttered environment. Simard and Ferrie [16] developed an algorithm that updates a tessellated model over time. Once again, it assumes that there is only one object and is geared toward modeling terrain and not smaller, mobile objects.

Let us now examine the increase in complexity required for incremental modelling. Not only must new points be found in the input images for addition to the model, these must be compared to the current model. After comparison, a decision must be made on whether to replace, modify, merge with, or add to, existing data. Another issue is closing the 3D model to form a solid model. This must occur once the model has rotated far enough in front of the cameras. Coupled with this are computational restrictions: model updating must occur on-line
whilst the object tracker is operating. This complexity can be better understood by examining a possible incremental-modelling algorithm:

- Extract a DM,
- Select potential vertices,
- Compare current vertices in the model with the DM, and:
  - Remove parts of the model that do not match the input data, and
  - Fine-tune vertex locations if necessary.
- Compare each vertex with the current model and decide to either:
  - Insert the new vertex,
  - Move an existing vertex to a better location,
  - Add the new vertex (if it is outside the model’s current surface), or
  - Discard.
- Update the TMs, and
- Add new TMs if necessary.

There exist several difficult sub-problems in the algorithm outlined above. Each step requires analyzing input data and making a complex set of decisions. For example, updating the TMs requires methods for selecting TM locations, texture resolution, and, updating existing TMs with new data. One may note that, the first two steps in the algorithm are the same as for periodic replacement. Hence, periodic replacement can be used as a starting point. Improving and augmenting the target object’s model over time is a large scale research project with the potential to provide improved object tracking and system reliability. It is, however, beyond the scope of this Thesis.

In this Thesis, the periodic replacement scheme has been used. It is simple, fast, and allows for rapid evaluation of the feasibility of building a model on-line for object tracking.
Gradual model improvement requires considerably more effort, both in terms of computational power, and in terms of research and development. Experimental evidence suggests that periodic replacement is sufficient for slow-maneuvering object tracking.

3.5 Segmenting the Object of Interest from the Background

The modelling procedure detailed can effectively build a surface model of the entire scene. What is required, however, is a model of only the OI. This requires some form of image segmentation. Although many image segmentation algorithms exist (e.g., [70]-[72]), most are not suitable for OI segmentation, as such algorithms mostly divide the image up into regions of similar colour. Objects usually have multi-coloured surfaces, resulting in over segmentation with colour-based algorithms. Additionally, most are difficult to implement in real-time, precluding such algorithms for use in our object-tracking system.

Rather than trying to develop a system to extract any object from any background in real-time, the proposed method uses segmentation information already obtained in a separate part of the modelling system. In Section 2.2, an OI selection methodology was presented that separates and selects an OI. Segmentation is part of this selection process.

Recalling the Interest Filters (IFs) presented in Section 2.2.1, one may note that a bank of IFs highlights regions of input images that are of interest; namely, regions that are likely to belong to an OI. Regions not marked of as interest, most likely do not belong to the OI. This can be combined with other information in order to separate the OI from the background.

In order to achieve segmentation using an IM, the IM is first thresholded. Subsequently, the modeler only adds vertices (principal-points) for pixels marked as of interest. Thus, the criteria for adding principal points becomes: $T(x, y) > \tau_T$ and $S(x, y) > \tau_S$, where $S(x, y)$ is the interest value for pixel $(x, y)$, and $\tau_S$ is the interest threshold.

This segmentation method is illustrated in Figure 3.7. An IF bank operates on an image
from the centre camera of five, Figure 3.7 (a), resulting in the IM in Figure 3.7 (b). This IM clearly highlights only parts of the OI, which is the Olimex box in the foreground. As a result, the model built using the IM for segmentation includes only the OI, Figure 3.7 (c).

There are two different situations in which segmentation must occur: when building the initial model, and when rebuilding the model. Whilst the modelling procedure remains the same, the input information is slightly different. For example, when rebuilding the model, one already has the object’s pose and approximate size (based on the previous model). These attributes aid the segmentation process. Each of the two situations are addressed separately below.
3.5.1 Initial Object Segmentation

When building the initial object model, segmentation is largely handled by the OI selector. The OI selector outputs a region and an IM. The region is built using a motion segmentor. Thus, segmentation is performed by using the IM as detailed above, and by restricting vertices to occur within the OI region. In this way, the resulting OI model includes just the OI region. The IM as well as the OI region are used in order to provide finer segmentation. The OI region may be rough, and include parts of other objects too. The IM could eliminate these small sub-regions containing other objects.

The above procedure can be seen in the experiments conducted for the OI Selector (Section 2.2.3) in Figure 2.11. This has been included here as Figure 3.8 for clarity. Together with the IM in Figure 3.8 (b), the segmentation seen in Figure 3.8 (a) restricts the model to the OI, Figure 3.8 (c).

Figure 3.8: The motion-segmentation map (a), IM (b), and resulting model (c) for a real-world motion sequence.
3.5.2 Segmentation for Model Rebuilding

Segmentation when rebuilding the object model is more complex than building the initial model. By this stage, the OI selector has been removed and no OI region is provided. The IF bank used by the OI selector may be insufficient to isolate the OI; it may, after all, highlight parts of multiple objects. Some extra data is required for successful segmentation.

The object tracker and previous OI model provide us with a certain amount of additional information. The object tracker provides the OI’s estimated pose in the current time-step, and also an estimate of the object’s velocity (both linear and angular). The previous OI model can be used to calculate a bounding rectangle containing the OI. This is achieved by projecting the model’s bounding sphere onto the image plane at the OI’s current estimated pose and calculating the bounding rectangle that contains this sphere.

The bounding rectangle is a segmented region containing the OI, and possibly, parts of other objects. Using the IM as outlined above results in further segmentation. When combined, these two techniques can isolate the OI, enabling the OI model to be rebuilt in the presence of background clutter.

An additional possibility is to change the IF filter bank after OI selection. What helped select an OI may not be optimal for rebuilding the OI. For example, an IF could be added that highlights regions that are moving under the same transformation as the OI. This would take the OI velocity as estimated by the object tracker, and compare multiple input frames, highlighting regions that moved under the same transformation. This would filter out all objects moving in different directions.

The techniques described above use existing information and known properties of the OI in order to separate it from the background. This enables real-time, or near-real-time operation. As a result, the modeller can segment and build a model of an OI on-line.
3.6 Experiments

As with the OI selector, the modeller’s performance was analysed in terms tracking objects using the model. The results were also compared to tracking an object with an a priori known 3D model (i.e., without the modeler being in the loop).

3.6.1 The Implemented Depth-Map Extractor

In order to be able to test the operation of the modeller, a DME algorithm had to be implemented. It was decided to use an algorithm similar to that presented by Yang et al. [73] since it uses 3D graphics hardware for speed, and is one of the few real-time DME algorithms in the literature. Yang et al.’s algorithm [73] projects the images from two or more cameras out into 3D space and onto a series of planes for comparison. These planes are parallel to the image plane of the reference camera (i.e., the camera that this DM is referenced to). The difference between [73] and the implemented algorithm lies in that the candidate DM in our case is non-planar and is iteratively improved rather than using Yang et al.’s plane-sweeping algorithm. A smoothness constraint has also been added in order to reduce errors. The proposed DME algorithm provides a more accurate DM whilst still operating in real-time (at 100 fps when processing 256×256 images from five cameras using a Radeon X800 graphics card).

Other DME algorithms could also be used. For example, since the development of the proposed modeller, Yang et al. have introduced a more advanced real-time modeller [74]. While using this newer algorithm could potentially provide a better DM, it was decided that the additional implementation time would not yield sufficient improvement to warrant this. Moreover, the existing implementation is adequate to confirm the overall proposed methodology’s effectiveness.
3.6.2 Experimental Set-up

The experimental set-up was similar to that used for testing the OI selection system (Chapter 2). Five cameras with a horizontal field-of-view of 34° and a resolution of $(512\times256)$ pixels each were used for capturing motion sequences. Synthetic sequences were generated using the same camera calibration data as for the real sequences. Five cameras provide more accurate DMs with the DME detailed above. Results with fewer cameras were found to be too noisy.

Synthetic motion sequences were generated using OpenGL and included a single-point light source placed above and to the right of the scene. Optionally, an image could be inserted as background in order to simulate a cluttered background. Real motion sequences were captured with a Sony SNC-RZ30N camera. The OI was placed on a precision $x$-$y$ stage. As only one camera was available for the experiments, it was placed on a precision linear stage and moved to mimic five static cameras. Each (simulated) camera’s view was recorded separately. The high precision of the motion stages was relied upon to produce exactly the same motion for each different view.

The OI selector was not used for these experiments. Instead, the modeller was provided with the known initial pose. There are two reasons for this: it prevents the OI selector from influencing the results, and it enables direct comparison of the tracked reference frame to the actual OI reference frame. The same two IFs used in the OI selector tests were used here once again. They are the colour IF, and background-subtraction based IF, Section 2.2.3.

3.6.3 Results

The modeller was tested on several synthetic motion sequences with various compositions, such as: a sequence of a cube with a blank background, Figure 3.9; a sequence of the same cube with a cluttered background, Figure 3.10; and, a sequence of a globe moving in front of a cluttered background, Figure 3.11. Each sequence was 240 frames long corresponding to 8 seconds of video at 30 fps.
The use of a blank background tests the on-line modeling ability without the issue of object segmentation. It also provides a reference against which the proposed object-segmentation methodology’s effectiveness can be compared. The sequences with cluttered backgrounds test the proposed system’s ability to separate an object from static background clutter. Finally, the globe model provides a test of the system’s ability to track objects that are more complex than a cube.

In all of the three motion sequences, the OI moves along the path shown in Figure 3.12 at
67 mm/s and rotates about the $y$-axis (vertical axis) at 0.18 rad/s. The sharp change in trajectory tests the system’s ability to track a maneuvering target. The OI started at 1000 mm from the camera, with the camera directed parallel to the $z$-axis (i.e., facing up the page in Figure 3.12).

![Figure 3.12: The path taken by the OI in synthetic sequences.](image)

Positional and orientational tracking errors for Figure 3.9’s motion sequence are shown in Figure 3.13 (a) and (b), respectively. The first 5 frames of the sequences were used to initialize the background-subtracter IF. In all motion sequences, the $y$-axis error remained small. This is to be expected as the OI remained predominantly along the $x$-axis, thus, minimizing the cross-coupling between the $z$- and $y$-axis errors. Errors along the $z$-axis (i.e., the optical-axis) are largest as motion/position along the optical axis is harder to extract than for the other axes.

For the globe sequence, the obtained tracker’s accuracy is similar to that for the previous two motion sequences for the simple cube, Figure 3.14.

In order to obtain some measure of the modeller’s performance for object tracking, the above motion sequences were also tested using just the object tracker minus the modeling system. In these tests, the tracker was given the exact model of the OI that was used to generate the synthetic motion sequences. The errors obtained were approximately 5 mm positionally,
Figure 3.13: The positional (a) and orientational (b) tracking errors for the sequence in Figure 3.9.
Figure 3.14: The positional (a) and orientational (b) tracking errors for the sequence in Figure 3.11.
and the orientation errors were \(0.5^\circ\) for the cube sequences and \(0.9^\circ\) for the globe sequence, respectively. As expected, an exact model results in more precise tracking (approximately \(10\times\) in this case). Thus, one can conclude that improving real-time modelling would improve tracking accuracy. However, one must note that, in this Thesis, the models were built on-line, and tracking was successfully achieved with these approximate models; namely, it has been shown that, an \textit{a priori} unknown object can be successfully modelled and tracked in real-time.

The final test was on real-world data. A small rectangular box was placed on an \(x\)-\(y\) table. This mimics an industrial scenario where an object is placed on a conveyor or some other device. In such a scenario, a robot arm might need to intercept the OI. Hence, the OI must be tracked. In contrast to the simulations above, one now has two moving objects, the OI and the conveyor. The conveyor will most likely be moving at the same speed as the OI, preventing the use of motion-based segmentation to extract the OI. The OI’s colour is different from that of the conveyor (or \(x\)-\(y\) table in this case). Hence, filtering based on colour was used in order to separate the OI from the \(x\)-\(y\) table. Filtering out background objects was still achieved using the background-subtraction IF. This is an example of how additional filters can be added and/or customized for a particular application.

Space constraints on the physical set-up required modifying the path taken by the OI to that shown in Figure 3.15. Its velocity along Section 1 remained the same as for the synthetic sequence. The velocity along Section 2 of the path was 60 mm/s. Due to equipment limitations, the OI did not rotate. Its initial distance from the camera was 900 mm.

The model obtained for the real-world box is shown in Figure 3.16. This figure shows several frames of the tracker’s projection overlaid over the input image. Input images are darkened so that the OI model that was built by the modeller can be seen clearly. The models in Figures 3.16 (a) to (c) are slightly different since the model was rebuilt once every 30 frames, as per the proposed model-updating scheme.

Whilst the extracted model does not cover the entire visible portion of the OI, it encom-
passes enough for the object tracker to reliably track the OI. Partial models are perfectly adequate providing they include sufficient surface features to enable tracking. The resulting tracking errors are shown in Figure 3.17. These errors cannot be directly compared to those for the synthetic image sequences above since the object’s size is different and the distance from the camera has also changed. What can be noticed is a slight increase in errors due to camera-calibration inaccuracies. Despite this, the OI was tracked to within approximately 30 mm positionally and 14° orientationally.

**Modeller Speed**

One of the major requirements of the modeller is that it run on-line in real-time or near real-time. The model is not useful if the OI has moved so far from its initial pose that tracking fails. Therefore, it is important to examine the modeller’s speed.

In the experiments, the modeler built models containing 126 triangles at a rate of about 13 fps. The object tracker, on the other hand, was capable of tracking the OI using those models at rates of 45 to 70 fps. Since, most cameras operate at 30 fps (for NTSC, 25 fps for PAL), one can note that the OI’s model can be updated once per second (i.e., every 30 frames) and still successfully track the OI.
Figure 3.16: The tracked model overlaid over darkened frames #5 (a), #125 (b), and #240 (c) of a real-world motion sequence.
Figure 3.17: The positional (a) and orientational (b) tracking errors for the sequence in Figure 3.16.
The above frame rates were obtained using a 2.0 GHz Athlon 64 CPU and a Radeon X800 pro graphics-card. Since performing these experiments, computer and graphics hardware has progressed further significantly. It can safely be assumed that the modeller’s (and object tracker’s) performance will improve with newer hardware.

### 3.7 Discussion and Recommendations

The experiments above demonstrate the feasibility of using on-line modelling for real-time 6-dof object tracking of *a priori* unknown objects. The currently implemented modeller can construct a model rapidly with sufficient accuracy to facilitate tracking. However, tracking accuracy was measured to be about ten times worse than for an accurate model. Therefore, there is room for improvement.

A weakness in the current implementation is the DME algorithm. It provides DMs in real-time, but, results include errors and there are objects, such as those with strong specular highlights, that it has difficulty with. This is a known problem with most DME algorithms. Ideally, the DME should, using only two cameras, provide DMs that are accurate enough that the noise-reducing median filter becomes obsolete. Improving the DM extractor could greatly improve the quality of the resulting models and, thus, improve the accuracy of the overall object-tracking system.

Another potential limitation is the use of the Delaunay triangulation. This implicitly assumes a convex 2D border. Currently, the algorithm can cope with irregular shaped objects provided that those irregularities lie within the 2D object boundary (i.e., within the silhouette). For example, a concave object could have a deep depression in its surface and be modelled correctly until the object is rotated such that this depression causes the object’s silhouette to become non-convex. At this point, the model would include small portions of the background in this concave region. The object tracker may or may not be able to continue tracking under such conditions. Extracting an object’s silhouette and removing all triangles that lie outside the
silhouette could potentially solve this problem.

The experiments above were performed using two different types of IFs. Using a wider range of IFs could create an IF framework that is useful for a wider range of applications. For example, an IF could use velocity information obtained from the tracker to aid in separating the OI from clutter. Such an IF would provide more generic segmentation when the model is being rebuilt.

The final recommendation is to develop a real-time modeller capable of progressively updating and refining the model. Reasons for this have been discussed previously. Essentially, this would provide a system that allows the model to become more accurate as time progresses. The more accurate the model, the more accurate the tracking would be.

### 3.8 Summary

The proposed modeller facilitates tracking of \textit{a priori} unknown objects. It builds a model online which the object tracker uses to track the object. This overcomes a major limitation of model-based trackers, namely, the need for a model of the target being tracked to be given beforehand. Experiments in which a model was built and, then, used by a 6-dof object tracker confirmed the viability of this methodology.

Rather than trying to build a precise (or optimal) model, accuracy is traded for speed. The modeler rapidly builds an approximate model represented as a tessellated (meshed) surface and a TM. Projective texture mapping provides the OI’s surface details that are used by the object tracker with almost no computation. The OI is separated from background clutter using the IF framework presented in Section 2.2.1. This framework provides real-time segmentation by using known properties of the OI (including application-specific knowledge).

Models obtained from the currently implemented modeller result in object-tracking accuracies about ten times worse than achieved with an exact model. This suggests that real-time
modelling is a field requiring further research. Future work should focus on improving DME, and on the development for a modeller that progressively improves the model over time. However, the viability of on-line modelling for real-time 6-dof pose tracking of \textit{a priori} unknown objects has been demonstrated.
Chapter 4

3D Visual-Model-Based Object Tracking

4.1 The Core Object Tracker

The proposed core object-tracker algorithm (without any of the extensions described later in this Chapter) performs the following sequential tasks:

- Predict the pose of the OI at the current time-step,
- Project the OI’s visual model onto the image plane(s) at its current predicted pose using OpenGL (Equation (2.1)) to produce a virtual image,
- Estimate the motion between the virtual projected image and the actual input image (i.e., the motion between the predicted pose and the OI’s actual pose) by solving the 6-dof optical flow (Equation (4.5) below) via a least-squares method, and
- Estimate the OI’s pose using the predicted pose and the estimated motion (Equation (4.7) below).

The above procedure is repeated for every time-step, providing a continuous stream of pose estimates. Optical flow is used to estimate the motion between the predicted and actual poses, not the actual motion of the object. Additionally, OpenGL produces a mask and z-buffer that are
used to separate the OI from the background and provide depth information, respectively. The depth information in the z-buffer allows 6-dof motion to be calculated using a single camera.

A motion predictor is used to minimise the difference between the projected and actual OI. The prediction is based on the estimated poses of the OI in previous time-steps. This minimises the possibility of losing track of the OI whilst maximising accuracy. Optical flow is a linearization procedure, thus, minimising the initial error minimises errors introduced due to non-linearities [75].

The 2D position and appearance of the projected OI model is determined by the relative pose between the OI and camera. Hence, for situations in which the camera is mounted on an Autonomous Vehicle (AV) (i.e., it is not static), the motion predictor predicts the relative velocity between the camera and the OI (the reference frame is attached to the AV). If desired, absolute poses could be used but, then, two motion predictors would be required; one for the AV, and one for the OI. Inertial sensors and the AV’s control output (desired motion) could also be used as inputs to the motion predictor for increased accuracy. For simplicity, in this Thesis, it will be assumed that only one motion predictor is used for the relative pose, and no other external sensors are involved.

### 4.1.1 6-dof Optical Flow

Motion estimation (or predicted pose-error estimation) is performed using gradient-based optical flow. This has its basis in the Lucas and Kanade algorithm [76]. Their algorithm assumes brightness constancy [6]. Namely, it is assumed that any change in pixel intensity is due to motion alone (i.e., the total brightness remains constant). Whilst this may not strictly be true, it is nevertheless a useful approximation to make. The optical-flow equation is

\[
\begin{bmatrix}
\nabla_x I(x, y, t) & \nabla_y I(x, y, t) & \nabla_t I(x, y, t)
\end{bmatrix}
\begin{bmatrix}
\frac{u_x}{v_x} \\
\frac{u_y}{v_y} \\
1
\end{bmatrix} = 0,
\]

(4.1)
where $\nabla_x$ and $\nabla_y$ are the $x$ and $y$ derivatives respectively and $\nabla_t$ is the time derivative. $I(x, y, t)$ is the intensity of Pixel $[x \ y]^T$ at time $t$. Solving (4.1) for a set of pixels yields the mean 2D velocity for that set of pixels $[v_x' \ v_y']^T$.

Equation (4.1) above provides a relationship between image data for a set of pixels and 2D motion. An OI can move freely in 3D space with 6 dof. Thus, the 2D on-screen velocity in (4.1) must be related to 6-dof motion. This relationship can be obtained by differentiating the projection equation (Equation (2.1)) with respect to time, yielding:

$$v' = \frac{d p'}{d t} = \frac{1}{p_z'} M_{\text{int}} M_{\text{ext}} (v + \omega \times p), \quad (4.2)$$

where $v'$ is the projected 2D velocity corresponding to the linear 3D velocity $v$ and the angular velocity $\omega$. Substituting Equation (4.2) into Equation (4.1) yields

$$\begin{bmatrix} \nabla_x I & \nabla_y I & \nabla_t I \end{bmatrix} \left( \frac{M_{\text{int}} M_{\text{ext}} (v + \omega \times p)}{p_z'} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) = 0. \quad (4.3)$$

Generally, Equation (4.3) cannot be solved directly since $p_z'$ – the $z$-axis distance to the object at each pixel – is not known. In our case, however, OpenGL places per-pixel depth information ($p_z'$) in a z-buffer as part of its visibility mechanism. When projecting 3D objects to render a 2D image, it must be determined which surfaces are visible and which are occluded. The z-buffer maintains the z-distance to the object drawn at each pixel, thus, allowing new surfaces to be discarded if they are occluded by surfaces that have already been drawn. As a result, Equation (4.3) can be solved directly by extracting $p_z'$ from the z-buffer. This novel combination of model projection, the z-buffer, and optical flow facilitates full 3D 6-dof pose tracking, even if only one camera is available. For multiple cameras, each camera has its own $M_{\text{int}}$ and $M_{\text{ext}}$ matrices that map 3D coordinates from a common global reference frame to their respective image planes.
Angular Velocity About the OI’s Own Origin

A potential problem with the optical-flow equation (Equation (4.3)) is that the angular velocity is calculated about the centre of the world reference frame. Preferably, however, the angular velocity should be about the OI’s own origin. This can be achieved via one more transformation:

$$ p_{rel} = p - p_{obj} \quad (4.4) $$

where $p_{obj}$ is the OI’s origin with respect to the world reference frame. Replacing $p$ with $p_{rel}$ in (4.3) yields

$$ \begin{bmatrix} \nabla_x I & \nabla_y I & \nabla_t I \end{bmatrix} \begin{bmatrix} M_{int} M_{ext} (v + \omega \times p_{rel}) \frac{p_z'}{p_z} \\ 0 \\ 0 \\ 1 \end{bmatrix} = 0. \quad (4.5) $$

Solving (4.5) yields the linear velocity of the OI’s origin and its angular velocity about its own origin, expressed in the world reference frame.

4.1.2 Projecting the Model

The OI’s model and pose as well as the camera projection matrix are all passed to OpenGL for rendering. In this way, a virtual image of the OI at its predicted pose is rendered. This is overlaid on top of the input image to minimise the difference between the two images. Additionally, pixels belonging to the OI are marked in the stencil buffer. This results in three images: a virtual image (Figure 4.1 (a)), a DM contained in the $z$-buffer (Figure 4.1 (b)), and an OI mask in the stencil buffer (Figure 4.1 (c)). Each image provides different information: the virtual image provides the projected OI’s visual appearance, the mask is used to isolate the OI from background clutter, and the $z$-buffer allows the optical-flow equation to be solved for full 6-dof motion. In essence, the mask performs image segmentation without requiring additional processing.
Figure 4.1: A virtual-image (a), depth information (b) in the z-buffer, and (c) a mask in the stencil buffer.

**Z-Buffer to Depth-Map Transformation**

In OpenGL, the z-buffer is used to perform occlusion. It ensures that objects are correctly occluded, regardless the order in which they are drawn. In order to maximize accuracy, the z-buffer is non-linear and scaled to fit between near- and far-clipping planes [77]. Better accuracy is available for objects closer to the camera. For conversion from z-buffer format to physical coordinates, the following formula is used

\[
z_c = \frac{-z_{far} \cdot z_{near}}{z_{far} - z_{buff} \cdot (z_{far} - z_{near})},
\]

where \(z_c\) is the depth in camera coordinates, \(z_{near}\) and \(z_{far}\) are the distances to the near- and far-clipping planes, respectively, and \(z_{buff}\) is the z-buffer’s value. One can note that the z-buffer data must be in floating-point format and be scaled such that \(z_{buff} \in [0, 1]\).

The near- and far-clipping planes specify the valid range for \(z_c\) (Figure 4.2). If part of an object is closer to the camera than \(z_{near}\) or farther away than \(z_{far}\), then, its \(z_c\) will be clipped in the z-buffer and image corruption may occur. As a consequence, \(z_{near}\) and \(z_{far}\) must be chosen to encapsulate the entire working area of the OI. However, placing the clipping planes too far apart could hinder accuracy. Accuracy would be particularly poor if \(z_{near}\) is too close to the camera, as then, the bulk of the z-buffer range would represent a small area in front of the camera, leaving no numerical precision for the rest of the scene.
4.1.3 Estimating Motion

The OI’s pose is updated by using optical flow to calculate the motion between the estimated pose (in the virtual image) and the actual pose (in the real image). Linear and angular velocities for the OI are calculated using (4.5), the 6-dof equation for optical flow. The value of $p_z'$ for each pixel is extracted from the $z$-buffer. The $z$-buffer is the key component that allows direct 6-dof motion estimation. Motion is estimated using a least squares method for only those pixels marked by the OI mask (stencil buffer) as belonging to the OI, thus, effectively ignoring background clutter. Note that, the calculated velocities are for the motion between the estimated pose and the actual pose, not the actual motion of the OI; that is, the motion is in fact the error between the OI’s predicted and actual poses. The OI’s actual pose is estimated by moving the predicted pose by the estimated motion for one time-step:

$$P_{est} = \begin{bmatrix} R(\omega) & v \\ 0^T & 1 \end{bmatrix} P_{pred},$$

Equation (4.5) defines a motion constraint based on the data from a single pixel. A group of pixels belonging to a single object corresponds to a system of linear equations (i.e., a system of motion constraints). This can be solved using any common technique for solving a system
of linear equations. The proposed technique uses least-squares as it is fast, and thus suitable for real-time operation. Other techniques such as robust estimation [78] or total least squares [79] could also be used, if necessary. These may yield more accurate results, though, at the cost of greater computation time. One can note that, while (4.5) is defined as the motion between two frames in a sequence, in this case, it is the motion between the virtual image, \( I_v \), and the input image, \( I_i \).

The complete motion-estimation procedure is as follows:

- Blur both the virtual image, \( I_v \), and the input image \( I_i \), with a Gaussian filter,
- Calculate the spatial derivative (gradient) of \( I_v \), (i.e., \( \nabla I_v \)),
- Calculate the “time” derivative: \( I_i - I_v \),
- Assemble the constraints from the derivative images, and
- Solve (4.5) for the motion.

Blurring ensures that the image is locally linear. The size of the blur is related to the maximum motions that can be tolerated. For example, a Gaussian blur sigma value of 1.8 (as used in our experiments) would result in reliable motion estimation of up to 1.22 pixels in any direction.

The final aspect of the algorithm to discuss is thresholds. Not every pixel can be used for optical flow. Also, some pixels may give erroneous results that could result in large outliers, to which least squares is sensitive. In order to eliminate these, two thresholds are introduced herein. The first ensures that a minimum gradient exists for a pixel, namely:

\[
|\nabla I_v| > \tau. \tag{4.8}
\]

A pixel with no gradient, contains no motion information; a pixel with a miniscule gradient is likely to be dominated by the effects of noise. Using (4.8) eliminates both of these.

The other threshold relates to rejecting erroneously large motions. Since the value of \( \sigma \) for the Gaussian blurring filter controls the maximum motion that can reliably be estimated,
it makes sense to discard all motion constraints whose estimated motions are too large. This corresponds to:

\[
\frac{|\nabla_t I_v|^2}{|\nabla_x I_v|^2 + |\nabla_y I_v|^2} < \tau_{\text{vmax}}^2,
\]

where \(\tau_{\text{vmax}}\) is the maximum motion that can be estimated reliably given the Gaussian filter’s value for \(\sigma\). Within the current implemented algorithm, \(\tau_{\text{vmax}}\) is equal to 1.22.

4.2 Improving Robustness Through Extensions

The basic tracking algorithm above has some limitations: diffuse lighting that minimises the existence of shadows is required; also, the disparity between the predicted and actual OI poses must be kept to within a few pixels on-screen (i.e., sudden accelerations may not be tolerated). In the real-world, both directional lighting and larger motions occur. Therefore, techniques have been developed in order to address these limitations.

4.2.1 Local Illumination Normalization

The visual appearance of objects depends on both its surface features, and the lighting conditions under which it is viewed; that is, shading is illumination and location dependent. The OI’s model contains no shading information. Performing optical flow between an unshaded projection and a real (shaded) image violates the brightness-constancy assumption. In this case, some of the brightness changes are due to comparing the same OI under different lighting conditions. This disparity could cause the tracker to fail.

One possible solution to the above shortcoming would be to perform shading during the projection of the OI’s model. Equations for simulating illumination are well known and existing real-time 3D graphics applications can perform this operation with increasing realism. However, 3D graphics applications use 3D scene models which include lighting; whereas, in the case of object tracking, the lighting is unknown. It would not be feasible to extract the
required illumination information fast enough for use in real-time tracking. Hence, some other technique of minimising the effects of illumination would be required.

Little has been reported in the literature that tackles the illumination-variation problem. Matsushita et al. [80] have developed a technique that uses eigenspaces and visual hulls. Unfortunately, it expects the bulk of the scene to be static and does not eliminate shadows of moving objects. Instead, herein, a Local Illumination Normalization Filter (LINF) has been developed. This filter operates under the assumption that shadows are generally low-frequency components of an image. Thus, the LINF normalizes the image such that the mean colour of a local region (defined by the cut-off frequency of a low-pass filter) is mid-gray:

$$I'(x, y) = I(x, y) \frac{0.5}{(G \ast I)(x, y)}.$$  \hspace{1cm} (4.10)

where \((G \ast I)(x, y)\) is the low-pass version of the input image \(I(x, y)\) obtained by convolving \(I(x, y)\) with a Gaussian filter kernel. A problem with (4.10) that is common to all image-processing operations involving division is that; low-light regions can amplify noise, and/or cause division-by-zero errors. In order to cope with this, the ideal LINF (4.10) is modified with a noise reduction constant, \(\epsilon_n\), resulting in:

$$I'(x, y) = \frac{0.5(I(x, y) + \epsilon_n)}{(G \ast I)(x, y) + \epsilon_n}.$$  \hspace{1cm} (4.11)

The above prevents the denominator from becoming too small, which would cause image noise to be amplified greatly. The \(\epsilon_n\) term in the numerator ensures that the local mean output colour remains mid-gray, even if \(I(x, y)\) is zero; without this term in the numerator, \(\epsilon_n\) would pull the local mean output colour down to zero as the local intensity level drops. The optimal value of \(\epsilon_n\) is related to the noise level in the image; the higher the noise level, the larger \(\epsilon_n\) should be.

Equation (4.11) eliminates the bulk of all shading effects whilst preserving the higher frequency components used by the optical flow algorithm. Figure 4.3 shows the effect of this filter on a photograph of Convocation Hall Building at the University of Toronto. The high-
frequency components of the image are preserved whilst low-frequency shading is removed. Other low-frequency detail is also lost in the process, but this is acceptable.

![Images of Convocation Hall Building](a) (b)

Figure 4.3: A photo of Convocation Hall Building at the University of Toronto shown in its original form (a), and after it has been filtered by a LINF (b).

The LINF is a preprocessing step for our object-tracking algorithm. By normalizing both the input and projected images, the difference between them due to shading is minimised, thus, resulting in improved robustness to shading. One can note that this does not provide complete removal of shading. Some shadow edges are still visible in Figure 4.3(b). Rather, it reduces shading effects fast enough to be used for real-time object tracking. Experimental results showing the effectiveness of LINF for object tracking are presented in Section 4.4.2 below.

### 4.2.2 Multi-Scale Optical Flow

Gradient-based optical flow assumes local linearity of the input images. Local linearity is enforced by blurring input images with a Gaussian filter. The larger the blur, the larger the local region of linearity is, the larger the motions that can be tolerated. As a trade-off, greater blurring results in slower and less accurate tracking. In standard optical-flow algorithms that estimate the motion field, multi-scale techniques may be used to overcome this limitation (e.g., [75], [81], and [82]).
Multi-scale optical-flow algorithms, typically, begin by forming a pyramid of down-sampled images. Optical flow is, then, performed on each level of the pyramid, starting with the lowest-resolution image. The resulting motion field from each level is used to warp the first input-image at the next resolution level. The warping corresponds to the estimated motion field. This process continues until the highest resolution level has been processed. Summing the motion fields from each level in the pyramid would yield the overall motion.

Instead of warping the images, the OI’s estimated pose is updated starting with the smallest resolution for coarse motion. Next, the object is reprojected at the next resolution up in the pyramid and the motion is recalculated. In each step, the pose estimate is updated by the calculated motion (or offset). The motion updates become more accurate each time the resolution is increased. In this manner, large motions are handled efficiently at coarse resolutions, without losing the accuracy obtained from motion calculations at the original resolution.

Minor changes are required to the optical-flow equation (Equation (4.3)) in order to incorporate multi-scaling. Most notably, the intrinsic camera parameters (\(M_{int}\)) are different for down-sampled images. However, rather than change the intrinsic parameters, it is easier to note that the distance between pixel centres in down-sampled images differ from the original image by a scale factor. In this way, Equation (4.3) becomes:

\[
\begin{bmatrix}
\frac{\nabla_x I}{s} & \frac{\nabla_y I}{s} & \nabla_t I
\end{bmatrix}
\frac{1}{p_z'} M_{int} M_{ext} (v + \omega \times p) = 0,
\]

where \(s\) is the scale factor between the down-sampled image and the original image.

4.2.3 Occlusion Rejection

A major problem in object tracking is coping with partial occlusions. The tracking algorithm can cope with background clutter, but not with occlusions. The mask generated in OpenGL’s stencil buffer masks off only objects in the background, not those in front of the OI. In order to continue tracking despite occlusions, the filtering out of occluded regions is required.
In [83], Pan and Hu suggest that occlusion is a cyclic problem: the occlusion must be known before the OI can be located, but the occluded part can only be determined after determining the OI location. Whilst this may be true to a certain extent, the motion calculations are referenced to the virtual image (i.e., the predicted pose), and not the actual input image (containing the OI at its actual pose). Hence, one needs to determine which pixels in the virtual image belong to occluding objects in the input image. Knowing which parts of the OI are occluded relative to its actual pose in the input image is not required.

What causes tracking failure can be seen in Figure 4.4. Pixels used by optical flow are marked in green. One can notice how the cube in this Figure is being used in motion calculations, even though it is not part of the OI. At these pixels it is effectively trying to calculate the optical-flow between the OI model (the Olimex box), and the cube. This is physically meaningless and violates the brightness-constancy assumption that is fundamental to optical flow. Brightness constancy assumes that the total brightness remains constant, implying that any local intensity changes are purely due to motion. Thus, one can conclude that most occluded pixels violate brightness constancy.

Unfortunately, there are no simple tests for detecting brightness-constancy violations. Nevertheless, if local brightness variations are, in fact, due to motion instead of occlusion, then, the
image intensity gradients in the virtual and input images ($\nabla I_v$ and $\nabla I_i$, respectively) should be of similar direction and magnitude. If this is not the case, then, brightness constancy has been violated, and it is highly unlikely that the pixels in question belong to the same object. For example, if a pixel is part of a white-to-black edge in one image, but a red-to-black edge in the other, then, most likely, they come from two different objects, and attempting to calculate optical flow between them is physically meaningless.

Using the principle above, a series of tests have been devised to discard occluded pixels. These tests work on colour channels separately. Therefore, the Colour Channel $j$ in Image $I_i$ is denoted as $I_{i,j}$, where $j \in \{R, G, B\}$, the Red, Green, and Blue colour channels, respectively.

The first and simplest test is to check whether there is sufficient gradient in the input image, i.e.,

$$|\nabla I_{i,j}| > \tau_{gm},$$

for all colour channels ($t_{gm}$ is the minimum gradient magnitude). This is the same test performed by the original algorithm on the virtual image, $I_v$. Essentially, if there is no gradient in $I_v$ and $I_i$, then, motion cannot be estimated using that pixel. if, on the other hand, one has a gradient but not the other, then, the pixels in question probably belong to different objects (or different parts of the same object).

If a pixel passes the first test (Equation (4.13), the next step is to examine gradient direction consistency. Denoting the angle difference between the gradients in $I_v$ and $I_i$ with $\langle \nabla I_v, \nabla I_i \rangle$, this test is defined as:

$$\langle \nabla I_v, \nabla I_i \rangle < \tau_\theta, \forall j : \{|\nabla I_{i,j}| > \tau_{gm} \land |\nabla I_{i,j}| > \tau_{gm}\}. \quad (4.14)$$

Equation (4.14) implies that the difference between the gradient directions in the images should be less than the threshold $\tau_\theta$ for all colour channels in which the gradient magnitudes in both images (i.e., both $I_v$ and $I_i$) are above threshold $\tau_{gm}$. $\tau_\theta$ should be large enough to allow OI rotation, but be small enough that the angle can only be measured in colour channels in which there is a gradient; ignoring all other channels prevents random angles, due to noise, from
causing the algorithm to reject valid pixels. This test operates on the basis that the gradients should be facing in roughly the same direction. If they are not, then, either the pixels belong to different objects, or extreme rotation is present, in which case the pixel in question is unsuitable for motion analysis anyway.

The final test checks whether the gradient magnitudes are similar. Gradient-based optical flow assumes local linearity. Hence, the gradient magnitudes should be similar, but not necessarily equal. Thus, the final test is:

$$||\nabla I_v| - |\nabla I_i|| < \tau_{md},$$  \hspace{1cm} (4.15)

where $\tau_{md}$ is the threshold marking the maximum allowable gradient magnitude difference.

If a pixel passes all the above tests (Equations (4.13) to (4.15)), then, it is passed on to the optical-flow solver; otherwise, it is rejected. In this manner, occluded regions are rejected along with any other region that violates the brightness-constancy assumption.

**Incorporating Occlusion Rejection into the Tracker**

The sequence of tests described above operates during the pre-processing stage. It is implemented on the GPU for performance. Thus, the overall procedure can be summarized as:

- Predict the pose in the current time-step,
- Project the OI’s model onto the image plane (i.e., generate the virtual image),
- Perform all filtering operations (e.g., LINF, blurring, and derivative calculations)
- Perform occlusion rejection as per Equations (4.13) to (4.15), and colour-gradient redundancy,
- Calculate the motion between the virtual and input images, and use this to estimate the actual pose of the OI, and
- Repeat this procedure for the next time-step.
Occlusion Rejection Side Effects

While the goal of the occlusion rejection algorithm is to facilitate tracking in the presence of partial occlusions, there are a few positive side effects. For example, the OI’s predicted pose may be significantly far from the actual pose of the OI, that parts of the projected image may cover background regions instead of the OI. Previously, these pixels would have been included in optical-flow calculations, even though they belong to different objects. The occlusion rejection algorithm filters out those regions as well. This will be seen in experiments (Section 4.4.5).

The key to the side effects is that, the algorithm rejects any pixels that violate brightness-constancy, regardless of whether they are caused by partial occlusions or not. This results in an overall improvement in robustness as it removes partial occlusions, occluded background regions (as described above), and other errors such as noise in dim regions, modelling errors, and specular highlights (which are not present in the model).

4.3 Improving Speed Through Extensions

Speed is another critical requirement for the proposed 6-dof tracking system to be used in real-time. Various techniques are proposed herein that boost speed. These include the extensive use of OpenGL and GPUs as well as a technique that exploits colour-gradient redundancy.

4.3.1 Using a Bounding Sphere

The OI, typically, only occupies part of an image. Therefore, the proposed object tracker operates only on a sub-window enclosing the OI (i.e., a bounding rectangle). This improves performance by reducing the required computation during filtering, the required graphics card memory to system-memory bandwidth, and the required amount of system memory.

In order to facilitate efficient selection of the sub-region to process, a bounding sphere
encloses the OI model. The bounding-box of the bounding sphere’s projection is the region that will be processed by the optical-flow algorithm. Figure 4.5 shows the 2D geometry in projecting this bounding sphere onto the image plane. Here, \( p \) is the position of the OI and \( r \) is the bounding-sphere’s radius. The camera’s parameters (from the matrix \( M_{int} \)) are: \( f \), the focal length, \( s_{x/y} \), the \( x \) or \( y \) scale factor and \( o_{x/y} \), the \( x \) or \( y \) offsets for the image’s origin in image coordinates (the last two parameters are not shown in Figure 4.5 to avoid clutter).

\[ \psi_{x/y} = \tan^{-1} \left( \frac{p_{x/y}}{p_z} \right), \tag{4.16} \]

\[ \theta_{x/y} = \sin^{-1} \left( \frac{r}{\sqrt{p_{x/y}^2 + p_z^2}} \right), \tag{4.17} \]

\[ \text{top\_edge} = \frac{f}{s_y} \tan (\psi_y + \theta_y + o_y), \tag{4.18} \]

\[ \text{bottom\_edge} = \frac{f}{s_y} \tan (\psi_y - \theta_y + o_y), \tag{4.19} \]

\[ \text{left\_edge} = \frac{f}{s_x} \tan (\psi_x - \theta_x + o_x), \tag{4.20} \]

\[ \text{right\_edge} = \frac{f}{s_x} \tan (\psi_x + \theta_x + o_x). \tag{4.21} \]
One can note that, above, \( x/y \) denotes \( x \) or \( y \), depending upon which axis (the \( x \)-axis or \( y \)-axis) is being examined.

### 4.3.2 Using the GPU and OpenGL

Computer-graphics hardware has been used extensively as co-processors. In particular, modern graphics cards include a GPU that can be programmed with custom algorithms. GPUs are well suited for vector operations and can, typically, perform multiple vector operations simultaneously. The computation power of a GPU generally exceeds that of CPUs, with the restriction that it is not a general purpose processor, and hence is suited only for certain tasks. Nevertheless, image processing, which forms the bulk of the computational load, can be performed quickly and efficiently on the GPU, leaving the CPU free for other tasks. Additionally, graphics hardware is used herein for projecting the OI model. This is a task in which graphics cards excel, and can perform far more quickly and efficiently than would a CPU.

The possible bottleneck that exists with graphics hardware use is the bus connecting it to the main processor. This has a limited bandwidth. For example, the AGP 8× bus can transfer 2100 MB/s to the graphics card and 266 MB/s from the card [84]. This can easily become the performance-limiting bottleneck. The more modern PCI-Express bus can manage 4 GB/s in both directions [84], but this still remains a possible bottleneck. For this reason, as much processing as possible is performed on the graphics card before transferring the bare minimum amount of data to main-memory required to achieve a task. Transferring back and forth is avoided.

### 4.3.3 Colour-Gradient Redundancy

Colour images contain more information about a scene than do black and white images. As colour cameras are readily available, it would be desirable to exploit the extra information. Calculating optical flow from colour image sequences yields improved accuracy, [85]. While
improved accuracy is desirable, colour images also contain three times as much data, thus, increasing the computation time for an already computationally-intensive task. This is also three times as much data that must be transferred from graphics RAM to main memory.

Several researchers, such as Barron and Klette [85] as well as Andrews and Lovell [86], have suggested various colour-space transformations to reduce the number of colour channels from three to two. This section presents a novel new technique that provides a threefold reduction in data without a noticeable loss in accuracy. It stems from an important observation: *due to the nature of visible matter, image gradients of the individual colour channels are generally aligned.*

Although this may seem counter intuitive at first, since, theoretically the gradients of each colour channel are independent; due to the nature of visible matter, colour transitions are from one discrete colour to another, cross gradients rarely occur. Effectively, gradients in each RGB colour channel are generally aligned. This fact can be exploited in order to provide a threefold reduction in the amount of data to be transferred and processed, without any significant loss of information.

*Measuring Pixel Directedness*

The fact that colour gradients are aligned, can be confirmed by analysing the distribution of gradients for a pixel in 2D space. Let us consider the matrix of gradients for the colour channels of a pixel:

\[
A = \begin{bmatrix}
\nabla_x I_R & \nabla_y I_R \\
\nabla_x I_G & \nabla_y I_G \\
\nabla_x I_B & \nabla_y I_B
\end{bmatrix},
\]

(4.22)

where \(\nabla_x\) and \(\nabla_y\) denote the \(x\) and \(y\) gradients, and \(I_R, I_G,\) and \(I_B\) are the Red, Green, and Blue components of the current pixel, respectively. The gradients have a major and minor direction,
which can be determined by decomposing $A$ using Singular Value Decomposition (SVD),

$$A = UDV^T,$$  \hfill (4.23) 

where $U$ is a $(3 \times 2)$ orthonormal matrix, $(D)$ is a $(2 \times 2)$ diagonal matrix containing the singular values of $A$, and $V$ is a $2 \times 2$ orthonormal matrix. The columns of $V$ associated with the maximum and minimum singular values in $D$ are the major and minor gradient directions, respectively. Instead of calculating the SVD for every pixel, it would be easier to use the relationship between SVD and eigenvector decomposition. Thus, for any pixel, the directedness matrix, $C$, is defined as:

$$C = A^T A.$$  \hfill (4.24) 

Combining Equations (4.23) and (4.24) yields,

$$C = V DDV^T = V \Sigma V^T,$$  \hfill (4.25) 

which is the eigenvector decomposition of $C$. Therefore, the eigenvectors $v_{1,2}$ of $C$ are also the major and minor directions of the distribution. The ratio between the maximum and minimum eigenvalues ($\lambda_1$ and $\lambda_2$, respectively) yields the ratio between the major and minor axes of the gradient distribution. The larger the ratio, the more uniform (or “directed”) the gradients are. By definition, $C$ is positive semi-definite, namely,

$$\lambda_1 \geq \lambda_2 \geq 0.$$  \hfill (4.26) 

If the ratio $\lambda_1/\lambda_2 \geq 10$, then, the pixel’s gradients can be considered to be approximately aligned. This corresponds to the gradient along the major axis being an order of magnitude greater than those along the minor axis.

**Experimental Verification**

Colour-gradient redundancy was verified experimentally by examining “directedness” of pixels. Several images were processed using the following procedure:
• Filter the input image with a Gaussian blur filter,
• Calculate the $x$- and $y$-axis gradients,
• Calculate the directedness matrix $C$ for each pixel,
• Calculate the eigenvalues $\lambda_{1,2}$ of $C$, and
• Output the following statistics for all pixels with $\lambda_1 \geq \tau$ ($\tau$ is the minimum gradient magnitude threshold):
  - The total number of pixels with $\lambda_1 \geq \tau$,
  - The mean maximum eigenvalue $\lambda_1$,
  - The mean minimum eigenvalue $\lambda_2$,
  - The ratio of the mean eigenvalues, and
  - The number of pixels with $\frac{\lambda_1}{\lambda_2} \geq 10$.

The objective is to obtain statistics of the gradients that would be used by the optical-flow algorithm. This is why the image is blurred and pixels with $\lambda_1 < \tau$ are discarded. This corresponds to steps taken for optical flow. For the analysis, the Gaussian blur’s sigma value was set to 1.8 and $\tau = 984 \times 10^{-6}$. These values match the parameters used in tracking experiments.

Statistics for an example set of images [87]-[90] are given in Table 4.1. The overall mean maximum eigenvalue, $\lambda_1$, of $C$ for all images is $6.87 \times 10^{-3}$ and the mean minimum eigenvalue, $\lambda_2$, is $34.1 \times 10^{-6}$. This results in a max/min ratio of about 200, suggesting that gradients are strongly aligned. As a ratio of 10 or more is considered to imply that the gradients are approximately aligned, the percentage of pixels below this threshold is of interest. On average, for the images considered, less than 2% of pixels had a ratio below 10. More importantly, pixels containing ratios below 10 often corresponded to phantom feature-points created by the crossing of occlusion boundaries, rather than actual feature-points. Such phantom feature-points are not reliable regions for tracking as they move independently of the OI.
Table 4.1: Image gradient statistics.

<table>
<thead>
<tr>
<th>Image</th>
<th>Number of Pixels with $\lambda_1 \geq \tau$</th>
<th>Mean $\lambda_1$ ($10^{-3}$)</th>
<th>Mean $\lambda_2$ ($10^{-6}$)</th>
<th>Min. $\lambda_1/\lambda_2$</th>
<th>% of pixels with $\lambda_1/\lambda_2 &lt; 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cones</td>
<td>71110</td>
<td>4.91</td>
<td>120</td>
<td>1.03</td>
<td>9.5</td>
</tr>
<tr>
<td>Convocation Hall</td>
<td>119180</td>
<td>12.9</td>
<td>12.6</td>
<td>1.19</td>
<td>0.30</td>
</tr>
<tr>
<td>Cube</td>
<td>73434</td>
<td>8.72</td>
<td>11.5</td>
<td>1.26</td>
<td>0.27</td>
</tr>
<tr>
<td>Iceberg</td>
<td>99110</td>
<td>3.37</td>
<td>6.57</td>
<td>3.60</td>
<td>0.050</td>
</tr>
<tr>
<td>Mt. Ngauruhoe</td>
<td>187924</td>
<td>3.9</td>
<td>26.3</td>
<td>1.07</td>
<td>1.3</td>
</tr>
<tr>
<td>Sandford Fleming Building</td>
<td>165735</td>
<td>10.1</td>
<td>1.73</td>
<td>1.07</td>
<td>0.67</td>
</tr>
<tr>
<td>Sunset</td>
<td>42548</td>
<td>4.42</td>
<td>17.6</td>
<td>2.67</td>
<td>0.51</td>
</tr>
<tr>
<td>Vinca Flower</td>
<td>60388</td>
<td>5.37</td>
<td>24.4</td>
<td>1.15</td>
<td>1.54</td>
</tr>
<tr>
<td>Water Lilies</td>
<td>155912</td>
<td>7.87</td>
<td>85.9</td>
<td>1.05</td>
<td>5.03</td>
</tr>
</tbody>
</table>

The above results are summarized visually in Figure 4.3.3 (b), which illustrates the various gradient regions for the cube image shown in Figure 4.3.3 (a). Black regions correspond to regions that would be discarded for optical flow due to inadequate gradients (i.e., $\lambda_1 \geq \tau$). White spots (in top left and bottom left of Figure 4.3.3 (b)) mark pixels with $\lambda_1/\lambda_2 < 10$. The vast majority of non-black pixels are coloured red. These are pixels with aligned gradients (i.e., $\lambda_1/\lambda_2 \geq 10$).

**Compressing Data by Exploiting Colour-Gradient Redundancy**

Given that colour-gradient alignment has been confirmed, this property can now be exploited. With all three colour channels being aligned, only one gradient and difference value need to be processed per pixel. All that is required is a suitable transformation. Simply converting the image to grayscale and, then, performing optical flow would not suffice as this would effectively be dropping back down to grayscale optical flow. This would result in lost detail in
regions where the contrast is caused by colour variations rather than intensity variations.

A seemingly obvious choice for the data-reduction transform would be to use the singular vectors corresponding to the maximum singular value of $A$. However, calculating the singular vectors requires significant computational resources, including performing several square-root operations. Instead, one can recognize that the gradients are aligned, and simply add the gradients. The only concern is that some gradients may be of opposite signs, resulting in them cancelling out one another. The following steps overcome this problem:

1. Calculate the directedness matrix $C = \begin{bmatrix} a & c \\ c & d \end{bmatrix}$,
2. If $d > a$, multiply the gradients and their corresponding difference values by the sign of the gradient’s $y$ component,
3. Else, multiply the gradients and their corresponding difference values by the sign of the gradient’s $x$ component, and
4. Sum the resulting gradients and difference values to produce one gradient and one difference value.

Difference values are the differences between virtual images and input images, on a per-pixel basis (i.e., the time derivative of the motion sequence). Multiplying by the sign of the
CHAPTER 4. 3D VISUAL-MODEL-BASED OBJECT TRACKING

The experimental set-up was the same as for previous chapters. The object tracker was run isolated from all the other support modules. Without the OI selector and modeller in place, the initial pose and an exact model were provided. This provides the most accurate representation of the object tracker’s performance. The goal was to measure the performance of the algorithm and its sub-components, and analyse the robustness to variations in image composition. Included in this is confirming that the proposed extensions to the tracking methodology improve overall performance and robustness.

In the experiments, a Kalman Filter (KF) [91] was employed as motion predictor. It is acknowledged that more sophisticated motion predictors exist that could improve the robustness of the motion predictor. For the purpose of testing, however, simplicity is important, as it allows better analysis of the object-tracking algorithm. If a complicated motion predictor were used, it could be hard to determine whether an exhibited behaviour is due to the tracking algorithm, or the predictor.

A more advanced motion predictor could be used later. Possible candidates include Hujic et al.’s [92] approach, which used a fading-memory filter in conjunction with the KF. Other options include an Extended KF (EKF) (Lippiello et al. [93]), Extended-Extended KF (EEKF) (Yang and Welch [94]), Unscented KF (UKF) (Ponsa et al. [95]), or interacting-multiple-models (e.g., [96, 97]) that switch between multiple dynamic models depending on the OI’s behaviour.
4.4.1 Expected Accuracies

It is insightful to examine the accuracies that can be expected before delving into the experimental results. Due to the non-linear nature of 3D to 2D projection, accuracy varies with an object’s size, distance from the camera, and location on screen. Moreover, there is cross-coupling between the errors in each of the six dofs, namely, an error in one dimension can cause errors in other dimensions as well. In order to obtain an idea of what level of accuracy can be expected, an object lying on the optical axis that is placed at the maximum distance from the camera that the physical set-up allowed (i.e., the maximum distance from the camera that an OI could have) is examined.

As images are sampled in 2D space, one can expect that object features can be localized to the nearest pixel (sample). This applies to both localizing points during calibration and to object tracking. Whilst achieving sub-pixel accuracy is possible, for this analysis, it is assumed that the combined calibration and tracking error is up to 1 pixel on-screen each. This provides us with a reference to which experimental results can be compared.

In experiments, the camera had a resolution of (512×256) and its horizontal field-of-view\(^1\) was 34°. The OI, in this analysis, is a cube measuring (96×96×96) mm, placed at 1210 mm from the optical centre. Under these conditions, using Equation (2.1), an on-screen positional error of 1 pixel corresponds to a 1.4 mm error in estimated 3D position parallel to the image plane (i.e., \(x\)- or \(y\)-axis error). If, on the other hand, there is an error in on-screen size (as opposed to position) of \(\pm 2\) pixels, the estimated 3D will have an error of \(+34\) or \(-32\) mm along the optical axis (i.e., the \(z\)-axis), respectively. A \(\pm 2\) pixel error in size corresponds to an error in the OI’s outer edges of 1 pixel each. On-screen errors can also cause errors in the orientation estimate (imagine a rotation such that the OI’s outer edges are displaced by 1 pixel). For the \(x\)-, \(y\)-, and \(z\)-axes (in camera coordinates), these errors are: 14°, 14°, and 1.2°.

\(^1\)The horizontal field-of-view is the angle about the optical centre spanned horizontally by the camera’s image sensor.
respectively. As world coordinates will rarely line-up with camera coordinates, it is best to expect errors to be up to 42 mm positionally and 15° in orientation. If errors are below this level, then, sub-pixel accuracy has been achieved.

The above errors assume that errors along each dof are decoupled from those of the other dofs. In reality, however, they are dependent (i.e., coupled). For example, an error of 14° about the x- or y-axis could cause an accompanying displacement of 7.3 mm in 3D position and vice versa. Errors also depend on the visual content. Additionally, the magnitudes and sizes of errors change as the OI moves away from the optical centre. Hence, the above numbers should be considered approximate only.

The above analysis shows that tracking is more accurate for 2D motion parallel to the image plane than along the optical axis. Hence, for OIs moving predominantly on a 2D plane, a bird’s-eye (top) view would provide the highest accuracy. Unfortunately, this is not possible for many robotic systems as the camera may have to be mounted on the robot itself. As autonomous robots is the main target application for the proposed object-tracking system, the experiments that follow have the camera placed close to the ground looking sideways, similar to cameras mounted on mobile robots.

4.4.2 Robustness to Lighting

In real-world environments, the tracker must cope with a wide range of different lighting conditions and environmental clutter. The following experiments evaluate the tracker’s robustness to several different conditions. For these experiments, the OI followed the path shown in Figure 4.7. The abrupt sharp changes in direction simulate a maneuvering OI.

Figure 4.8 shows frames from several different motion sequences with different image compositions. In order, from 4.8 (a) to (f), the sequences have: a cube with a white background and reasonably diffuse (fluorescent) lighting, directional lighting from the left side, low-lighting conditions, a complex background (i.e., the laboratory), a spinning globe, and a rectangular
Figure 4.7: The path taken by the OI.

box. In the low-light motion sequence, the camera has compensated somewhat for the low lighting. As a result, the image looks brighter but contains more noise. The spinning globe sequence was generated synthetically and includes a directional light placed diagonally above and to the right of the scene.

The tracking errors for Figures 4.8 (a) to (f) are shown in Figures 4.9 to 4.14, respectively. The first thing to be noticed in these figures is that the observed errors are smaller than the pixel-accurate error bounds calculated in Section 4.4.1, i.e., sub-pixel accuracy is achieved. Comparing the results for all the sequences, it can be noted that bright diffuse lighting is the preferred lighting. Accuracy decreases as the lighting conditions deviate from these conditions. However, even for the low-light sequence, Figure 4.11, sub-pixel accuracy is still achieved, with maximum positional error being 24 mm and orientational errors being less than $2.9^\circ$.\footnote{The errors statistics are for the absolute distance and total orientational errors (sum of orientational errors for all axes), respectively.}

When the OI model is projected using the tracked pose, its motion is visually indistinguishable from actual OI’s motion.

The object tracker is able to track an OI in the presence of a complex background. This is shown in Figure 4.12. Some difficulty may be encountered if the background contains similar structure to the outer edges (i.e., the OI’s occlusion boundary).

The rotating globe sequence, Figure 4.8 (e), demonstrates tracking an object that is more...
Figure 4.8: Frames from motion sequences with diffuse lighting (a), directional lighting from the side (b), low light conditions (c), a complex background (d), a globe as OI (e), and a cardboard box and complex background (f).
complex object than the cube and box used in the other sequences. Errors obtained, Figure 4.13, show that objects with complex surface patterns can be tracked accurately with the proposed algorithm. Moreover, the globe model contains 2500 triangular surfaces as opposed to the 8 rectangular surfaces in the cube model. There was no noticeable reduction in algorithm speed despite the added complexity. Increasing the number of surfaces to 100000 caused the frame-rate to drop from (80 to 100) down to (71 to 91). Hence, the algorithm scales well with OI geometry complexity.

In the final sequence, (f), a box with complex texture was used as the target. A model of the box was built manually by measuring its dimensions and capturing its surface texture using the camera. The tracking accuracy is similar to those for the other objects tested in the previous sequences, Figures 4.14.

![Figure 4.9: The positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (a).](image)

Effectiveness of the LINF

The experiments above include the usage of the LINF module whose task it is to filter out shading effects. The effectiveness of the LINF can be noted by comparing tracking results
Figure 4.10: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (b).

Figure 4.11: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (c).
Figure 4.12: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (d).

Figure 4.13: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (e).
Figure 4.14: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.8 (c).

when the LINF is used, and without. In these experiments, only the core tracking algorithm and the LINF module were used; none of the other tracking extensions (e.g., colour-gradient redundancy) were present. Synthetic image sequences were chosen for this test since they allow complete control over the lighting, including testing in the presence of 100% ambient light. With pure ambient lighting, the tracking errors were similar regardless of whether the LINF was present. Once directional lighting is introduced, the LINF provides significantly better performance, Figure 4.15 (b). Without illumination normalization, the tracker diverged from the true OI trajectory in the presence of lighting effects, Figure 4.15 (a). The motion sequence for Figure 4.15 contained a distant light source that was placed above and to the right of the camera.

4.4.3 Robustness to Modelling Errors

The OI models provided in the experiments above were carefully constructed to be as accurate as possible. A successful tracker must be able to cope with imperfect models. Tests were run in which a geometric modeling error (in the object’s dimensions) was deliberately introduced
4.3D VISUAL-MODEL-BASED OBJECT TRACKING

Figure 4.15: The positional tracking errors in the presence of shading without (a) and with (b) using the LINF.

to the tracker’s model of the OI. Modelling errors of up to 22% were tolerated for the synthetic sequences tested. Above 22%, the tracker diverged from the true OI trajectory (Table 4.2).

4.4.4 Robustness to Large Motions

Tracking robustness to large motions was examined using motion sequences with increasing velocities, Figure 4.8 (d). The same path as in Figure 4.7 was followed, but the velocities were doubled.

The object tracker was set to use two pyramid levels. It was determined that more levels would cause the estimated pose to oscillate, probably caused by the sub-sampled images becoming too small. Thus, one may conclude that the number of pyramid levels that can be used depends on the on-screen size of the object.

In theory, the maximum allowable motions are decided by the Gaussian blur sigma value, and the number of pyramid levels. Input images are blurred to enforce local linearity for optical flow. A sigma value of 1.8 was used resulting in a theoretical reliable motion estimation of up to 1.22 pixels in any direction (2D on-screen). Due to the cube’s distance from the camera, this
Table 4.2: Tracking error as a function of modeling error.

<table>
<thead>
<tr>
<th>Geometric-Modelling Error (%)</th>
<th>(Maximum Tracking Error / Cube Edge Length) × 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.9</td>
</tr>
<tr>
<td>2.5</td>
<td>12.2</td>
</tr>
<tr>
<td>5</td>
<td>17.8</td>
</tr>
<tr>
<td>10</td>
<td>23.5</td>
</tr>
<tr>
<td>15</td>
<td>17.14</td>
</tr>
<tr>
<td>20</td>
<td>20.0</td>
</tr>
<tr>
<td>22.5</td>
<td>21.0</td>
</tr>
<tr>
<td>25</td>
<td>(OI lost)</td>
</tr>
</tbody>
</table>

corresponds to approximately 2 mm/frame. Thus, with two pyramid levels, one would expect a maximum allowable velocity of approximately 4 mm/frame or 120 mm/s (moving to the right, and slightly away from the camera). This assumes that the 3D motion is predominantly parallel to the camera’s image plane.

The results are summarized in Table 4.3. The first failure occurs in Sequence 3, with an initial velocity of approximately 134 mm/s. This is roughly what the theoretical maximum velocity is. However, the algorithm successfully tracks the OI (the cube) for 8.8 seconds, and the failure point occurs at a sharp change in direction of motion. The OI also touched an edge of the image, resulting in incorrect output from the filters due to lack of data. The sharp change in direction implies that the motion predictor estimates the predicted pose further from the actual pose for that frame than if no prediction were used. The filter boundary results in possible corrupt data for a narrow band of pixels along the image edge. The tracker had already successfully tracked the OI through three previous sharp trajectory changes.

In Sequence 4, the tracker successfully initiates tracking on an object with an initial velocity of 268 mm/s. This is above the expected maximum speed. However, the object tracker
uses a motion predictor. Thus, it is not velocity, but the change in velocity that would cause large offsets between the predicted and actual poses. It fails at the second sharp corner in the trajectory. Examining the tracking output, it was noted that the object tracker almost fails initially, with the pose estimate lagging behind the actual pose of the first few frames. The motion predictor quickly helps eliminate this offset.

Finally, in Sequence 5, the tracker fails immediately. The initial velocity of 537 mm/s is too much for the uninitialized tracker. The object-tracker’s motion predictor starts with zero velocity as the initial velocity is unknown.

Given the results above, one may conclude that, the maximum trackable velocity should be somewhere between 134 mm/s and 268 mm/s. It is important to note that, this is for the given experimental set-up only. A change in camera parameters, the OI’s distance from the camera or its trajectory would affect these limits. In particular, larger motion can be tolerated along the camera’s optical axis as this results in smaller changes on-screen. Given that only two pyramid levels could be used reliably for multi-scale optical flow, it is clear that further research could be performed in order to expand the allowable range even further.

<table>
<thead>
<tr>
<th>Seq. #</th>
<th>Initial Velocity mm/s</th>
<th>Success/Fail</th>
<th>Failure Frame #</th>
<th>Failure point</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(30, 15)</td>
<td>Success</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(60, 30)</td>
<td>Success</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(120, 60)</td>
<td>Fail</td>
<td>263</td>
<td>Fourth direction change (OI at image edge).</td>
</tr>
<tr>
<td>4</td>
<td>(240, 120)</td>
<td>Fail</td>
<td>83</td>
<td>Second direction change.</td>
</tr>
<tr>
<td>5</td>
<td>(480, 240)</td>
<td>Fail</td>
<td>1</td>
<td>Immediate tracking failure.</td>
</tr>
</tbody>
</table>
4.4.5 Robustness to Occlusions

Tracking accuracy was tested both with and without the occlusion-rejection algorithm enabled. All test sequences were based on a single sequence of images containing a cardboard box as OI. Sequences containing partial occlusions were generated from this base sequence by compositing other objects over the sequence. This provides us with a reference, or control sequence, and a range of different types of occluding objects. The thresholds were set to:

\[ \tau_{gm} = 98.4 \times 10^{-6}, \tau_{\theta} = 0.90, \text{ and } \tau_{md} = 39.2 \times 10^{-3}. \]

Figure 4.16 shows a sequence in which the partial occlusion is caused by a vertical white bar. Pixels that are used in optical-flow motion calculations are marked in green for both the basic tracking algorithm, Figure 4.17 (a), and with the occlusion-rejection algorithm, Figure 4.17 (b). In Figure 4.17 (b), the original algorithm incorrectly uses pixels within the white bar as part of motion calculations, resulting in tracking failure, Figure 4.18. When the occlusion-rejection algorithm is added, motion calculations no longer include the white bar, Figure 4.17 (b), resulting in successful tracking, Figure 4.19, with errors below \( \pm 10 \text{ mm} \) positionally and \( 2.5^\circ \) orientationally, respectively. This level of accuracy is similar to the level of accuracy achieved when no occlusions are present.

Figure 4.16: A white bar partially occludes the Olimex box, which is the OI.

The white bar is quite a simple occluding object. Figure 4.20 shows a frame from a se-
Figure 4.17: Without occlusion rejection, the tracker incorrectly uses pixels belonging to the white bar for tracking (a); the occlusion-rejection algorithm filters out the white bar (b).
Figure 4.18: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.17, when tracking without the occlusion-rejection algorithm.

Figure 4.19: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.17, when tracking with the occlusion-rejection algorithm.
quence in which the cube – that was used as OI in previous experiments – is the occluding object. Once again, the original algorithm fails to track in the presence of partial occlusion, Figure 4.21, but the new algorithm tracks it accurately until close to the end of the sequence, Figure 4.22. At the end of the sequence the estimated pose starts deviating from the true pose. This is due to one of the cube’s edges being similar enough to the OI that it is not perfectly removed by the occlusion-rejection algorithm. However, it should also be noted that the OI is mostly occluded at this point, leaving a small number of pixels available for motion calculations. Thus, a small number of erroneous pixels have a larger impact on the motion estimate.

Figure 4.20: Without occlusion rejection, the tracker incorrectly uses pixels belonging to the white bar for tracking (a); the occlusion-rejection algorithm filters out the cube (b).
Figure 4.21: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.20, when tracking without the occlusion-rejection algorithm.

Figure 4.22: Positional (a) and orientational (b) tracking errors for the sequence in Figure 4.20, when tracking with the occlusion-rejection algorithm.
For reference, the occlusion-rejection algorithm was also tested on the original, unoccluded, sequence, Figure 4.23. Close inspection of Figure 4.23 shows that the occlusion-rejection algorithm has removed pixels in regions with unreliable data. As a result, tracking errors have dropped from $\pm 8$ mm to $\pm 6$ mm positionally, and from $3.5^\circ$ to $2.5^\circ$ orientationally, respectively. This suggests that the occlusion-rejection algorithm may improve overall robustness and accuracy beyond simply coping with occlusions. It also eliminates other regions that violate the brightness-constancy constraint upon which the optical-flow algorithm is based.

Adding the occlusion-rejection algorithm did not noticeably lower the tracking speed. Both

![Figure 4.23](image-url)
the original algorithm, and the new one operated on the given motion sequences at 40-70 fps on a 2.0 GHz AMD Athlon 64 3000+ running Windows XP with a Radeon X800 Platinum Edition GPU.

As with any algorithm, some limitations exist. Objects with similar edge characteristics or surface features could pass the occlusion-rejection algorithm and cause a corresponding decrease in accuracy (e.g., in the last sequence, Figure 4.23). Fortunately, this algorithm does not exclude the use of other techniques. It rapidly removes many occluded, or otherwise invalid, pixels. Other algorithms could operate on the remaining pixels, thus, improving robustness further. For example, robust estimation methods could be used in order to filter out outliers. Up to now, iterative methods, such as Robust Estimation [78], have been avoided due to their computational expense and the need to maintain real-time performance. However, with advances in computational power and suitable acceleration structures, this may become a viable technique.

4.4.6 Colour-Gradient Redundancy – Speed

In the tests run, the tracker processed image sequences at approximately 80 to 100 fps (depending on the distance to the OI) for the cube and globe models. The larger Olimex box resulted in tracking of 40-70 fps, due to its larger on-screen size. These results were obtained with a 2.0 GHz AMD Athlon 64 3000+ with Windows XP GPU was a Radeon X800 Platinum Edition. This frame-rate is far above the minimum 10 fps required for real-time robot navigation operations. Increasing the processing rate further would be possible by making certain optimizations. For example, the vector and matrix operations could be performed using Single-Instruction Multiple-Data (SIMD) extensions such as SSE and SSE2. These extensions allow arithmetic operations to be performed simultaneously on multiple data values.

Of particular interest is to determine how much of a performance boost the colour-gradient redundancy did the data-reduction algorithm produce. Theoretically, the increase should only
be threefold. Instead, it was found that using the colour-gradient redundancy data-reduction algorithm resulted in a fivefold increase in speed over using normal colour optical-flow. The additional increase can be attributed to the effect of CPU caches and/or optimizations in the graphics drivers. Access to data in a cache is faster than from main memory. Hence, if the reduced data fits inside the cache entirely, a large increase in speed can be observed.

One concern is that the fivefold performance boost will come at the cost of decreased accuracy. In order to determine this, tracking errors obtained when exploiting colour-gradient redundancy were compared to those obtained when raw colour data was processed directly. These errors are shown in Figures 4.24 and 4.25, respectively. Comparing the errors obtained with the data reduction algorithm to those obtained without using the data-reduction algorithm, one notes that the error ranges are quite similar. Both have mean errors of approximately 2.2 mm and 1.0°, and maximum errors of approximately 5.5 mm and 3.1°. The errors statistics are for the absolute distance and total orientational errors (sum of orientational errors for all axes), respectively. This confirms that the proposed colour optical-flow algorithm provides significant performance increase, without adversely affecting accuracy.

4.5 Example Application – Convoying

This section details the use of the object-tracking system in a target application: convoying, also known as platooning. The basic concept is to have a lead vehicle whose trajectory is followed by one or more trailing vehicles. Each of the trailing vehicles maintain a constant separation distance (or a constant separation time). Studies performed by Hochstadter and Cremer [98], Wu et al. [99], and others suggest that autonomous convoy driving has the potential to reduce traffic congestion. Such convoys would have faster response time that human drivers, allowing vehicle separation distances to be reduced safely. Single-driver freight convoys would also be possible, both on the road and in industrial settings.

Another possible application would be shadow vehicles [100]. A shadow vehicle is a ve-
Figure 4.24: Positional (a) and orientational (b) tracking errors for the motion sequence shown in Figure 2.12 obtained using the colour-gradient redundancy algorithm.

Figure 4.25: Positional (a) and orientational (b) tracking errors for the motion sequence shown in Figure 2.12 obtained when the raw-colour data was used for tracking.
Vehicle driving slowly at the start of a section of highway containing road works. It is there to protect the workers as they slowly move along the highway performing maintenance operations, but places the shadow vehicle driver at risk from on-coming traffic. An autonomous shadow vehicle would eliminate this risk.

A number of different techniques for autonomous convoying have been presented in the literature. These include using radar [101], linear cameras [102] and matrix cameras [103]. Many of these techniques rely on markers or reflectors, which is undesirable as it limits convoying to vehicles with those markers. Computer vision could potentially be used for markerless convoying. Marapane et al. [104] describe a convoy system that uses stereo cameras to obtain 3D position at 15 Hz. However, their algorithm’s core is a 2D tracking algorithm and no orientation is obtained. Orientation information is useful for a convoy control system as it provides the lead vehicle’s direction of travel without resorting to differentiation, which is sensitive to noise. Thus, a 6-dof pose tracker could be an effective component in such an autonomous convoying system.

A simple convoy-control system was implemented in this Thesis that uses the proposed 6-dof pose tracker. The lead vehicle is the system’s OI. A single camera captures images of the lead vehicle and the tracker outputs the trajectory of the lead vehicle as a stream of 6-dof poses. This stream of poses is used by the convoying control algorithm to determine what control outputs to pass to the following vehicle’s drive system. The control algorithm follows the path of the lead vehicle and maintains a constant following distance. Following the lead vehicle’s path avoids cutting corners and hitting objects/pedestrians. Unlike some other convoying systems that control only speed, this system controls both speed and steering. As a result, the system is fully autonomous.

The implemented convoy-control system only requires 2D (3-dof) pose. However, the additional 3 dof provided by the object tracker could be used by a more advanced system to extract the terrain contour along the desired trajectory. This is a possible future extension of the convoying system.
4.5.1 Autonomous Convoy Controller

A simple discrete-time state-space controller was implemented. Whilst a more sophisticated controller could be utilised to achieve better performance, the focus of this simulated experiment was on the use of the object tracker. The controller takes the pose trajectory given by the visual tracker and outputs linear acceleration and angular velocity commands to the drive system.

Given a following vehicle (Vehicle \( i \)) and a lead vehicle (Vehicle \( i-1 \)), the control law is as follows:

\[
\begin{bmatrix}
    a_i \\
    \omega'_i
\end{bmatrix} = \begin{bmatrix}
    c_1(d_i - v_i) + c_2(v_{i-1} - v_i) \\
    c_3(\phi_i - \omega_i) + c_4(\psi_i - \omega_i)
\end{bmatrix}, \tag{4.27}
\]

where \( a_i \) and \( \omega'_i \) are the linear acceleration and angular velocity commands for Vehicle \( i \), respectively. \( D_i \) is the distance between Vehicle \( i \) and Vehicle \( i-1 \), along the trajectory followed by Vehicle \( i-1 \). \( v_i \) and \( \omega_i \) are the linear and angular velocities of Vehicle \( i \), respectively. \( \phi_i \) is the rotation between the current direction of travel and the next desired position. \( \psi \) denotes the angle between the current 2D orientation (\( \theta_i \)) and the desired orientation in the next time-step.

None of the control inputs above correspond directly to the stream of poses output by the object tracker. The stream of poses must be interpreted in order to obtain \( v_{i-1}, \phi_i, \) and \( \psi \). Let \( A_i \) be the set of all \( N \) poses of Vehicle \( i-1 \) between the current positions of Vehicles \( i \) and \( i-1 \):

\[
A_i = \left\{ \bigcup_{n=1}^{N} P_{n,i-1} \right\}, \tag{4.28}
\]

where \( P_{n,i-1} \) is the pose of Vehicle \( i-1, n \) steps away from Vehicle \( i \). The following distance, \( d_i \), is calculated using the following equation:

\[
d_i = \sum_{n=1}^{N-1} \| p_{n,i-1} - p_{n+1,i-1} \| + \| p_{1,i-1} - p_1 \|. \tag{4.29}
\]

The 3D positions \( p_{n,i} \) and \( p_i \) are obtained from the pose matrices (Equation (1.1)). The variable \( p_i \) contains the current estimated position of Vehicle \( i \) (i.e., the last estimated position of the lead vehicle).
The value of $\phi_i$ is calculated as the angle between the following Vehicle $i$’s direction of travel, and a point $p_{j,i-1}$ along the lead vehicle’s trajectory that is a set distance ahead. The variable $j$ is a non-negative integer. Thus, $\phi_i$ is

$$\phi_i = \angle (p_{j,i-1} - p_i, v_i),$$

where $v_i$ is the current traveling direction of Vehicle $i$. Likewise, the next desired orientation ($\theta_d$) is taken to be the orientation obtained from $P_{1,i-1}$. Thus,

$$\psi_i = \theta_d - \theta_i,$$

where $\theta_i$ is the 2D orientation of Vehicle $i$. Once all of the variables above have been calculated, they are passed on to the controller.

### 4.5.2 Overview of Complete Convoy-Control Algorithm

The complete convoy-control algorithm for the following vehicle (Vehicle $i$) is as follows:

- Obtain the lead-vehicle’s (Vehicle $i-1$) current pose using the visual object tracker,
- Add this pose to the array of lead vehicle poses, $A_i$,
- Calculate the input values to the controller and execute the control command, and
- Remove all poses from $A_i$ that are now behind the following vehicle (Vehicle $i$).

Repeat the above procedure for each time-step.

### 4.5.3 Experimental Set-up

The complete visual convoy-control system was tested via simulation of a two-vehicle convoy. The lead vehicle was set to follow a predefined trajectory that included both acceleration and deceleration whilst. The simulator generated images from the viewpoint of the following vehicle and passed those to the object tracker. Control outputs from the convoying algorithm were,
then, passed to a simulated vehicle drive system. The simulated vehicles were $55 \times 55 \times 55$ units in size. A virtual camera was placed at the following vehicle’s centre, facing forward.

In a real convoy, neither vehicle would follow a perfect trajectory. Bumps in the road could cause image jitter that would result in the lead vehicle appearing to shake in the image sequence captured by the virtual camera. In order to simulate this, Gaussian noise was added to the 3D position of the lead vehicle relative to the camera (on the following vehicle) before generating the virtual camera’s images. The added noise was set such that 99.73% percent (the $\pm 3\sigma$ limits) of the resulting 2D on-screen errors are within $\pm 4$ pixels: this results in frame-to-frame motion of up to $\pm 8$ pixels, a large amount of noise for the object-tracker to cope with.

The following vehicle’s camera has a resolution of $512 \times 256$. At a following distance of 300 units, the lead vehicle is approximately $200 \times 200$ pixels on-screen. Figure 4.26 shows a sample (synthetic) image of the lead vehicle. The simulated camera in the following vehicle was set to a capture-rate of 30 Hz, corresponding to the standard NTSC video rate. However, the algorithm itself operated at 33 to 35 Hz (much faster than the minimum 10 Hz required). In practise, therefore, the algorithm would be capable of operating at standard video rates. This was achieved using a 2.0 GHz AMD Athlon 64 3000+ with Windows XP and a Radeon X800 Platinum Edition GPU.

Figure 4.26: A lead vehicle image as captured by the following-vehicle’s camera.
4.5.4 Results

In the results provided below, the lead vehicle accelerated to 720 units/s in 2 s. Following this, it executed a left-hand turn followed immediately by a right-hand turn at constant speed. Finally, the lead vehicle decelerated to a stand-still in 2 s. The set separation distance was 300 units between the lead vehicle’s centre and the centre of the following vehicle’s camera.

Figures 4.27 (a) and (b) show the $x$- and $y$-axis trajectories for the lead vehicle, respectively, along with the tracker’s estimates. Likewise, the orientational trajectories are displayed in Figure 4.27 (c). Figure 4.28 shows the tracking errors for these trajectories over time. RMS errors for the $x$- and $y$-axis were 0.31 and 0.35 units, respectively; RMS orientational errors were $0.51^\circ$. All maximum errors were within acceptable limits. These figures show that the lead-vehicle’s trajectory estimates are very accurate, and follow the actual trajectory closely. Only the 3 dof used for tracking are shown in these figures; it is noteworthy to remember that the other 3 dof are also being tracked.

In the simulations, the autonomous convoy controller successfully maintained a separation of 300 units to within $\pm 5$ units, Figure 4.29. This corresponds to a separation of about 0.4 seconds. Additionally, the perpendicular distance (error) between the following- and lead-vehicle trajectories was less than 3 units, Figure 4.30. This demonstrates the potential of using the proposed visual object-tracking based convoy-control methodology: It can autonomously maintain safe following distance and follow the correct trajectory when traveling with a separation of just 0.4 seconds between vehicles.

---

3Separation time between vehicles is the amount of time that the vehicles would take to cover the distance between them.
Figure 4.27: The lead vehicle’s $x$ (a), $y$ (b), and $\theta$ (c) trajectories.
Figure 4.28: The $x$ (a), $y$ (b), and $\theta$ (c) tracking errors.
Figure 4.29: The following distance as a function of time.

Figure 4.30: The perpendicular trajectory error.
4.6 Summary

This Chapter has presented the core tracking algorithm plus various extensions developed for improved robustness and speed. The result is an object tracker that can track the 6-dof pose of an OI in real-time, given a model. It works by using optical flow to measure the pose disparity between a projection of the model at a predicted pose, and the image of the actual OI. It can track with sub-pixel accuracy.

A key element in the operation of this algorithm is extraction of 3D information from the projection of the OI model. This model projection is performed using 3D graphics hardware via OpenGL. As part of this process, a $z$-buffer is produced, containing the depth value for each pixel. This is the information lost in 3D to 2D projection, which usually prevents 6-dof pose tracking with a single camera. By extracting this depth information, 6-dof motion can be calculated with only a single camera.

Various techniques have been developed to improve robustness. The first is the LINF, which normalizes images such that the mean colour in a local region is mid-gray. This reduces the effect of shadows on the optical-flow calculations. Experimental results show that the resulting object tracker can operate successfully over a range of different lighting conditions, including situations such as directional lighting, in which the core tracking algorithm would fail. Experimental results have also shown that the object tracker can cope with significant errors in the model.

Another extension adapts multi-scale optical flow for 6-dof pose tracking. This ensures that the object tracker can cope with large motions, i.e., large disparities between the predicted and actual poses. In the experiments performed, velocities of up to $(120, 60)$ mm/s were tracked reliably. Tracking could even be achieved up to $(240, 120)$ mm/s, but with noticeable lags in the pose estimate, and the possibility of tracking failure with sharp changes in direction.

The final robustness extension is a technique for coping with partial occlusions. This is particularly important for real-world applications in which the OI may be partially occluded
from view by static or dynamic obstacles. This extension operates on the principle of rejecting pixels which violate the brightness-constancy assumption that underpins optical-flow. This prevents the optical-flow module from erroneously calculating the motion between disparate objects, or even, different parts of the same object. Experiments show sub-pixel tracking of partially occluded OIs in situations that the core algorithm would lose the OI. Moreover, a tangible increase in accuracy was observed when tracking an unoccluded OI.

The other extensions relate to boosting speed in order to obtain real-time performance. The algorithm makes extensive use of the GPU both for projecting images of the OI, and for performing image-processing. Additionally, a data-reduction algorithm was presented that provides a threefold reduction in data without losing the advantages of colour optical-flow. It exploits a property of images known as colour gradient redundancy, which can be characterized as: image gradients for the colour channels in an image are generally aligned. The measured performance increase was fivefold; higher than the threefold boost that was expected. This is due to the effects of things such as data caches.

The end result of all of the above is a real-time 6-dof pose tracker operating at frame-rates exceeding 30 fps (40-100 fps in experiments, depending on the OI size). At these frame-rates, the proposed algorithm is suitable for a range of robotic tasks. It can track in 3D using only a single camera.

Finally, an example use of the object tracker was presented; namely, a convoy control system. This system enables a vehicle to autonomously follow a lead-vehicle’s trajectory whilst maintaining constant separation distance. Such systems have applications such as reducing traffic congestion via closer car spacing, single-driver transport convoys, and, autonomous shadow vehicles that protect road workers. The object tracker was used to provide the relative pose of the lead vehicle over time. The convoy controller, then, extracted the required control inputs from this trajectory data, and sent control outputs to the vehicle’s drive system. In simulations, a separation distance of 300 units was maintained to within ±5 units, corresponding to a separation time of 0.4s. During the simulation, the lead vehicle executed accelerations and
turns.
Chapter 5

System Implementation – Simulations and Experiments

5.1 Implementation

Previous chapters have discussed the individual thesis contributions in isolation. This chapter combines all the proposed sub-algorithms (i.e., modules) into one object-tracking system and evaluates the system as a whole. The interactions between the modules are shown in Figure 5.1, which is a reprint of Figure 2.1.

The Object of Interest (OI) selector is an initialization module. It is run on start-up until an OI has been selected. Subsequently, the OI selector module initializes tracking by passing a region of interest to the modeller along with a generated initial pose. After initializing the modeller, the OI selector ceases operation. An alternative startup procedure could be user-based OI selection. In such a scenario, the motion segmentation would be displayed to the user, who would then choose the object he/she wishes the system to track. The region encompassing the OI, along with an initial pose would then be passed on to the modeller in the same manner as by the fully automated OI selection module proposed herein.
Both the modeller and the OI selector use the Interest Filter (IF) bank. Thus, the IF bank operates continuously, passing Interest Maps (IMs) to either the OI selector, or to the modeller, as required. Due to one of the implemented IFs being based on a background-subtraction algorithm, the IF bank must run even on frames in which the model is not rebuilt in order to maintain the background.

The modeller does nothing until the OI selector provides the initial pose and region of interest. At this point, it generates the initial model around the provided initial pose. One may note that the OI selector’s initial pose generator temporarily borrows the Depth-Map Extractor.
(DME) from the modeller as this is required in order to generate an initial pose. Future IFs may require Depth Maps (DMs) in order to operate. Hence, the DME could be treated as a separate module that is primarily used by the modeller.

After the initial model is built, the object tracker is enabled. The OI model is passed to the tracker, and it proceeds to track the object. The tracker outputs the estimated pose for every time-step. Once every $n$ frames, the tracker passes the current estimated (tracked) pose to the modeller and triggers a model rebuild. The new model replaces the old one, and tracking proceeds.

In summary, the complete tracking procedure, starting with initialization, is as follows:

- **Initialization:**
  - The IF bank is initialized using the first $m$ frames (e.g., first 5-10 frames),
  - The OI selector searches for a region that is of interest and meets all criteria for an OI,
  - If a region is found that meets all OI criteria and is of greatest interest,
    - An initial pose is generated for the selected region (i.e., region $R$), and
    - The OI selector is shutdown, and modelling begins (i.e., proceed to modelling).
  - Else, the previous step is repeated for the next frame (i.e., the OI selector continues its search).

- **Modelling:**
  - A DM is obtained,
  - A 3D surface of region $R$ is built that is referenced to the provided pose,
  - The projective texture is extracted, and
  - The completed OI model is passed on to the object tracker for the tracking phase.

- **Tracking:**
– The OI’s pose in the current frame is predicted,
– The OI model is projected onto the image plane at its predicted pose, forming a virtual image,
– Optical flow estimates the motion between the projected OI (in the virtual image) and the actual OI (in the input image),
– The predicted pose is transformed by the estimated motion to produce the output pose estimate for the current frame,
– If \( n \) frames have passed,
  * The bounding box for the OI in the current frame (calculated from the bounding volume) becomes region \( R \), and
  * Region \( R \) and the current estimated pose are passed to the modeller and the modelling procedure is repeated to build a new model (i.e., return to Modelling).
– The next frame is processed from the start of the tracking procedure.

5.2 Complete System Test

The complete tracking system was tested using both real, and synthetic motion sequences. The experimental set-up is identical to that of the modeller experiments (Chapter 3). An OI was placed on a precision \( x-y \)-stage and a colour camera was used to capture images from five viewpoints, simulating the use of five cameras. As with previous experiments, two IFs were used: a colour IF, and a motion IF based on background subtraction. The colour IF was tailored to the specific OI being tracked in each case.
5.2.1 Tracking Results

Results for Synthetic Image Sequences

Figure 5.2 shows an example frame for a globe sequence in which the OI selector recognized the globe as an OI, and initialized tracking of that globe. In this sequence, the globe rotated at constant velocity whilst following the path in Figure 5.3. The IF bank was set to highlight the globe, and OI selection proceeded as shown in Figure 5.4. The modeller built a model on-line, Figure 5.5(a), and the tracker proceeded to track the OI (i.e., the globe), Figure 5.6.

Figure 5.2: Frame 21 from a synthetic sequence taken by the central camera.

Tracking errors are within 30 mm positionally, and $5^\circ$ orientationally, Figure 5.6. These tracking errors are approximately six times larger than when tracking with an ideal model, in which case the positional and orientational errors were 5 mm and $0.9^\circ$, respectively (Figure 5.7).

Chapter 3 discussed that rebuilding the OI model periodically could result in the drifting of the reference frame. This can be noted in the error plots in Figure 5.6. Whilst this could likely be solved by further research into real-time modelling for object tracking, it is deemed to be beyond the scope of this Thesis. This Thesis presents a first step in the development of 6-dof real-time object tracking of a priori unknown objects, where modelling is just one of several sub-processes.
Figure 5.3: The path taken by the OI in synthetic motion sequences.

Figure 5.4: The segmentation-map (a) and interest-map (b) for Frame 21, Figure 5.2.
In the object-tracking system’s present state, the reference frame drift is slow: namely, the current tracking system should be acceptable for short-term tracking. The proposed tracking system would also be usable for initial tracking of known objects. In such cases, the OI selector would select an object, and the object tracker would track the object using an approximate model built by the on-line modeller. In parallel with this, an external system would work to recognize the object, and load the exact model. At this point, the on-line modeller could be disabled, and the precise model would be used for tracking. This would allow time for object recognition to complete without risking losing track of the OI.

**Results for Real-World Image Sequences**

Figure 5.8 shows an example frame from a real-world sequence taken by the central camera (Camera 3 of five). On startup, the object-tracking system uses the first few frames to initialize the IF bank’s filters. At Frame 10 (Figure 5.8), the OI selector segments the input image
Figure 5.6: Positional (a) and orientational (b) tracking errors for the motion of the object in Figure 5.2, using the approximate OI model built on-line.
Figure 5.7: Positional (a) and orientational (b) tracking errors for the motion of the object in Figure 5.2, using an a priori provided model.
(Figure 5.8 (a)) whilst the IM (Figure 5.8 (b)) highlights the desired object, which was selected as OI. The selected OI region is passed it on to the modeller for modelling, Figure 5.10(a). With the model built, the object tracker proceeds to track the object. The modeller rebuilds the OI model every 30 frames. Several frames showing the tracking and modelling processes are shown in Figure 5.10. The OI path was the same as those used in Chapter 3, Figure 5.11.

Figure 5.8: Frame 10 from a real-world sequence taken by the central camera.

Tracking errors for the motion of the object in Figure 5.8 are shown in Figure 5.12. Errors were within 15 mm positionally and $10^\circ$ orientationally. With a perfect model, tracking errors for the same sequence was to within 10 mm positionally and $5^\circ$ orientationally, Figure 5.13. Thus, for this particular sequence, using an approximate model that was built and updated on-line resulted in errors that were approximately 1.5-2 times larger, when compared to tracking using a perfect model.

Comparing the results for the synthetic motion sequence above with the real-world motion sequence, it can be noted that the increase in errors when switching from a perfect model, to one built on-line, was two to four times, instead of six times. This lower increase tracking errors obtained for the real-world sequence is due to the OI not rotating in the real world sequence. When an object is rotated, errors become more apparent; thus, one would expect worse tracking accuracy under those circumstances.
Figure 5.9: The segmentation map (a) and IM (b) for Frame 10, Figure 5.8.

Figure 5.10: The generated model and reference frames at Frame #10 (a), #120 (b), and #230 (c), respectively, for the motion of the object shown in Figure 5.8.
5.3 An Example Application – Tracking for Dynamic Camera Reconfiguration Systems

The primary goal of this Thesis is to provide real-time 6-dof object tracking of \textit{a priori} unknown objects for autonomous systems. This experiment links the proposed object tracker with a dynamic camera-reconfiguration algorithm under development in our laboratory [105]. Geared toward autonomous surveillance, the dynamic reconfiguration algorithm moves cameras in order to maximise the visibility of an OI that may be \textit{a priori} unknown. This reconfiguration includes avoiding partial occlusions. Cameras can be mounted on linear and/or rotary stages. Such camera reconfiguration would be useful for a number of applications, including:

- Active face/object recognition (i.e., obtain the best frontal view of the subject/OI for most reliable recognition),
- Security operations; provide a surveillance operator with the maximum visibility of the chosen subject,
- Targeting narrow viewing angle cameras or sensors for biometric purposes (e.g., automatic iris recognition), and
- Object analysis; obtaining a set of views covering an OI's entire surface.
Figure 5.12: Positional (a) and orientational (b) tracking errors for the object in Figure 5.8, using the approximate OI model built on-line.
Figure 5.13: Positional (a) and orientational (b) tracking errors for the object in Figure 5.8, using an *a priori* provided model.
Such camera-reconfiguration systems are likely to encounter objects/subjects that they have not encountered before, i.e., \textit{a priori} unknown objects. In order to achieve optimal camera reconfiguration, the \textit{a priori} unknown OI’s future poses must be predictable, a task performed by the OI tracking system proposed in this Thesis.

\subsection*{5.3.1 The Camera-Reconfiguration Algorithm}

A broad overview of the reconfiguration algorithm is included here. Only the detail level required to understand the system is given. In-depth description of the algorithm is provided in [105]-[107].

At the core of the camera-reconfiguration system is the relative comparison of visibility of the OI to each of the cameras. The current poses of the cameras, obstacles, and the OI are combined into a visibility metric, \( V \). Of these input variables, the reconfiguration system has control over only the camera poses. \( V^j_i \) denotes the visibility for Camera \( i \) at Demand Instant \( j \) at time \( t_j \), as a function of \( P^j_{C_i} \), the pose of Camera \( i \) at Demand Instant \( j \). A demand instant is a point in time at which the active-vision system wishes to capture images. This can occur at a slower rate than the frame-rate of the object tracker. The visibility metric, \( V^j_i \), will be defined later; first, let us examine the overall algorithm. In an environment with \( n_{cam} \) cameras, \( n_{obs} \) obstacles, and with prediction performed over the time horizon ending at Demand Instant \( m \):

- For each Demand Instant, \( t_j, j \in [1..m] \), perform the following:
  - For each camera, \( C_i, i \in [1..n_{cam}] \), solve the following:
    * Given:
      \[ P^0_{C_i}, P^0, u^0, P^0_{obs_k}; k \in \{1..n_{obs}\} \]
    * Maximize:
      \[ Pr = g(V^l_i); l \in \{1..j\} \]
    * Subject to:
\[
\begin{align*}
\cdot & \quad P^1_{C_i} \in F_i \\
\cdot & \quad P^1_{C_i} \in A^1_{i} \\
\cdot & \quad V^1_l \geq V_{\text{min}}; l \in \{1..j\}
\end{align*}
\]

End of loop.

\[\text{• Repeat while: } t_{\text{proc}} < t_{\text{max}}.\]

Above \(P^j\) is the OI pose at Demand Instant \(j\), \(P^j_{\text{obs}_i}\) is the pose of Obstacle \(i\) at Demand Instant \(j\), \(n^j\) is the feature vector of the OI at Demand Instant \(j\), \(F_i\) is the discretized set of feasible camera poses for Camera \(i\), \(A^j_i\) is the discretized set of achievable poses for Camera \(j\) at Demand Instant \(j\), \(V_{\text{min}}\) is a threshold for minimum visibility, \(t_{\text{proc}}\) is the time spent processing data, and \(t_{\text{max}}\) is the maximum amount of processing time allowed before a final pose set must be chosen. The Feasible Poses, \(F_i\), are poses that the camera can achieve within the physical limits of the hardware (e.g., the length of a linear stage). The Achievable Poses, \(A_i\), on the other hand, are all poses that the camera can reach in time for the next Demand Instant. This is limited by the speed of the reconfiguration actuators. The performance function, \(Pr\), is a measure of success in achieving maximum visibility. This maximization problem seeks to first maximize the visibility of the OI in the immediate future, Demand Instant \(t_1\), for all cameras. If sufficient processing time remains, the system attempts to maximize expected visibility at \(t_1\) and \(t_2\), and then \(t_1, t_2, t_3\), and so on.

\textit{Camera Agents}

The reconfiguration system comprises multiple agents, each performing a single task. Every physical sensor (i.e., camera) has its own agent that is responsible for performing the optimization problem presented above. The visibility metric function is critical to the choice of poses:
where $W_{\text{area}}$, $W_{\text{dist}}$, and $W_{\text{angle}}$ are weighting constants for the three metrics of: visible OI area, zoom factor (size of OI in the camera view), and, angular distance from a view with the OI centered in the image. The vectors $t_1$ and $t_2$ are the two tangent points to an ellipse bounding the OI. These points lie on lines which pass through the focal point of the camera. The sum is over all distinct, visible portions of this line, $L_i$, (a total of $n_{\text{area}}$) – namely, the sum gives the total length of all the line segments together, from $t_1$ to $t_2$, that are visible (not outside the limits of the camera field of view, and not occluded by any obstacles). This sum is normalized by the maximum possible value, which is simply the total length of the line segment between the two tangent points (complete visibility). The vector $f$ is the focal point of the camera, and $c_{\text{obj}}$ is the optical centre of the OI. The centre term gives the distance from the focal point to the OI centre; essentially, it is a metric of the size of the OI in the final image. It is normalized by $d_{\text{max}}$, which is the maximum possible distance from the focal point of this camera to an object that is considered to be within the confines of the workspace. Finally, $\phi$ is the angle between the focal line of the camera and the line passing through both the camera rotation center, $c_{\text{cam}}$, and the OI center, $c_{\text{obj}}$. It is a measure of centering of the OI in the camera image, and is normalized by the maximum possible angle, $\phi_{\text{max}}$.

**Central-Planning Agent**

Each of the camera agents sends ranked pose lists to the central-planning agent. From these, the planner must determine which cameras are to be assigned to the OI for the current instant, and which will remain unassigned. A final set of poses for all cameras is, then, determined using a set of rules:

- Cameras with a visibility metric less than a minimum, $V_{\text{min}}$, at all poses, are unassigned.
The $M$ highest visibility cameras are assigned and all others unassigned.

- Unassigned cameras are asked to re-evaluate the visibility metric for additional demand instants. From these, the unassigned cameras are moved in anticipation of potentially optimal viewpoints at instants further into the future.

- For assigned cameras, a weighted sum of metrics is evaluated. Normally, this would include feedback from the active-vision algorithm about desired future views of the OI.

Using these rules, the system will look ahead and improve future visibility as well as current visibility, whenever possible.

**Pose Prediction Agent**

The purpose of the pose-prediction agent is to provide estimates of the OI’s future poses as well as the future poses of the obstacles. A number of options are available, such as the Kalman Filter (KF) [91] or derivatives thereof (e.g., [93] and [95]). The predictor tends to place more emphasis on near-future visibility, because prediction uncertainty increases as the prediction time-span increases.

**Referee Agent**

The referee agent enforces global rules that are not imposed directly by the optimization problem, or by the other agents [106]. Such rules are highly application specific, and vary in purpose – for example, specification of a minimum quality of service.

**5.3.2 Experimental Set-up**

The physical set-up is depicted in Figure 5.14. Cameras 1 to 3 were reconfigurable, and had a horizontal field-of-view of $24^\circ$. In contrast, the tracking cameras had a field of view of $34^\circ$. All
reconfigurable (OI recognition) cameras were mounted on rotary stages; Camera 1 was also mounted on a linear stage, enabling it to translate sideways.

Figure 5.14: The experimental set-up with all cameras at their default poses.

Five stationary cameras were allocated for the purposes of object tracking. Five cameras are necessary due to the limitations of the current DME algorithm. It is envisioned that improvements in DME technology could reduce this to two cameras in the future. Then, each of the mobile cameras could be replaced with a stereo pair for seamless multi-camera tracking across large areas. The five (tracking) cameras were simulated by mounting one SNC-RZ30N camera on a linear stage, and capturing five images from the different camera locations.

The experiment can be considered to be quasi-real-time due to the use of simulated cameras. Whilst the images were not captured in real-time, it is important to note that object tracking was carried out in real-time, i.e., 30 Hz. From this stream of poses, the camera-reconfiguration algorithm extracted the horizontal 2D position and orientation at every demand instant. These demand instances occur at a rate that is significantly less than 30 Hz due to the camera-reconfiguration system not needing to operate at such a high rate. Maintaining a high frame-rate for tracking minimises the motion between frames in order to achieve robust tracking.
The OI was a band-aid box that was mounted on a precision \( x-y-\theta \)-stage. Additionally, two stationary obstacles (vertical columns) were present. These caused partial occlusions of the OI that the camera-reconfiguration system had to avoid.

### 5.3.3 Results

All cameras were initially placed at their default poses, Figure 5.14. Frames 0 to 8 were used for initialization of the IFs. At Frame 9, the OI selector identified the band-aid box, an OI model was built, and tracking proceeded. The OI was tracked to within 16 mm positionally, and 6° orientationally.

Once tracking was initiated, camera reconfiguration was also started. Three of the resulting camera configurations are shown in Figure 5.15; the left column shows the result of camera reconfiguration for ideal tracking (i.e., ground-truth data was passed to the reconfiguration algorithm), whilst the right column shows the camera poses when the object tracker was used. Clearly, the camera-reconfiguration outcome is almost identical in both cases. On closer inspection, one can note that the cameras are directed at a slightly different point on the OI (the point at which the optical-axis lines converge in the figure). This is due to the slight difference in reference-frame position on the OI; the reference frame generated by the initial pose generator is close to, but not quite at, the centre of the OI. The tracking error also contributes to the slight difference in the target point. However, with a maximum error of 16 mm, the tracked pose remains within the object itself.

Figure 5.16 shows Frame 153 as captured by each of the three reconfigurable (recognition) cameras for both ideal (left column) and actual (right column) tracking. In all images, the OI is fully visible and at, or close to, the centre of the image frame.

The visibility metric values for all cameras are plotted in Figure 5.17. Values for the tracked case are equal to, or slightly below those for the ideal case. Thus, the camera-reconfiguration system is operating successfully using the data provided by the proposed object tracking sys-
Figure 5.15: The camera configuration at Frame #45 (a) and (e), #81 (b) and (f), #117 (c) and (g), and #153 (d) and (h) for the ideal (a) to (d), and tracked (e) to (h) cases.
Figure 5.16: The OI at Frame 153, as viewed by Camera #1 (a) and (b), Camera #2 (c) and (d), and Camera #3 (e) and (f) for the ideal (a), (c), and (e), and tracked (b), (d), and (f) cases.
tem. Moreover, the OI was selected automatically and no OI model was required \textit{a priori}, making the combined surveillance system fully autonomous.

\section*{5.4 Summary}

All of the individual modules presented in the previous chapters were combined herein to produce a single object-tracking system. This system was tested on both synthetic and real-world image sequences containing OIs whose models are not known \textit{a priori}. In these tests, the desired OI was automatically selected, a model built, and, the OI was tracked using the generated model. Additionally, the tracking system was used within a target application: dynamic camera reconfiguration.

The goal of dynamic camera reconfiguration is to move a set of cameras mounted on linear and/or rotary stages such that the overall visibility of an OI is optimal. In order to achieve this, the OI must be tracked. In the experiments, the proposed object-tracking system provided the tracking data. The visibility metric values obtained using the tracking system were close to, and sometimes even equal to, those obtained with ideal tracking (i.e., ground-truth pose data). Moreover, tracking was completely automatic, from OI selection, through to modelling and tracking.

In all of the above experiments, the object-tracking system autonomously selected the desired OI, and proceeded to track it in real-time. It was successful in tracking the 6-dof pose of OIs in the presence of background clutter, and even partial occlusions. However, there are still a few limitations. Currently, only two different types of IFs have been implemented. Also, the modeller has difficulty with occlusions and current DME algorithms are rather inaccurate. Additionally, there is the issue of reference frame drift when the modeller periodically rebuilds the OI model. However, real-time tracking of the 6-dof pose of an \textit{a priori} unknown object in the presence of clutter has been achieved. Existing other object trackers can only operate if a model is provided \textit{a priori}. With further research, the limitations above could be over-
Figure 5.17: The visibility metric for Cameras #1 (a), #2 (b), and #3 (c).
come. In the meantime, the current algorithm could be used for short-term tracking, or for initial tracking of known objects. Initial tracking would use the approximate modeller until an object-recognition algorithm had recognized the OI and loaded a precise model; this would eliminate the risk of losing track of the OI during the recognition phase.
Chapter 6

Conclusions and Recommendations

This Thesis has presented a complete methodology for tracking the 6-dof pose of an \textit{a priori} unknown Object of Interest (OI) in real-time. The proposed system performs all required operations from OI selection and modelling through to actual tracking. Use of this system within target applications such as convoying (Section 4.5) and camera reconfiguration for surveillance (Section 5.3) has been demonstrated.

6.1 Summary and Conclusions

The primary purpose of the proposed tracking methodology is to enable autonomous systems to sense objects within their environment and to interact with them. This places strict real-time constraints on the tracking system that require innovative methods in order to still achieve the desired goals. For example, pre-existing modelling techniques have been, typically, non-real-time; thus, the development of a new approach (i.e., a novel real-time modeller) was required.

As was stated previously, this Thesis divided the tracking task broadly into three sub-objectives: OI selection, modelling, and tracking. The first two relate to initializing and supporting the object tracker, whilst the final sub-objective is the object-tracking process itself.
Each corresponding module developed in this research contributes toward the overall goal of real-time tracking for autonomous systems. OI selection addresses selecting the OI in the first place as well as performing basic initialization. The modeller enables the proposed methodology to track \textit{a priori} unknown OIs by generating the required model on-line. Everything is brought together by the tracker which uses the provided OI model in order to track the selected OI in real-time.

\section*{6.1.1 OI Selection}

OI selection is a rarely-considered, but nevertheless important, problem. An object tracker would be of little use if it could not find, or choose, an OI to track. The proposed methodology offers a novel and extensible approach to this problem. Interest Filters (IFs) are connected serially in an IF bank. Each IF highlights regions of interest based on a set of criteria (e.g., colour, velocity, position, patterns, etc.). The filter bank progressively reduces the total area of interest resulting in a combined interest Map (IM) that highlights regions that match the desired OI. This can be customized to suit any application via selection of IFs, and by tuning the parameters of individual IFs (e.g., select the colours for a colour IF).

Motion segmentation is an efficient technique for separating an image into individual rigid objects that are moving independently. It is used herein primarily because the IM alone is not enough for segmentation. Next, a set of interest properties (e.g., mean interest level) for each segment (or region) is calculated, and the region with the greatest mean interest level that matches set criteria is chosen as the OI. If the required criteria are not met, it is assumed that there is no OI, and the OI-selection algorithm advances to the next frame in the motion sequence.

What constitutes an OI is application specific. Hence, identifying an OI is also application specific. Rather than solve the OI-selection problem for a single application (or a few applications), the OI-selection methodology outlined above provides a common framework that can
be adapted to a wide range of applications with varying requirements. It enables whatever knowledge of the OI’s properties are available to be leveraged, in order to find and select OIs without tying the system into a specific set of criteria.

The final task for the OI selection system is to generate an “initial pose,” or initial reference-frame. This is the reference frame that will be tracked over time. It is also the reference frame around which the 3D model will be built. Initial-pose generation is based on the visible part of the object. The reference frame is placed within the OI, close to the visible (2D) centre.

### 6.1.2 Real-Time Modelling

All 6-dof pose trackers require a 3D model. Previously, this requirement limited tracking to objects for which a 3D model was available *a priori*. This Thesis introduces a novel real-time modeller that rapidly builds a model of an *a priori* unknown object, on-line. Thus, the full 6-dof pose of unknown objects can now be tracked too, opening up a wider range of potential applications for such tracking methodologies.

The first step in building a model is Depth-Map Extraction (DME). This extracts the 3D structure of a scene. A possible alternative would be to use feature-point based 3D structure extraction. However, this would defeat the advantage of visual-model based object tracking, which is the ability to track objects without well-defined feature-points. Thus, objects with smooth curved surfaces can be modelled and, therefore, tracked as well.

After DME, a 3D tessellated mesh of triangles is built. The proposed method adds vertices to the mesh in a grid-like fashion. Vertices are only added in “textured” regions of the input image since depth cannot be obtained in smooth (textureless) regions. Feature points are also used as vertices since, typically, these points are located in shape-defining points on the object (e.g., sharp corners). Additional points are inserted in a grid-like fashion in order to ensure a minimum density of vertices, and obtain a reasonable approximation of the OI’s surface. This scheme provides a quick approximate surface without having to resort to computationally
costly techniques for selecting optimal vertex locations.

A critical part of the modelling process is to separate (or segment) the OI from the rest of the image. Otherwise, the model would contain all visible objects. The OI selector already has a system for segmentation, the IF bank. This is reused by the modeller. A thresholded version of the IM is used as a mask for the modeller, thus, no vertices are placed in regions that are marked as not of-interest. As a result, the model covers the OI, and not the rest of the scene. The motion segmentation map is also used as a mask. When a model is being regenerated, the bounding-box of the previous model is used as a mask instead of the motion segmentation map, which is no longer available. These techniques serve to further eliminate non-OI regions from the model.

3D geometry is only part of the OI model; the surface markings, or texture, is equally important for tracking. It is the optical flow of this texture that is used to track the OI. Rather than perform complex decisions on how the texture should be mapped onto the OI, projective texture mapping is employed. This method projects a texture onto a surface in the same manner that a physical projector would form an image on a wall (or screen). In doing so, the input image itself is used as the texture, and the only calculations involved are referring the projector’s pose to the OI’s reference frame. The projector’s parameters match the parameters of the camera that captured the image.

The entire modelling algorithm is fast enough to operate on-line. In this research, it operated at a rate of several fps. What differentiates this modeller from the other few real-time modellers in the literature is that it also performs segmentation. This is essential for tracking a single OI in a cluttered environment.

The cameras only see one side of the OI. Therefore, the model must be updated periodically as it translates and rotates. The current implementation periodically replaces the model with a new one, thus, enabling continual tracking regardless of how far the OI rotates. One current limitation is the issue of reference-frame drift. Because the model is periodically rebuilt around
an imprecise reference frame, it may drift relative to the point on the OI at which it should be anchored. Further research into real-time modelling will be required in order to overcome this limitation. Nevertheless, this Thesis presents a first step in the development of 6-dof real-time object tracking of \textit{a priori} unknown objects. To date, no existing literature has been found that addresses the issue of simultaneously modelling and tracking an OI in the presence of clutter.

6.1.3 Tracking

The ultimate goal of this Thesis is real-time 6-dof tracking. Once the OI has been selected, and a model built, the object-tracking algorithm takes over. In experiments, the proposed object-tracker achieved rates of up to 100 fps, which is far above the 30 fps of standard video. Thus, it is a true real-time object tracker.

The proposed tracking methodology begins by predicting the pose of the OI in the current frame. Next, a visual 3D model (obtained from the modeller, or built beforehand) is projected onto the image at its predicted pose. Optical flow is performed between the projected (virtual) image and the input (real) image. This estimates the offset (i.e., motion) between the predicted pose and the actual pose. Combining the predicted pose with the motion, the OI’s current pose is estimated.

One of the contributions to object tracking is exploiting 3D graphics hardware. In particular, the \textit{z}-buffer contains depth information for the projected image. This enables 6-dof optical flow to be calculated directly from a single image. The graphics-card’s GPU is also used for image processing, thus, freeing up the CPU for other tasks.

Various extensions to the core tracking algorithm have been developed in order to improve robustness and speed. For example, multi-scale optical flow was adapted from classical optical-flow algorithms for 6-dof tracking. This enlarges the allowable range of disparity between the projected OI and actual OI from 1-2 pixels to several pixels (approximately 8 pixels in experiments). Thus, robustness to large and erratic motions was increased.
Another contribution is the Local Illumination Normalization Filter (LINF). This filter normalizes the image such that the local mean colour is mid-gray. In doing so, the effect of lighting variations and shadows is greatly reduced, enabling tracking under a wide range of lighting conditions. LINF-based filtering does not eliminate lighting variations completely. Instead, it greatly reduces the effects of such lighting variations within the restrictions of real-time operation.

A major contribution to object tracking, and optical flow in general, is colour-gradient-redundancy based acceleration of optical flow. Colour images offer extra information, but also have three times the volume of data. This requires three times the memory bandwidth and three times more computation than gray-scale image sequences. Other attempts to reduce the data volume, whilst still taking advantage of colour, managed to reduce the data by about 33%. Colour-gradient redundancy, on the other hand, offers a threefold reduction in data, effectively reducing the memory and bandwidth requirements to the same level as for gray-scale image sequences. The proposed algorithm is based on the following observation: due to the nature of visible matter, image gradients of the individual colour channels are generally aligned. Because the gradients of the red, green, and blue colour channels are generally aligned, the gradients (and difference values) for all the colour channels can be merged. This resulted in a fivefold increase in tracking performance instead of threefold. Most likely, this is caused due to the reduced data fitting into CPU caches, resulting in faster access.

The final contribution to object tracking is a methodology for coping with partial occlusions. Lack of robustness to partial occlusions is a common criticism for visual-model/template based techniques. In this Thesis, the partial-occlusion problem is solved by comparing the image gradients of the projected and input images, and discarding pixels that have disparities exceeding set thresholds. The algorithm rejects pixels that violate the brightness-constancy assumption that underpins optical flow. This check ensures that optical flow is only calculated between the projected OI model and the OI, and not between the model and other objects.
6.2 Recommendations

As was mentioned above, this Thesis represents a large first step toward development of 6-dof real-time object tracking of a priori unknown objects. The desired goals have been met. However, further research would be required in order to bring this technology into real-world autonomous systems. This section covers areas in which such research could be performed that builds upon this Thesis.

6.2.1 OI Selection

The most obvious starting point for building upon the existing OI selection framework is the development of more IFs. At present, two IFs have been implemented; one filter is based on colour, the other highlights regions of motion. For example, an IF based on colour co-occurrence histograms [108] could provide greater control over specifying an OI’s visual attributes. Likewise, IFs that filter based on 3D position and velocity could result in a more flexible OI-selection system.

An IF of particular value would be one that filters based on known motion of the OI. This would be used during model-updating events. Given that the motion of the OI is known, an IF that highlights parts of the input image that is moving under the same transformation could be very effective at separating the OI from the rest of the scene.

Another area that could benefit from research is motion segmentation. The existing motion segmentor is 2D. Whilst this is adequate for some tasks, it is less effective for motion outside the 2D plane. A real-time 3D motion segmentor could alleviate this problem, and increase the range of situations that the OI selector could handle.
6.2.2 Real-Time Modelling

DME is currently the module that is restricting the object-tracker’s robustness the most. Real-time DME algorithms are still fairly new, and there is significant research still to be performed in this field. At present, such algorithms output DMs containing significant levels of noise, particularly pits and spikes. Occlusion boundaries also cause significant challenges, and large errors at such boundaries are often present. Additionally, it is currently difficult to obtain accurate DMs for highly specular objects. Any improvement in real-time DME methodologies would result in more accurate models, and hence, more accurate tracking.

Earlier, it was mentioned that periodically replacing the model with a new one opened up the possibility of reference-frame drift. An incremental modeller could alleviate this problem whilst also providing a progressively more accurate model over time. Little literature exists on incremental modelling; even less on real-time incremental modelling. In particular, no literature has been found, to date, that covers incremental modelling of a single OI in a cluttered environment. Thus, there is an opportunity for significant research in this relatively unexplored field. Development of an incremental modeller is essential for advancement of the object-tracking system proposed in this Thesis.

The current modeller uses a tessellated 3D surface with a texture map as surface. This is a common modelling scheme for 3D graphics and visualization. However, other modelling schemes do exist. For example, if an accurate DM could be obtained, a DM/texture combination could prove to be useful. Such a model would be available immediately, without requiring significant computation in order to construct a mesh.

One of the limitations of the current modeller is that it can only model objects with convex 2D silhouettes. Objects with more irregular boundaries could result in parts of the background being included in the model in concave regions (e.g., between a subject’s arm and main torso). This is due to Delaunay Triangulation assuming that the 2D shape is convex. A method of detecting and removing such regions, or avoiding them completely, would increase the range
of objects that can be accurately tracked.

The final issue with modelling is coping with occlusions. The proposed object tracker is able to maintain accurate tracking in the presence of partial occlusions. In contrast, the modeller performs almost no checking for occlusions, and hence, could include occluded regions in a model. This was beyond the scope of this Thesis, but does need to be addressed.

6.2.3 Tracking

There are several steps that should be taken before moving the proposed tracking methodology into the *real world*. The most important item that needs to be developed, is detection of tracker failure. Given that the target application is for robotic and autonomous systems, it is essential that the overall control system knows when tracking has failed, and is able to respond appropriately. This requirement will become more important as robots move from the industrial realm of factories into the domestic world. Consumer robots such as robotic vacuum cleaners are only the first in what is expected to be a wave of consumer robots. The vehicle convoying application simulated in this Thesis, for example, could face catastrophic consequences if the following vehicle continued to respond to tracking data after a tracking failure. Detection of tracking failure appears to be almost completely unexplored at present, possibly because tracking itself is such a challenge. Indeed, it was beyond the scope of this Thesis too. However, this would an be essential feature for real-world robotics.

The proposed algorithm features multi-scale techniques for robustness to large motions. There is, however, a limit as to how many hierarchical levels can be used. Eventually, the down-scaled image will be so small that performing 6-dof optical flow would be unfeasible. Use of 2D optical flow at at the lowest scales might increase robustness to large motions even further. In particular, 2D correlation-based optical flow is worth investigating.

Another factor that could limit the maximum OI speed that can be tracked is motion blur. This can be minimised to a certain extent by using a high shutter rate on the input cameras.
Where this is not possible, motion blur would increase the disparity between the OI model and the input images. There are methods of simulating motion blur in real-time graphics (e.g., [109]). Such techniques could form part of a scheme in order to track in the presence of motion blur. Motion blur actually provides queues as to the direction of motion, which could be exploited if such motion blur could be interpreted adequately.

This Thesis proposed a novel method for removing occluded regions from motion calculations in real-time. Experiments showed this to be highly effective. However, objects with similar markings could still confuse the algorithm. Thus, combining the proposed occlusion-rejection algorithm with other techniques is worth examining. Possibly, block-based robust-estimation, when combined with the existing algorithm, could increase occlusion robustness further, without losing real-time performance.
References


[82] Z. Zhang, J. Li, and X. Wei, “Robust computation of optical flow field with large motion,” in *IEEE Int. Conf. on Signal Processing*, vol. 1, Beijing, China, September 2004, pp. 893–896.


[85] J. Barron and R. Klette, “Quantitative color optical flow,” in *Int. Conf. on Pattern Recognition*, vol. 4, Quebec City, Canada, August 2002, pp. 251–255.


Principal Component Analysis

Principal Component Analysis (PCA) is a technique that captures defining properties of a data set whilst lowering the dimensionality. Lower dimensionality allows analysis to be performed more quickly. It can be used in image/pattern recognition (e.g., face recognition, [110]), and also has applications in image compression. This appendix approaches PCA from an image-recognition viewpoint.

PCA is essentially a vector-space transformation. This transformation is constructed such that the maximum variability between samples (e.g., images) in a data set is concentrated along the first basis vector. Each following basis-vector contains less and less information. For example, a data-set of images of faces could be transformed such that the first 5-10 basis vectors of the transformed space are sufficient in order to distinguish between these images, or other similar images.

The appropriate transformation is found via Singular-Value Decomposition (SVD) [111]. By its very nature, SVD generates basis vectors ranked in order of decreasing singular values. This is the exact property that is required by PCA.
A.1 Procedure

One of the best methods to understand PCA is to use it for a target application. This section outlines an example procedure for using PCA in recognition tasks. This Thesis used PCA for pose recognition of a single object. Such recognition requires two steps, building a database, and the recognition task itself. The process is similar for face and object recognition too.

A.1.1 Database Training

When using PCA for object recognition (or orientation recognition as in this Thesis), one starts with a database of training images. All images in the database are vectorized; namely, instead of representing the image as a matrix of pixels, the pixel and colour values are represented as a single column-vector of numbers. All the vectorized images are assembled into a single $\frac{m}{n}$ matrix, $A$. Thus, each column in $A$ corresponds to a single image/data-entry out of $n$ images of $m$ elements.

Next, the mean image, $\bar{a}$, is subtracted from all columns in $A$. If $a_i$ is Column $i$ of $A$, then

$$\bar{a} = \frac{1}{n} \sum_{i=1}^{n} a_i,$$

(A.1)

and

$$B = A - \bar{a}1^T,$$

(A.2)

where $1$ is an $m \times 1$ vector of ones. $B$ is the mean-adjusted version of matrix $A$. Subtracting the mean removes any DC offset from the data, thus, ensuring that the principal components span the variability in the data, not the DC offset. Matrix $B$ is decomposed via SVD:

$$B^T = UDV,$$

(A.3)

where $U$ is a unitary $m \times m$ matrix, $D$ is an $m \times m$ diagonal matrix of singular values, and $V$ is an $m \times n$ matrix containing orthonormal basis vectors for the singular values. PCA can
also be formulated using a covariance matrix and eigenvalue decomposition; only the SVD formulation is presented here.

At this stage, the number of dimensions of the transformed space can be chosen, i.e., how many principal components to use. The singular values indicate how much variability occurs along each basis-vector. They are arranged in order of decreasing magnitude. Thus, a cut-off can be chosen below which the basis vectors are discarded. This is achieved by forming a matrix, $V'$, which comprises the first $l$ columns of $V$.

The final step in database training is to build the database itself. Denoting $D$ as the database matrix, the database is calculated as follows:

$$ D^T = B^T V'. $$

(A.4)

Each column in $D$ corresponds to a single image in the database.

### A.1.2 Recognition

For recognition, an input image, $I$, is vectorized, producing a vector, $i$. This must be transformed into the principal component space:

$$ i'^T = i^T V'. $$

(A.5)

With the image transformed to the same vector space as the database, the image can now be compared to the images in the database. One common method is to measure the Euclidean distance between the images in the database (i.e., the columns of $D$ which are denoted as $d_j$) and the input image:

$$ d_j = |i' - d_j|, 1 \leq j \leq l. $$

(A.6)

Any database image with a Euclidean distance, $d_j$, less than a set threshold is a potential match to the input image. Out of the images which meet this criterion, the Database Image $j$ with the smallest Euclidean distance to $i'$ (i.e., $\min(d_j)$) is the closest match. In this Thesis, each
image in the database corresponds to a specific orientation of the Object of Interest (OI). Thus, the recognized pose would be the pose associated with the closest match. Similarly, in face recognition, the human subject corresponding to the closest match would be the recognized individual.
Appendix B

Harris Feature Points

In 1988, Harris and Stephens published a feature-point detector that has become a major component for many computer-vision algorithms [65]. Feature points are located by analysing the orientations of image gradients within a local area. If there are little or no gradients, then, there are no features present; if the gradients are predominantly all in the same direction, then, there is an edge present, but not necessarily a feature point; if, on the other hand, the gradients within a local area point in multiple directions, there is a feature point. Thus, the Harris feature-point detector can actually detect lines and edges too. However, feature points are usually of greater interest since they can be localized in two dimensions, whereas lines and edges cannot.

Harris feature-point detection starts with finding the image gradients, i.e., $\nabla_x I$ and $\nabla_y I$. From these gradients, a $2 \times 2$ matrix, $M$ is formed:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix}, \quad (B.1)$$

where $A$, $B$, and $C$ are images defined as:

$$A = (\nabla_x I)^2 * w, \quad (B.2)$$

$$B = (\nabla_y I)^2 * w, \text{ and} \quad (B.3)$$
\[ C = (\nabla_x I \cdot \nabla_y I) \ast w, \]  

(B.4)

where the symbol \( \ast \) denotes a convolution operation. The function \( w(u, v) \) is a Gaussian filter kernel:

\[ w(u, v) = e^{-\frac{u^2 + v^2}{2\sigma^2}}, \]  

(B.5)

where \( u \) and \( v \) are 2D coordinates.

It is important to realize that the matrix \( M \) exists for each pixel, namely, there is an “image” of \( M \). However, it is easier to examine feature-point detection on a per-pixel (i.e., single matrix) basis. Henceforth, it will be assumed that \( M \) is for a single pixel.

The eigenvalues of \( M \) relate to the major and minor gradient directions. If \( \lambda_1 \) and \( \lambda_2 \) (the eigenvalues of \( M \)) are both large, that pixel can be labelled as a feature-point. Pixels with one large eigenvalue correspond to edges, whilst pixels with zero (or almost zero) eigenvalues are featureless.

Calculating the eigenvalues would be computationally expensive, fortunately this is not necessary. The following two relationships can be used in order to identify feature-points without resorting to full eigenvalue/eigenvector decomposition. First, a few definitions:

\[ tr(M) = A + B = \lambda_1 + \lambda_2, \text{ and} \]

(B.6)

\[ det(M) = AB - C^2 = \lambda_1 \lambda_2, \]  

(B.7)

where \( tr(M) \) and \( det(M) \) are the trace and determinant of \( M \), respectively. Given the trace and determinant of \( M \), the following formula can be used to identify feature-points:

\[ R = det(M) - k \cdot tr(M)^2, \]  

(B.8)

where \( k \) is a constant (often set to 0.04). \( R \) is positive in corner regions (i.e., feature points), negative for edges, and close to zero in featureless areas. Thus, pixels whose value for \( R \) exceed a set threshold are considered to be feature points.
There is one issue with the above scheme; instead of generating one feature point per corner, it is likely to generate a cluster of feature points. Applying a local non-maximal suppression filter will remove these extra points, leaving a single feature point per corner/feature.
Appendix C

Delaunay Triangulation

A common problem in computational geometry is generating “good” triangulations from a set of points. Developed by Boris Delaunay [112], Delaunay triangulation generates a mesh of triangles for a given set of 2D points, that avoids thin “sliver” triangles. This is particularly useful in situations such as generating height maps, or single-sided models such as is generated by the modeller presented in this Thesis. Avoiding thin triangles ensures that the resulting mesh is the most accurate representation of the underlying surface for the given set of points. The largest disparity between a mesh and a 3D surface would likely occur at the mid-point along a triangle’s longest side. Thus, minimising the number of thin triangles minimises the lengths of the sides, thus, also minimising the disparities.

At the heart of Delaunay triangulation is the concept of a triangle’s circum-circle. A circum-circle is defined as the circle that contains all vertices of a triangle on its perimeter. This is shown in Figure C.1.

Delaunay triangulation can be summarized by a single rule: a triangle’s circum-circle should not enclose points/vertices belonging to a different triangle. It is acceptable for a point/vertex to be located on another triangle’s circum-circle perimeter, but such vertices may not be inside the circum-circle. If the triangles in a mesh comply with this rule, the mesh is a
Delaunay triangulation.

There are various different algorithms for performing Delaunay triangulation. One common technique with these algorithms is to perform triangle edge flipping; given two adjacent triangles, the common edge is “flipped” to the other two vertices, Figure C.2. Triangle edges can be flipped, until all triangles obey the circum-circle rule. Details on algorithms can be found in computational geometry literature, e.g., [66].