DRAFT COPY ONLY

Segmenting and Tracking Visual Entities

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"

der Georg-August-Universität Göttingen

im Promotionsprogramm
der Georg-August University School of Science (GAUSS)

vorgelegt von
Jérémie Papon aus Summit, NJ, USA

Göttingen, 2014

DRAFT COPY ONLY

Segmenting and Tracking Visual Entities

A DISSERTATION PRESENTED BY

JÉRÉMIE PAPON

TO THE FACULTY OF NATURAL SCIENCES AND MATHEMATICS

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SUBJECT OF
COMPUTER SCIENCE

GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN
GÖTTINGEN, GERMANY
MARCH 2014

Referentin/Referent: Prof. Dr. Florentin Wörgötter
Koreferentin/Koreferent: Prof. Dr. Dieter Hogrefe
Tag der mündlichen Prüfung: TBD

The canonical version of this document is the electronic copy maintained in the Github repository by the author. At this time, it is maintained at:

https://github.com/jpapon/papon_thesis/thesis.pdf

Much of the code created as a result of the research in this thesis is freely available under a BSD license as part of the Point Cloud Library:

http://www.pointclouds.org/

The code for the Oculus Vision System (see Appendix A) created as part of this thesis is freely available under GPLv3:

https://launchpad.net/oculus

For other usage, contact jpapon@gmail.com.

Segmenting and Tracking Visual Entities

Abstract

Abstract text.

Contents

The work described in this thesis has appeared in the following publications:

Papon, J.; Kulvicius, T.; Aksoy, E.; Wörgötter, F., "Point Cloud Video Object Segmentation using a Persistent Supervoxel World-Model," *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Nov. 2013.

Papon, J.; Abramov, A.; Schoeler, M.; Wörgötter, F., "Voxel Cloud Connectivity Segmentation - Supervoxels for Point Clouds," *Computer Vision and Pattern Recognition (CVPR) 2013*, June 2013.

Papon, J.; Abramov, A.; Wörgötter, F., "Occlusion Handling in Video Segmentation via Predictive Feedback," *European Conference on Computer Vision (ECCV) 2012*. Workshops and Demonstrations, Lecture Notes in Computer Science Volume 7585, 2012, pp 233-242.

Papon, J.; Abramov, A.; Aksoy, E.; Wörgötter, F., "A modular system architecture for online parallel vision pipelines," *Applications of Computer Vision (WACV) 2012*, pp.361-368, Jan. 2012.

Stein, S.; Schoeler, M.; Papon, J.; Wörgötter, F., "Object Partitioning using Local Convexity," *Computer Vision and Pattern Recognition (CVPR) 2013*, June 2014..

Stein, S.; Wörgötter, F.; Schoeler, M.; Papon, J.; Kulvicius, T., "Convexity Based Object Partitioning For Robot Applications," *Robotics and Automation (ICRA), 2014 IEEE/RSJ International Conference on*, June. 2014.

The research leading to this thesis was supported with funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, Xperience and grant agreement no. 269959, Intellact.

List of Figures

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

Some Quote.

Quoteauthor Lastname

1 Introduction

The adult human brain is able to process a bewilderingly large amount of visual data with ease. Nevertheless, at some point before the eyes first open, the brain is a blank slate, upon which the concepts of color, motion, objects, and so forth must be inscribed. At some point we learn to track objects as coherent wholes as they move, and we learn to break our world into meaningful and useful parts as we interact with it. In this work we argue that the two concepts are inexorably linked, that visual tracking creates the objects we see around us. We propose that without motion, and the ability to track motions in a coherent way, the notion of distinct objects is a meaningless one.

1.1 Problem Definition and Motivation

This chapter presents the motivation behind our work by first discussing the three underlying problems that will be addressed throughout. To summarize into a brief statement, we would say our overall goal is the decomposition of video into semantically meaningful entities. That is, to move from the base *low-level* pixels which compose an image to *high-level* structures which are more representative of how a human would understand the scene. In the following sections we will introduce the three constituent underlying sub-problems: Image Segmentation, Multi-Target Tracking (MTT), and Video Object Segmentation (VOS).

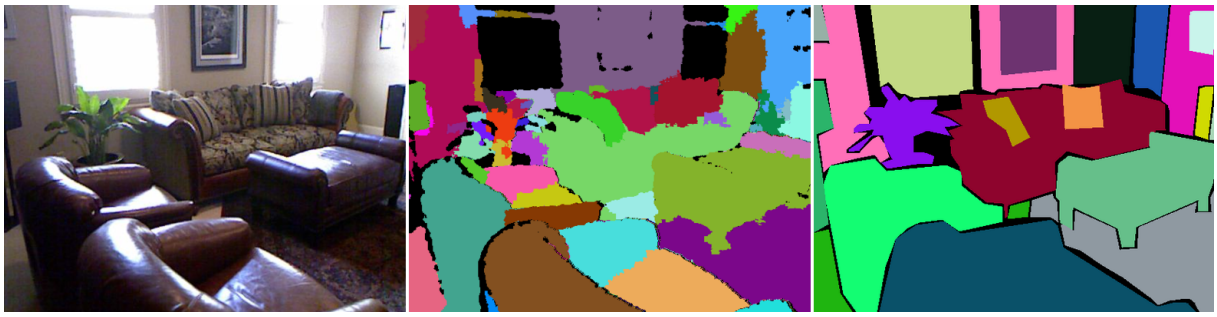


Figure 1.1.1: Example of Segmentation and Ground Truth. From left to right we have an image, a segmentation from a computer vision algorithm, and a human-annotated ground truth labeling. Here labels are represented by different colors. We shall use this convention throughout the rest of this work.

1.1.1 The Image Segmentation Problem

Image segmentation aims to divide the set of pixels in an image into a number of distinct subsets, where each subset represents some semantically meaningful entity (e.g., an object - see Fig. **??**). This is a notoriously tricky business, particularly because it is something that humans are able to do intuitively. This ease with which humans can segment visual scenes is highly deceptive; Marvin Minsky, one of the pioneers of Artificial Intelligence (AI), famously assigned one of his students "computer vision" as a summer undergraduate project in 1966. Nearly half a century later, despite the extensive effort to solve it, image segmentation, the first step on the long road to complete "computer vision", remains an unsolved problem.

Some citations about image segmentation by humans (from CVPR paper)

The reason for this is two-fold: firstly, there are many technical or physical challenges associated with properly dividing an image into separate objects. Among these, shadows, occlusions, reflections, imaging noise and so forth can all greatly affect the results of image segmentation. Consider, for instance, a partial occlusion as in Fig. **??**. A human can easily identify that the parts on either side of the occluding object belong to same object. This is accomplished using what we shall refer to as *high-level* knowledge throughout this work - in this case, knowledge of the complete nature of an object.

This leads us to the second challenge in image segmentation, which is that, generally speaking, there is no "correct" solution to the problem. A perfect labeling for one application might be useless in another. This is even more of a problem when we are discussing segmentation separate from any application, as is the case with standard image segmentation benchmarks (which are use to quantify algorithm performance). These benchmarks use ground-truth image labels (manually created by humans) to score the output of different algorithms. Unfortunately, the correctness of different labellings is highly subjective, and hand-drawn labels from people can differ radically.

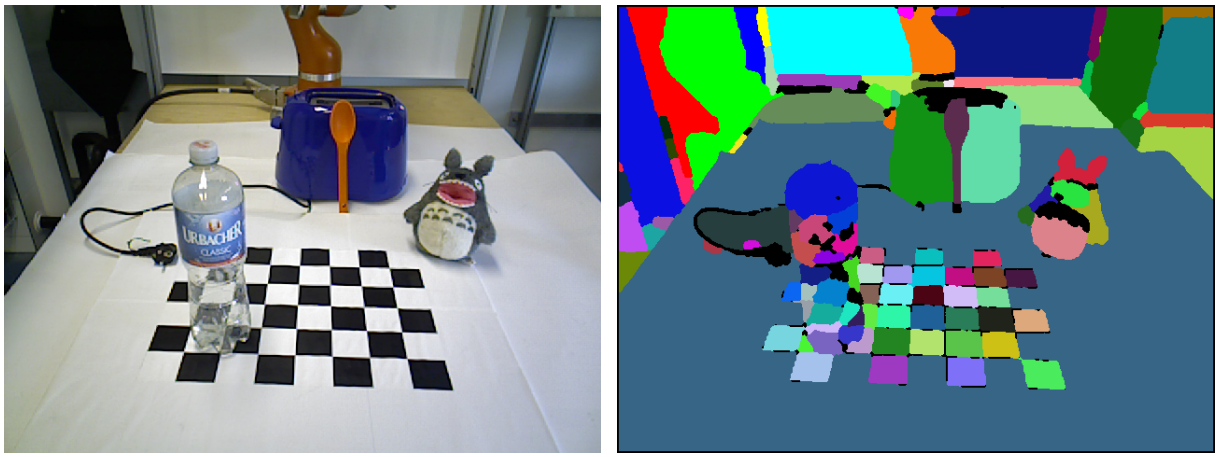


Figure 1.1.2: Technical Difficulties of Segmentation. Here we see some of the myriad of technical difficulties present in color-based segmentation, such as transparent objects (the water bottle), partial occlusions (the toaster), objects with strong color differences (the little monster), and similarities in color (the bottle cap to the table).

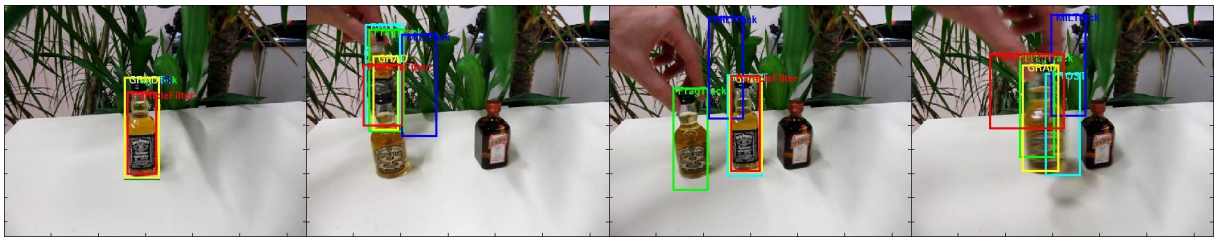


Figure 1.1.3: Example of Visual Tracking. This shows outputs from various trackers in a standard video tracking benchmark. Some of the difficulties of tracking can be seen- in particular complex backgrounds, motion blur, partial occlusions (second frame from left) and even full occlusions (right-most frame).

1.1.2 The Tracking Problem

Multi-target visual tracking (MTVT) is a crucial challenge for many computer vision applications such as visual surveillance, action recognition, and robotic imitation learning. In many such functions, visual tracking serves as the precursor to all further high-level inference, making robust tracking fundamental to the success of a large variety of intelligent systems. The general goal of MTVT is to sequentially estimate the number of targets and their corresponding states (e.g., position, velocity). This is accomplished by associating noisy observations over time with the entities which produced them. Tracking links targets into sequential states (known as *tracks*), typically using some a-priori detection model, such as tracking a human face in video based on a facial model. In general, this is done by estimating some state for each tracked object (e.g. a bounding box around a person's face) given an observation (e.g., the output of a face detector).

The primary challenge in MTVT is the data association problem - deciding which tracked target a particular observation belongs to. Confounding this is the additional null possibility, where an observation belongs to none of the tracked targets. Closer examination reveals that the difficulties are related to those of image segmentation, simply extended into the temporal dimension. In particular, interacting

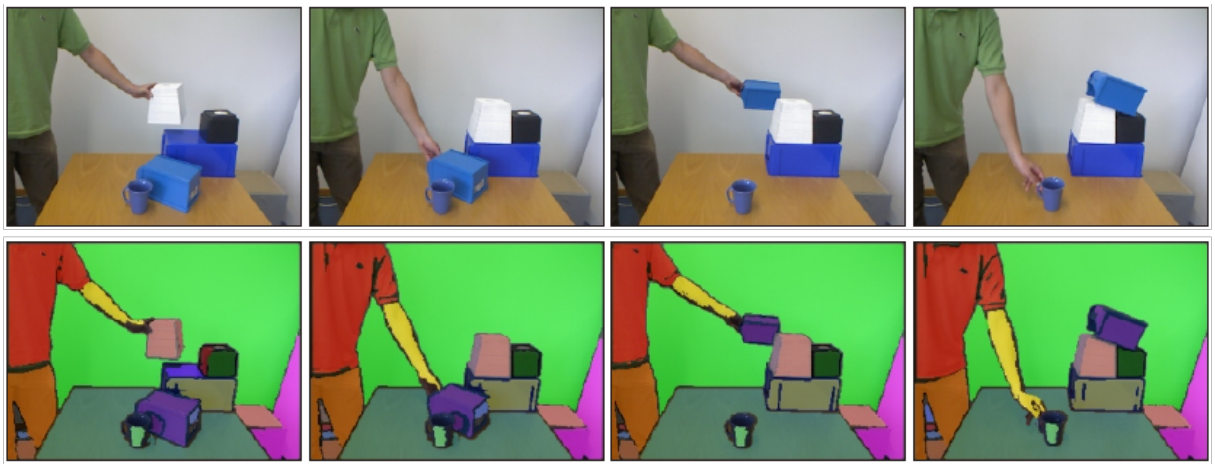


Figure 1.1.4: Example of Video Object Segmentation - from [3]. This shows the goal of VOS - to extract a dense labeling (labels here are shown as distinct colors) for every frame, maintaining temporal consistency of objects. For many applications it is of vital importance to make the labeling consistent from frame to frame, that is, to maintain object identities.

and occluded targets are especially challenging.

1.1.3 Video Object Segmentation - Segmentation In Sequential Frames

VOS attempts to cluster pixels of video frames into segments which are both spatially and temporally coherent. While similar to MTVT, VOS goes a step beyond localizing tracked objects, in that it makes an association decision for each observed pixel; in addition to estimating overall state, it must re-estimate spatial extent every frame. Additionally, VOS has the additional consideration that target appearance models are unknown a-priori, and are subject to arbitrary changes over time.

One interesting aspect of video segmentation is that it has the potential to be more accurate than single image segmentation, as it can take advantage of the temporal coherence of objects to infer information about the objects in a scene. Unfortunately, the addition of the temporal domain brings along new challenges as well; for instance that pixels which should be grouped across time may not be continuously visible, as in the case of partial or full occlusions. Additionally, the added dimension increases the computational complexity of the problem, making accurate segmentation a costly procedure. Temporal information also increases the exposure of the algorithm to noise, as each image frame is a separate noisy measurement. This adds a large amount of uncertainty to the problem, since measured values (i.e., of color) for an object can show significant variation over time.

1.2 State of the Art

1.2.1 Segmentation and Superpixels

Segmentation of scenes into objects remains one of the most challenging topics of computer vision despite decades of research. To address this, recent methods often use hierarchies which create a rank

order that build bottom-up from small localized superpixels to large-scale regions [**? ? ?**]. As an alternative, researchers have also pursued strictly top-down approaches. Such methods began with coarse segmentations using multiscale sliding window detectors [**?**], later progressing to finer grained segmentations and detections based on object parts [**? ?**]. These two avenues of research led naturally to methods which *combine* bottom-up hierarchy building with top-down object- and part-detectors [**? ? ?**]. While these approaches have yielded quite good results even on complex, varied data sets, they have lost much of the generality of learning-free approaches. In general the most powerful methods to-date use trained classifiers for segmentation [**? ?**]. This means they cannot be applied to arbitrary unknown scenes without being retrained, requiring the acquisition of a new data-set tailored to each test environment.

1.2.2 Multi-Target Visual Tracking

MTVT is a well-established field, which goes back over thirty years [19]. In this work we use Sequential Bayesian Filtering (SBF), a technique which recursively estimates the time-changing posterior distribution of target states given all previous observations. We use a Sequential Monte Carlo method known as Particle Filtering to approximate the posterior, an approach which was first introduced to the vision community by Isard and Blake [26] and has been the subject of much subsequent research extending it to multiple targets [24, 47, 48]. For details on how one approximates SBF using a Particle Filter we refer the reader to Appendix B.

Initially, researchers pursued two avenues of research in extending a single Particle Filter to multiple targets. The first was to represent all targets jointly in a single particle filter by assigning individual particles to particular labels [46]. This means that, for a given total number of particles, there will be fewer for each individual target - resulting in reduced accuracy. The second approach was to add additional dimensions to the state space for each additional target [43]. Unfortunately, this approach quickly increases the dimensionality of the state space, resulting in a need for a very high number of particles for the filter to remain accurate. In both approaches, the computational complexity increases exponentially as targets are added (for constant level of accuracy), as both seek to find a joint solution. As a consequence of this, it is beneficial to use a separate particle filter for each target. One way of doing this is to add factors to the observation and/or process models of the filters which explicitly model occlusions and interactions between targets [29**?**].

Another approach which has generated much interest is to use the output of detectors as the basis for tracking. Known as *tracking-by-detection*, these methods typically use independent particle filters to maintain tracks [10, 13], and place the focus of the problem on the data association step, wherein detections are assigned to targets. While there are several classical approaches for solving this association problem from Sonar and Radar research [20, 38], a greedy approach is typically sufficient given a good association scoring function [10, 49].

1.2.3 Video Object Segmentation

There are many existing VOS methods, which can be classified based on three parameters; whether they are on- or off-line, whether they are dense or sparse, and whether or not they are supervised. We can reduce the comparison-space of related work by comparing only with algorithms which have the same three parameters as this work - on-line processing (the algorithm may only use past data), dense segmentation (every pixel is assigned to a spatio-temporal cluster), and unsupervised operation. Four state-of-the-art segmentation algorithms meet these requirements: Mean-shift video segmentation (MSVS) [36], Multiple hypothesis video segmentation (MHVS) from superpixel flows [45], Propagation, validation, and aggregation (PVA) of a preceding graph [30], and Matching images under unstable segmentations [23]. Of these methods, none are able to handle full occlusions; in fact only MHVS considers occlusions, and it is only able to handle partial occlusions for a few frames, and does not consider full occlusions. Even state of the art off-line methods such as that of Brendel and Todorovic [11] only handle partial occlusions, claiming that "complete occlusions ... require higher-level reasoning".

In [35] Papadakis and Bugeau use a dynamical model to guide successive segmentations, along with an energy function minimized using graph cuts to solve the label association problem. They formally model visible and occluded regions of tracked objects, tracking them as distinct parts. While they do consider occlusions, they do not maintain a world model, and as such their methodology must fail under complete occlusions. Additionally, they formally model visible and occluded parts of the tracked objects, and so the method does not scale well with an increasing number of objects, and thus is better suited to extracting the silhouettes of a few objects than performing a full segmentation. Other methods, such as [1], are severely limited in that they require pre- computed models which are calibrated to a ground plane in order to resolve occlusions. Recent work in MTVT [34] successfully tracks multiple objects using a segmentation and association approach and adaptive 3D appearance models, but is limited by the need to align model point clouds to the observed data every frame, as well as the need for a ground plane. This precludes it from handling occlusions, as once a target is no longer observed, its track must be terminated.

Should we add some lines about applications, or is it obvious?

1.3 Outline and Contributions

This thesis is organized as follows: First, in Chapter 2 we present a hybrid VOS / MTT technique for 2D data. We describe the method, briefly describe the segmentation algorithm it relies on, and present results on a tracking benchmark. In Chapter 3 we present the concept of a persistent 3D voxel world model. We begin by briefly introducing some core concepts of acquisition and representation of 3D pointcloud data, then present Voxel Cloud Connectivity Segmentation (VCCS), a method for extracting a graph of 3D voxel patches from pointcloud data. We then discuss how to add pointclouds sequentially to the model in a way that allows voxels to persist through occlusions. In Chapter 4 we describe a method for using particle filters to track multiple rigid objects in pointcloud video data and present

results of tracking performance on artificial data. In Chapter 5 we combine the methods described in prior Chapters into a system which can produce full video segmentation of pointcloud videos. We show that the system is highly robust to occlusions and noisy data, and give results for the application of semantic understanding of human actions. Finally, in Chapter 6 we discuss the findings and experimental results of this work, possible future work, and conclude.

Each of the Chapters in this thesis contain novel contributions to the field:

Add citations for these to relevant papers

- **Chapter 2** contains the 2D segmentation through relaxation technique.
- **Chapter 3** contains the Supervoxel clustering method, as well as the scheme for maintain voxels in an Octree through occlusions.
- **Chapter 4** has the scheme for accelerating 3D particle filter tracking through stratified sampling of the model-space.
- **Chapter 5** has the techniques used to generate full segmentations based upon the results from multiple independent trackers. Additionally, Appendix A presents the Oculus Vision System, a computer vision system created over the course of the research for this thesis.

Some Quote.

Quoteauthor Lastname

2

Video Segmentation by Relaxation of Tracked Masks

In the beginning, 3D data, especially video data, was not readily available. As such, researchers were forced to make due with strictly 2D video, which is inherently ambiguous in many situations. In particular, partial and full occlusions are particularly vexing problems in 2D video - not least because understanding of 2D video is so easy for humans, yet so difficult to interpret algorithmically. Indeed, knowledge of object permanence, that is, the understanding of how to correctly interpret occlusions, is something that humans acquire very early on in their lives [**?**], but has yet to be successfully implemented in a fully automated VOS system. Even after decades of research, state-of-the-art methods still have trouble correctly resolving partial occlusions, and typically fail completely after even the briefest of complete occlusions.

In this Chapter, we shall present our attempts towards resolving the object permanence problem with 2D data, as well as advance color-based VOS in general. In particular, we seek to overcome two of the main drawbacks of the color-based video segmentation method developed by Abramov et al. [2] (and indeed, of color-based VOS in general). The first of these is the correct tracking of objects through partial and full occlusions, which we proposed to solve using a layering of deformable object masks that are allowed to interact and compete for "ownership" of pixels. The second is to allow for object identities to be maintained through sudden and/or fast movements - something that was not possible due to the core assumptions of the algorithm. To correct for this, we tracked the masks with a set of particle filters, a class of Bayesian predictive filters which are well known for their ability to handle difficult trajectories [24, 47, 48].

The underlying principle guiding the proposed algorithm is to use predictions from Bayesian filter-

ing to inform segmentation of higher-level temporal object correspondences. It is well known that sequential Bayesian estimation methods perform well in difficult tracking scenarios [18], and, under the Markov assumption, are computationally less demanding than video segmentation techniques such as MHVS [45], which consider many prior frames. Particle filtering is one such method which has been shown to approximate the optimal tracking solution well, even in complex multi-target scenarios with strong nonlinearities [24, 47, 48].

2.1 Overview of the Algorithm

Before proceeding to discuss elements in detail, we shall first give a brief overview of the algorithm (depicted in Figure 2.1.1). We begin by performing an initial segmentation (using any method) on the first frame $\mathbf{F}_{t_0}$ to generate an initial set of labels $\mathbf{S}_{t_0}$. An initial set of particles is generated for each label, and color histogram features are computed for each particle (as in [37]). Thus each object k at initial time t_0 is specified by a set of N_k particles $\mathbf{X}_{t_0}^{k,1:N_k}$, each of which contains a representation of the object, specified by a pixel existence map $\mathbf{M}$, a reference color histogram $\hat{q}$, a position shift vector $\mathbf{p}_{\mathbf{t_0}}$, and a velocity vector $\mathbf{v}_{\mathbf{t_0}}$.

The particles are then propagated in time independently, shifting their existence maps to new regions of the image. These shifted maps are used to generate measured color histograms from the next frame, which are evaluated to determine similarity to the object's reference histogram. The set of particles for each object is then combined to create an overall object pixel likelihood map. The pixel likelihood maps for all objects are then further combined with each other to create a label association likelihood map. In this likelihood map, each pixel is a Probability Distribution Function (PDF) specifying the probability that the original image pixel was generated from an observation of a particular object.

The label association likelihood map is then sampled using a per-pixel selection procedure (as described in Section 2.3.2) to generate a candidate label image, $\tilde{\mathbf{S}}_{t_0+1}$. This candidate image is used as the initialization for the Metropolis-Hastings algorithm with annealing of Abramov et al. [2], which updates the labels iteratively until an equilibrium segmented state is reached. The segmentation result, $\mathbf{S}_{t_0+1}$ is subsequently used to update the set of particles via three mechanisms; birth, decay, and repopulation. Birth is used for new labels in the segmentation output, and consists of initializing a new set of particles. Decay occurs when a label is not found in the segmentation output, and consists of killing a number of the particles of the missing label. The most commonly occurring mechanism, repopulation, occurs for all previously existing object labels which are found. Repopulation rejuvenates the set of particles for an object by replacing a number of particles in the set with new particles based on the relaxed segmentation result.

2.2 Tracking Object Masks

We shall now describe each of the parts of the algorithm given above in further detail, beginning with a description of how we track object masks using particle filters. First we will briefly review the basic

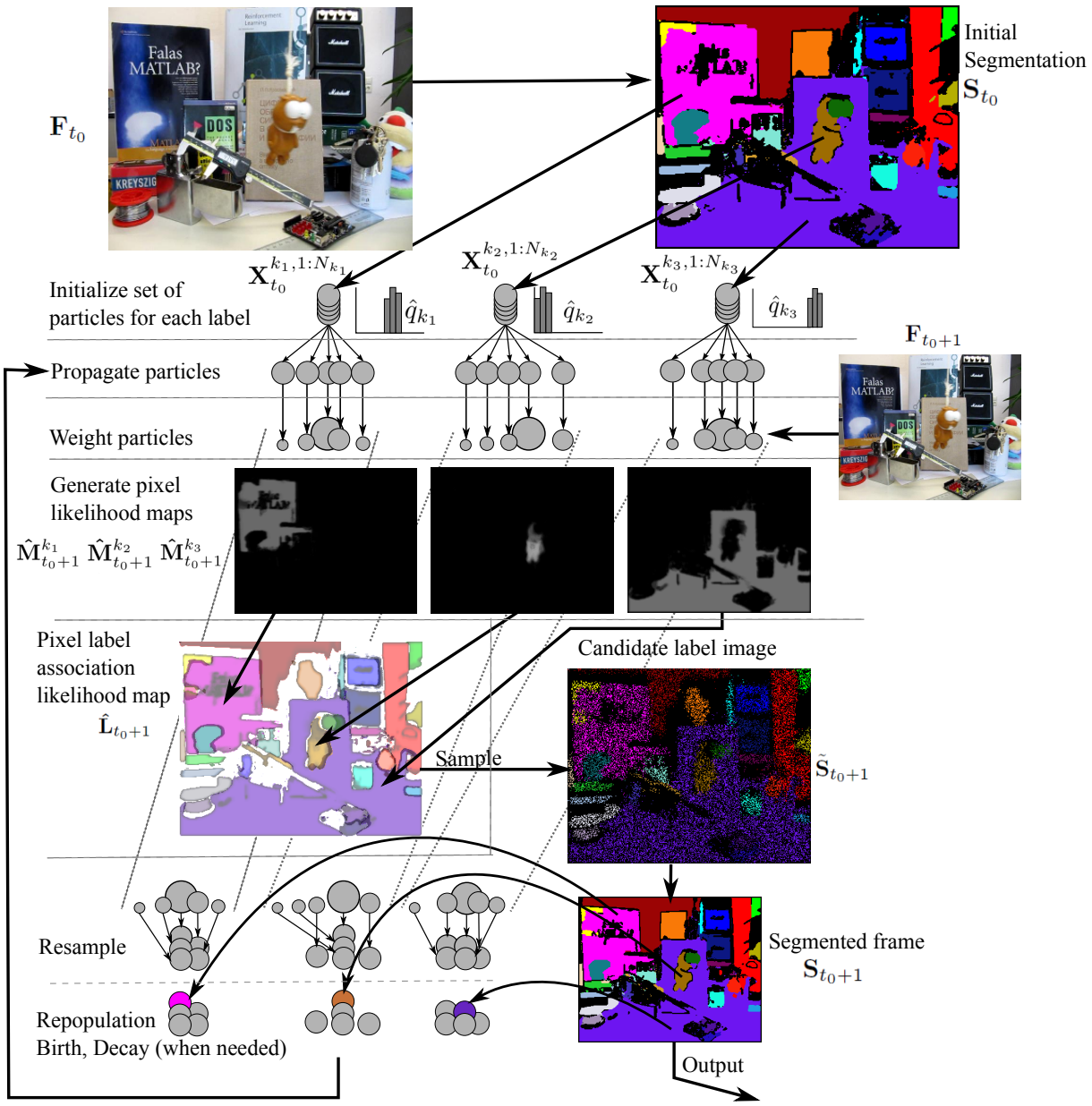


Figure 2.1.1: Flow of algorithm for one time step, shown for three labels (k_1, k_2, and k_3). For a description, see Section 2.1.

principles of sequential Bayesian estimation and particle filtering, and then show how they can be used to predict pixel-level label associations in order to seed a segmentation algorithm.

2.2.1 Sequential Bayesian Estimation

Sequential Bayesian estimation uses a state space representation, in which a state vector $\mathbf{x}_t$ describes the hidden state of a dynamic system. Bayesian estimation attempts to determine the posterior distribution of the state given all prior observations $\mathbf{z}$, i.e., $p(\mathbf{x}_t|\mathbf{z}_{1:t})$. This is accomplished using a two step recursion which first generates a hypothesis of the current state conditioned on the previous state and then performs a Bayes update using the new observation. These steps are known as the prediction and filtering steps, respectively.

The prediction step estimates the current distribution given all prior observations, or

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})d\mathbf{x}_{t-1}. \tag{2.1}$$

This prediction requires the specification of a stochastic *dynamic model*

$$\mathbf{x}_t = f_t(\mathbf{x}_{t-1}, \mathbf{v}_t), \tag{2.2}$$

where $\mathbf{v}_t$ is the process noise, which characterizes the state transition density $p(\mathbf{x}_t|\mathbf{x}_{t-1})$. The dynamic model takes advantage of knowledge of the system to generate reliable predictions of how the state evolves.

The filtering step uses Bayes rule to update the predicted density by conditioning it on the new observation $\mathbf{z}_t$:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{1:t-1})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})}. \tag{2.3}$$

This requires the specification of an observation, or measurement, model

$$\mathbf{z}_t = h_t(\mathbf{x}_t, \mathbf{w}_t), \tag{2.4}$$

where $\mathbf{w}_t$ is the measurement noise, which characterizes the observation density $p(\mathbf{z}_t|\mathbf{x}_t)$. Once the filtered, or posterior distribution is determined, an estimate of the state can be made using a variety of techniques (e.g., maximum a-posteriori, mean-shift).

Dynamic Model

In our method, the state of a particle consists of four elements; the pixel existence map $\mathbf{M}$, a reference color histogram $\hat{q}$, a position shift vector $\mathbf{p}$, and a velocity vector $\mathbf{v}_t$. Of these, only the position shift and velocity evolve over time, so we adopt the state vector

$$\mathbf{x}_t = [p_x v_x p_y v_y]^T, \tag{2.5}$$

where (p_x, p_y) denotes the accumulated shift of the pixel existence map in the image plane, and (v_x, v_y) the map velocity in the image plane. Motion is modeled using a constant velocity model in discrete time with uniform sampling period T, giving the dynamic model

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{v}_t, \tag{2.6}$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

and noise $\mathbf{v}_t$ is assumed to be zero mean Gaussian with fixed covariance.

Measurement Model

In our method measurements are taken by calculating a color histogram, q_t for the region lying within the shifted pixel existence map $\mathbf{M}$. That is, for particle n of object k,

$$q_t^{k,n} = hist(\mathbf{F}_t \cap \mathbf{M}_t^{k,n}). \tag{2.8}$$

Color histograms are three dimensional, with 8 bins for each of the color components hue, saturation, and value. As in [37], a Gaussian density is used for the observation density $p(\mathbf{z}_t|\mathbf{x}_t)$, that is

$$p(\mathbf{z}_t|\mathbf{x}_t) = \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{\Delta(\hat{q}, q_t)^2}{2\sigma^2}, \tag{2.9}$$

where $\Delta(\hat{q}, q_t)$ is the Bhattacharyya distance (as proposed in [17]) between the reference histogram $\hat{q}$ for the particle and the measured histogram q_t for time t. The Bhattacharyya distance is a standard measure of similarity between discrete probability distributions, and is defined as

$$\Delta(\hat{q}, q_t) = \sqrt{1 - \sum \sqrt{\hat{q} q_t}}. \tag{2.10}$$

2.2.2 Parallel Particle Filters

Except in special cases (e.g., Kalman Filter), closed-form solutions to Equations (2.1) and (2.3) are not available. Particle Filters are a Monte-Carlo method designed to approximate the posterior distribution with a weighted set of random samples. There are many excellent descriptions of the mechanics of particle filtering available (such as [18]), so we shall avoid presenting them here, and proceed directly to presenting the details of our algorithm.

The predictive portion of the method uses multiple Sequential Importance Resampling (SIR) filters in parallel to track multiple targets (labels) simultaneously. At this stage in the algorithm targets are assumed independent and interaction between labels is therefore not considered (interaction is accounted

for later, as described in Section 2.3). Particles are first propagated using the constant velocity dynamics model, and their predicted existence maps $\tilde{\mathbf{M}}^{k,n}$ are used to generate a measured histogram, q_t. Particles are weighted based on (2.9), and then normalized as a set for each label k. Systematic resampling is used to prevent particle degeneracy, due to its speed and good empirical performance [18].

The resulting distributions from the weighting procedure are used to generate object pixel likelihood maps for each label,$\hat{\mathbf{M}}^k_{t+1}$, which are then combined into the label association likelihood map $\hat{\mathbf{L}}_t$, as described in Section 2.3. A realization of this likelihood map can then be relaxed to produce a final segmented output, $\mathbf{S}_t$.

2.2.3 Particle Birth, Repopulation, & Decay.

One key improvement of the proposed algorithm over prior particle filtering methods is its use of the segmentation result $\mathbf{S}_t$ to update the particle sets. This allows the creation of new targets, adaptation to changing target appearance, and gradual elimination of targets which are no longer observed. This is accomplished via three mechanisms, which we term, respectively, birth, repopulation, and decay.

Birth occurs when a label which has not existed previously is found in the segmentation output $\mathbf{S}_t$, or more formally $\{k \notin \mathbf{S}_{1:t-1}, k \in \mathbf{S}_t\}$. It consists of generating a set of particles $\mathbf{X}^k$ for the new label using $\mathbf{S}_t$ to initialize an existence map $\mathbf{M}^k_t$ and $\{\mathbf{F}_t \cap \mathbf{M}^k_t\}$ to calculate a reference color histogram $\hat{q^k_t}$.

Repopulation is a key component of the algorithm, as it allows the pixel likelihood map for an object, $\hat{\mathbf{M}}^k$, to adapt over time to the changing appearance of the object. Every iteration, all previously existing object labels which are found in $\mathbf{S}_t$ are repopulated by replacing some particles in the set with particles generated from $\mathbf{S}_t$ and $\mathbf{F}_t$. Particles are chosen for replacement using stratified sampling, at a rate specified by parameter λ_r. The repopulation mechanism gradually modifies the object "model" through the addition of particles which have an updated existence map and color histogram (coming from the segmentation result). We use the term model here loosely, since there is in actuality no explicit model for any of the objects - merely a pixel likelihood map generated at each time step from the objects constituent particles and the current image frame.

Stratified replacement and relatively low repopulation rates are used to help keep the influence of erroneous hypotheses to a minimum, but as with any adaptive method, they can occasionally lead the tracker astray. Replacement of particles, rather than updating of a central model, helps to reduce this problem, since a few erroneous particles will generally not completely derail the algorithm. Nevertheless, future work could investigate strategies that allow pruning of unlikely hypotheses without negatively affecting occlusion handling.

Decay occurs when a label is not found in the segmentation output, $k \notin \mathbf{S}_t$. Particles are selected from k using random sampling, at a rate determined by the decay rate λ_d, and are pruned; they are no longer considered when filtering k. This reduces the number of active particles for the label in the next iteration, N^k_{t+1}, by approximately $\lambda_d N^k_t$. If the number of active particles for a label falls below a certain threshold, N_{min}, then the set of particles for the label is deleted, and the object is no longer tracked. If a label which was being decayed is observed again, i.e., $\{k \notin \mathbf{S}_{t-1}, k \in \mathbf{S}_t\}$, then the label is revived by replacing particles which had been killed with new particles, which are initialized as in the repopulation

step.

2.3 Extracting a Dense Image Labeling

The middle portion of Figure 2.1.1 depicts how the candidate label image, $\tilde{\mathbf{S}}_t$, is generated. The candidate label image is a summary of the accumulated knowledge of the particle filters; it is a prediction of what the segmented scene should look like. That is to say, it is a pixel-wise realization of the label association likelihood map $\hat{\mathbf{L}}_t$, which is constructed by combining the object pixel likelihood maps (which approximate the posteriors of the particle sets). $\tilde{\mathbf{S}}_t$ is the seed of the segmentation kernel, which uses pixel values from $\mathbf{F}_t$ to perform the relaxation process and generate a dense label image. In this section we will describe the process of generating the object pixel and label association likelihood maps, and then explain how the predictive loop allows occlusion handling without explicit object relationships or depth modeling.

2.3.1 Object Pixel Likelihood Maps.

The object pixel likelihood map for a particular object k is the weighted sum of the pixel existence maps of all of its labels,

$$\hat{\mathbf{M}}_t^k = \sum_{n=1}^{N_k} w_t^{k,n} \mathbf{M}^{k,n}. \tag{2.11}$$

Because the weights have been normalized, the pixel values in $\hat{\mathbf{M}}_t^k$ will be in the range $[0, 1]$. High pixel values will occur in regions which are present in the existence maps of highly weighted particles, or alternatively, are present in many particles with average weight.

2.3.2 Label Association Likelihood Map.

The label association likelihood map $\hat{\mathbf{L}}_t$ is a combination of all the object pixel likelihood maps, such that each pixel contains a discrete probability distribution giving the likelihood of the pixel belonging to a certain label. Additionally, a likelihood, p_0, for the pixel belonging to no label is inserted to allow pixels where no label has high likelihood to remain unlabeled in $\tilde{\mathbf{S}}_t$. More formally,

$$\hat{\mathbf{L}}_t = \bigcup_{n=1}^{K} \hat{\mathbf{M}}_t^n + p_0. \tag{2.12}$$

Each pixel of $\hat{\mathbf{L}}_t$ is then normalized, such that the sum of the discrete probabilities sums to one. The candidate label image can then be generated by taking a realization of $\hat{\mathbf{L}}_t$ to select pixel label values. Examples of the result of this process, $\tilde{\mathbf{S}}_t$, can be seen in Figures 2.1.1 and 2.6.1.

2.4 Occlusion Handling.

Occlusion relationships are handled naturally, since foreground objects will tend to have a strong peak in their weight distribution, corresponding to those particles which align properly with $\mathbf{F}_t$. Objects they occlude will have a flat particle weight distribution, since there will exist no shifted existence map which contains a color distribution which matches the reference histogram. This is due to the fact that the occluding objects and objects surrounding the occluded object have color distributions which differ from the occluded object. Let us assume foreground object j is contained by occluded object k, that is

$$\mathbf{M}_t^{j,n} \subset \mathbf{M}_t^{k,n}. \tag{2.13}$$

We also assume that the number of particles is sufficiently large such that

$$\exists\, \mathbf{M}_t^{j,n} \in \mathbf{M}_t^{j} : hist(\mathbf{F}_t \cap \mathbf{M}_t^{j,n}) \approx \hat{q}^{j,n}. \tag{2.14}$$

If $hist(\mathbf{F}_t \cap \mathbf{M}^{k,n}) \neq hist(\mathbf{F}_t \cap \mathbf{M}^{j,n})$, that is, the objects have different color distributions, then from (2.13) and (2.14), it follows that [1]

$$\nexists\, \mathbf{M}_t^{k,n} \in \mathbf{M}_t^{k} : hist(\mathbf{F}_t \cap \mathbf{M}_t^{k,n}) \approx \hat{q}^{k,n} \tag{2.15}$$

and therefore that

$$\begin{aligned} &min_{1:N_j}\{\Delta(\hat{q^{j,n}}, hist(\mathbf{F}_t \cap \mathbf{M}_t^{j,n}))\} < \\ &\quad min_{1:N_k}\{\Delta(\hat{q^{k,n}}, hist(\mathbf{F}_t \cap \mathbf{M}_t^{k,n}))\} \end{aligned} \tag{2.16}$$

and thus

$$max_{1:N_j}\{w_t^{j,n}\} > max_{1:N_k}\{w_t^{k,n}\}. \tag{2.17}$$

This means that in the label association likelihood map $\hat{\mathbf{L}}_t$, the occluding object will have a higher likelihood then the occluded. The candidate label image, $\tilde{\mathbf{S}}_t$ will therefore tend to favor occluding object labels, which will dominate the occluded object label during the segmentation relaxation process.

2.5 Segmentation using Superparamagnetic Clustering

To adjust the candidate label image $\tilde{\mathbf{S}}_t$ to the current frame $\mathbf{F}_t$, we use a real-time image segmentation algorithm based on superparamagnetic clustering of data [8]. The method of superparamagnetic clustering represents an input image being segmented by a Potts model, with pixel color vectors arranged on the sites of a two-dimensional (2D) lattice, where each pixel is featured by an additional variable, called a spin. This allows the segmentation problem to be formulated as a minimization problem which seeks

[1] This also assumes that the areas surrounding the occluded object also have different color distributions.

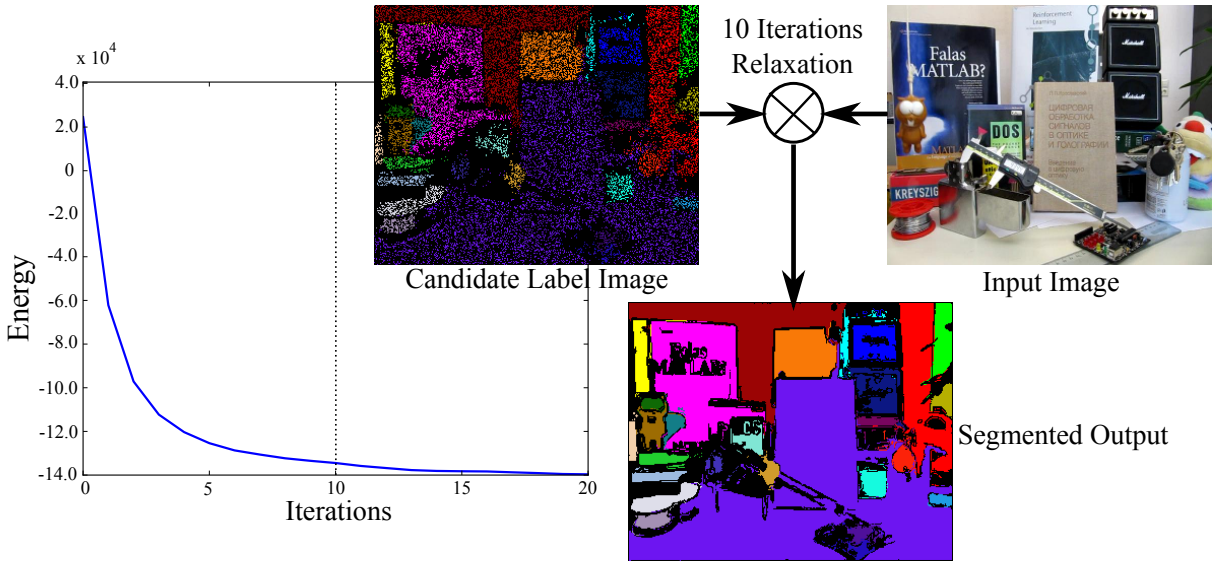


Figure 2.5.1: The relaxation process causes the energy of the label image to converge after few iterations (outcome after 10 iterations shown here). This results in efficient calculation of an accurate and temporally coherent segmentation.

to find the equilibrium states of the energy function in the superparamagnetic phase. In this equilibrium state regions of aligned spins coexist and correspond to a natural partition of the image data [8]. Since every found segment carries a spin variable which is unique within the whole image, the terms *spin* and *label* are equivalent here. The equilibrium states are found by the use of the highly parallel Metropolis algorithm with a simulated annealing, called *relaxation process*, implemented on a Graphics Processing Unit (GPU) [2]. In this work, the relaxation process adjusts the predicted candidate label image to the current frame.

Superparamagnetic clustering of data was chosen due to its flexibility in allowing the use of any initialization state; there are no particular requirements to the initial states of spin variables. The closer the initial states are to the equilibrium, the less time the Metropolis algorithm needs to converge. This property makes it possible to achieve temporal coherency in the segmentation of temporally adjacent frames by using the sparse label configuration taken from the candidate label image for the spin initialization of the current frame. A final (dense) segmentation result is obtained within a small number of Metropolis updates. Conventional segmentation methods do not generally have this property and cannot turn a sparse segmentation prediction into dense final segments which preserve temporal coherence. Moreover, since the method can directly use sparse predictions as the seed of the segmentation kernel, we can avoid the costly and error-prone block-matching procedure required to find label correspondences in other work, such as in Brendel and Todorovic [11] or Hedau et al. [23]. Figure 2.5.1 illustrates the relaxation process, and the convergence of energy after a small number of iterations.

2.6 Experimental Results

In order to evaluate performance, we compare our method to the state of the art on several challenging video tracking benchmark sequences which are available online[2]. It should be noted that, as opposed to the other tracking algorithms, we do not pre-select a region to track, and track fully deforming object masks (rather than a rectangle). Additionally, we employ no learned or a-priori specified models, use 50 particles per label, and only have two parameters; the repopulation and decay rates λ_r and λ_d, which were both held constant at 0.05 throughout testing. Results are compared to the PROST [41], MilTrack [7], FragTrack [5], and ORF [40] tracking algorithms. Further details concerning the parameters used for the above algorithms in the benchmarking can be found in [41].

We shall not evaluate the visual quality of segmentation results here for a couple of reasons. First, detailed evaluation of the visual quality of super-paramagnetic clustering has been presented in [2] in great detail. The visual quality of the segmentation results obtained from this work do not differ significantly from these results, with the exception of labels having continuity through occlusions. Secondly, it is directly acknowledged in other VOS work that the methods fail under partial [30, 36] or full [11, 45] occlusions. As such, comparing performance to other VOS methods is somewhat unreasonable. Rather, the better comparison is to the state of the art in tracking methods, which attempt to handle full and partial occlusions.

In order to compare with the other methods, we needed to output a tracking rectangle for each frame. To do this, once the sequence was segmented, we found the segment which corresponded to the object to track in the first frame, and then took the bounding-box which contained it in each frame as the tracking rectangle. This bounding-box was then compared to ground-truth using two measures; Euclidean distance and the PASCAL-challenge based score proposed in [41]. The latter compares the area of intersection of the ground truth and tracked box with the union of the same. When this is greater than 0.5, the object is considered successfully tracked. Table 2.6.1 gives our results, as well as the results for the other methods.

Testing showed that, when certain assumptions hold, our algorithm performs on par with, and in some cases outperforms, state of the art tracking algorithms. This is the case for the *liquor*, *lemming*, and *board* sequences. In the *lemming* sequence, frames of which are shown in Figure 2.6.1, our algorithm outperforms the other methods in cases of occlusion, especially when the tracked object is fully occluded. While other methods offer false positives and erroneous tracks, our method decays the label for the object and avoids proposing incorrect tracking solutions. In the *liquor* sequence, our algorithm adapts to the changing appearance (size, shape, and color) of the tracked bottle, allowing it to maintain performance on par with the other algorithms, in spite of the difficulties of segmenting transparent objects. In the *board* sequence, our method successfully adapts to the rapidly changing appearance of the tracked board as it rotates, allowing it to maintain an accurate track and outperform the other methods.

In addition to showing the strengths of our method, a weakness was also highlighted by the bench-

[2] http://www.GPU4Vision.org

Table 2.6.1: PROST dataset benchmark results. Top and bottom tables are average pixel error and PASCAL based scores, respectively

Sequence	PROST	MIL	Frag	ORF	HybridPF
Lemming	25.1	14.9	82.8	166.3	19.8
Box	13.0	104.6	57.4	145.4	114.1
Liquor	21.5	165.1	30.7	67.3	25.5
Board	39.0	51.2	90.1	154.5	30.9
Lemming	70.5	83.6	54.9	17.2	73.9
Box	90.6	24.5	61.4	28.3	7.5
Liquor	85.4	20.6	79.9	53.6	54.2
Board	75.0	67.9	67.9	10.0	71.4

mark sequences. The *box* sequence demonstrated the limitations of using unsupervised color-based segmentation to initialize the objects to track. In the sequence, the object to track contains strong color differences, which are segmented into different initial regions. As the object moves around, the particles for these regions are attracted to other objects it passes over which have similar color.

2.7 Discussion

In this chapter we presented a new method for performing on-line, dense, unsupervised video segmentation which uses tracking as the basis for segmentation. We have given results which show that the method is able to resolve occlusion relations between objects without explicitly modeling them, and can maintain consistent labels for objects, even when they leave and re-enter the field of view. Additionally, we have shown that the method is able to adapt to rapidly changing appearance of tracked objects, producing consistent segmentations over lengthy video sequences. A GPU version of the algorithm has been developed that can achieve near real-time levels 10 fps at 640x480 resolution on an i7 standard desktop. One sequence we tested showed the vulnerability of the method to situations where objects have similar color distributions. A way to resolve this is to consider additional features such as object geometry in the measurement function. To measure such geometric features requires the addition of depth to our observations; we shall investigate adding this extra dimension in the following chapters.

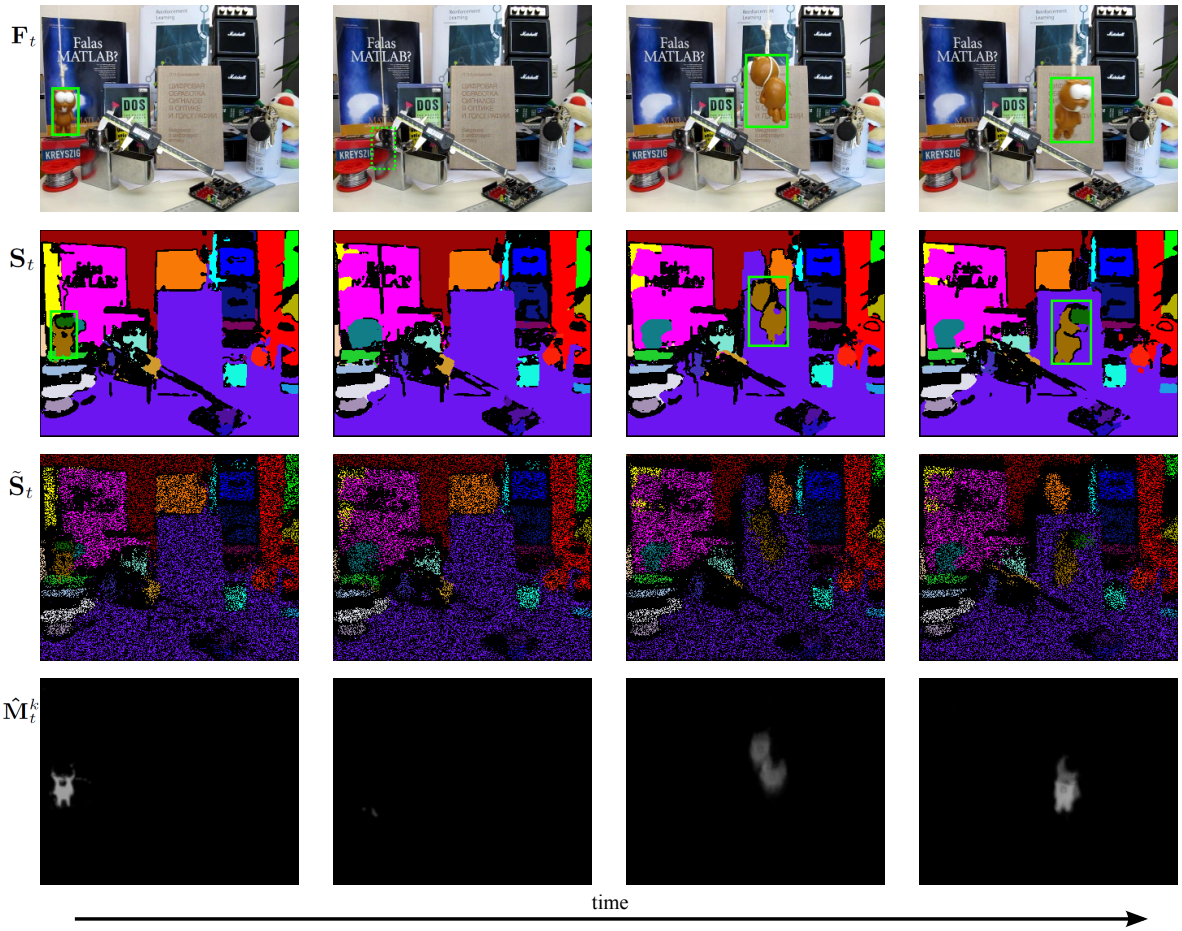


Figure 2.6.1: Output frames from the *lemming* sequence, in which a target is completely occluded (for $\sim$ 20 frames, second column) and changes significantly in appearance. The object which is tracked for comparison to other algorithms is highlighted with a green box. $\mathbf{F}_t$-Original frames from movie. $\mathbf{S}_t$-The output of the segmentation algorithm. $\tilde{\mathbf{S}}_t$-The candidate label image constructed by taking a random draw from $\hat{\mathbf{L}}_t$, the label association likelihood map. $\hat{\mathbf{M}}_t^k$-The overall object pixel likelihood map for the lemming label, created by combining the set of particles for the label. Intensity represents the sum of the normalized weights of the set of particles.

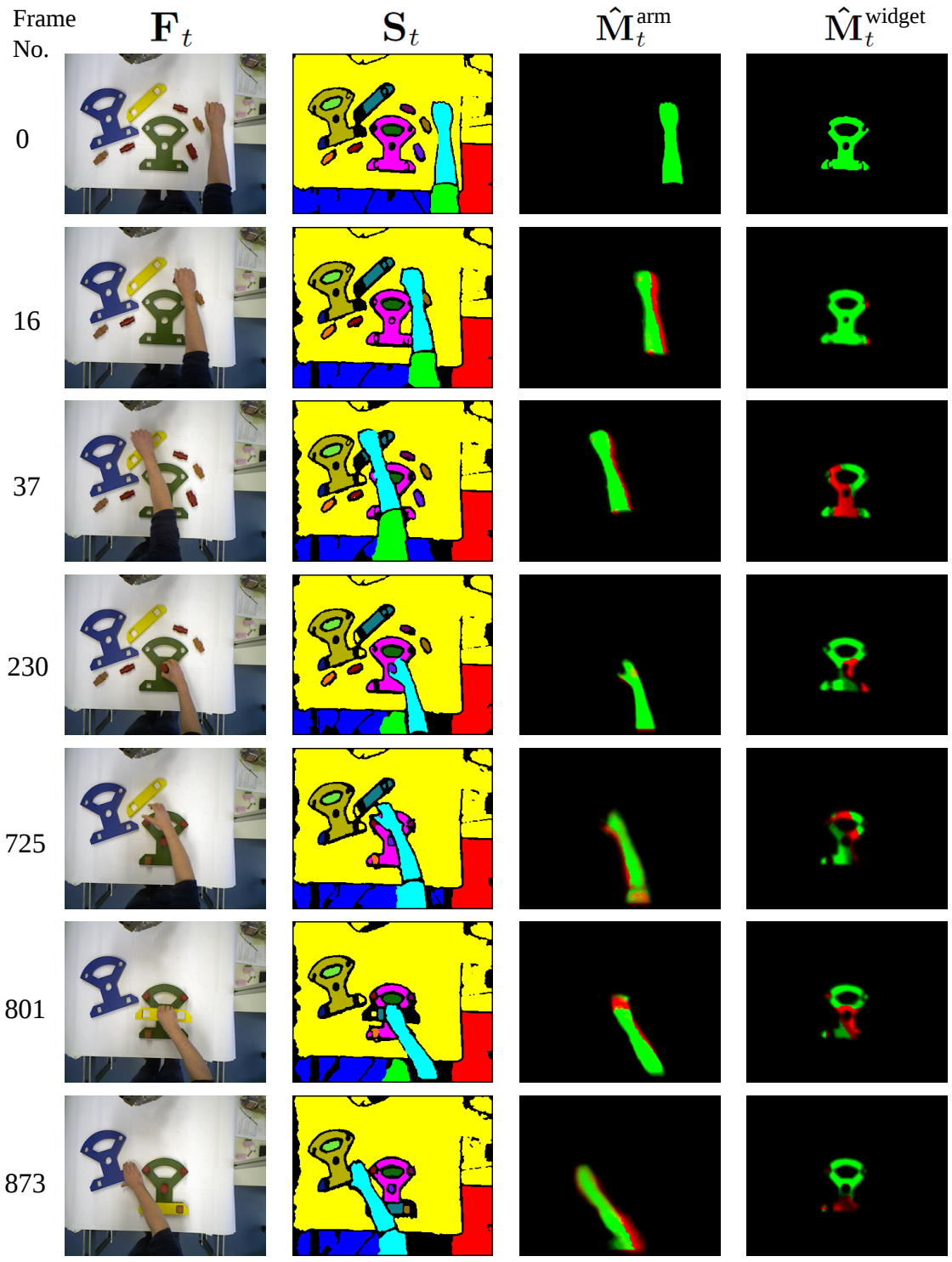


Figure 2.6.2: Results of segmentation on Cranfield Benchmark Sequence. Green masks show observed pixels, while red masks show occluded pixels which are believed to belong to the object.

Some Quote.

Quoteauthor Lastname

3

Patch-based Perceptual World Model

The widespread availability of cheap 3D sensors has had a profound impact on the world of computer vision. Researchers have found that they no longer need to attempt to find heuristic tricks which can create an artificial three dimensional interpretation from a two dimensional image. The advent of the Kinect (and other, related inexpensive RGB-D sensors) has allowed direct progression to high-level concepts and rules which the human mind uses when first learning to understand the real world - a completely different approach than trying to mimic the behavior of the mind when it is adapting those rules (learned from a life-time of 3D stereo data) in order to interpret some new 2D image.

In this chapter we shall present our work in creating a full 3D artificial world model which can be used for higher level semantic understanding of both single frames and video. The model we present begins with point clouds, relying on the general framework set up in the Point Cloud Library [1] (which we have both made use-of and contributed-to extensively as part of this work). Point clouds are a useful way of representing the data obtained from RGB-D sensors. The pixel coordinates and depth value coming from the RGB-D pair can be transformed into an (x, y, z) point, and the RGB information can be attached to this point.

[1] http://www.pointclouds.org/

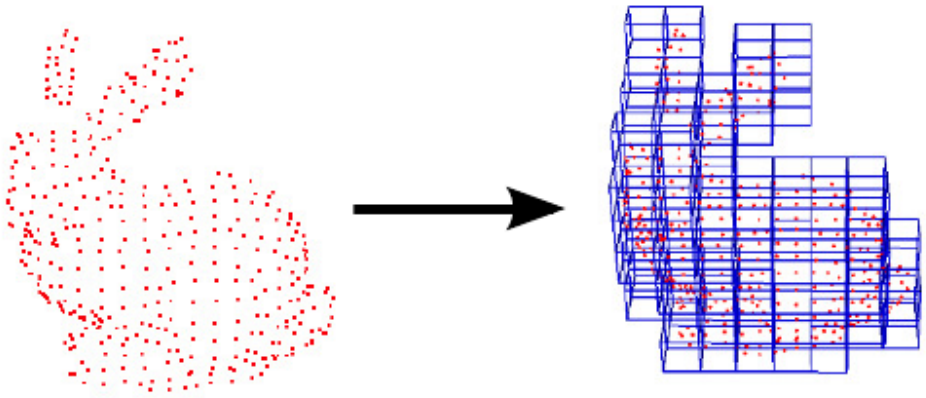


Figure 3.1.1: Illustration of Voxelization. On the left we have a point cloud of the "Stanford Bunny". This cloud is inserted into the voxel grid shown on the right, where all points falling within on grid unit, or voxel, are combined. From [**?**]

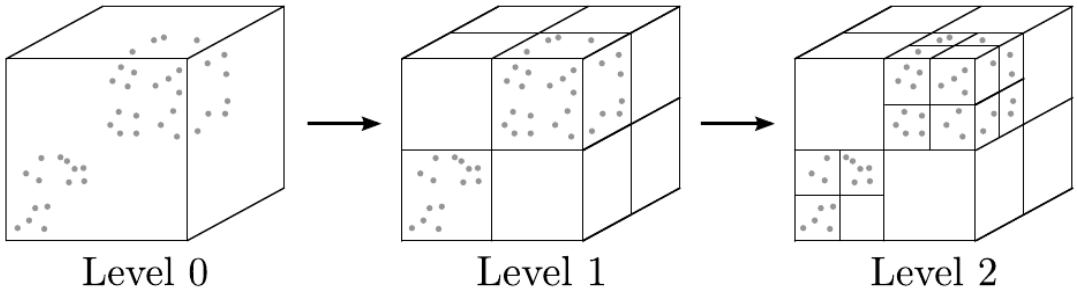


Figure 3.1.2: Use of an octree for voxelization. The points are grouped into voxels by recursively subdividing the bounding box into its eight constituent octants. This recursion terminates when the box size has edge length equal to the voxel leaf size R_{voxel}.

3.1 Voxelization

The resolution of a standard RGB-D camera such as the Kinect is 640x480 pixels, yielding about 300,000 points per frame. While for static image segmentation this might be an acceptable amount of data, for video segmentation it is simply too much data to process directly in reasonable run times (on standard hardware). Because of this, a common pre-processing step is to down-sample point clouds using a *voxel-grid* filter, a process known as *voxelization*.

3.2 Octree Adjacency Graph

Adjacency is a key element of the proposed method, as it ensures that supervoxels do not flow across object boundaries which are disconnected in space. There are three definitions of adjacency in a vox-

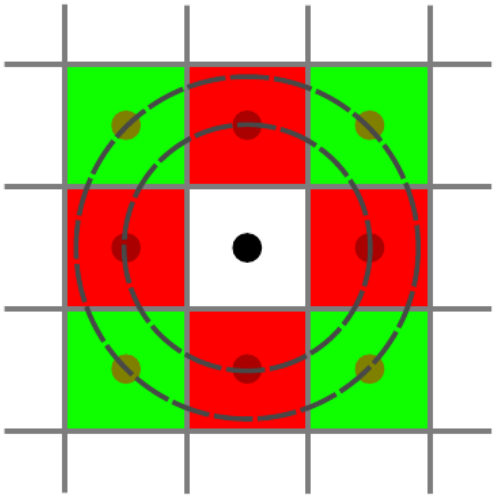


Figure 3.1.3: Standard adjacency of pixels in a 2d grid. Neighbors can share edges (4-neighborhood) or vertices (8-neighborhood).

elized 3D space; 6-,18-, or 26-adjacent. These share a face, faces or edges, and faces, edges, or vertices, respectively. In this work, whenever we refer to adjacent voxels, we are speaking of 26-adjacency.

As a preliminary step, we construct the adjacency graph for the voxel-cloud. This can be done efficiently by searching the voxel kd-tree, as for a given voxel, the centers of all 26-adjacent voxels are contained within $\sqrt{3} * R_{voxel}$. R_{voxel} specifies the voxel resolution which will be used for the segmentation (for clarity, we shall simply refer to discrete elements at this resolution as voxels). The adjacency graph thus constructed is used extensively throughout the rest of the algorithm.

We need to introduce supervoxels properly

3.3 Spatial Cluster Seeding

The algorithm begins by selecting a number of seed points which will be used to initialize the supervoxels. In order to do this, we first divide the space into a voxelized grid with a chosen resolution R_{seed}, which is significantly higher than R_{voxel}. The effect of increasing the seed resolution R_{seed} can be seen in Figure 3.3.2. Initial candidates for seeding are chosen by selecting the voxel in the cloud nearest to the center of each occupied seeding voxel.

Once we have candidates for seeding, we must filter out seeds caused by noise in the depth image. This means that we must remove seeds which are points isolated in space (which are likely due to noise), while leaving those which exist on surfaces. To do this, we establish a small search radius R_{search} around each seed, and delete seeds which do not have at least as many voxels as would be occupied by a planar surface intersecting with half of the search volume (this is shown by the green plane in Figure 3.3.1). Once filtered, we shift the remaining seeds to the connected voxel within the search volume which has

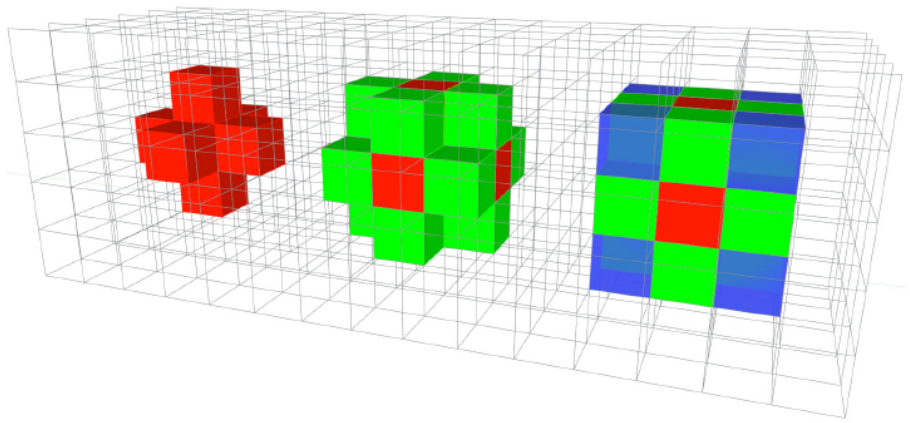


Figure 3.1.4: Adjacency in a 3d voxel grid. The 6-, 18-, and 26-neighborhoods share a face, edge, and vertex, respectively.

the smallest gradient in the search volume. Gradient is computed as

$$G(i) = \sum_{k \in V_{adj}} \frac{\| V(i) - V(k) \|_{CIELab}}{N_{adj}}; \tag{3.1}$$

we use sum of distances in CIELAB space from neighboring voxels, requiring us to normalize the gradient measure by number of connected adjacent voxels N_{adj}. Figure 3.3.1 gives an overview of the different distances and parameters involved in seeding.

Once the seed voxels have been selected, we initialize the supervoxel feature vector by finding the center (in feature space) of the seed voxel and connected neighbors within 2 voxels.

3.4 Cluster Features and Distance

VCCS supervoxels are clusters in a 39 dimensional space, given as

$$\mathbf{F} = [x, y, z, L, a, b, \text{FPFH}_{1..33}], \tag{3.2}$$

where x, y, z are spatial coordinates, L, a, b are color in CIELab space, and $\text{FPFH}_{1..33}$ are the 33 elements of Fast Point Feature Histograms (FPFH), a local geometrical feature proposed by Rusu et al. [**?**]. FPFH are pose-invariant features which describe the local surface model properties of points using combinations of their k nearest neighbors. They are an extension of the older Point Feature Histograms optimized for speed, and have a computational complexity of $O(n \cdot k)$.

In order to calculate distances in this space, we must first normalize the spatial component, as distances, and thus their relative importance, will vary depending on the seed resolution R_{seed}. Similar to the work of Achanta et al., [**?**] we have limited the search space for each cluster so that it ends at the

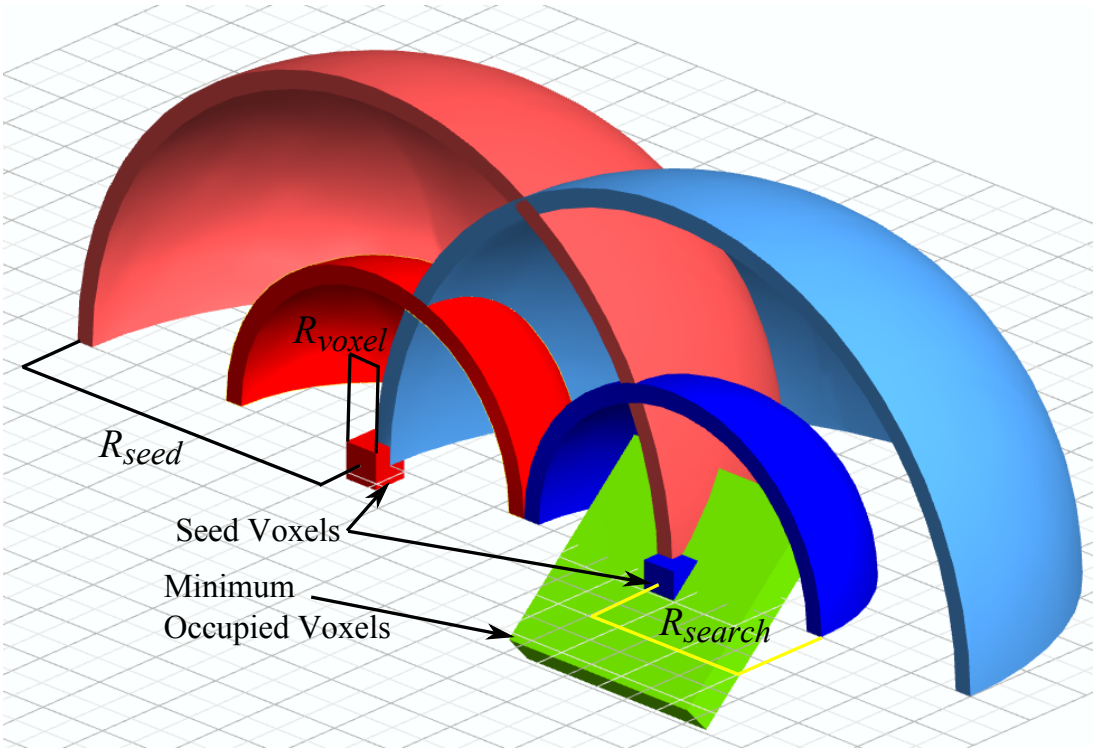


Figure 3.3.1: Seeding parameters and filtering criteria. R_{seed} determines the distance between supervoxels, while R_{voxel} determines the resolution to which the cloud is quantized. R_{search} is used to determine if there are a sufficient number of occupied voxels to necessitate a seed.

neighboring cluster centers. This means that we can normalize our spatial distance D_s using the maximally distant point considered for clustering, which will lie at a distance of $\sqrt{3}R_{seed}$. Color distance D_c, is the euclidean distance in CIELab space, normalized by a constant m. Distance in FPFH space, D_f, is calculated using the Histogram Intersection Kernel [**?**]. This leads us to a equation for normalized distance D:

$$D = \sqrt{\frac{\lambda D_c^2}{m^2} + \frac{\mu D_s^2}{3R_{seed}^2} + \varepsilon D_{HiK}^2}, \tag{3.3}$$

where λ, μ, and ε control the influence of color, spatial distance, and geometric similarity, respectively, in the clustering. In practice we keep the spatial distance constant relative to the other two so that supervoxels occupy a relatively spherical space, but this is not strictly necessary. For the experiments in

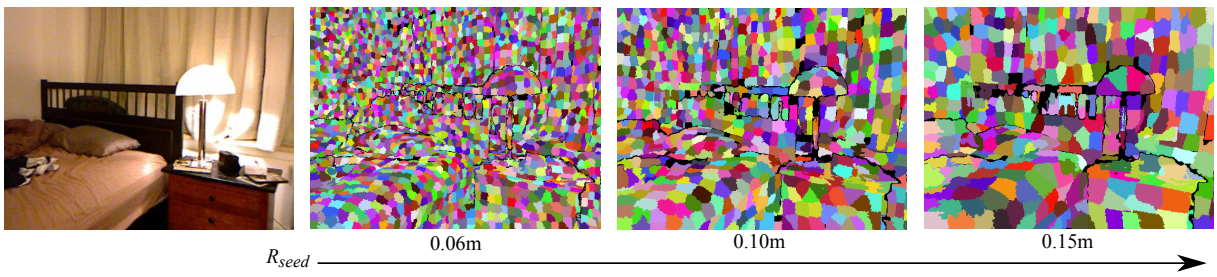


Figure 3.3.2: Image segmented using VCCS with seed resolutions of 0.1, 0.15 and 0.2 meters.

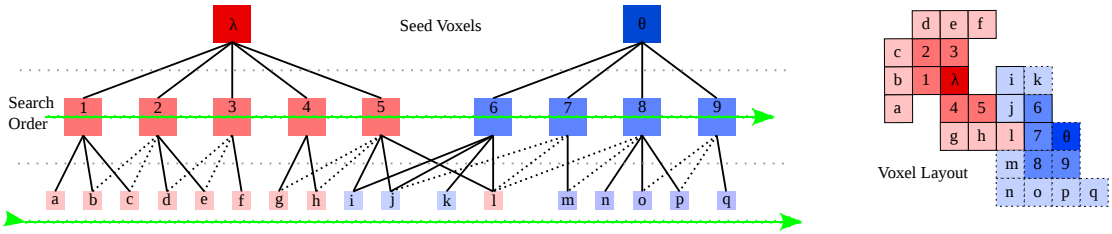


Figure 3.5.1: Search order for the flow constrained clustering algorithm (shown in 2D for clarity). Dotted edges in the adjacency graph are not searched, as the nodes have already been added to the search queue.

this paper we have color weighted equally with geometric similarity.

3.5 Flow Constrained Region Growing

Assigning voxels to supervoxels is done iteratively, using a local k-means clustering related to [**? ?**], with the significant difference that we consider connectivity and flow when assigning pixels to a cluster. The general process is as follows: beginning at the voxel nearest the cluster center, we flow outward to adjacent voxels and compute the distance from each of these to the supervoxel center using Equation 3.3. If the distance is the smallest this voxel has seen, its label is set, and using the adjacency graph, we add its neighbors which are further from the center to our search queue for this label. We then proceed to the next supervoxel, so that each level outwards from the center is considered at the same time for all supervoxels. We proceed iteratively outwards until we have reached the edge of the search volume for each supervoxel (or have no more neighbors to check).

This amounts to a breadth-first search of the adjacency graph, where we check the same level for all supervoxels before we proceed down the graphs in depth. Importantly, we avoid edges to adjacent voxels which we have already checked this iteration. The search concludes for a supervoxel when we have reached all the leaf nodes of its adjacency graph or none of the nodes searched in the current level were set to its label. This search procedure, illustrated in Figure 3.5.1, has two important advantages over existing methods:

1. Supervoxel labels cannot cross over object boundaries that are not actually touching in 3D space, since we only consider adjacent voxels, and

2. Supervoxel labels will tend to be continuous in 3D space, since labels flow outward from the center of each supervoxel, expanding in space at the same rate.

Once the search of all supervoxel adjacency graphs has concluded, we update the centers of each supervoxel cluster by taking the mean of all its constituents. This is done iteratively; either until the cluster centers stabilize, or for a fixed number of iterations. For this work we found that the supervoxels were stable within a few iterations, and so have simply used five iterations for all presented results.

3.6 Depth Adaptive Grid

So far we have described the main algorithm for segmentation. Next we will introduce a generally applicable depth transform, which improves this specific analysis but can be used for all types of image analyses using algorithms with a fixed scale of observation on RGB-D data from a single RGB-D camera. In our case, we address shortcomings of the voxel grid VCCS is based on.

It is evident that observations from a single RGB-D camera have a significant drawback - the point density, and thus available detail of the scene geometry, falls rapidly with increasing distance from the camera. In addition, the levels of both quantization and noise grow quadratically [**? ?**], leading to a further degradation in the quality of geometric features. This change in point density with depth creates a tradeoff between capturing small-scale detail in the foreground (using small voxels) and avoiding noise in the background (using large voxels). This is a general problem which occurs in all algorithms working with a fixed scale (for example a radius search) on point clouds created from a single view.

We propose to compensate for the loss of point density and quantization with increasing depth z by transforming the points into a skewed space using the transformation $T : (x, y, z) \rightarrow (x', y', z')$ with

$$x' = x/z, \quad y' = y/z, \quad z' = \log(z) \tag{3.4}$$

The division of the x and y coordinates by z reverses the perspective transformation, equalizing the point density in the x-y-plane. Transforming the z coordinate helps to deal with the effects of depth quantization by compressing points as depth increases. It is easy to show that the transformation has the following property:

$$\frac{\partial x'}{\partial x} = \frac{\partial y'}{\partial y} = \frac{\partial z'}{\partial z} = \frac{1}{z} \tag{3.5}$$

Because the derivatives are equal, the local coordinate frame is stretched equally along all axes by the transformations. The important thing about this property is, that small cubic voxels are still cubic after the transformation. This leaves the geometry of space basically untouched in the foreground (if the voxel size is chosen sufficiently small), while voxels in the background are skewed and grow, to compensate for reduced amount of detail available in the data.

Rather than transforming the clouds back and forth, we instead transform the bins of the octree itself, creating an octree where bin volume (and thus, voxel size) effectively increases with distance from the camera viewpoint. Doing this directly within the octree allows us to determine adjacency as before (neighboring bins), even though distance between neighboring voxels increases with distance from the camera. Fig. 3.6.1 illustrates this advantageous effect of this transformation on the segmentation.

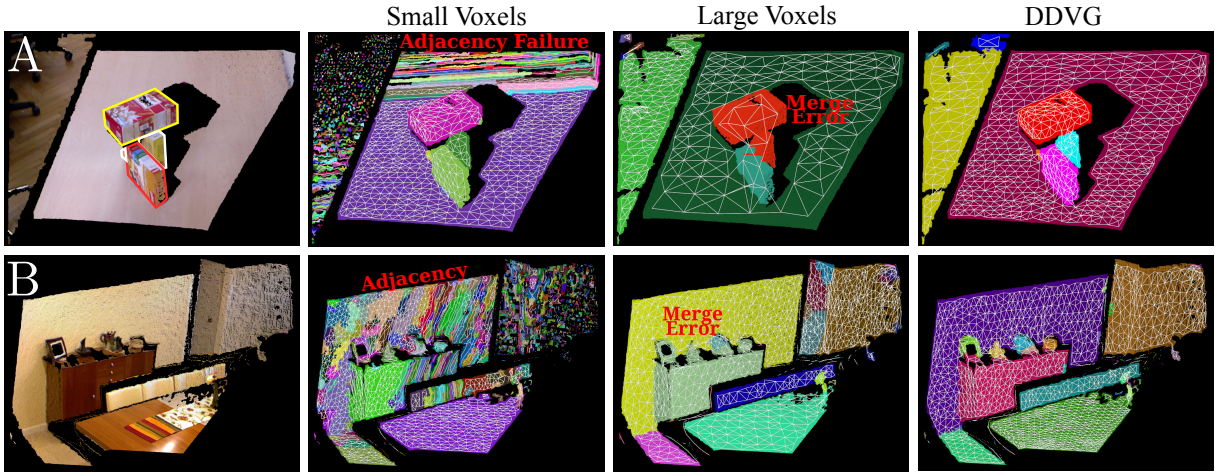


Figure 3.6.1: Two example point clouds **(A,B, left)** showing the need for the Depth Dependent Voxel Grid. For better visibility outlines have been drawn around the boxes in A. Using *Small Voxels* objects close to the camera can be segmented, but adjacency breaks down as the depth increases and the point density decreases. Using *Large Voxels* corrects the adjacency graph in the background, but leads to objects being merged in the foreground due to the coarse resolution. Using *DDVG*, the scale of the voxels gradually increases with distance from the camera – adapting to the increased noise level and lower point density – consequently adjacency is maintained and the segmentation of scenes with large depth variance is possible using fixed parameters. Note, the flat rug on the table in B does not differ enough from the table's surface and cannot be segmented by any purely depth dependent method.

3.7 Locally Convex Connected Patches (LCCP)

As an example of an application of supervoxels and the adjacency octree, we shall briefly present a segmentation method which segments the supervoxel adjacency graph by classifying whether the connection $e = (\vec{p}_i, \vec{p}_j)$ between two supervoxels is convex (=valid) or concave (=invalid).

Extended Convexity Criterion (CC) Consider two adjacent supervoxels with centroids at the positions $\vec{x}_1, \vec{x}_2$ and normals $\vec{n}_1, \vec{n}_2$. Whether the connection between these is convex or concave can be inferred from the relation of the surface normals to the vector joining their centroids.

The angle of the normals to the vector $\vec{d} = \vec{x}_1 - \vec{x}_2$ joining the centroids can be calculated using the identity for the dot product $\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos(\alpha)$ with $\alpha = \measuredangle(\vec{a}, \vec{b})$. For *convex* connections, α_1 is smaller than α_2 (see Fig. 3.7.2 A). This can be expressed as:

$$\alpha_1 < \alpha_2 \Rightarrow \cos(\alpha_1) - \cos(\alpha_2) > 0 \Leftrightarrow \vec{n_1} \cdot \hat{d} - \vec{n_2} \cdot \hat{d} > 0,$$

where $\hat{d} = \frac{\vec{x_1} - \vec{x_2}}{||\vec{x_1} - \vec{x_2}||}$. Similarly, for a *concave* connection we get:

$$\alpha_1 > \alpha_2 \Leftrightarrow \vec{n_1} \cdot \hat{d} - \vec{n_2} \cdot \hat{d} < 0.$$

Note that these operations are commutative, thus the choice of which patch is $\vec{x}_1$, does not change the result. Also the criterion is still valid if the $\vec{x}_i$ are displaced, as long as they stay within the surface.

To compensate for noise in the RGB-D data, a bias is introduced to treat concave connections with

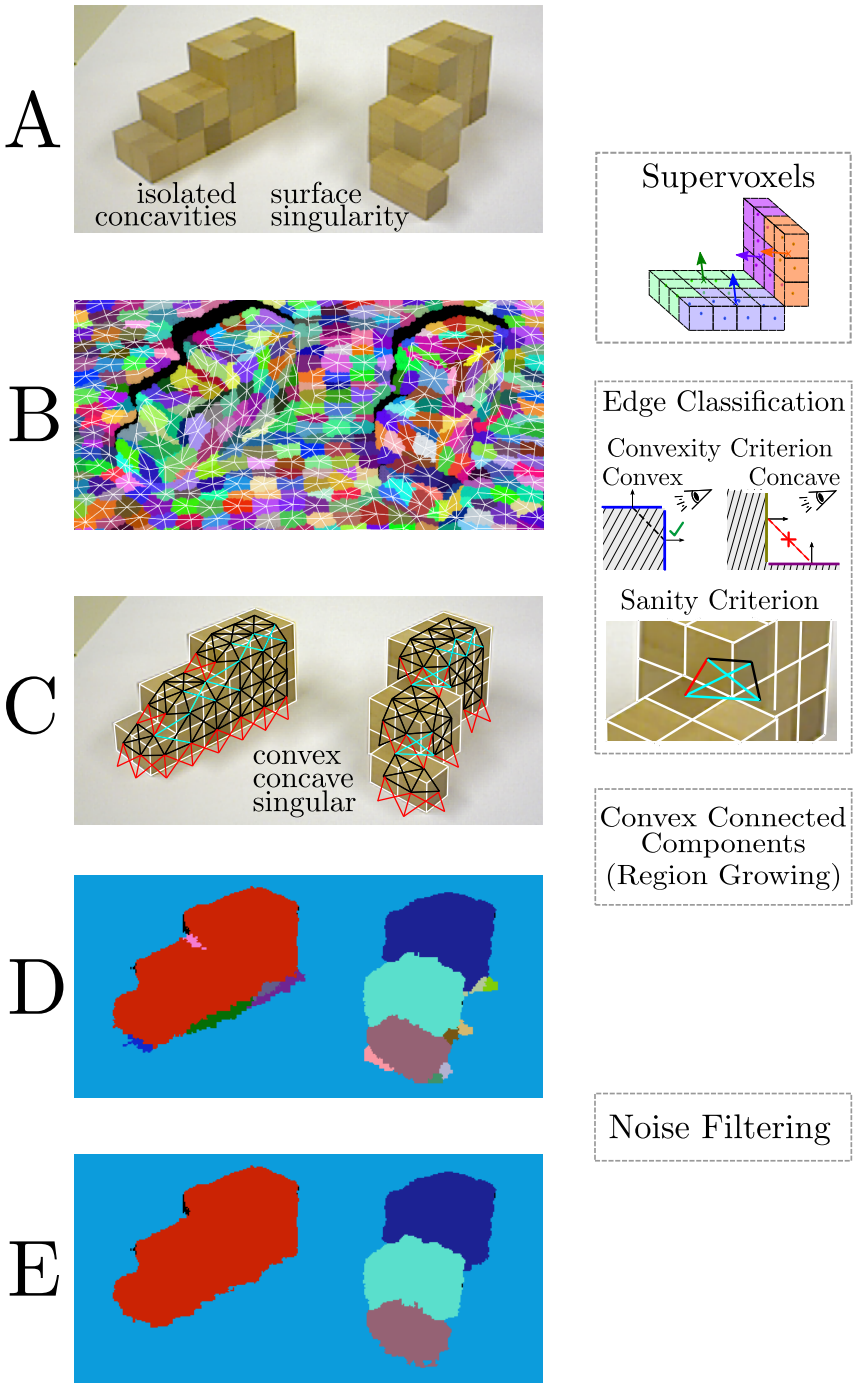


Figure 3.7.1: Flow diagram of the segmentation algorithm. **A)** RGB images corresponding to the point clouds of the scene. The red lines show two isolated concavities. The blue box shows an area with a surface singularity. **B)** Supervoxel adjacency graph. **C)** Model depicting the classified graph. Black lines denote convex connections, red lines concave ones and turquois lines singular connections (those, where two patches are connected only in a single point). **D)** Segmentation result; object labels are shown by different colors. **E)** Final result after noise filtering. The right column illustrates the supervoxel patches and the convexity and sanity criteria used for edge classification.

very similar normals, that is

$$\beta = \measuredangle(\vec{n}_1, \vec{n}_2) = |a_1 - a_2| = \cos^{-1}(\vec{n_1} \cdot \vec{n_2}) < \beta_{\text{Thresh}} \,,$$

as convex, since those usually represent flat surfaces. Depending on the value of the *concavity tolerance threshold* β_{Thresh}, concave surfaces with low curvature are seen as convex and thus merged in the segmentation. This behavior may be desired to ignore small concavities. We set:

$$\text{CC}_b(\vec{p}_i, \vec{p}_j) := \begin{cases} \text{true} & (\vec{n_1} - \vec{n_2}) \cdot \hat{d} > 0 \ \vee \ (\beta < \beta_{\text{Thresh}}) \\ \text{false} & \text{otherwise.} \end{cases} \tag{3.6}$$

where the variable CC_b defines the *basic convexity criterion*. However, local errors in the feature estimation caused by noise in the data can propagate very easily, potentially leading to errors in the resulting segmentation. This also makes the recognition of small concavities harder, as subtle features are more sensitive to noise. To improve on this we – finally – also include neighborhood information in the classification of edges: For a convex edge $e = (\vec{p}_i, \vec{p}_j)$, we require that there exists a common neighbor $\vec{p}_c$ of $\vec{p}_i$ and $\vec{p}_j$ that has a convex connection to both.

Thus we define *extended convexity* CC_e:

$$\begin{aligned} \text{CC}_e(\vec{p}_i, \vec{p}_j) = \text{CC}_b(\vec{p}_i, \vec{p}_j) \wedge \text{CC}_b(\vec{p}_i, \vec{p}_c) \\ \wedge \text{CC}_b(\vec{p}_j, \vec{p}_c) \end{aligned} \tag{3.7}$$

With extended convexity, more evidence is necessary for a connection to be labeled as convex. Our implementation is based on the idea proposed by Moosmann [**?**]. In some cases results are already satisfactory even without using this type of neighborhood information. Thus, in the results section we will always denote whether this is used or not.

Region Growing: The second problem discussed in the Introduction, the danger of splitting objects along *isolated concavities*, can be addressed in the next step. For this we need to find all *clusters* of supervoxels, which belong to the same subgraph of valid convex connected edges. This is accomplished using a region growing process: First, an arbitrary seed supervoxel is chosen and labeled. This label is then propagated over the graph with a depth search that is only allowed to grow over convex edges. Once no new supervoxel can be assigned to the segment, we choose a new seed supervoxel that has not been labeled and propagate the new label as before, repeating the process until all supervoxels have been labeled. Note that all our criteria are commutative, so the output of the region growing does not depend on the choice of the seeds.

3.8 Experimental Results

3.8.1 Datasets

In the following sections we present quantitative results for VCCS and LCCP. We compare to state-of-the-art methods on the *NYU Indoor Dataset*[**?**] and *Object Segmentation Database*[**?**].

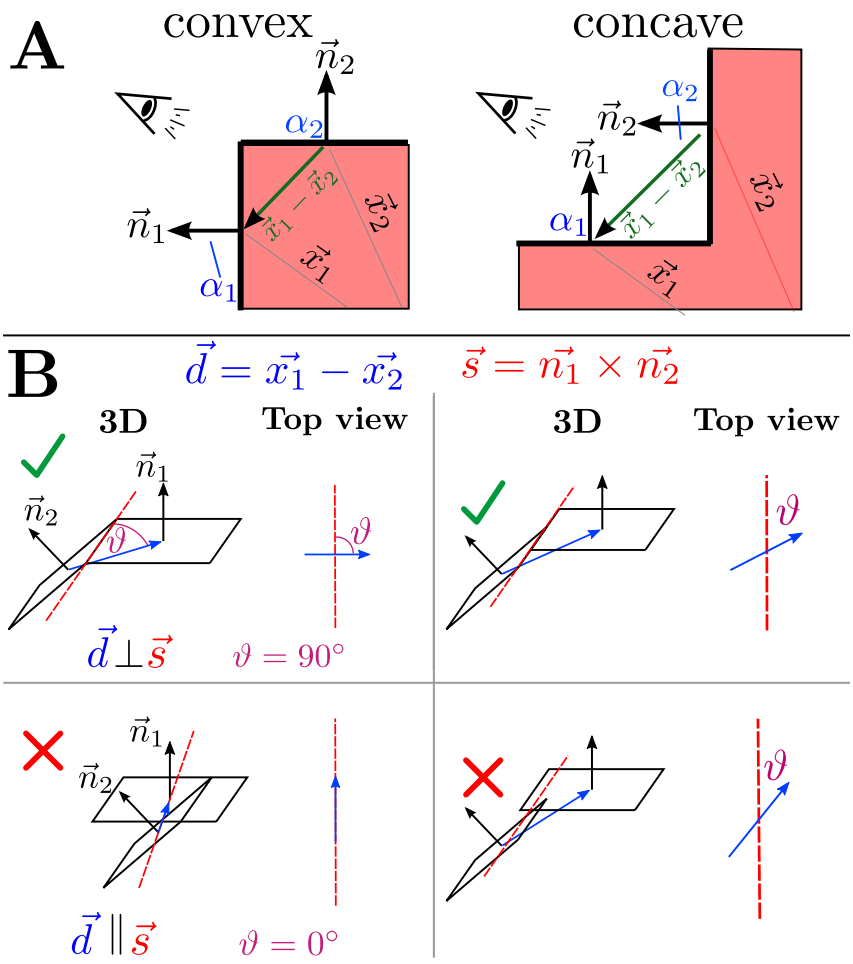


Figure 3.7.2: A) Illustration of measures used in the basic convexity criterion. **B)** Illustration of the sanity criterion for decreasing values of ϑ. Singular connections are characterized by small values of ϑ.

Object Segmentation Database (OSD)

The *Object Segmentation Database* (OSD-v0.2) was proposed by Richtsfeld *et al.*[**?**] in 2012. It consists of 111 cluttered scenes of objects on a table, taken with close proximity to the pictured objects. The scenes contain multiple objects, which have mostly box-like or cylindrical shape, with partial and full occlusions and heavy clutter in 2D as well as 3D. Importantly, most objects in the data set are *simple*, that is, consist of only a single part. This makes the ground-truth data relatively non-ambiguous.

The qualitative examples (Fig. 3.8.1) show that our algorithm performs very well in the segmentation of these cluttered scenes. The object separation can be intuitively understood: all objects present in the scenes are separated by concave boundaries, i.e. a line connecting neighboring surfaces of two different objects always travels through "air". This is also true for the boundary between an object and the supporting surface. As a consequence, objects that have a convex shape are correctly captured as one segment and separated from the other objects. Hollow objects (bowls, cups etc.) can be observed to show multiple segments inside, because the orientation of surface normals changes strongly on these concave surfaces. Despite most objects being simple, some objects have meaningful parts, such as handles or bottle caps, which are segmented by our algorithm.

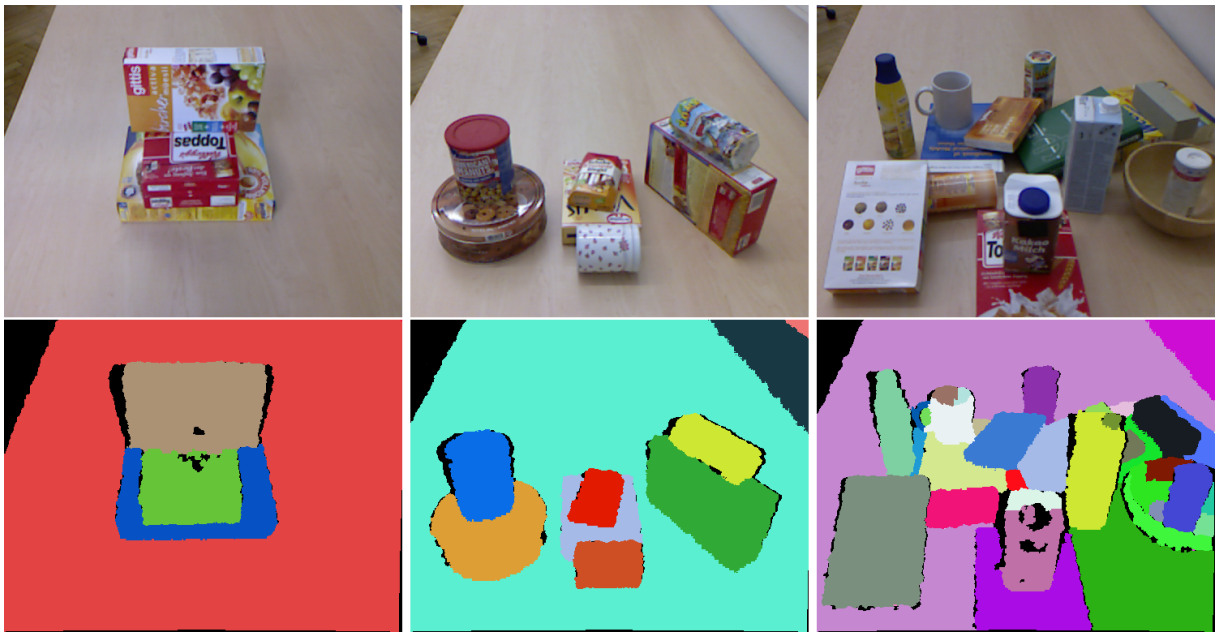


Figure 3.8.1: Example results for the OSD dataset. Points beyond a distance of 2m were cropped for visualization. Parameters: $R_{voxel} = 0.005$, $R_{seed} = 0.02$, $\beta_{\text{Thresh}} = 10°$.

The quantitative results (Table **??**) demonstrate that our approach is able to compete with state-of-the-art methods in the task of segmenting cluttered scenes with 'single-part' objects. Compared to the learning-based method from [**?**] we achieve better object separation (F_{us}), but higher oversegmentation error (F_{os}). The latter is because we sometimes detect object parts (*e.g.* handles) and we do not utilize model fitting, which helps against noise. Comparing to the learning-free method of Ückermann *et al.* [**?**] we obtain results within a standard deviation. Note that our actual goal is to partition complex objects into parts, which is dissimilar from the other two methods.

Figure 3.8.2: Example results for scenes from the NYU dataset using unsmoothed depth. Black areas indicate missing depth. Top row: rgb images. Mid. row: segmentation result. Bottom row: ground truth. Parameters A-C: $R_{voxel} = 0.0075$, $R_{seed} = 0.03$ and $\beta_{\text{Thresh}} = 8^{\circ}$. Parameters D-E: $R_{voxel} = 0.01$, $R_{seed} = 0.04$ and $\beta_{\text{Thresh}} = 10^{\circ}$ (identical to quantitative results, see Tab. 3.8.1).

NYU Indoor Dataset (NYU)

The *NYU Indoor Dataset*[2] (NYUv2) from Silberman *et al.*[**?**] is a large and complex dataset, consisting of 1449 cluttered indoor scenes. The data consists of pairs of aligned RGB and depth images, along with human annotated densely labeled ground truth. The images were captured in diverse indoor scenes, and present many difficulties for segmentation algorithms such as varied illumination and many small similarly colored objects. Examples of typical scenes are shown in Figure 3.8.5. One main difficulty presented by the dataset is that the distance to objects from the camera is quite large in the dataset. This results in significant depth quantization artifacts as well as few data points for many objects. Additionally, depth is often missing for extensive portions of many of the images, due to limitations of the Kinect sensor (e.g. reflective, transparent surfaces - windows are especially problematic). Silberman *et al.* attempt to correct for these errors using a hole filling algorithm (smoothdepth), which estimates depth for missing areas based on the scheme from Levin *et al.*[**?**].

3.8.2 Supervoxels

In order to evaluate the quality of supervoxels generated by VCCS, we performed a quantitative comparison with three state-of-the-art superpixel methods using publicly available source code. We selected the two 2D techniques with the highest published performance from a recent review [**?**]: a graph based method, GCb10 [**?**][3], and a gradient ascent local clustering method, SLIC [**?**][4]. Additionally, we

[2] http://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html

[3] http://www.csd.uwo.ca/~olga/Projects/superpixels.html

[4] http://ivrg.epfl.ch/supplementary_material/RK_SLICSuperpixels/index.html

Figure 3.8.3: Example of hole-filling for images after returning from voxel-cloud to the projected image plane. Depth data, shown in the top left, has holes in it, shown as dark blue areas (here, due to the lamp interfering with the Kinect). The resulting supervoxels do not cover these holes as shown in the bottom left, since the cloud has no points in them. To generate a complete 2D segmentation, we fill these holes in using the SLIC algorithm, resulting in a complete segmentation, seen in the top right. The bottom right shows human annotated ground truth for the scene.

selected another method which uses depth images, DASP[**?**][5]. Examples of over-segmentations produced by the methods are given in Figure 3.8.5.

Returning to the Projected Plane

RGB+D sensors produce what is known as an organized point cloud- a cloud where every point corresponds to a pixel in the original RGB and depth images. When such a cloud is voxelized, it necessarily loses this correspondence, and becomes an unstructured cloud which no longer has any direct relationship back to the 2D projected plane. As such, in order to compare results with existing 2D methods we were forced to devise a scheme to apply supervoxel labels to the original image.

To do this, we take every point in the original organized cloud and search for the nearest voxel in the voxelized representation. Unfortunately, since there are blank areas in the original depth image due

[5] https://github.com/Danvil/dasp

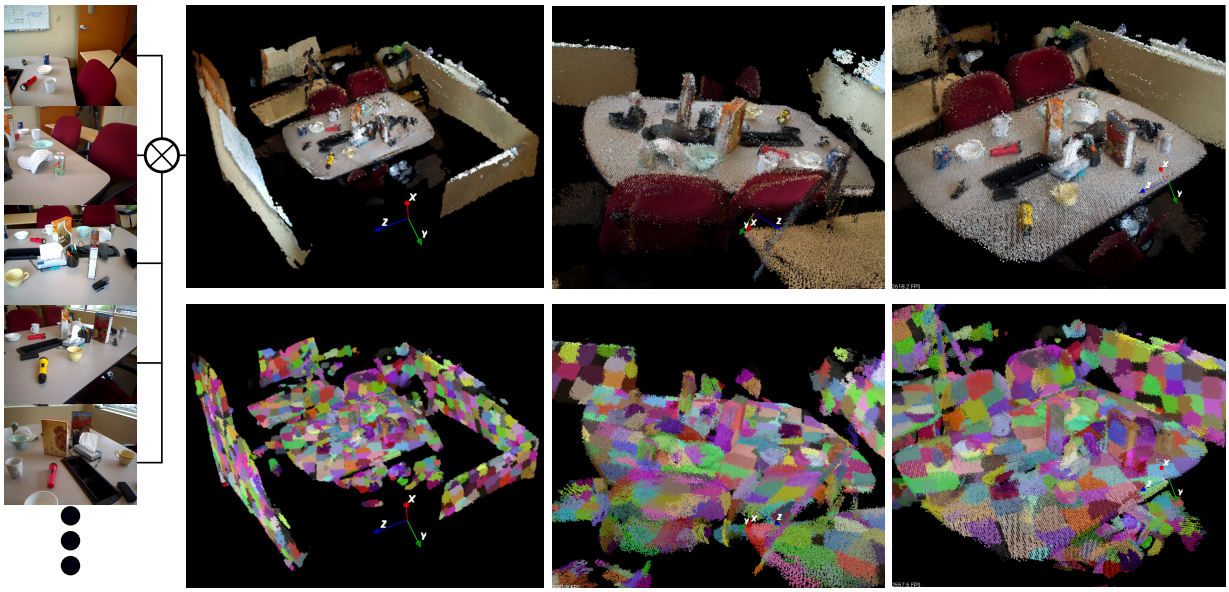


Figure 3.8.4: Over-segmentation of a cloud from the RGB-D scenes dataset[28]. The cloud is created by aligning 180 Kinect frames, examples of which are seen on the left side. The resulting cloud has over 3 million points, which reduces to 450k points at $R_{voxel} = 0.01m$ and 100k points with $R_{voxel} = 0.02m$. Over-segmentation of these take 6 and 1.5 seconds, respectively (including voxelization).

Figure 3.8.5: Examples of under-segmentation output. From left to right- ground truth annotation, SLIC, GCb10, DASP, and VCCS. Each is shown with two different superpixel densities.

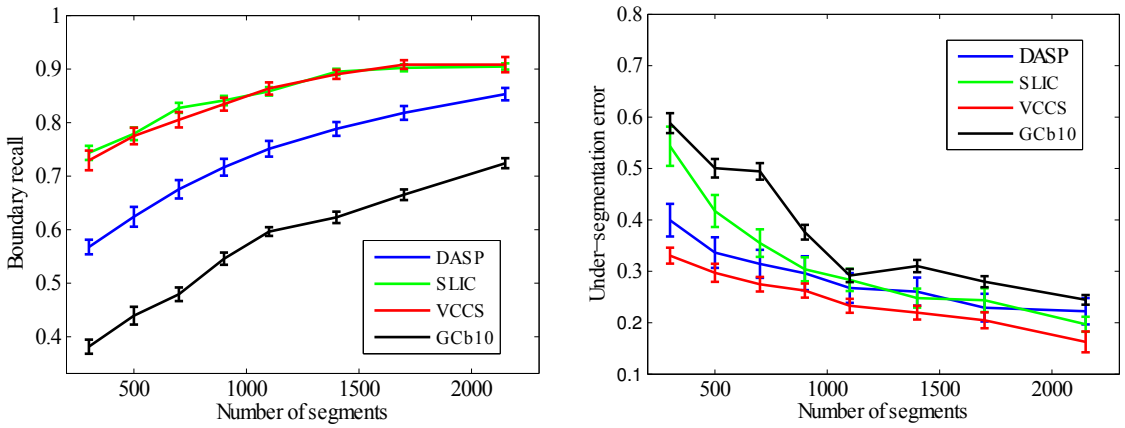


Figure 3.8.6: Boundary recall and under-segmentation error for SLIC, GCb10, DASP, and VCCS.

to such factors as reflective surfaces, noise, and limited sensor range, this leaves us with some blank areas in the output labeled images. To overcome this, we fill in any large unlabeled areas using the SLIC algorithm. This is not a significant drawback, as the purpose of the algorithm is to form supervoxels in 3D space, not superpixels in the projected plane, and this hole-filling is only needed for comparison purposes. Additionally, the hole filling actually makes our results worse, since it does not consider depth, and therefore tends to bleed over some object boundaries that were correctly maintained in the supervoxel representation. An example of what the resulting segments look like before and after this procedure are shown in Figure 3.8.3.

Evaluation Metrics

The most important property for superpixels is the ability to adhere to, and not cross, object boundaries. To measure this quantitatively, we have used two standard metrics for boundary adherence- boundary recall and under-segmentation error[**? ?**]. Boundary recall measures what fraction of the ground truth edges fall within at least two pixels of a superpixel boundary. High boundary recall indicates that the superpixels properly follow the edges of objects in the ground truth labeling. The results for boundary recall are given in Figure 3.8.6. As can be seen, VCCS and SLIC have the best boundary recall performance, giving similar results as the number of superpixels in the segmentation varies.

Under-segmentation error measures the amount of leakage across object boundaries. For a ground truth segmentation with regions $g_1, \ldots, g_M$, and the set of superpixels from an over-segmentation, $s_1, \ldots s_K$, under-segmentation error is defined as

$$E_{useg} = \frac{1}{N}\left[\sum_{i=1}^{M}\left(\sum_{s_j | s_j \cap g_i} |s_j|\right) - N\right], \tag{3.8}$$

where $s_j \mid s_j \cap g_i$ is the set of superpixels required to cover a ground truth label g_i, and N is the number

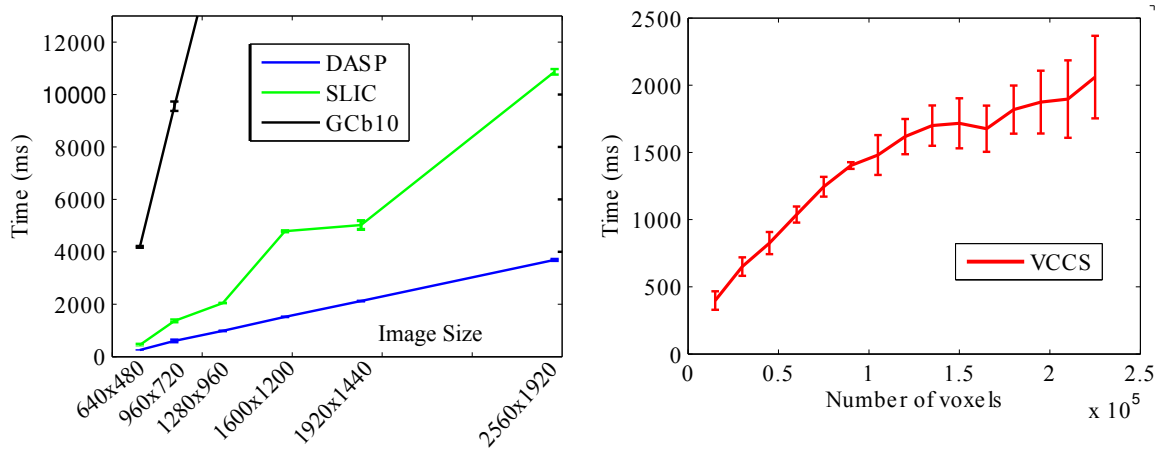


Figure 3.8.7: Speed of segmentation for increasing image size and number of voxels. Use of GCb10 rapidly becomes unfeasible for larger image sizes, and so we do not adjust the axes to show its run-time. The variation seen in VCCS run-time is due to dependence on other factors, such as R_{seed} and overall amount of connectivity in the adjacency graphs.

of labeled ground truth pixels. A lower value means that less superpixels violated ground truth borders by crossing over them. Figure 3.8.6 compares the four algorithms, giving under-segmentation error for increasing superpixel counts. VCCS outperforms existing methods for all superpixel densities.

Time Performance

As superpixels are used as a preprocessing step to reduce the complexity of segmentation, they should be computationally efficient so that they do not negatively impact overall performance. To quantify segmentation speed, we measured the time required for the methods on images of increasing size (for the 2D methods) and increasing number of voxels (for VCCS). All measurements were recorded on an Intel Core i7 3.2Ghz processor, and are shown in Figure 3.8.7. VCCS shows performance competitive with SLIC and DASP (the two fastest superpixel methods in the literature) for voxel clouds of sizes which are typical for Kinect data at $R_{voxel} = 0.008m$ (20-40k voxels). It should be noted that only VCCS takes advantage of multi-threading (for octree, kd-tree, and FPFH computation), as there are no publicly available multi-threaded implementations of the other algorithms.

3.8.3 Locally Convex Connected Patches (LCCP)

We compare segments against ground truth using three standard measures: *Weighted Overlap* (WOv), which is a summary measure proposed by Silberman *et al.* [**?**], as well as *false negative* (*fn*) and *false positive* (*fp*) scores from [**?**] and *over-* (F_{os}) and *under-segmentation* (F_{us}) from [**?**].

Example scenes in Fig. 3.8.2 show that the general object separation is still very good, which is expected from the results presented in the previous section. In contrast to the simple objects from the OSD dataset, however, in the NYU dataset the objects of interest are complex objects from various as-

Method	Learned Features	Depth Data	WOv
LCCP	NO LEARNING	depth	53.6%
	NO LEARNING	smoothdepth	53.8%
LCCP + ext. convexity	NO LEARNING	smoothdepth	57.6%
Silberman *et al.*[**?**]	RGB	-	50.3%
	Depth	both	53.7%
	RGB-D	both	60.1%
	RGB-D + Support + Structure classes	both	61.1%
Gupta *et al.*[**?**]	gPb-ucm Gradients (from [**?**])	-	55.0%
	gPb-ucm + Depth + Concavity Gradients	both	62.0%

Table 3.8.1: Comparison of different segmentation methods on the NYU dataset using weighted overlap *WOv*. LCCP results were produced with voxel size $R_{voxel} = 0.01$, seed size $R_{seed} = 0.04$ and concavity tolerance angle $\beta_{\text{Tresh}} = 10^{\circ}$.

pects of everyday life. As a consequence, these are mostly composed of multiple parts and thus our algorithm reveals interesting partitions. To give a few examples: In scene (A) the cupboard is partitioned into the top plate and its various drawers and the toilet is segmented into the flushing tank, the seat and the base. Scene (B) presents how a human is partitioned into hand, arm-bed, upper/lower part of the body and legs. For the sofas in scene (D) we get the two arm-rests, the back-rest and the seating. Note that in these cases the segments represent "nameable regions", which is also the case for many other segments (we discuss this further in Section **??**). It is evident that the ground truth data will then disagree with our labeling, which results in unjustified errors. For example, in scene (A) the ground truth considers the sink to be an important part, while the drawers are not labeled individually. Despite this disagreement, we do not consider either of the labelings wrong in general, but see them as different views on the data which are in many cases equally justifiable.

Because we designed our algorithm to detect object parts defined only by geometry, very thin objects like the posters on the background wall in scene (C) are not recognizable. Also the challenging data quality sometimes leads the part partitioning to fail, as seen for the human in scene (E). To show how well the general idea of part partitioning using concave boundaries works, we present results for a complex object with high data quality in the next section.

The quantitative results (Table 3.8.1)[6] show that our algorithm is able to produce good results on the challenging dataset. Despite being much simpler and without requiring learning on human annotated ground-truth, we compete with the approach from [**?**] when only depth information is used. Additionally, we still achieve 93% of their score when comparing against the more complex feature spaces used in conjunction with learning-based algorithms. We should emphasize that our competitors do not aim for object parts but rather for "whole object" detections, specifically, those whole objects learned from this particular annotated ground truth. Conversely, our method establishes a general rule for object-ness that does not depend on this particular dataset, nor on the whims of a particular human annotator.

[6] Updated results for [**?**] are available at http://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html.

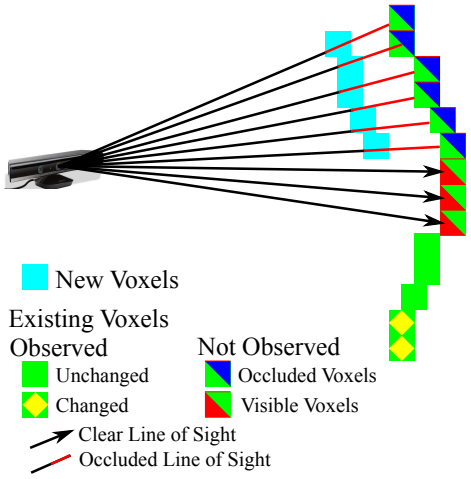


Figure 3.9.1: Categorization of voxels based on new frame of data. Voxels fall into three categories, they are either new, observed or not observed in the frame. Furthermore, observed voxels can either have changed or remained the same, while voxels not observed in the frame are either occluded or no longer exist (in which case they should be deleted).

3.9 Sequential Update of Perceptual Model

Adding newly observed points to an existing supervoxel octree is accomplished through a three stage process. First, we must insert the points into the octree, and initialize new leaves for them if they did not exist previously. This results in an octree where leaves fall into three possible categories (illustrated in Fig. 3.9.1; they are either new, observed, or unobserved in the most recent observation. Handling of new leaves is straightforward; we simply calculate adjacency relations to existing leaves and flag them as unlabeled.

To determine whether a leaf which existed previously has changed, we test the distance between the centroid of the points falling within its voxel (from the new frame) and its previous centroid. This is done in the same feature space used for growing the supervoxels, that is, we test whether the normal, color, and spatial location have varied more than a threshold value. This threshold is set to a relatively low constant value so that it favors false-positives (finding change when there was none), as they do not impact the tracking performance of the algorithm, but only have a slight effect on its run-time. If a leaf is found to have changed, we remove its previous labeling. We also perform a global check to see if more than half of a supervoxels support has changed; if so, we completely remove the supervoxels label from all of its constituent voxels.

Finally, we must consider how to handle leaves which were not observed in the inserted point cloud. Rather than simply prune them, we first check if it was possible to observe them from the viewpoint of the sensor which generated the input cloud. This occlusion check can be accomplished efficiently using the octree by determining if any voxels exist between unobserved leaves and the sensor viewpoint. If a clear line of sight exists from the leaf to the camera, it can safely be deleted. Conversely, if the path is obstructed, we ”freeze” the leaf, meaning that it will remain constant until it is either observed or passes the line of sight test in a future frame (in which case, it can be safely deleted). This occlusion testing

means that tracking of occluded objects is trivial, as occluded voxels remain in the observations which are used for tracking.

Once the octree voxels have been updated, we then proceed to update the supervoxels as before. That is, first we generate new seeds in regions of large unlabeled voxels, and then conduct the iterative region growing. This results in new supervoxels in regions which are new or changing, while leaving supervoxels in static and occluded regions unchanged. This reduces the tracking and segmentation problem to finding the best joint association of these new supervoxels with those from the prior time-step.

3.10 Discussion

In this Chapter we have presented several new concepts- the octree adjacency graph, supervoxels, a segmentation method which uses supervoxels, as well as a way to sequentially update an octree with new frames of data. Additionally, we have presented quantitative and qualitative results which demonstrate the usefulness of these techniques. In particular, LCCP has demonstrated the usefulness of a patch-based adjacency-graph interpretation of 3D Point Cloud data. We believe the results we achieved stem from two core properties: the ability of supervoxels to efficiently encode local regions, and the usefulness of a 3D adjacency-graph in resolving situations which are ambiguous in a 2D representation.

Some Quote.

Quoteauthor Lastname

4

Model-Based Point Cloud Tracking

Now that we have established a reduced, stable world model in which voxels persist through occlusions, the next step is to adapt the general framework of Sequential Bayesian Estimation to track models within this 3D voxel model. The core concepts remain the same as discussed in Chapter 2 (for a brief introduction, see Appendix B), and we will again use a bank of parallel particle filters, due to their robustness to noise (of which there is quite a bit in Kinect data) and ability to handle non-linear dynamics.

4.1 Particle Filters in 3D

The general concept of particle filtering remains the same as the 2D version discussed in Chapter 2, with the primary changes lying in how we score individual particle predictions using the measurement model. The models have also been changed from 2D masks to 3D Point Cloud models which rely on point to point correspondence rather than a global histogram distance score. The dynamic model is an extension from 2D pixel position to real-world 3D coordinates which also include orientation.

4.2 Model Representation

One of the main limitations of the 2D projected mask model discussed in Chapter 2 is that the masks of objects are not invariant to pose changes - in general, rotation of an object will change the shape of its mask and distribution of its color histogram. As we now have the ability to observe the full 3D shape of

an object, we choose to represent objects as clusters of points which correspond to the exterior of the object. A visual representation of such a model is given in Figure **??**.

Pointcloud representation of an object model

Points for objects are stored in a model-centered reference frame (which we shall denote with superscript ^{m}), with each point containing an XYZ position, an RGB color for the point, as well as a surface normal vector. That is, each point p of the model k consists of a nine-dimensional vector:

$$p_k^m = [x^m, y^m, z^m, R, G, B, n_x, n_y, n_z], \tag{4.1}$$

and a model for an object O_k consists of a vector of n_k such points p^m:

$$O_k^m = [p_0^m \ldots p_{n_k}^m]. \tag{4.2}$$

It is important to note that the points of an object model given above are model-relative - they must be transformed into the world coordinates in order to evaluate their fit to observations. This will be discussed further in the next Section.

4.3 Dynamic Model

In the 2D tracker presented previously, the time-dependent state vector of a particle consisted of a position shift vector $\mathbf{p}_t = [p_x, p_y]$ and a velocity vector $\mathbf{v}_t = [v_x, v_y]$. The natural extension of this to 3D is to simply add a third p_z and v_z element to each. Of course we should note that the x and y dimensions here in our 3D representation are distinct from those in 2D, which represented pixel coordinates in the image plane. Here our positional coordinates represent real-world distances from a fixed origin (typically the camera "pin-hole" position). It is also important to note that coordinates in our 3D representation are originally in a continuous space - though we discretize them using the octree model discussed in the previous Chapter. For clarity, we shall simply denote coordinates in the world reference frame with no superscript.

While this straightforward extension gives us a reasonable 3D equivalent to our 2D tracked masks, we now have full 3D models, and so it makes sense to use a state vector which takes advantage of it. As such, we further extend the state vector for position and velocity to allow for rotations of the model around the object reference frame x-axis (roll - γ), y-axis (pitch - β), and z-axis (yaw - α). This yields a

position state vector for particle j at time t of

$$\mathbf{x}_t^j = [d_x, d_y, d_z, \gamma, \beta, \alpha]. \tag{4.3}$$

Each object model is tracking using a set of N such particles. We shall now generally omit the object variable k in our notation for clarity. Even though we omit the k, the reader should assume that the following equations are for individual object models, and that we have a set of N independent particles for each object. Additionally, we have velocity state vector

$$\mathbf{v}_t = [v_x, v_y, v_z, v_\gamma, v_\beta, v_\alpha], \tag{4.4}$$

which is not tracked individually per particle, but rather as a whole for the model.

As before, motion is modeled using a constant velocity model in discrete time with a variable sampling period T, giving the dynamic model

$$\mathbf{x}_t = \mathbf{x}_{t-1} + T\mathbf{v}_{t-1} + \boldsymbol{\omega}, \tag{4.5}$$

with noise vector $\boldsymbol{\omega}$ assumed to be zero mean Gaussian with fixed covariance. Particle velocities are updated after weighting of individual particles using the measurement model, and are a weighted average of the change in position

$$\mathbf{v}_t = \frac{1}{TN} \sum_{j=1}^{N} w_j (\mathbf{x}_t^j - \mathbf{x}_{t-1}^j), \tag{4.6}$$

where w_j is the normalized weight for particle j.

Tracking independent velocities for each particle doubles the dimensionality of the state-space, requiring a proportional increase in the number of particles. While the use of independent velocity states potentially helps in complicated tracking scenarios, in our experiments we were unable to observe any tangible benefit. Moreover, in order to avoid instability in the tracking results we needed to double the number of particles for a given noise level, doubling the processing time required. As such, we have chosen to use the above "group-velocity", and leave it to future work to investigate the possibility of independent velocity states.

4.4 Measurement Model

As points for the model are given in a model-centered frame of reference, we must transform them to the world frame them using a 3D affine transformation quaternion:

$$\mathbf{B}^j = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & d_x \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & d_y \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

which we use to transform the extended position vector for each point in the model:

$$p^m = [x^m, y^m, z^m, 1], \tag{4.8}$$

yielding positions in the world frame for each of our η model points for a particular particle j:

$$\begin{bmatrix} \mathbf{p}_1^j \\ \mathbf{p}_2^j \\ \vdots \\ \mathbf{p}_\eta^j \end{bmatrix} \begin{bmatrix} [x_1, y_1, z_1, 1]^\mathsf{T} \\ [x_2, y_2, z_2, 1]^\mathsf{T} \\ \vdots \\ [x_\eta, y_\eta, z_\eta, 1]^\mathsf{T} \end{bmatrix} = \begin{bmatrix} \mathbf{B}^j & 0 & \dots & 0 \\ 0 & \mathbf{B}^j & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{B}^j \end{bmatrix} \begin{bmatrix} [x_1^m, y_1^m, z_1^m, 1]^\mathsf{T} \\ [x_2^m, y_2^m, z_2^m, 1]^\mathsf{T} \\ \vdots \\ [x_\eta^m, y_\eta^m, z_\eta^m, 1]^\mathsf{T} \end{bmatrix}. \tag{4.9}$$

Once we have our transformed points, we then must establish correspondences between each particle's model points and a world point. This is done so that we may score how well a particular particle matches the current world model observation. That is, for each transformed point $\mathbf{p}_{1\dots\eta}^j$, we select corresponding point $\mathbf{p}^*$ in the observation which has minimal spatial distance.

Do I need to put an equation formalizing this? Maybe a figure?

To find these correspondences, we first compute a KD-tree in the spatial dimensions for the world model points. This allows us to efficiently search for the nearest point to each transformed point. We create this tree for the world model rather than the transformed model (even though the former has more points) as there is only one world, but many particles and models. Computing it for the models would require a KD-tree for each particle in each model. Additionally, computing it for the world allows us to take advantage of sampling strategies (discussed later in this Chapter) which significantly reduce our run-time complexity.

Once we have selected (with replacement) an observed point correspondence for each model point, we must calculate an un-normalized weight $\tilde{w}^j$ corresponding to the similarity of the transformed points to the world observation. This is accomplished by summing the individual correspondence scores computed using weighted distance in world-, color-, and normal-space:

$$\tilde{w}^j = \sum_1^\eta \frac{1}{1 + \frac{\mu \|\mathbf{p}_{xyz}^j - \mathbf{p}_{xyz}^*\|}{R_{voxel}} + \frac{\lambda D_c(p_{RGB}^j, p_{RGB}^*)}{m} + \varepsilon \|\mathbf{p}_{n_x n_y n_z}^j - \mathbf{p}_{n_x n_y n_z}^*\|}, \tag{4.10}$$

where we follow the convention given Section3.4. That is, μ, λ, and ε are weighting constants, D_c is euclidean distance in HSV space, and m is a normalizing constant. We do not normalize normals, as they are already unit vectors. In our experiments we typically set the weighting factors to $\mu = 1, \lambda = 2$, $\varepsilon = 1$, as this balances the scoring between color and geometric shape, and found experimentally that it produced consistently good tracking results. The calculated particle weights $\tilde{w}^j$ are then normalized, and a final state estimate can be computed by taking the weighted average of all particles

$$\mathbf{x}_t = \sum_{j=1}^N w_j \mathbf{x}_t^j, \tag{4.11}$$

and the group-velocity can be computed using Equation 4.6.

4.5 Stratified Correspondence Sampling

While the tracking method discussed above works on recorded data, it can only achieve 10 frames per second on current hardware for single objects. Moreover, speed of tracking is highly dependent on the size of object models as well as voxel resolution used. With this in mind, we use a sampling scheme which selects a limited number of points from the model to transform and test. By doing this, we achieve linear asymptotic time complexity for the particle filter with respect to the number of particles - there is no dependence on the number of points in the models or the voxel resolution used. The only step which is dependent on the number of input points is the KD-tree construction, but this is only done once (for the world model), is done as a pre-processing step regardless (for normal computation), and is computed by a highly optimized external library.

The sampling scheme is as follows. We set a constant number of sample points per particle N_s, and then divide the model into N_t strata, such that $N_t = N_m/N_s$, where N_k is the number of points in model k. We then select a point from each stratum using uniform sampling, and proceed to transform and score it as described in previous Sections. As an additional step, we also select $\frac{N_s}{2}$ points uniformly from the entire model. Using strata ensures we have coverage of the whole model, while sampling randomly from the entire distribution improves occlusion performance.

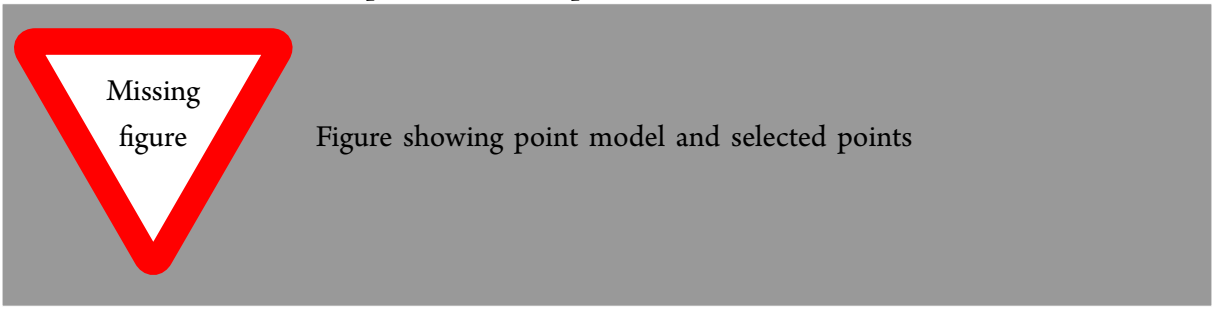


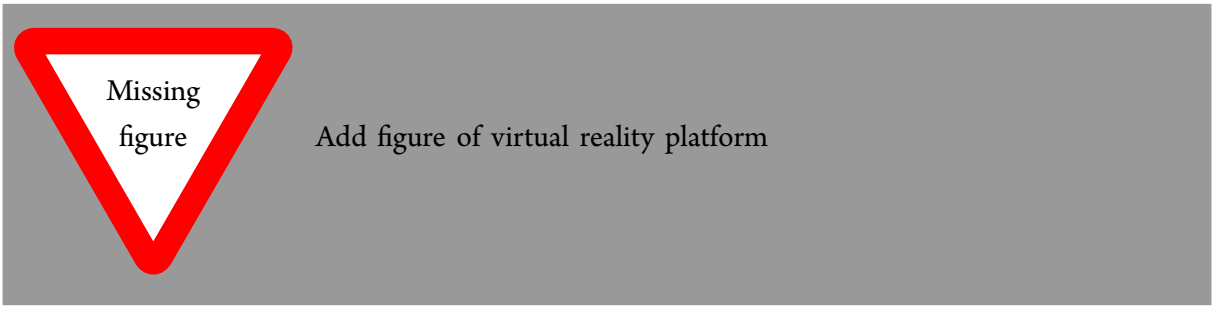
Figure showing point model and selected points

While sampling will tend to produce noisier tracking results for low N_s, it also greatly reduces the computational complexity. This allows one to greatly increase the number of particles for a given desired frame-rate. In the results presented next we shall demonstrate that this trade-off allows us to significantly increase accuracy for a given frame-rate and reduces run-time complexity to the point that we can track 6 DoF pose for multiple objects in real-time.

4.6 Results on Virtual-Reality Data

Before we present results on real data, we shall first quantify tracker performance using Virtual-Reality (VR) data. VR data is primarily useful because it allows us to easily obtain accurate ground-truth data for object tracks and poses - something that is generally not possible with real data [**?**]. With this in mind we have developed a VR setup and benchmark [**?**] which simulates the platform we shall use in the real data shown later. The virtual platform has many simulated cameras, but in these tests we use

two simulated Kinect RGB-D sensors with overlapping fields of view covering the entire work area. The sensor simulation can generate benchmark images and point clouds with controlled levels of quality, from "ideal" to "real". [1] In the following tests we used the standard depth error of the Kinect as our lowest noise level and compared the algorithms with less accurate testdata.



Once again, our benchmarking task is assembly of the well-established "Cranfield" set [16, 33, 42] (see Fig. **??**).The actual assembly actions in VR are carried out and tracked with a data-glove, generating accurate, objective ground-truth data, e.g. exact object positions as well as detailed information on the timing and existence of object manipulations and spatial relations between manipulated objects. Comparisons of real and virtual images were made in the project FastMap [**?**] and showed that artificial and real images led to very similar result in computer vision algorithms.

In our first experiment, we shall demonstrates the effectiveness of our stratified sampling strategy. We used a simple scenario in which we must track two bolts as they are picked up and inserted into a faceplate. To simplify the analysis, we fix the run-time variable by setting a desired frame-rate of 20fps [2], and then determined experimentally the number of particles for which this frame rate was sustainable at different degrees of sampling. Degree of sampling is simply the ratio of the number of points sampled to the number of points present in the model, $\frac{N_s}{N_m}$. Table 4.6.1 gives the average frame rates and standard deviation measured during the runs. Figure **??** shows average error rates for the different degrees of sampling. Additionally, Figure **??** shows the affect on error and run time of increased particle number for fixed degrees of sampling. Finally, Figure 4.6.1 gives an example of what the tracked output looks like versus ground truth - it is evident that the tracker is able to follow the objects quite well in terms of translation. Rotations, on the other hand, are much noisier, due to the rotational symmetries present in the tracked object (the bolt).

[1] Here, "real" is defined by the similarity of outcomes when real and simulated data are processed by libraries such as OpenCV and PCL, e.g. color histograms (RGB deviation, RGB saturation), edge detection, SURF feature detection and RANSAC feature similarity.

[2] All these experiments were run on the same machine - an i7 980x with 32g of memory

Table 4.6.1: Measured frame rates of tracker during tracking of 2 bolt VR scenario.

Degree of sampling	1.0	0.8	0.6	0.4	0.2	0.1
Particles	0	0	0	0	0	0
Avg time per frame (ms)	0	0	0	0	0	0
Std. Deviation (ms)	0	0	0	0	0	0
Avg Framerate (fps)	0	0	0	0	0	0

Screenshots from this scenario

Plot showing track error vs degree of sampling. Also show number of particles on y axis. Fixed run-time.

Plot showing track error vs degree of sampling. Also show number of particles on y axis. Fixed run-time.

Figure 4.6.2 demonstrates one intended application of the tracking - recognizing the different actions of the Cranfield assembly sequence. In the sequence all consistently tracked objects are represented by graphs in which nodes represent manipulated objects and edges indicate whether two objects touch each other or not. We can discretize the graph sequence into decisive key frames, which occur whenever a new node or edge is formed or an existing edge or node is deleted. Sequences of all extracted key frames are employed for measuring semantic similarities, yielding action recognition as described in [**? ?**]. Fig. 4.6.2 depicts sample key frames with tracked objects (each is indicated with a different color) and

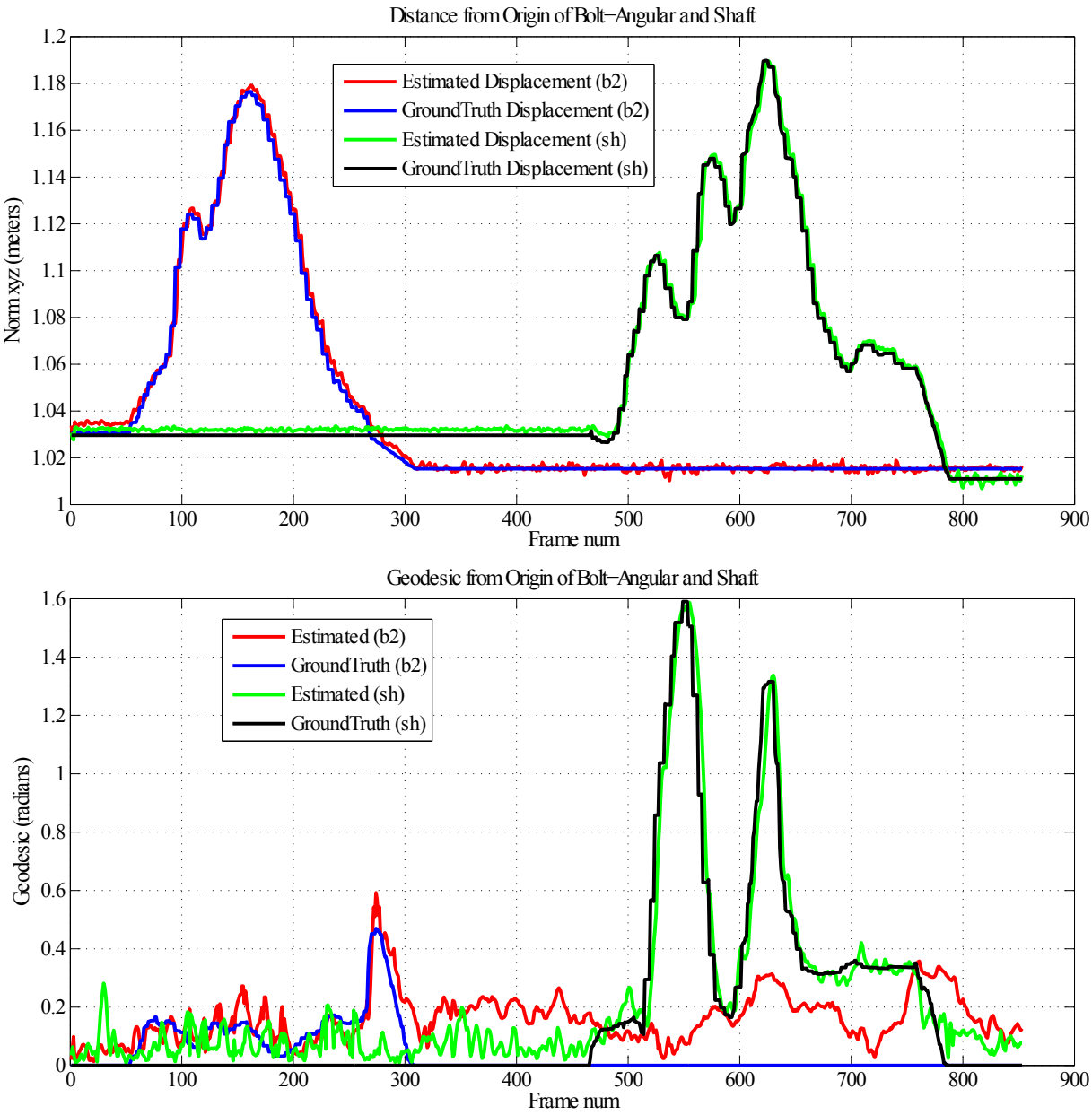


Figure 4.6.1: Tracked Output vs Ground Truth Artificial Sequence. The top panel shows position in terms of XYZ displacement and the bottom shows rotation in terms of geodesic. The location is generally tracked quite well, while the rotation is noisy due to the rotational symmetry of the two tracked objects (a bolt-angular and shaft).

corresponding graphs at which a new action is recognized.

4.7 Results on Real Data

In this section we present results on 10 different recordings of humans assembling the Cranfield benchmark. Recordings were made on the MARVIN platform at the University of Southern Denmark, and

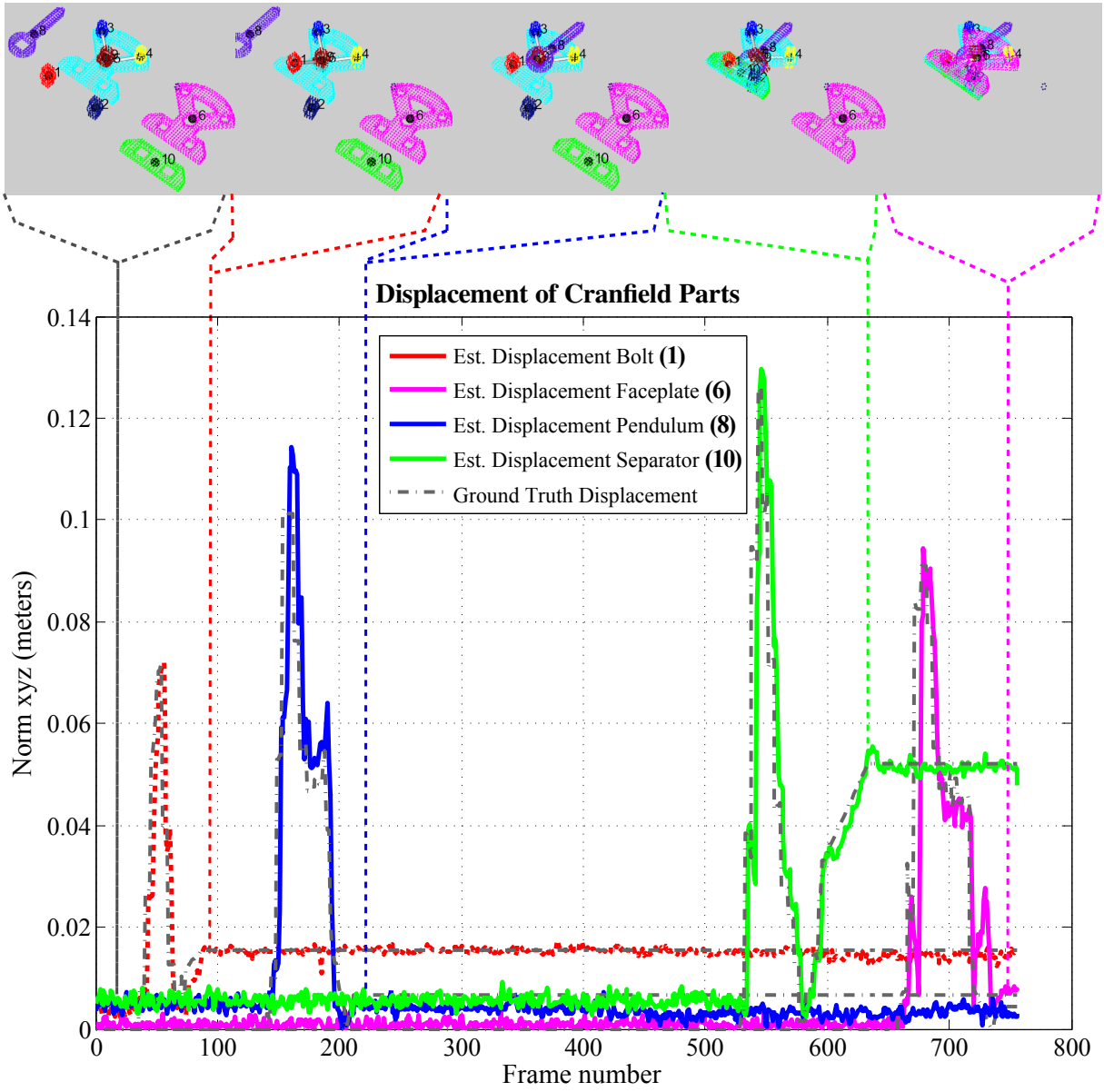


Figure 4.6.2: Segmentation of Cranfield Sequence into Keyframes - Tracked objects are monitored for when interactions between them occur, yielding keyframes which correspond to semantically important frames. Results here are shown for an artificial sequence with depth and RGB noise added.

use 2 Kinect RGB-D cameras [3], positioned in the same place as in the VR simulation. The 10 recordings consist of 2 assemblies each by 5 different people, where the people were following assembly instructions presented by the planning system of the IntellAct project. A description of the planner is beyond the scope of this work (we refer the reader to **??**), but for our purposes we just need to know that the order of assembly is random.

As we do not have ground truth poses for this dataset, we evaluate tracker performance by comparing

[3]It is well-known that multiple Kinect sensors sharing a common field of view will cause IR interference, resulting in poor depth reconstructions. A known solution, which the platform incorporates, is the use of vibrating motors mounted on the Kinect sensors [12]. This method has been shown to effectively blur out the noisy contributions of external sensors, while maintaining a high depth reconstruction quality.

actual human actions to recognized actions. That is, we check that the final tracked positions for the objects is the correct one for the assembly task.

Add results from these experiments

Some Quote.

Quoteauthor Lastname

5

Tracking Based Point Cloud Video Segmentation

So far, we have presented a 2D particle-filter based VOS method, developed a 3D point-cloud based world-model, and shown how it is possible to track within this model using particle filters. What remains is to bridge the gap between the tracked model results presented in the previous Chapter, and the supervoxel model presented in the preceding one. There are many avenues available through which to proceed in doing this, and which one is optimal remains an open question. In this Chapter we shall describe the avenue which we pursued, show our results, and discuss where future research should lead.

Before we begin, it will probably be helpful if we show visually exactly what we are trying to achieve. Figure **??** gives such an outline; the core idea is that we want to have a full segmentation of each frame. While the tracking we have presented estimates a 6DoF pose for models in each frame, we now want to label every observed voxel (or supervoxel - which amounts to the same thing). Moreover, we want to use tracking as the basis for this labeling, due to its ability to maintain object identities through occlusions, sudden movements, and other difficult situations.

Figure showing supervoxels, tracked models, associations to achieve segmentation

5.1 Tracked Model Representation

The first issue that must be addressed is deciding at what "level" the tracking should be done - at an object level, or at the supervoxel level. While ideally one could directly track supervoxels themselves, this is generally not feasible due to the aperture problem seen in neural visual fields [32]; local motion

can only be estimated perpendicular to a contour that extends beyond its field of view [44]. This means that in order to properly estimate motion of supervoxels, we must extend the field of view considered significantly beyond the size of the supervoxel itself; in fact, our aperture must contain the borders of the object in question, otherwise pairwise association of supervoxels is indeterminate.

Figure showing aperture problem?

As such, we must track higher level groupings - that is to say either objects, or at least object parts. For the experiments we present later in this chapter, we use a simple plane fitting and removal algorithm to remove supporting surfaces, followed by a euclidean clustering of the remaining supervoxels as in [39]. This was done to simplify the experiments, though we should stress that any clustering method could be used to initialize the tracked object models, and indeed, one avenue of current research is using the LCCP segmentation presented in Chapter 3 to initialize (and perhaps re-initialize) the models. In any case, regardless of the segmentation used, the supervoxel clusters are used to initialize the models which will be tracked.

Similar to the previous Chapter, models consists of clouds - though now we use supervoxels rather than voxels, as seen in Figure **??**. Other than this consideration, the models are essentially identical to those given in Equations 4.1 and 4.2. Just as each voxel in those equations is an average of the many points contained within the voxel, now each supervoxel's centroid is the average of all the voxels it contains. The main difference between the two on a local level is that supervoxels are not uniform in shape. Because of this, an important component of the supervoxel-based model which is not considered in the voxel version is the supervoxel connectivity graph. Additionally, supervoxels also have the additional information provided by a label - this is particularly important for non-moving objects, whose supervoxel labels will generally not change.

Figure showing points, voxels, and supervoxels (with connectivity) in a model.

5.2 Supervoxel-Based Particle Filters

Tracking of the segmented models is accomplished using a bank of independent parallel particle filters, just as in the previous chapter. As our models now consist of supervoxels, so too must our observations - thus we use the supervoxels produced using the persistent scheme discussed in Chapter 3. The observation model measures distance in a feature space of spatial distance, normals, color (in HSV space), and labels. Weights of predicted states $\mathbf{x}_t^j = [d_x, d_y, d_z, \gamma, \beta, a]$ are measured by associating supervoxels from the transformed models to the observed supervoxels nearest in space. Particles are then weighted by measuring total distance in feature space, just as in (4.10), with the addition of a binary label term,

$$W_L = \begin{cases} 1, & L_p = L_{p^*} \\ \frac{N_k - 1}{N_k}, & L_p \neq L_{p^*} \end{cases} \tag{5.1}$$

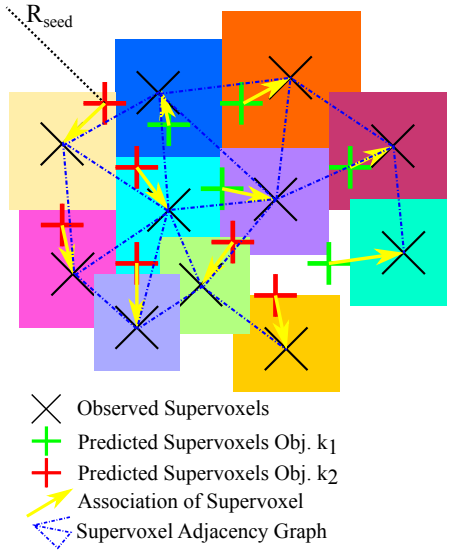


Figure 5.3.1: Association of observed supervoxels with predicted model supervoxels using global energy.

which results in the augmented distance function

$$\tilde{w}^j = \sum_{1}^{\eta} \frac{1}{1 + \frac{\mu \|\mathbf{p}^j_{xyz} - \mathbf{p}^*_{xyz}\|}{R_{voxel}} + \frac{\lambda D_c(p^j_{RGB}, p^*_{RGB})}{m} + \varepsilon \|\mathbf{p}^j_{n_x n_y n_z} - \mathbf{p}^*_{n_x n_y n_z}\| + \nu W_L}. \tag{5.2}$$

As before, we adopt the notation p for the supervoxel and $p*$ for its corresponding supervoxel in the observation.

KLD sampling [21] is used to dynamically adapt the number of particles to the certainty of predictions. As matching supervoxel labels gives a high certainty of a correct prediction, objects which are not moving, and therefore have static supervoxel labels, need very few particles for accurate tracking. The details of KLD sampling are beyond the scope of this work, but we refer the reader to [21] for an in-depth description of their operation. Outside of KLD sampling and the differences in the measurement function noted above, the particle filters function just as described in the previous chapter. The end result at each time-step are independent predictions of 6DoF object state, allowing a transformation of the model roughly aligning it with the currently observed supervoxels.

5.3 Association by Joint Energy Minimization

The additional step that we must take to extract a full segmentation (rather than only object tracks) is to associate the observed supervoxels to the predictions coming from the particle filters. Therefore we must solve the multiple target data association problem. This is accomplished using an energy minimization which seeks to find an optimal global association of supervoxels to predictions. To do this, we first create a list of all observed supervoxels which lie within a radius R_{seed} of each predicted supervoxel coming from the particle filters (see Fig. 5.3.1). Then we select all supervoxels which could only

be associated with one possible object, associate them, and remove them from further consideration.

To associate the remaining observed supervoxels, we determine which objects are competing for them, and then find the predicted supervoxel from each object which lies closest to them in the feature space (using spatial location, normals, and color as in (**??**)). We adopt a RANSAC-like approach, similar to [25], to sample from the set of possible associations and determine a global association which best aligns the predictions to the observed supervoxels. Additionally, we use a weighted sampling strategy where the likelihood of assigning object k as the label L of supervoxel p falls off with increasing distance from the object centroid C_k

$$\mathcal{L}(L_p = k|C_k) = \frac{1}{C_k}. \tag{5.3}$$

To score a set of assignments, we compute a global energy, given in (5.4). Each global label association $\mathcal{A}$ consists of local associations a which assign an object label k to each observed supervoxel p. The first summation term, $\sum_p \|p_k^* - p\|$, measures error in feature space between the observed supervoxel and the closest supervoxel in its associated predicted object p_k^*.

$$E_{\mathcal{A}} = \prod_{a \in \mathcal{A}} \Delta_k \left(\sum_p \|p_k^* - p\| + \lambda \sum_{(p,p') \in \mathcal{N}} \delta(L_p \neq L_{p'}) \right) \tag{5.4}$$

The second summation is a smoothing prior which considers the adjacency graph of observed supervoxels. For every observed supervoxel, we compare its assigned label L_p to the label of all supervoxels p' which lie within its adjacency neighborhood $\mathcal{N}$. We adopt the Potts model as in [9], where $\dot{\delta}()$ is 1 if the specified condition holds, and 0 otherwise, and λ is a weighting coefficient which controls the importance given to spatial continuity of labels.

Finally, the multiplicative term $\prod_{a \in \mathcal{A}} \Delta_k$ controls for the expansion or contraction of object volumes through the number of observed supervoxels associated with them. Δ_k penalizes for changes in volume by increasing the energy for deviations from unity in the ratio of observed supervoxels assigned to an object $\hat{N}_k$ with the number in the object model itself $\hat{N}_k$, that is

$$\Delta_k = \begin{cases} \hat{N}_k / N_k & \text{if } \hat{N}_k \geq N_k \\ 2 - \hat{N}_k / N_k & \text{if } \hat{N}_k < N_k \,. \end{cases} \tag{5.5}$$

Once the energy arrives at a stable minimum, we extract the resulting association of observed supervoxels to predicted results, and use them to update the tracked models.

5.4 Alignment and Update of Models

The joint energy minimization results in a global association $\mathcal{A}$ which assigns observed supervoxels to tracked objects. In order to use this to update the object models, we determine a transform which aligns it to the internal representation stored by the particle filter. As an initial guess, we use the inverse of the predicted state, and then use an iterative closest point [14] procedure to refine the transform such that

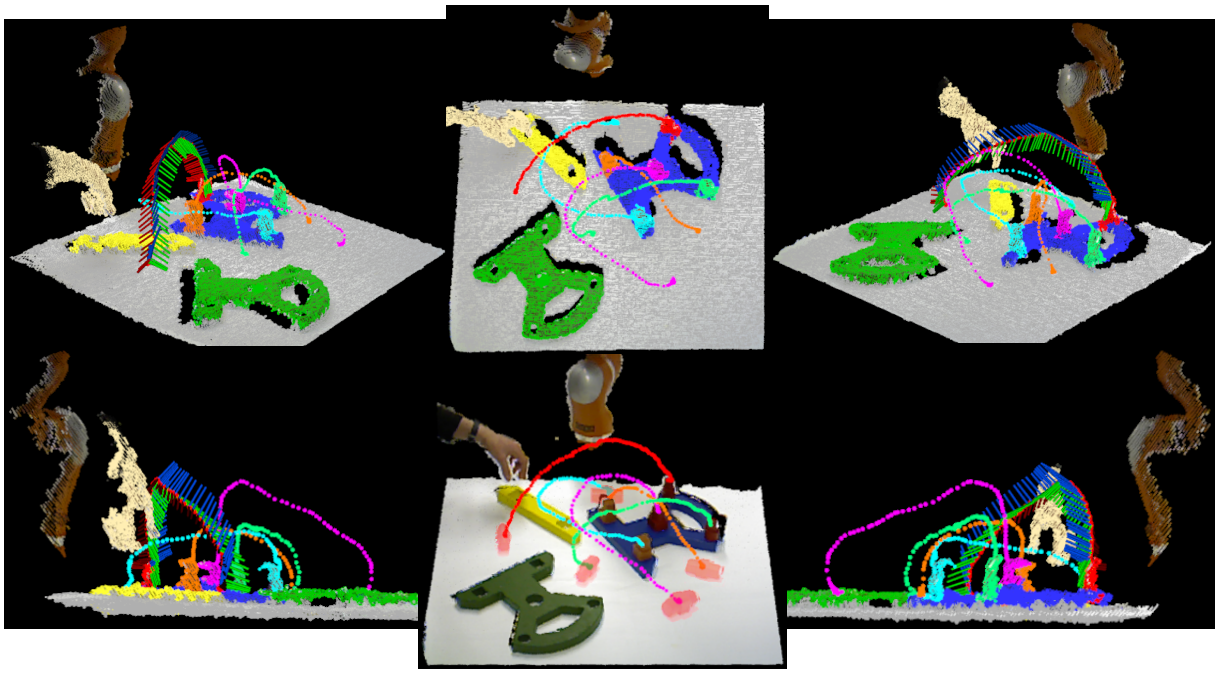


Figure 5.3.2: Result of tracking and segmentation on Cranfield scenario from different views. Here the tracks are shown as dots of the color of the tracked label for each timestep. Initial locations of the pegs are shown in the middle bottom frame as semi-transparent masks. Calculated orientation is shown for the red peg with a set of axes every second time-step; these axes show pose in a frame relative to the start.

the set of observed supervoxels best aligns with the model prior. We then replace the model prior with the new observed supervoxels.

As a final step, we use the refined transform to update the states of the particles. To do this, we shift each particle x_i towards the refined state $\hat{x}$, weighting the importance given to the refined state by a constant factor ε

$$x'_{i \in L} = (1 - \varepsilon)x_i + \varepsilon\hat{x}\,. \tag{5.6}$$

For this work, we found that an ε of 0.5 effectively removes noise (jitter) introduced by the replacement of the tracked model. Additionally, we correct the internal motion model of the particle filters to correspond to the new updated state.

5.5 Experimental Results

In order to demonstrate the usefulness of the proposed method, in this Section we first provide results from two successful applications. Both applications use the Cranfield scenario [15] as in the previous chapter. Figure 5.3.2 show the results of tracking and segmentation (only the pegs are shown in Figure 5.3.2 to avoid clutter) using our Cranfield pieces. It can be seen that the algorithm is able to successfully extract full segmentation throughout the video.

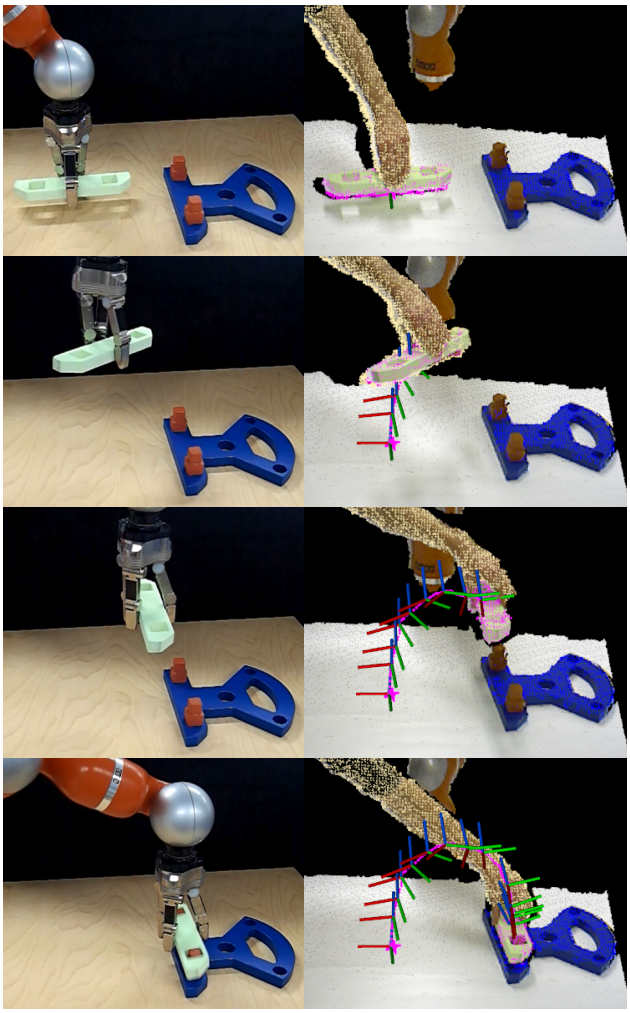


Figure 5.5.1: Kuka LWR arm imitating trajectory and pose learned from tracked human demonstration.

5.5.1 Imitation of Trajectories for Robot Manipulation

The standard way of teaching robots to perform human-like actions is imitation learning, also called programming by demonstration [**? ?**]. There are several ways to demonstrate movements: 1) recording movements in joint-space (joint angles) or target-space (Cartesian space) by ways of a motion capture device (requires putting markers on human body), 2) using kinaesthetic guidance (guiding a robot's movements by a human hand), or 3) via teleoperation (controlling a robot via joystick). The only way to obtain motion trajectories from human observation in a "non-invasive" procedure is by using stereo vision [**?**], however, usually it is model based. The tracking algorithm we have presented here can be used as an alternative method to obtain motion trajectories (in Cartesian space) in a model-free way.

To demonstrate this, we applied our tracking algorithm to obtain human motion trajectories in Cartesian space including orientation of manipulated object (in total six DoFs). We tested it using a recording of the Cranfield scenario where, first, we let a human demonstrate the action and then reproduced it using a KUKA Light Weight Robot (LWR) arm [**?**]. Specifically, here we imitate a human putting the separator block on the pegs. To generate trajectories for the robot from human demonstrations, we

used a modified version of Dynamic Movement Primitives [**?** **?**] (DMP) and learning method as described in [**?**]. We used Cartesian impedance control and, thus, generated six DMPs (three for motion of the end-effector in Cartesian space and three for orientation of the hand) based on trajectories obtained from the tracking algorithm. Here we used 100 equally spaced kernels with width $\sigma = 0.05$ for each dimension (for more details please refer to [**?**]). As demonstrated in Fig. 5.5.1 and the supplementary video, trajectories obtained by the proposed tracking algorithm are sufficiently accurate to allow reproduction of the human motion.

5.5.2 Semantic Summaries of Actions

Need to show results on the Odense recordings instead, but how to show it makes a difference. Maybe use it to do object classification at the end based on what the tracked action was? Or just say we generate SECs without needing object models.

A fundamental task for intelligent autonomous robots is the problem of encoding long chain manipulations in a generic way, for use in tasks such as learning and recognition. As a demonstration of the usefulness of the proposed tracking framework, we use a recently introduced novel Semantic Event Chain (SEC) approach [6] which converts each segmented scene to a graph: nodes represent segment (i.e. object) centers and edges indicate whether two objects touch each other or not. By using an exact graph matching technique the SEC framework discretizes the entire graph sequence into decisive main graphs. A new main graph is identified whenever a new node or edge is formed or an existing edge or node is deleted. Thus, each main graph represents a "key frame" in the manipulation sequence. Figure 5.5.2 shows a few detected sample key frames from the long Cranfield action. While the complete action has in total 1453 frames, the SEC representation reduces it to just 35 key frames, each of which represents a topological change in the scene.

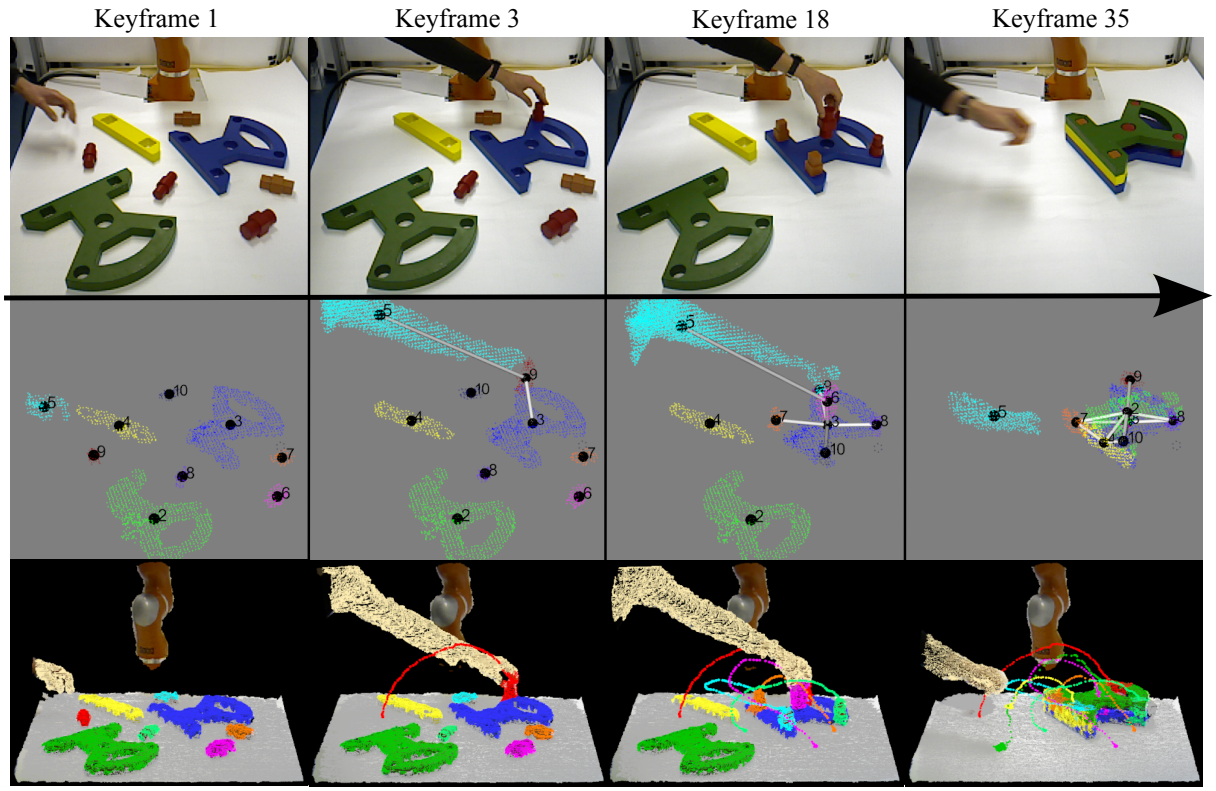


Figure 5.5.2: A few example key frames extracted from the long Cranfield action. Numbered nodes represent interacting objects, while edges show touching relations between objects. Each keyframe represents a topological change in the scene - here we show 4 of the 35 keyframes.

Some Quote.

Quoteauthor Lastname

6 Conclusions

Throughout this work, we have had one goal in mind; to develop video segmentation which has the reliability of tracking methods. The primary reason for doing this was to ensure the temporal consistency of segments through extended video clips of, in particular, indoor assembly tasks. Difficulties presented in such videos include partial and full occlusions, sudden and fast displacements of objects, different objects with similar or identical appearance, and objects which cannot be segmented based on color. Additionally, assembly tasks require a high degree of precision, particularly when it comes to the relative pose of parts when they are interacting (e.g. putting a bolt in a hole).

Our intended application for the segmentation of such videos was the general bootstrapping of assembly task understanding. If one can correctly track objects and their parts through a task without a-priori knowledge, it should be possible to use this to teach a robotic system from scratch. Not only this, but if one is able to correctly track full 6 DoF pose throughout a human-demonstrated assembly task, it is possible to learn trajectories that a robot can use to directly imitate effective motion paths.

6.1 Summary of Contributions

We began in chapter 2 by presenting a 2D VOS method that made use of our proposed methodology; to use tracking as the basis for video segmentation. In particular, we showed how particle filters, a class of Sequential Bayesian Monte Carlo methods, can be used to predict what the next frame's segmentation should look like. These proposed segment masks were then combined using a weighted sampling

strategy and then refined to fit observed image data using a relaxation process.

Next, in chapter 3 we introduced RGB-D sensors, how they can be used to produce 3D point cloud data, and how this data can be organized using an octree structure. We then presented a specialized type of octree - the adjacency octree - which we developed to allow quick and efficient traversal between neighboring voxels within the tree. We subsequently showed how the adjacency octree can be used to efficiently further sub-divide voxel data into localized patches, called supervoxels, using our Voxel Cloud Connectivity Segmentation (VCCS). The utility of supervoxels was then demonstrated by showing their effectiveness in segmenting static scenes using a local convexity criterion (LCCP). The effectiveness of VCCS and LCCP were then demonstrated by showing their favorable results on a large state-of-the-art benchmark as compared to other state-of-the-art methods. Finally, we conclude the chapter by presenting how an adjacency octree containing supervoxels can be sequentially updated with new frames of data without deleting potentially occluded voxels.

We then extended the particle filter framework in chapter 4 to 3D point clouds by formalizing the notion of a voxel-based measurement and dynamic model. While this straightforward implementation worked, we showed how it could achieve much faster (real-time on standard hardware) run times and accuracy by sampling correspondences. This improvement was then quantified using artificial data generated using a virtual reality simulator. As a final demonstration of the effectiveness of the tracker, we presented results on recordings of humans constructing the Cranfield benchmark. This showed how the tracker can be used to distill semantic understanding from a video sequence.

Finally, in chapter 5 we tackled the problem of extracting full segmentations from tracked results. To do this, we first showed how the 3D particle filter presented previously could be extended to work on supervoxels. Then we showed how the adjacency graph of supervoxels could be used along with a global energy minimization to resolve interactions between trackers and produce a full segmentation consistent with the tracked poses. As a final demonstration of the presented methodology, we give results on several recordings of manipulation actions.

Another important contribution was the development of the open-source Oculus vision system discussed in Appendix A. This system served as the platform on which much research has been done over the past several years and was a key tool in publications by several other researchers. Finally, we would like to note that most of the algorithms discussed in this work have been released as open-source to the vision community as part of the Point Cloud Library [1]. We consider both of these important contributions, as the open sharing of code is vital to the advancement of the discipline of Computer Science.

6.2 Shortcomings of VOS Benchmarks

Evaluation of segmentation algorithms is a notoriously vexing problem due to the inherent ambiguity of what constitutes a "correct" segmentation. As such, in our work we have determined that we should avoid making concrete decisions on segmenting objects in single works, and instead chose to limit our-

[1] http://www.pointclouds.org/

self to the lower, supervoxel level. While this is not an entirely satisfactory solution, we felt that there is simply not enough information in a single frame to extract meaningful segmentations accurately. Indeed, one cannot really tell the granularity with which a scene should be segmented into distinct objects until they see some action.

Benchmarking of 3D point cloud segmentation is a young topic - in fact, the first extensive benchmark, the *NYU Indoor Dataset*, was not published until 2012 [**?**]. As such, it has many complications (that did not exist in 2D) which have yet to be resolved, such as that it is difficult, if not impossible, to make a 2D ground truth annotation correctly line up with the 3D point cloud representation. Even more to the point, there are still no 3D VOS benchmarks. In fact, even though the field is decades old, one must look to 2013 to find a 2D VOS benchmark [22]. There are many reasons for this lack of a proper benchmark, but the primary one is that is simply extremely time consuming to annotate ground-truth for even very short video sequences. Furthermore, labeling a single ground truth is even more difficult for video than single images, for instance, what happens when one takes a cap off of a bottle; should it be given a new label? If so, should it have had a separate label the entire time, or only once it is separated? What happens when objects become occluded and then reappear; should they be given new identities or maintain their old ones? If they keep their old ones, how long should we allow an object to be occluded for before we "forget" it?

Due to all of these concerns, we have made the decision in this work to benchmark our video segmentations by evaluating their ability to properly recognize actions. This, combined with qualitative results, effectively show the effectiveness of the method without the need to haggle over inscrutable questions such as "what constitutes an object"?

6.3 Limitations and Direction of Future Work

The main limitation of the 2D tracking framework presented in chapter 2 is that it can only "guess" at correct behavior when an object is occluded. Indeed, this is a general problem of trying to infer behavior in a 3D world from observations in a 2D projected plane. It is because of these ambiguities and an inability to resolve them in a comprehensive and satisfying manner that we proceeded to tracking in 3D using RGB+D observations.

While our persistent voxel world model is very effective at maintaining the existence of stationary objects through occlusions, it does not handle objects which move while they are occluded. Solving this problem at the low-level of voxels is an unresolved problem, the outlook of which is fairly bleak. Our attempts at solving the problem lead us to believe that higher level object knowledge is necessary to account for occluded motion. With this in mind we are investigating a way of associating occluded objects with their occluder so that they move with them. Another limitation which we are currently addressing is that our persistent voxel world model does not account for camera motion. There is some existing work on real-time camera pose estimation, and we are hoping to incorporate such a method into our system in the near future.

As with any VOS method, our final result has a few important limitations. One of these is the need

to set a rate at which to allow models to change. In many cases, objects are rigid, and do not change; allowing them to change only adds instability to the segmented output. Conversely, some objects are deformable, or not entirely visible in the scene, and will need to change to be correctly segmented. Another limitation is that we currently have no mechanism for adding new objects to the scene without reinitializing the tracking. Our current work focuses on correcting both of these limitations by incorporating the results of LCCP segmentation into the global minimization process to allow for parameter-free changing of models as well as the automated initialization of new tracks.

Bibliography

[1] V. Ablavsky, A. Thangali, and S. Sclaroff. Layered graphical models for tracking partially-occluded objects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[2] A. Abramov, K. Pauwels, J. Papon, F. Worgotter, and B. Dellen. Real-time segmentation of stereo videos on a portable system with a mobile gpu. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(9):1292–1305, Sept 2012. ISSN 1051-8215. doi: 10.1109/TCSVT.2012.2199389.

[3] A. Abramov, K. Pauwels, J. Papon, F. Worgotter, and B. Dellen. Depth-supported real-time video segmentation with the kinect. In *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*, pages 457–464, Jan 2012. doi: 10.1109/WACV.2012.6163000.

[4] Alexey Abramov, Eren Erdal Aksoy, Johannes Dörr, Florentin Wörgötter, Karl Pauwels, and Babette Dellen. 3d semantic representation of actions from efficient stereo-image-sequence segmentation on gpus. In *International Symposium 3D Data Processing, Visualization and Transmission*, 2010.

[5] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[6] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research*, 30(10): 1229–1249, 2011.

[7] B. Babenko, Ming-Hsuan Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[8] M. Blatt, S. Wiseman, and E. Domany. Superparamagnetic clustering of data. *Physical Review Letters*, 76(18):3251–3254, 1996.

[9] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Machine Intell.*, 23(11):1222–1239, Nov 2001. ISSN 0162-8828. doi: 10.1109/34.969114.

[10] M.D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(9):1820–1833, Sept 2011.

[11] W. Brendel and S. Todorovic. Video object segmentation by tracking regions. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[12] D Alex Butler, Shahram Izadi, Otmar Hilliges, David Molyneaux, Steve Hodges, and David Kim. Shake'n'sense: reducing interference for overlapping structured light depth cameras. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, pages 1933–1936. ACM, 2012.

[13] Yizheng Cai, Nando Freitas, and JamesJ. Little. Robust visual tracking for multiple targets. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, volume 3954 of *Lecture Notes in Computer Science*, pages 107–118. Springer Berlin Heidelberg, 2006. doi: 10.1007/11744085_9.

[14] Dmitry Chetverikov, Dmitry Stepanov, and Pavel Krsek. Robust euclidean alignment of 3d point sets: the trimmed iterative closest point algorithm. *Image and Vision Computing*, 23(3):299 – 309, 2005. ISSN 0262-8856. doi: 10.1016/j.imavis.2004.05.007.

[15] K Collins, AJ Palmer, and K Rathmill. The development of a european benchmark for the comparison of assembly robot programming systems. In *Proceedings of the 1st Robotics Europe Conference, Brussels*, pages 27–28, 1984.

[16] K. Collins, A. J. Palmer, and K. Rathmill. The development of a European benchmark for the comparison of assembly robot programming systems. In *Robot technology and applications (Robotics Europe Conference)*, pages 187–199, 1985.

[17] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2000.

[18] Arnaud Doucet, Nando De Freitas, and Neil Gordon, editors. *Sequential Monte Carlo methods in practice*. 2001.

[19] T.E. Fortmann, Y. Bar-Shalom, and M. Scheffe. Multi-target tracking using joint probabilistic data association. In *Decision and Control including the Symposium on Adaptive Processes, 1980 19th IEEE Conference on*, volume 19, pages 807–812, Dec 1980.

[20] Thomas E. Fortmann, Y. Bar-Shalom, and M. Scheffe. Sonar tracking of multiple targets using joint probabilistic data association. *Oceanic Engineering, IEEE Journal of*, 8(3):173–184, Jul 1983.

[21] Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *The International Journal of Robotics Research*, 22(12):985–1003, 2003. doi: 10.1177/0278364903022012001. URL http://ijr.sagepub.com/content/22/12/985.abstract.

[22] Fabio Galasso, Naveen Shankar Nagaraja, Tatiana Jiménez Cárdenas, Thomas Brox, and Bernt Schiele. A unified video segmentation benchmark: Annotation, metrics and analysis. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, 2013.

[23] V. Hedau, H. Arora, and N. Ahuja. Matching images under unstable segmentations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, june 2008.

[24] C. Hue, J.-P. Le Cadre, and P. Perez. Tracking multiple objects with particle filtering. *IEEE Transactions on Aerospace and Electronic Systems*, 38(3):791 – 812, jul 2002.

[25] Hossam Isack and Yuri Boykov. Energy-based geometric multi-model fitting. *International Journal of Computer Vision*, 97:123–147, 2012. ISSN 0920-5691. doi: 10.1007/s11263-011-0474-7.

[26] Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[28] K. Lai, Liefeng Bo, Xiaofeng Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817 –1824, may 2011. doi: 10.1109/ICRA.2011.5980382.

[29] O. Lanz. Approximate bayesian multibody tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(9):1436–1449, Sept 2006.

[30] Siying Liu, Guo Dong, Chye Hwang Yan, and Sim Heng Ong. Video segmentation: Propagation, validation and aggregation of a preceding graph. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[32] D. Marr and S. Ullman. Directional selectivity and its use in early visual processing. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 211(1183):151–180, 1981. doi: 10.1098/rspb.1981.0001.

[33] D. Martinez, G. Alenya, P. Jimenez, C. Torras, J. Rossmann, N. Wantia, E. E. Aksoy, S. Haller, and J. Piater. Active learning of manipulation sequences. In *IEEE Int Conf Robotics and Automation (ICRA)*, 2014.

[34] Dennis Mitzel and Bastian Leibe. Taking mobile multi-object tracking to the next level: People, unknown objects, and carried items. In *Computer Vision – ECCV 2012*, volume 7576 of *Lecture Notes in Computer Science*, pages 566–579. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33714-7.

[35] N. Papadakis and A. Bugeau. Tracking with occlusions via graph cuts. *IEEE Trans. Pattern Anal. Machine Intell.*, 33(1):144 –157, Jan. 2011.

[36] Sylvain Paris. Edge-preserving smoothing and mean-shift segmentation of video streams. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *European Conference on Computer Vision (ECCV)*, volume 5303 of *Lecture Notes in Computer Science*, pages 460–473. Springer Berlin / Heidelberg, 2008.

[37] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *European Conference on Computer Vision (ECCV)*, volume 2350 of *Lecture Notes in Computer Science*, pages 661–675. Springer Berlin / Heidelberg, 2002.

[38] D.B. Reid. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on*, 24(6):843–854, Dec 1979.

[39] R.B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4, May. doi: 10.1109/ICRA.2011.5980567.

[40] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2009.

[41] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof. Prost: Parallel robust online simple tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[42] C. Schou, C. F. Carøe, M. Hvilshøj, J. S. Damgaard, S. Bøgh, and O. Madsen. Human assisted instructing of autonomous industrial mobile manipulator and its qualitative assessment. In *AAU Workshop on Human-Centered Robotics*, pages 22–28, 2012.

[43] D. Schulz, W. Burgard, D. Fox, and A.B. Cremers. Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1665–1670 vol.2, 2001.

[44] Shinsuke Shimojo, Gerald H Silverman, and Ken Nakayama. Occlusion and the solution to the aperture problem for motion. *Vision research*, 29(5):619–626, 1989.

[45] Amelio Vazquez-Reina, Shai Avidan, Hanspeter Pfister, and Eric Miller. Multiple hypothesis video segmentation from superpixel flows. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *European Conference on Computer Vision (ECCV)*, volume 6315 of *Lecture Notes in Computer Science*, pages 268–281. Springer Berlin / Heidelberg, 2010.

[46] J. Vermaak, Arnaud Doucet, and P. Perez. Maintaining multimodality through mixture tracking. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1110–1116 vol.2, Oct 2003.

[47] J. Vermaak, S.J. Godsill, and P. Perez. Monte carlo filtering for multi target tracking and data association. *IEEE Transactions on Aerospace and Electronic Systems*, 41(1):309 – 332, jan. 2005.

[48] B.-N. Vo, S. Singh, and A. Doucet. Sequential monte carlo methods for multitarget filtering with random finite sets. *IEEE Transactions on Aerospace and Electronic Systems,*, 41(4):1224 – 1245, oct. 2005.

[49] Bo Wu and Ram Nevatia. Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors. *International Journal of Computer Vision*, 75(2):247–266, 2007. ISSN 0920-5691.

Appendices

A The Oculus Vision System

A.1 Motivation

There is great interest in development of complex vision systems for robotic vision applications. Such research has strict requirements; these systems must operate in real-time, using input from multiple sources, and typically consist of multiple algorithms which work in concert to produce useful output with minimal delay. Consequently, the architecture which binds algorithms and input sources together has become an increasingly important factor. In this Appendix we shall present a vision architecture we developed over the course of the thesis work which uses modular plugins, a novel buffering scheme, and GPU memory optimizations to allow real-time performance of an online vision system, even with complex pipelines and algorithms developed by independent researchers.

A primary concern when developing such complex vision systems lies in how to properly integrate algorithms developed by different researchers, often from multiple institutions. Typically, computer vision researchers develop solutions tailor-made for their particular problem, without concern over the difficulties involved in integrating their particular algorithm into a large system. To ease this integration process, we provided a plugin interface. The plugin system allows independently developed algorithms to communicate with the architecture's central memory management system, interact with the GUI, define their own unique data types, and integrate into systems with plugins developed by other researchers.

Another motivation for developing a vision architecture is the desire to enable the use of complex algorithmic layouts in an online system. In particular, interest in creating loops that allow high level algorithms (i.e. which come late in the pipeline) to feedback and improve the output of low level vision methods. Traditional online vision pipeline architectures cannot accommodate such loops in an adequate way, as at any given moment each portion of the pipeline is processing data from different instants in time.

Existent vision system architectures also do not support the use of GPUs in a fully integrated way, leading to inefficient use of the device and communication with device memory. The presented method incorporates specially designed GPU data-containers to ensure optimal PCI-bus use through a pre-caching scheme and concurrent memory transfers. In addition to these, extensibility is ensured through an interface which allows user-defined data-container handling, allowing plugin developers to explicitly define how the memory manager shares data between the host and device. In this Appendix we will present an overview of our system, describe a typical system configuration used for robotics, and then

give performance figures from a demonstration setup.

A.2 System Architecture

Our vision system is a plugin shell which provides an easy-to-use API for interacting with the GUI, memory management system, and visualization components. In order to ensure expandability, such a system must provide straightforward communication and interaction between plugins created independently, while employing strong-typing checks to ensure only valid plugins may be inter-connected. In addition, it must ensure that plugins have the flexibility to define their own methods for visualization. Finally, the system must ensure that each plugin is self-contained, and executes within its own thread (or threads). This is especially important for fast execution on modern processors, where the number of cores can match, or even exceed, the number of plugins one is running.

In the next subsections, we shall describe how our architecture accomplishes these goals while requiring as little computational and communication overhead as possible. Small overhead is especially important in the case of real time video processing, where relatively large images must be processed at fast frame rates.

A.2.1 Execution Flow

At its core, the architecture provides a shell which consists of a GUI for loading plugins and visualizing data, a system for storing plugin output to file, and a buffering/memory-management system for handling data. This functionality is contained in the *Main Thread* and *Memory Manager Thread* shown in Figure A.2.1. Users build their system by adding plugins, configuring their options via the GUI, and then connecting the plugins to each-other. The user can also save/load a fully configured system as an XML file. Once a vision system has been built, the user can control execution using the frame rate module, which controls the firing rate of the system clock.

As the whole system runs asynchronously in independent threads, the clock trigger acts as the initial starting point for each frame. This means that any source plugins, such as a stereo camera rig or a video file reader plugin, must connect to the frame rate module. As a trigger arrives at each plugin, a triggering signal is sent to the memory manager, telling it to generate a *DataContainer* for the plugin's output. The plugin is then triggered, causing it to execute its processing functionality and generate output, which it stores in the location assigned to it by the memory manager. The plugin then generates another triggering signal, which is connected to both the memory manager and whatever ensuing plugins use the output as their input. When a plugin has multiple inputs, it will loop inside its execution thread, waiting until all inputs for a frame have arrived before executing. This is accomplished by each thread having its own input queue map; it is important to note though, that these queues contain no actual data (and thus minimal overhead), and merely serve as a message passing system. The signaling and triggering system employs the open-source Qt signal & slot architecture. In particular, the system makes use of Qt's ability to queue signals for execution as they arrive at a thread.

A.2.2 Plugin Development and Interaction

The functionality of the system is provided primarily via plugins. A plugin consists of a shared library which is loaded dynamically at run-time. The system is based on the low-level Qt plugin API, which facilitates development and ensures compatibility across different platforms. Plugins inherit from a pure abstract interface class which defines a protocol for communicating with the core application. This permits plugins to define input and output types and pass messages to/from the GUI and memory manager.

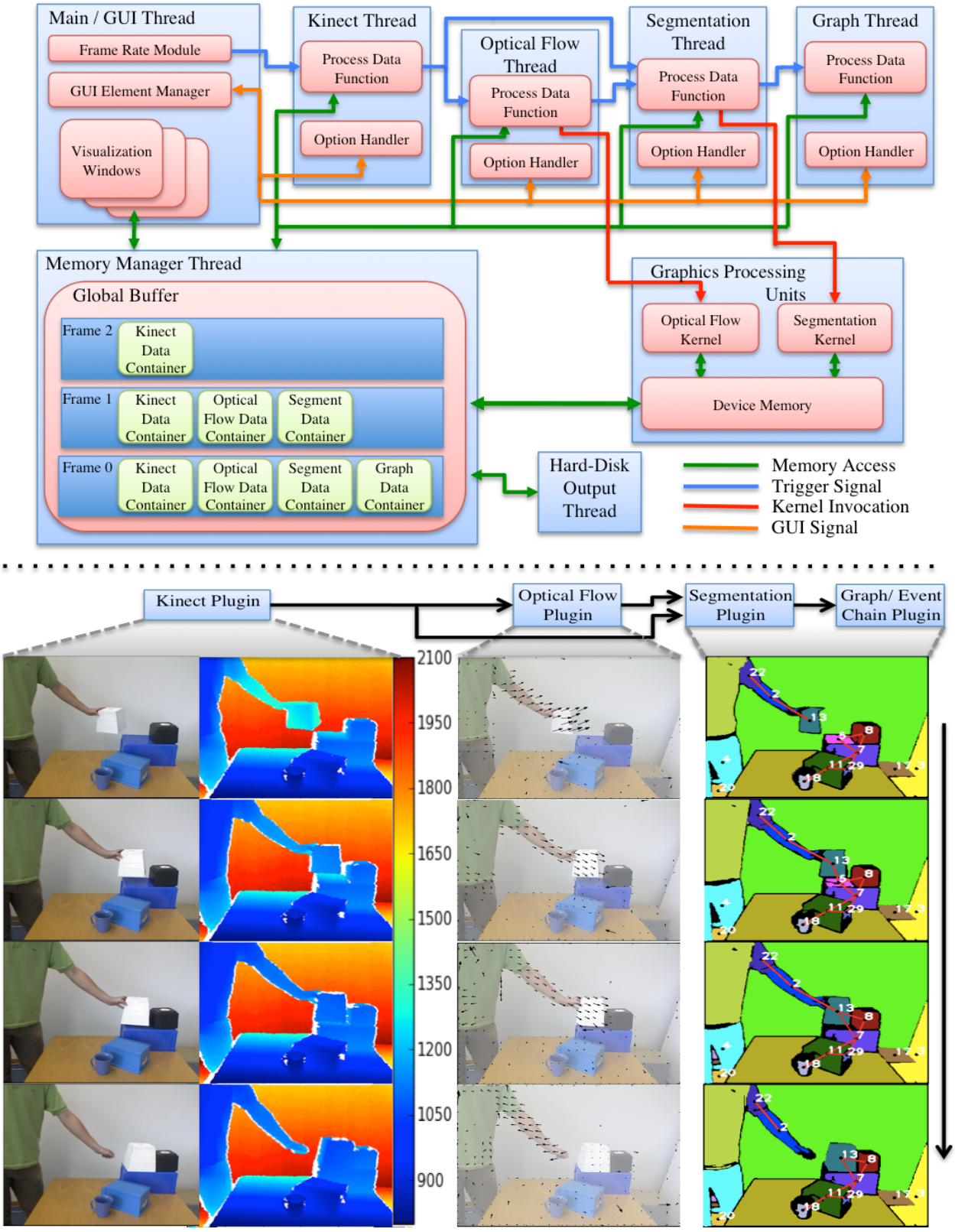


Figure A.2.1: Overview of the system architecture and demonstration system output for four frames. The colums show output from the different components; from left to right, Kinect image and depth (in mm), optical flow, and graphs overlaid on segmentation plugin output. This type of output can be seen live in any number of visualization windows within the GUI.

Developers are required to implement a *processData* function, which receives input and writes to an output *DataContainer*. The developer can optionally create any number of GUI elements (e.g. sliders, buttons) using the interface functions. Plugins specify how many inputs they require, and give the possible types for these inputs. Communication between plugins is accomplished through a standardized data container interface. The core architecture contains commonly used data container implementations, such as *StereoImageContainer*. Plugins may define their own specialized data containers which are loaded at runtime with the plugin. For example, the Segmentation plugin has its own container type *SegmentationData*, which contains a list of labeled segments, metadata about the segments, and labeled images. The standardized data container interface allows for any plugin to refer to a new container class without actual knowledge of the container itself other then the string identifiers of its members (e.g. "Segment Labels"). Correct handling of access to these members is accomplished through dynamic dispatch using the virtual lookup table. This ensures that a plugin written by one researcher can be easily used as input to another's, as long as they know the proper identifiers and underlying formatting of the data.

A.2.3 Visualization

During the development and use of a vision system, it is of utmost importance to be able to visualize what is occurring at every stage of the system pipeline. As such, our system allows users to create any number of visualization windows which can select any plugin to display (and which part of the plugin's output to display, e.g. left or right image). If a developer creates their own data container for a plugin, they can define a special visualization callback function as part of this container. The system will automatically detect this callback when the plugin is loaded, and use it for visualizing the plugin's output. Developers can specify multiple methods for visualizing the plugin; the GUI for visualization will allow selection of which to display.

Visualization windows read directly from the global buffer, and as such have a small memory overhead. Additionally, visualization runs in the GUI thread, rather than in any of the plugin threads. If a plugin slows down the system, visualization (and the GUI) will remain responsive, allowing the user to troubleshoot. This also means that visualization that requires computation, such as labeling an image with text or vector graphics, will have a negligible effect on the actual frame throughput of the system. If visualization lags behind the system output, frames are automatically skipped on an interval that allows visualization to maintain synchronization with the rest of the system. This is of particular importance in an online system, such as our real-time robotic application, where visualization lagging behind processing can cause confusion or even errors.

A.3 Memory Architecture

The memory management system has been designed to allow distributed development and computing, complex system pipelines incorporating feedback loops, and efficient use of the GPU as a computational resource. The following subsections will describe how these design goals have been achieved by illustrating our *Global Buffer* design and explaining how it manages GPU memory.

A.3.1 Global Buffer

Our global buffer concept was designed to overcome the limitations of standard online vision pipelines. In a standard online pipeline a local buffering scheme is used; each algorithm has an input buffer, where data accumulates while it is waiting to be processed. Such a setup is adequate as long as the pipeline remains unidirectional, but complications arise in using feedback loops. Figure A.3.1 compares a stan-

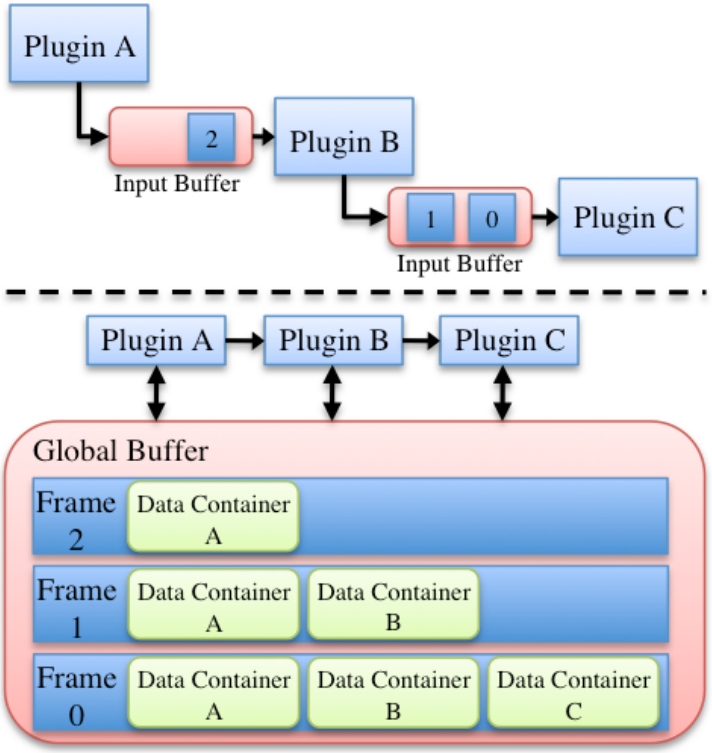


Figure A.3.1: A typical buffering scheme (top) and our buffer (bottom).

dard pipeline with our global buffer; unlike a typical buffering scheme, our global buffer maintains and manages all memory in a central location (and separate thread). The global buffer is responsible for dynamic allocation of all data containers, maintaining reference counts, and determining when a frame can expire. Since the global buffer is responsible for maintaining memory, plugins use a message passing system to communicate. Plugins pass messages to each other to notify completion of a new frame, or to trigger a feedback mechanism. They also use the message passing system to request that the global buffer allocate a new data container for their output. When a developer creates a new type of data container, they use a simple interface to pass the global buffer a function pointer for creating an instance of their new data container type.

In order to fully understand the limitations of a standard buffering system, consider, for instance, the system shown at the top of Figure A.3.2. If the feedback mechanism is triggered for frame n, plugin B must return to frame n in order to modify how it was processed. This is not possible in the standard local buffer scheme, as that data was discarded after it was used as input to B. One possible solution is to maintain another local buffer for each plugin which contains data which has already been processed, but this quickly adds several degrees of complexity. In particular, garbage collection becomes very difficult, and management of these buffers when feedback does occur becomes unnecessarily convoluted.

The global buffer solves this by maintaining data in a more structured way. When a feedback mechanism is triggered for frame n the triggering plugin (D) sends a message to B, causing it to stop processing what it has scheduled, and revert to frame n. As frame n is still easily accessible in the global buffer, B can simply send a request for the pointer(s) to the input data container(s) it requires. The global buffer is guaranteed to still have the data for frame n, because D never produced an output for frame n, so the global buffer has not marked frame n as complete. Once B finishes processing frame n with its new feedback information, it will overwrite its old output for frame n (shown in orange) and then simply continue on as it would normally, processing frame $n+1$. The feedback corrected data will propagate

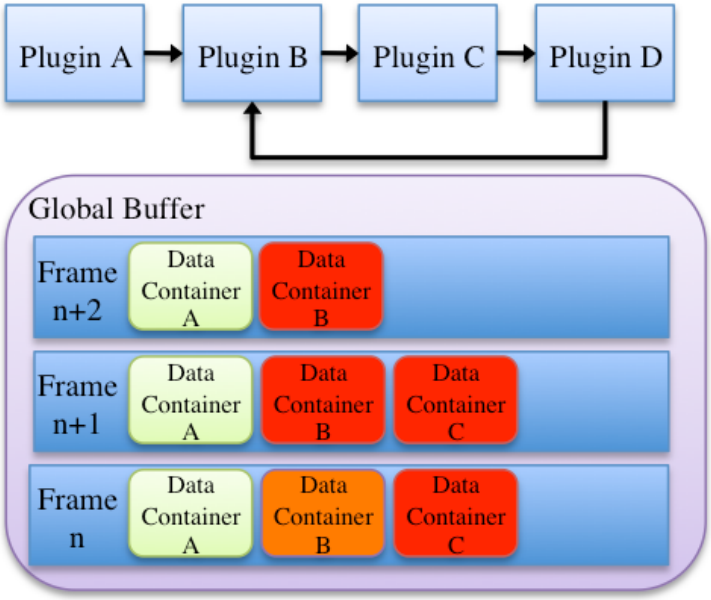


Figure A.3.2: Feedback using a global buffer

down the pipeline, and any data which is no longer valid (shown in red) will simply be overwritten. Infinite feedback loops are avoided by a preventing feedback from occurring more than once per plugin per frame.

A.3.2 GPU Memory Handling

While utilizing the massively-parallel GPU as a coprocessor has become increasingly common, how to integrate it effectively into an open vision architecture remains an open question. Particularly vexing is how to integrate it seamlessly into the memory system of such an architecture, as the GPU has separate physical memory, which is entirely distinct in both location and structure from that used by the CPU [**?**]. Data streaming through the system must be transferred to the GPU for modules which use it, and then transferred back out for visualization and used by modules later in the pipeline.

A naive implementation of this architecture would simply serialize the operations; when a module needs to use the GPU, it copies data to device memory, executes a kernel, and then copies the output back out to host memory. While this is still relatively efficient, it fails to fully take advantage of the pipelined streaming architecture, since the memory transfer bandwidth is idle while the kernel is executing. The architecture uses the streaming CUDA API to utilize this spare bandwidth, allowing it to perform concurrent asynchronous memory transfer and kernel execution.

As shown in Figure A.3.3, we utilize a pre-caching technique, whereby data for frame $n+1$ is transferred during the execution of frame n. When the kernel execution time is significantly longer than the transfer time (B), memory transfer is completely hidden, even with unidirectional memory. When kernel execution time is comparable to memory transfer time, only some of the transfer can be hidden (C), unless the hardware supports concurrent data transfers[1] (D).

[1]Concurrent data transfers are supported under the Fermi architecture[**?**]. Currently the Fermi Quadro and Tesla series cards have two Direct memory access (DMA) engines[**?**], allowing them to perform host-to-device and device-to-host operations simultaneously. The consumer Fermi cards (GTX 4xx, 5xx) only have a single DMA engine, so concurrent transfers are disabled on them.

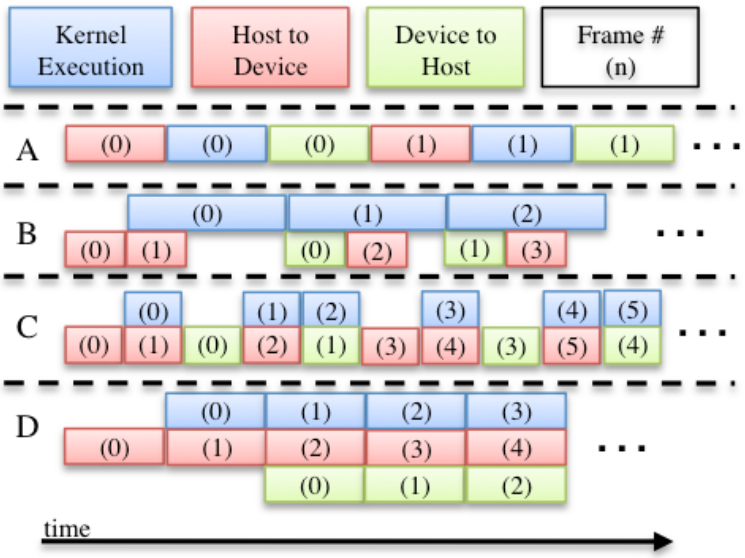


Figure A.3.3: Streaming; Concurrent kernel execution

A.4 Demonstration System

This section presents a real-time demonstration system, consisting of six plugins. The demonstration system calculates dense disparity using a standard stereo camera setup (rather than Kinect data) in order to show the flexibility of the architecture as well as highlight the speedup achieved via multithreading. Switching from Kinect input to a stereo camera setup is simply a matter of changing connections in the GUI. The pipeline described consists of plugins for reading and rectifying stereo data, calculating optical flow[**?**], computing disparity[**?**], segmentation and tracking[4], dense disparity estimation, and semantic graph and event chain generation[6**?**]. This type of a system configuration is used to recognize and learn object manipulation actions in a robotics context.

A.4.1 Image Acquisition

Video is acquired using a Firewire stereo camera rig. Triggering for image acquisition can be controlled using either an external hardware trigger or the architecture's software clock. Rectification is performed on the GPU (there is a separate plugin for calibration using a standard chessboard). Time from triggering to output of a rectified pair of stereo images is around 10ms at 1024x768.

A.4.2 Disparity and Optical Flow

Optical flow is computed using the GPU implementation [**?**] of a phase-based algorithm [**?**]. The algorithm tracks the temporal evolution of equi-phase contours by taking advantage of phase constancy. Differentiation of the equi-phase contours with respect to time yields spatial and temporal phase gradients. Optical flow is then computed by integrating the temporal phase across orientation. Estimates are refined by traversing a Gabor pyramid from coarser to fine levels. The plugin uses the five most recent frames to compute optical flow in the case of online video, but can also use "future" frames when working with recorded movies (this can slightly improve the quality of output flow).

Sparse disparity maps are computed on the GPU using a technique similar to optical flow [**?**]. Rather than use temporal phase gradients, the disparity algorithm relies on phase differences between stereo-pair rectified images. As with the optical flow algorithm, results are computed using a coarse to fine

pyramid scheme.

A.4.3 Segmentation and Tracking

The segmentation and segment tracking plugin has two roles; first, it partitions the image into labeled regions, as seen in the right-most column of Figure A.2.1, and second, it determines correspondences between frames to maintain consistent labeling. The segmentation algorithm is based on the work of Blatt et al. [8], which applies the Potts model in such a way that superparamagnetic phase regions of aligned spins correspond to a natural partition of the image data. Initial spins are assigned to pixels randomly, and then a Metropolis-Hastings algorithm with annealing [4] is used to iteratively update the spins until an equilibrium state is reached.

The Metropolis algorithm is implemented on the GPU[4], permitting real-time performance. The algorithm itself lends itself to efficient implementation on a GPU, as interactions are only computed locally (8 connected nearest-neighbors). Coupling interactions between pixels are determined using average color vector difference (in the HSV space) of nearest-neighbors. Additionally, when depth data is available, the algorithm prevents interactions between pixels if there is a significant difference in their depth values. This prevents coupling across regions which have similar color but discontinuous depth.

In addition to segmentation, the plugin maintains consistent labels for objects from frame to frame. This is accomplished by transferring spins between frames using output from an optical-flow plugin [4]. As such, only the first frame is actually initialized at random; subsequent frames are initialized using a forward-propagated version of the previous frame's equilibrium spins. This has two advantages; the number of iterations needed to reach equilibrium is greatly reduced since the spin distribution already approximates the final state, and the algorithm naturally tracks objects since spins (and thus labels) are maintained over time.

A.4.4 Semantic Graphs

The semantic graphs plugin constructs a symbolic 3D description of the scene from the segmentation results and disparity maps. Segments are used to construct undirected and un-weighted graphs (seen in the right-most column of Figure A.2.1; nodes are labeled with numbers and red lines are graph edges). Each segment is given a node and edges represent their three dimensional touching relations. Graphs can change by continuous distortions (lengthening or shortening of edges) or, more importantly, through discontinuous changes (nodes or edges can appear or disappear). Such a discontinuous change represents a natural breaking point: All graphs before are topologically identical and so are those after the breaking point. Hence, we can apply an exact graph-matching method [**?**] at each breaking point and extract the corresponding topological main graphs. The sequence of these main graphs thus represents all structural changes (manipulation primitives) in the scene.

This type of graph representation is then encoded by a semantic event chain (SEC), which is a sequence-table; rows and columns of which represent possible spatial relations between each segment pair and manipulation primitive. This final output can be used to classify manipulations and categorize manipulated objects for use in a robotics or human-computer interaction (HCI) setting[6**?**]. The primary advantage of this method is that actions can be analyzed without models or a-priori representation; the dynamics of an action can be acquired without needing to know the identities of the objects involved.

A.5 Results and Discussion

Testing was performed to compare single threaded with multi-threaded operation mode and to detect the impact of visualization on processing speed. Testing was performed on an Intel i7 (3.33Ghz, 8 exe-

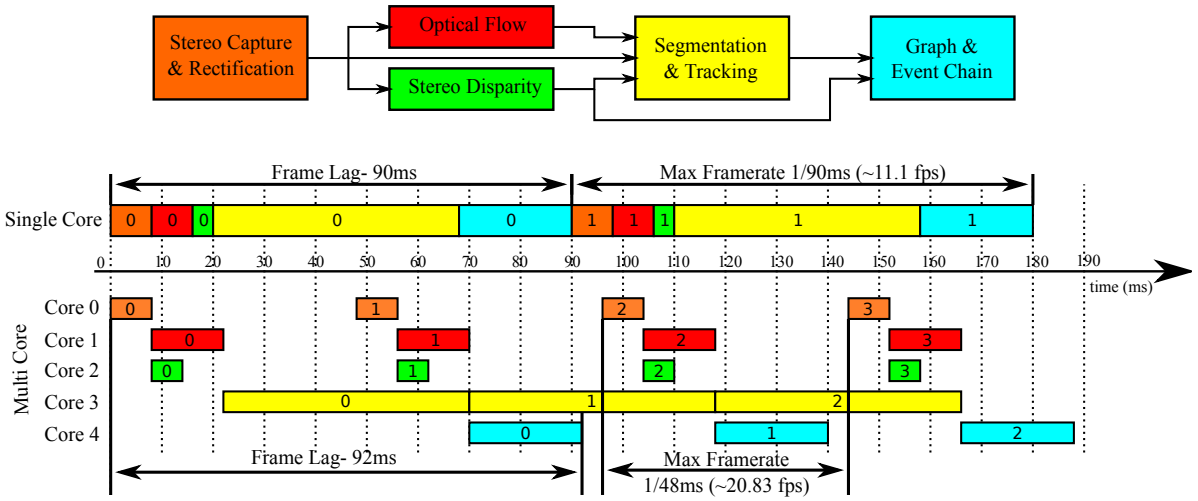


Figure A.4.1: Timing results for demonstration system; plugins are color coded and contain frame numbers. When run in single thread mode, short GPU operations such as optical flow are significantly faster due to reduced overhead; this results in slightly lower (2ms) frame lag. The true benefit of multi-threaded mode is the higher maximum frame-rate that can be achieved.

cution threads) system with an NVIDIA GTX 295 GPU. The demonstration setup depicted at the top of Figure A.4.1 was used for all tests. To determine if visualization had a negative impact, the tests were run with and without a visualization windows for each component, showing live views of their outputs. Timing measurements for plugins are the mean execution time per frame of a 1000 frame (640x480) stereo video sequence (frames of which are shown in Figure A.2.1), averaged over 10 runs. The code for the single and multi-threaded versions is identical with the exception of the movement of plugin objects to separate threads.

We measure performance by analyzing two key attributes of a pipelined vision real-time vision system. First, in terms of frame lag, that is time from frame acquisition to final output, multi-threaded mode is slightly slower than single-threaded. As shown in Figure A.4.1, this is due to relatively fast plugins which use the GPU (disparity and optical flow in this case). This can be attributed to the static overhead cost incurred by switching between threads while using the CUDA run-time API. The switching is relatively expensive for short GPU operations as it forces the CUDA driver to create and destroy GPU contexts[2]. This could be avoided by the addition of an additional GPU; in our demonstration system the driver is forced to change contexts as there are three threads (flow, disparity, segmentation) attempting to use two GPUs. Additionally, the architecture will soon be brought to the newest CUDA release, which allows context sharing between threads. It should also be noted that at higher resolutions multi-threaded mode overtakes single-threaded, as the overhead cost of context switching is outweighed by the gain from computing optical flow and disparity in parallel.

The second measure of performance, throughput, or maximum frame rate, shows a significant speedup in multi-threaded mode, almost doubling from 11.1 (stereo)fps to 20.83. While significant, the speedup is not equal to the number of execution threads used by the demonstration setup (six; one for each plugin and one for the GUI & memory manager). This less-than-optimal gain can be attributed to the fact that the demonstration system had one component, segmentation & tracking, which was significantly slower then the rest. As seen in Figure A.4.1, the entire system throughput is limited by the rate at which the segmentation plugin produces output.

[2]GPU contexts are analogous to CPU processes, and each have their own distinct address space. Each thread may only have one context active at a time, and contexts may not share threads. See [**? ?**] for more details.

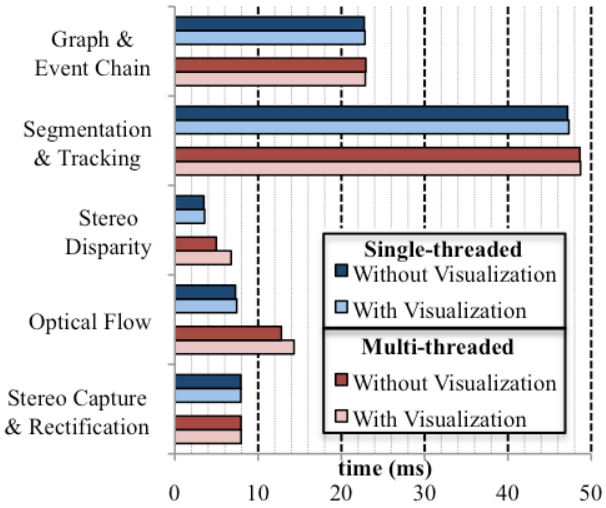


Figure A.5.1: Visualization has a slight impact on performance, but the effect is negligible in multi-threaded mode where the slight increases in processing time are hidden in the length of the longest component (in this case, segmentation).

As seen in Figure A.5.1, the addition of visualization components has a small impact on performance. This delay was most noticeable for the shorter components, disparity and optical flow, but never exceeded 2ms. Fortunately, this additional time does not affect throughput in multi-threaded mode, as it is hidden by the length of the longest component. The times with visualization were used for Figure A.4.1; clearly shortening the time of any component other than segmentation will have a negligible effect on performance. While the increase does not affect throughput, it has a slight effect on frame lag. Frame lag is less important than throughput for our research, but it should be noted that in certain cases, such as when quick reactions are required, frame lag may be an important performance measure.

A.6 Conclusion

Building a self-contained, efficient, and complete vision system acts as a significant barrier to entry for those wishing to develop and test new vision algorithms. We have presented a modular plugin environment, designed specifically for expandability and parallel architectures, which facilitates rapid distributed development of vision pipelines. Our plugin system allows simple collaboration between organizations, allowing developers to share algorithms easily, and without forcing them to share code. The architecture permits streaming use of the GPU as a coprocessor, efficient visualization of algorithm outputs, and the ability to use complex pipelines involving feedback mechanisms. The system architecture has been released released under an open-source GPL license[3].

[3] https://launchpad.net/oculus

B

Sequential Bayesian Estimation

Sequential Bayesian estimation refers to a class of approaches for estimating a varying unknown probability density function from a time series of noisy observations. These approaches use a state space representation, in which a state vector $\mathbf{x}_t$ describes the hidden state of a dynamic system. The goal is to estimate the posterior distribution of the state given all prior observations $\mathbf{z}$, i.e., $p(\mathbf{x}_t|\mathbf{z}_{1:t})$. This is accomplished using a two step recursion which first generates a hypothesis of the current state conditioned on the previous state and then performs a Bayes update using the new observation. These steps are known as the prediction and filtering steps, respectively.

The prediction step estimates the current distribution given all prior observations, or

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})d\mathbf{x}_{t-1}. \tag{B.1}$$

This requires the specification of a stochastic *dynamic model* to characterize the state transition density $p(\mathbf{x}_t|\mathbf{x}_{t-1})$:

$$\mathbf{x}_t = f_t(\mathbf{x}_{t-1}, \mathbf{v}_t), \tag{B.2}$$

where $\mathbf{v}_t$ is the process noise. The dynamic model takes advantage of knowledge of the system to generate reliable predictions of how the state evolves independent of observations.

The filtering step uses Bayes rule to update the predicted density by conditioning it on the new observation $\mathbf{z}_t$:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{1:t-1})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})}. \tag{B.3}$$

This requires the specification of a *measurement model* to characterize the observation density $p(\mathbf{z}_t|\mathbf{x}_t)$:

$$\mathbf{z}_t = h_t(\mathbf{x}_t, \mathbf{w}_t), \tag{B.4}$$

where $\mathbf{w}_t$ is the measurement noise. The marginal likelihood $p(\mathbf{z}_t|\mathbf{z}_{1:t-1})$ is constant relative to the state, and is generally ignored in practice and replaced with a simple normalizing factor.

Once the filtered, or posterior distribution is determined, an estimate of the state can be made using a variety of techniques (e.g., MAP, mean-shift).

B.1 Particle Filters

Unfortunately, except for in special cases (such as the linear Gaussian case with the Kalman filter) determining an exact solution for the posterior distribution is not feasible. As such, Particle Filter techniques were developed to approximate the posterior distribution. They use sequential Monte Carlo to directly implement the Bayesian recursion equations on a set of samples. The most common Particle Filter algorithm is Sequential Importance Sampling (SIS) recursively updates a set of N samples (particles) from the previous time step $\{\mathbf{x}_{t-1}^j, w_{t-1}^j\}$ in a two-step procedure:

1. **Predict:** Apply the dynamic model to find an estimate of the new state for each particle, $\tilde{\mathbf{x}}_t^{1..N}$. That is, draw samples from the state transition prior distribution $p(\mathbf{x}_t|\mathbf{x}_{t-1})$.
2. **Update:** Evaluate the weight for each particle using the observation density: $\tilde{w}_t^j = p(\mathbf{z}_t|\tilde{\mathbf{x}}_t)$ and then normalize.

The set of weighted particles $\{\mathbf{x}_t^j, w_t^j\}$ then approximates the posterior distribution, and an overall state estimate can be found using any appropriate method.

B.1.1 Resampling

An important issue with SIS is that for any finite number of particles the weights will tend to degenerate to the trivial set where all particles have weight zero except for one. This results in the observations having no effect on the particle trajectories, meaning the filter amounts to a random walk using the dynamic model. To avoid this problem, a resampling step was added [**? ?**] which generates a new particle set by sampling from the existing particle set. The simplest way of doing this is to simply sample from the multinomial distribution of the particle weights and then set all particle weights to $1/N$. While this *multinomial resampling* can be effective if employed judiciously, it can also lead to other problems, namely an increasing variance of the posterior distribution. To overcome this a variety of low-variance resampling techniques have been developed; we refer the reader to [**?**] for a concise summary of the different approaches.