University of Nevada, Reno

EFFICIENT OBJECT DETECTION AND TRACKING USING A NOVEL MSER-BASED APPROACH

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

by

Christopher J. King

Dr. Mircea Nicolescu / Thesis Advisors
&
Dr. Monica Nicolescu / Thesis Advisors

December, 2013

UNIVERSITY OF NEVADA RENO **THE GRADUATE SCHOOL**

We recommend that the thesis
prepared under our supervision by

CHRISTOPHER J. KING

entitled

EFFICIENT OBJECT DETECTION USING A NOVEL MSER-BASED APPROACH

be accepted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

Mircea Nicolescu, Ph.D., Advisor

Monica Nicolescu, Ph.D., Advisor

George Bebis, Ph.D., Committee Member

Mark Pinsky, Ph.D., Committee Member

Yantao Shen, Ph.D., Graduate School Representative

Marsha H. Read, Ph.D., Associate Dean, Graduate School

December, 2013

ABSTRACT

This dissertation introduces a novel, real-time, color-based, Maximally Stable Extremal Region (MSER) detection and tracking algorithm. Our algorithm combines MSER-evolution with image-segmentation to produce maximally-stable segmentation. This is achieved using a dual-pass segmentation algorithm that first clusters pixels into a hierarchy of detected regions using an efficient line-constrained evolution process, and then fills gaps between regions to produce dense image segmentation. The resulting region-set offers several unique advantages including fast operation, dense coverage, and temporal stability. It also facilitates efficient computation of additional features including line segments, corners, contours, color histograms, and others. Region tracking is accomplished by matching sub-regions identified around each region's perimeter to those in subsequent frames. Foreground segmentation is achieved by identifying regions that display consistent motion that differs from the background. A static background can improve results but is not required. If models already exist to describe an object, those models can be used to identify foreground regions directly. Regions are modeled and classified using simple color histograms compiled from the contained pixel-clusters. Simple predefined interactions are identified between people and objects using proximity measurements and relative motion vectors. If the interaction information suggests that a theft or other threat might be occurring, the system is capable of dispatching one or more automated robots to investigate or pursue an individual. The system was demonstrated on physical robots that were used in both surveillance and assistance-type scenarios.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation Award IIS-0546876 and by the Office of Naval Research Award N00014-06-1-0611.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I Introduction

The ever-declining price of cameras and data-storage has allowed surveillance systems to become increasingly common in both private and public areas. Despite this increase in coverage, the extent that a system can be used for real-time response is still limited by the number of human-viewers available to monitor the feeds, and in most cases, it is not feasible to have any viewers at all. Consequently, most modern surveillance systems are intended for only passive monitoring, meaning that the extent of their utility is in their ability to provide evidence that might aid detectives in solving a crime. This may help in the eventual apprehension and conviction of a perpetrator, but the extent that crimes are actually prevented is limited to the extent that a criminal is aware of, and dissuaded by, the presence of a camera. To make matters worse, recorded video from passive surveillance systems are generally only viewed when there is non-video evidence of a crime taking place (e.g. alarms, property damage, missing merchandise, eye-witness reports, etc.). In the absence of secondary evidence, it is likely that even recorded crimes go undetected and unpunished.

Given the limitations of traditional systems, considerable effort has been invested into creating intelligent surveillance systems that can autonomously detect criminal activity as it occurs (and ideally, even before it occurs). In its simplest form, an intelligent system might combine motion detection algorithms with traditional human-monitored video feeds. The motion detectors could alert humans to watch any video containing dynamic objects, thus reducing the amount of footage an individual must view. A slightly

more complex system might use feedback from those humans to improve the system's ability to discriminate between normal and abnormal video sequences, further increasing the system's ability to operate autonomously. In its most sophisticated form, an automated system should be able to operate without any human intervention at all. It would be able to identify and track all objects within each camera's field of view, it would locate and identify any people present, it would accurately label their behaviors and interactions, and it might even use current behaviors to predict a person's future actions. If such a system observed (or predicted) suspicious or unusual behavior, it should be able to take appropriate actions. Actions might include the sounding of alarms, locking down areas, calling in appropriate authorities, or even dispatching specialized robot responders.

Since the identification and prediction of behaviors could be generalized for use in other applications, the development of an automated security system would potentially extend far beyond surveillance, potentially being applied to any type of monotonous or dangerous monitoring environment. Vehicle monitoring could be used for real-time optimization of traffic flow and could automatically provide on-board navigation systems with current information about traffic conditions. Scientific studies requiring extended observation could be conducted without the physical presence of a researcher. The software could even be used to endow robots with the ability to autonomously interact with human users, ultimately allowing robots to fill niches in domestic, work, military, or extra-planetary environments. Furthermore, when considering the rapidly expanding use

of video applications on personal devices, the potential number of uses for such an algorithm will almost certainly extend beyond what is even imaginable.

Traditionally, surveillance system architectures have used approximately the same framework as is outlined below. The primary assumption made by the typical system is that video will be taken from a stationary camera. This ensures that pixels corresponding to specific parts of the scene during the initial frames of a video sequence will correspond to those same areas in subsequent frames. Changes in an environment can then be detected through the comparison of pixels between new and old frames.

i. **Foreground Segmentation:** Traditionally the identification of regions corresponding to moving foreground objects is achieved using a two-phase background modeling approach. In the initialization phase, a model is created using video taken of an empty scene (Toyama, Krumm, Brumitt, & Meyers, 1999), though some systems allow on-the-fly initialization (Stauffer & Grimson, 1999). Intensities (or colors) observed at each pixel location are compiled into a Gaussian distribution (Wren, Azarbayejani, Darrell, & Pentland, 1997), a mixture of Gaussians (Stauffer & Grimson, 1999), or into non-parametric models (Elgammal, Harwood, & Davis, 2000). Pixel models are generally updated using subsequent frames. At runtime, pixels from each video frame are compared to the corresponding pixel model. If an object has moved into the camera's field of view, pixels corresponding to the object are likely to differ from the initial distributions and segmentation is achieved by clustering these outlier pixels into foreground regions.

ii. **Feature Detection:** Segmented regions (pixel clusters) contain a huge amount of redundant information. To promote generalization and to increase the speed of region comparison, a reduced set of features is generally extracted. Features may include the region's average color (Zhu & Yuille, 1996) or a histogram of colors (Comaniciu, Ramesh, & Andmeer, 2003), the region's shape (Fieguth & Terzopoulos, 1997), inflection points along the contour (Langridge, 1982), lines (Beymer, 1991), corners (Harris & Stephens, 1988), blobs-like features (Lindeberg, 1988), texture-related features (Lowe, 2004), and others.

iii. **Region Classification (or the generation of new region-models):** Classification is achieved by comparing region features to those of existing models. Classification may be applied to individual objects (Grewe & Kak, 1995), or to an entire scene (Nowak, Jurie, & Triggs, 2006). It may be general (e.g. region is a human (Viola, Jones, & Snow, 2003)), or specific (e.g. region is Bob displaying a forward-facing pose (Steffens, Elagin, & Neven, 1988)).

iv. **Region Tracking:** Regions identified in each frame are matched to those of subsequent frames. Ideally, a region should be tracked over a sufficient duration to establish a motion trajectory. In cases where a tracked region has temporarily exited the frame or is otherwise lost, models can be used to recover and resume tracking upon the region's return. Models may also be necessary to resolve between objects and their shadows or between objects that are occluded (Tao,

Sawhney, & Kumar, 2002). In the absence of background modeling, models can be used to track foreground regions directly (Fieguth & Terzopoulos, 1997).

v. **Behavior Description:** Trajectory, classification, pose, and other information are used to estimate possible interactions between multiple objects (e.g. person ‘P’ is moving away from object ‘O’) or to estimate observed behavior (e.g. person ‘P’ has dropped and abandoned object ‘O’) (Oliver, Rosario, & Pentland, 2000) (Ivanov & Boblic, 2000).

vi. **Automated Response:** A system may simply alert humans to view the video footage for confirmation that suspicious behavior has taken place (Sillito & Fisher, 2008). However, a fully automated system would be expected to initiate its own investigative actions (e.g. better resolution images could be taken of the departing person ‘P’ and robots could be dispatched to inspect abandoned object ‘O’).

As mentioned above, the vast majority of surveillance systems are designed around the assumption that system cameras will remain relatively static throughout the period of observation. This assumption greatly simplifies the task of identifying moving objects and reduces demands on computer resources to the extent that real-time processing becomes feasible. Additionally, this strategy requires no prior information about the foreground, and under the right conditions, can be used to precisely identify the boundaries around each object.

Although the fixed camera constraint can be thought of as a reasonable assumption (since fixed cameras are common to most existing surveillance systems), it reduces a system's overall generalizability and greatly limits the ability for systems to adapt to a world with an ever-increasing number of mobile cameras. Even when a static camera is available, background models are highly sensitive to any movement or changes in the environment. For example, small variations in illumination can cause an entire image to be detected as foreground, resulting in the loss of tracked object. Consequently, a considerable amount of research has been dedicated to developing background modeling algorithms that can adapt to background changes. Although these strategies may be effective in extending the duration of reliable foreground tracking, they inevitably create a problem where pixels from foreground objects can assimilate into the background should the object ever stop moving. Although this behavior can be advantageous in some instances (where it facilitates the segmentation of objects that pass in front of other objects that have temporarily stopped), resolving this issue requires the addition of higher-level tracking algorithms. This is especially problematic when a non-moving object resumes motion since the area occluded by the assimilated object can suddenly appear as new foreground, resulting in the tracking of an object that doesn't exist.

An additional disadvantage to the background modeling approach is that it clusters all connected non-background pixels into a single region, preventing it from distinguishing between separate moving objects that touch or occlude one another. Even shadows and reflections are often indistinguishable from foreground. Again, researchers

have proposed various techniques to handle these situations but they often require foreground model information (Tao, Sawhney, & Kumar, 2002), which may not be available.

Although background modeling is the simplest and most widely used strategy for detecting foreground objects, the limitations imposed by this strategy have inspired researchers to experiment with other techniques. As mentioned in the outline above, when foreground feature-models are available, they can be used to identify and track foreground regions directly. Color is one of the more common features to use during foreground tracking since it is simple to extract and is relatively unaffected by motion, transformation, image blur, and partial occlusion (Comaniciu, Ramesh, & Andmeer, 2003). Recently, texture-based algorithms capable of identifying affine-invariant interest points have gained considerable popularity. These include the Edge-Based detectors (e.g. Canny (Canny, 1986)), Corner-Based detectors (e.g. Harris-Affine (Harris & Stephens, 1988)), Region-Based detectors (e.g. MSER (Matas, Chum, Urban, & Pajdla, 2002)), and others. These algorithms are designed to identify features in a reproducible way, even if the object displaying the features has changed its scale, position, or orientation.

When foreground models are not available, detecting foreground regions without the aid of background segmentation becomes considerably more complicated. If the system can be primed with initial foreground estimations, various tracking algorithms can be used to maintain those positions through subsequent frames. This allows hybrid systems to use background segmentation for initialization, followed by the activation of foreground tracking during periods of camera motion. If initialization is not possible,

foreground can be detected by identifying regions that move independently of the background (Sangi, Heikkila, & Silven, 2001). This requires the tracking of a sufficient number of image features in both foreground and background to accurately estimate local optical flow. Features that display spatially consistent motion are then clustered into independent regions. These strategies are not ideal for segmentation however, since they generally require both the foreground objects and the background to move in a rigid and linear way. Tracking an articulated human is particularly challenging, not only because of the high non-linearity of its movement, but because of the challenges associated with identifying features that persist through the range of motion of the self-occluding, non-rigid structure. To make matters worse, texture-based features are often detected with highest density along the boundary between the foreground and background, an area where feature tracking is particularly unstable and inaccurate.

To compensate for the limitations of motion-based segmentation, it is generally necessary to apply additional clustering techniques. Image segmentation uses color or texture information to identify pixel clusters, and is especially useful in its ability to maintain divisions that correspond to occlusion boundaries, as well as the boundaries separating surfaces and materials. Unfortunately, many of the existing segmentation algorithms are far too computationally intensive for use in real time (especially when run in parallel with optical flow estimation) and generally these algorithms provide only a single interpretation of how an image should be segmented, offering no alternatives in cases where there may be ambiguities in segmentation.

Given the limitations of existing foreground segmentation and tracking algorithms, a successful implementation of a generalized surveillance system will likely require either a completely unique strategy for processing low-level information, or an algorithm that can efficiently combine multiple techniques into a single architecture.

This dissertation presents an automated surveillance system that has been designed from the bottom up using a novel set of image processing algorithms. The system has the following contributions:

- We omit the requirement for using standard background modeling algorithms, allowing our system to free from the strict dependence on fixed cameras.
- Our algorithm tracks highly deformable regions that lack the distinct textural patterns required by other feature detection algorithms.
- Our algorithm detects multiple features, and does so in a way that allows the stability characteristics of each detected feature to mutually reinforce the others. The specific features detected by our algorithm include: Color-based Maximally Stable Extremal Regions (MSER), Canny lines, corners, and region contours.
- Our system implements every portion of the surveillance architecture, while operating in real-time using a standard laptop.

One of the most noteworthy features of our architecture is that we have omitted traditional background modeling algorithms and have therefore eliminated the strict requirement for using static-cameras. A consequence of this is that it slightly changes the

order of typical operations. In the traditional architecture, background segmentation algorithms are used to identify foreground regions, which are described using a reduced set of features. In our architecture, low-level features are detected throughout the image first, which are independently tracked and clustered to allow the identification of foreground objects. The differences from the standard approach are summarized below, followed by a more detailed discussion of each portion of the proposed architecture.

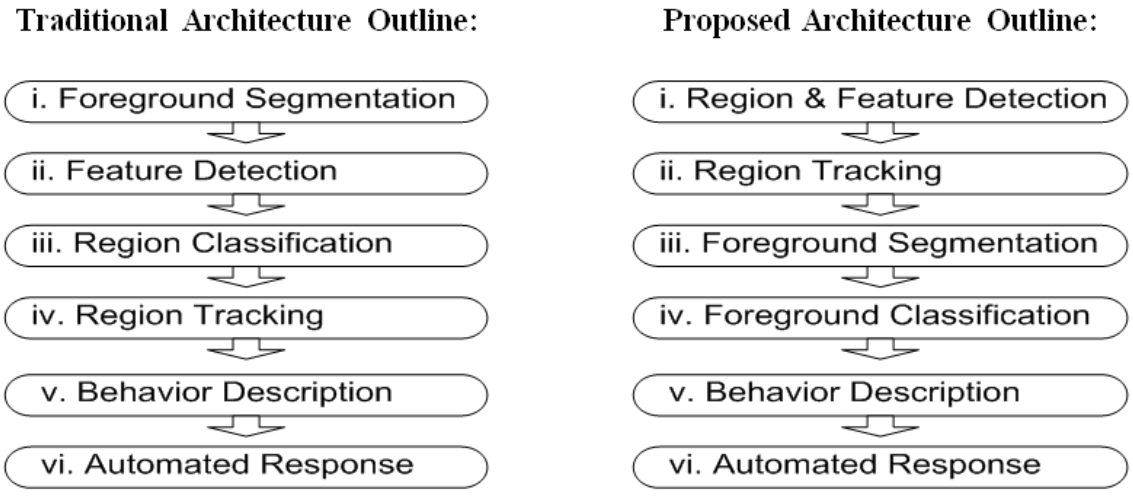


Figure I-1 Flowcharts of traditional and proposed surveillance architectures.

The specifics of each component of the proposed architecture are summarized below. Each component is illustrated using representative screen shots that were extracted while a human subject interacted with objects related to one of our “homework” scenarios:

i. **Region & Feature Detection:** At the lowest feature level, our framework combines the detection of Canny edges (Canny, 1986) with that of the color-based Maximally Stable Extremal Regions (MSER) (Forssen & Lowe, 2007), and does so in a way that provides detection results that surpass results obtained by running these algorithms independently. Since the detected regions are defined by both their region-related properties, and by the edges surrounding their perimeter, only minimal processing is needed to identify additional features including line segments, corners, contours, color histograms, and others. Another advantage of our approach is that the resulting set of detected regions provides dense image coverage, allowing it to be used for traditional image segmentation. If a sparse representation is desirable, a stability threshold can be applied to identify a subset of regions that are considered most stable. In both cases, our segmentation offers multiple interpretations of how an image can be segmented, with relative confidence estimates produced for each.

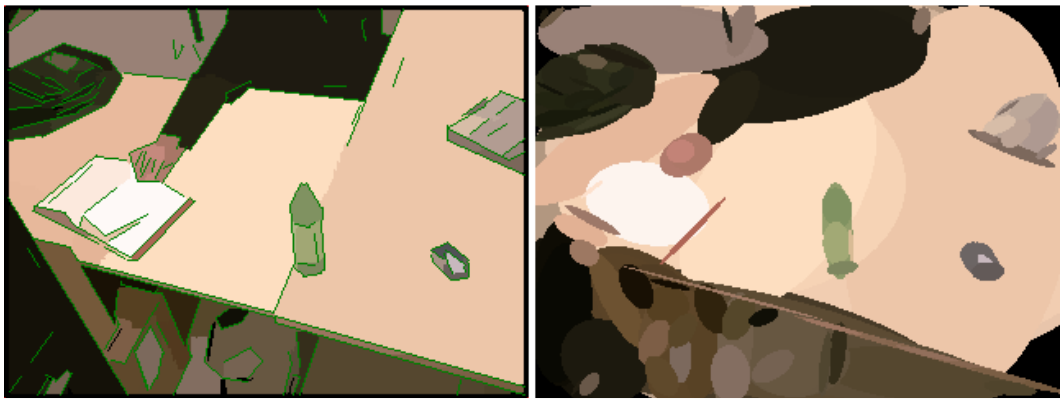


Figure I-2: Region detection using the proposed algorithm.

Figure I-2 shows detected lines (green) and segmented regions in the left image. The right image displays segmented regions using best-fit ellipses.

ii. **Region Tracking:** Our MSER algorithm divides each region into small sub-regions found around the region's perimeter. These perimeter segments are used for tracking regions between frames. Perimeter-segments are defined using the RGB color of a sub-region on the internal edge of the perimeter, the RGB color of a sub-region on the external edge of the perimeter, and a directional component extracted from the gradient at that location. The perimeter-segment elements are probabilistically matched to elements from subsequent frames and a greedy algorithm is used to estimate a likely correspondence between the frames.

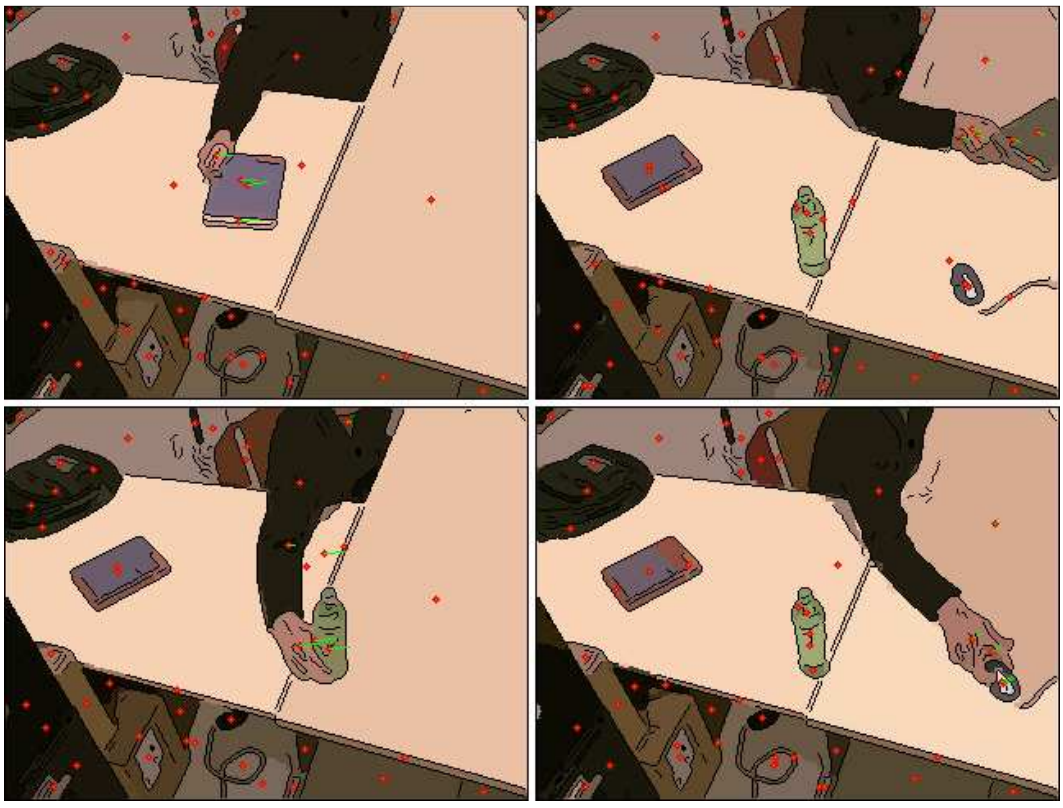


Figure I-3: Region tracking using the proposed algorithm.

Figure I-3 shows movement lines associated with each region. Red circles show the region centers. Green lines show computed motion between video frames.

iii. **Foreground Segmentation:** When foreground models of objects of interest are available, they are used directly to segment foreground objects from the background. When foreground models are not known, segmentation can be achieved by identifying objects that display motion that significantly differs from that of the background.

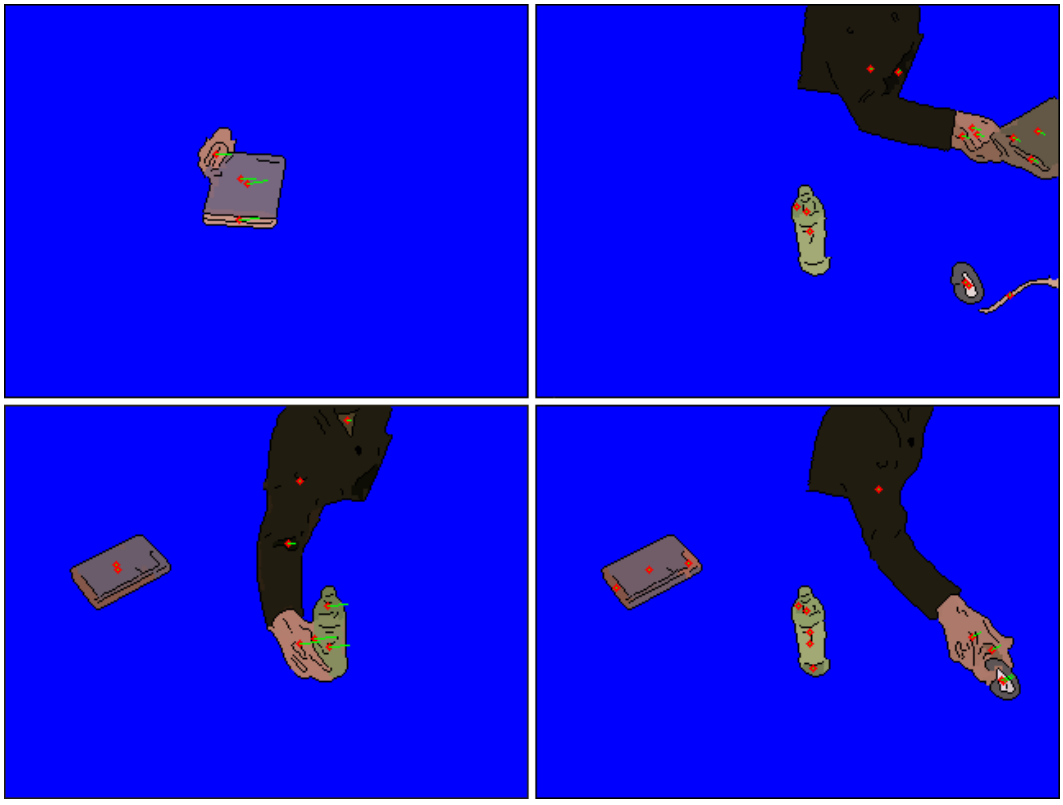


Figure I-4: Foreground segmentation using the proposed algorithm.

Figure I-4 shows regions that the system has determined to be foreground. Regions displaying consistent motion are shown in color. All other regions are shown in blue.

iv. **Foreground Classification:** Regions are modeled and classified using simple color histograms compiled from the contained pixel-clusters over multiple frames. Since complex object recognition and behavioral analysis was not the focus of this dissertation, simple labels were manually assigned to stored models (e.g. book, food, drink, etc.).

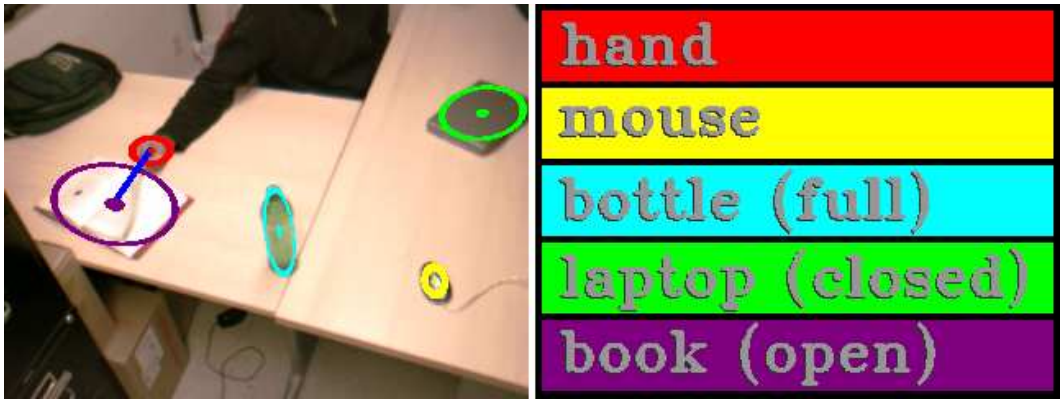


Figure I-5: Region classification using the proposed algorithm.

Figure I-5 shows classified regions highlighted using color-coded ellipses with object names displayed to the right using the same colors.

v. **Behavior Description:** Foreground objects are classified as '*active objects*' (those that were observed to move under their own power) and '*passive objects*' (those that were only observed to move when in close proximity of another object). Interactions are computed between object pairs that contain at least one '*active object*'. Interactions are a function of proximity and the relative direction of the movement vectors. These are generally classified into the three simple categories. In cases where both objects are '*active*', the categories are '*converging*', '*diverging*', and '*moving together*'. In cases where only one object

is '*active*' and the other is '*passive*', categories are '*reaching*', '*dropping*', and '*carrying*'. For purpose of demonstration, only simple pre-defined behaviors are recognized. Behaviors are assigned to every object/interaction combination. When a combination is observed with a relatively high probability over a sufficient number of frames, the system labels the behavior and initiates the response sequence.

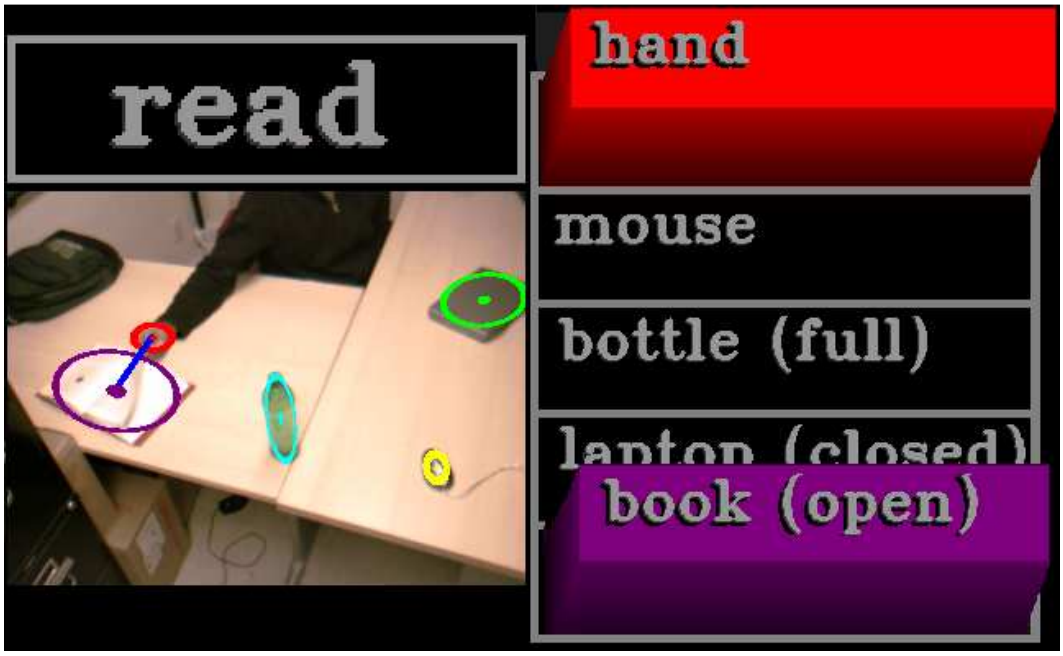


Figure I-6: Action classification using the proposed algorithm.

Figure I-6 shows classified regions highlighted using color-coded ellipses with labels displayed to the right of the image. The apparent "height" of the label is a measure of the interaction likelihood. In this frame, the system has identified the interaction between the "hand" and the "book" as being the most likely, and has drawn a blue line to connect the object pair.

vi. **Automated Response:** Predefined responses were assigned to behaviors in some of the demonstrations. When these behaviors are identified by the system, it initiates a response sequence involving investigation by one or more robots.

Figure I-7: Robot response using the proposed algorithm.

Figure I-7 shows a person who opened a laptop that he was not authorized to use. The system sent a patrol robot to take higher resolution pictures of the person.

It should be noted that our primary motivation for building the proposed system was to achieve real-time operation on a standard laptop, while still providing a high degree of functionality. For every part of our system, more accurate algorithms likely

exist. Though, in most cases, these algorithms are too computational for real-time operation, especially when put together into a complete system. To demonstrate real-time operation and potential applicability of our proposed segmentation and tracking algorithms, we apply them to several full functioning applications. These applications generally involve the observation and reaction to a person approaching, moving, or carrying various known objects. Much of this work has been published previously in book chapters (Kelley, et al., 2013) (King, et al., 2012) (Espina, et al., 2011) (Kelley, Tavakkoli, King, Nicolescu, & Nicolescu, 2010), conference articles (Kelley, King, Ambardekar, Nicolescu, Nicolescu, & Tavakkoli, 2010) (Tavakkoli, Kelley, King, Nicolescu, Nicolescu, & Bebis, 2008) (Kelley, Tavakkoli, King, Nicolescu, Nicolescu, & Bebis, 2008) (Tavakkoli, Kelley, King, Nicolescu, Nicolescu, & Bebis, 2007) (King, Palathingal, Nicolescu, & Nicolescu, 2007) (King, Palathingal, Nicolescu, & Nicolescu, 2007), and journal articles (Kelley, Tavakkoli, King, Ambardekar, Nicolescu, & Nicolescu, 2012) (King, Palathingal, Nicolescu, & Nicolescu, 2009) (Kelley, King, Tavakkoli, Nicolescu, Nicolescu, & Bebis, 2008).

The structure of the dissertation is as follows. **Chapter II** describes some previous work in the area of automated surveillance and is divided into six subchapters (i. – vi.) corresponding to the six parts of the '**Traditional Architecture Outline**'. **Chapter III** describes the proposed surveillance architecture and is divided into six subchapters (i. – vi.) corresponding to the six parts of the '**Proposed Architecture Outline**'. **Chapter IV** describes the scenarios used to validate different portion of our system. **Chapter V** concludes with a roadmap for future work.

Chapter II Previous Work

In the introduction a general outline was provided to describe the phases of a conventional surveillance system. This chapter covers some existing work that is related to the various parts of the automated surveillance architecture (listed in the '**Traditional Architecture Outline**'). This chapter will be divided into the following subchapters: **II.i Foreground Segmentation**; **II.ii Feature Detection**; **II.iii Region Tracking**; **II.iv Region Classification**; **II.v Behavior Description**; and **II.vi Automated Response**.

II.i Foreground Segmentation

Foreground segmentation is the identification of the image regions that correspond to potential objects of interest. In surveillance applications, objects are generally considered to be foreground if they are found to move independently of the background. This allows regions to be detected, even when no prior information is available to identify the object. Although some systems use flow-fields and other motion-detection algorithms to detect areas of movement, it is far more common to design algorithms around the assumption that the camera will be static and the background will remain relatively static through the duration of the experiment. This eliminates the need for complex region tracking and allows moving objects to be identified as those locations where the field of view has changed from previous frames.

II.i.1 Background Model-Based Segmentation

Background model-based segmentation is still one of the most frequently used strategies for identifying foreground objects. This technique relies on the assumptions that a surveillance camera is static and that objects of interest are those that move independently through the camera's field of view. Background models are generally constructed during an initialization period of the video feed that ideally contains no foreground objects. Depending on the requirements of the system, these models may or may not be updated at runtime. Segmentation involves the identification and clustering of pixels in subsequent frames that differ significantly from those in the background model.

The most significant problem associated with background segmentation is that the background model uses pixel-level features that are almost always fixed in relation to the camera. This makes the approach highly sensitive to even slight camera movement, illumination changes, reflections, shadows, occlusion, etc. The plots in **Figure II-1** show how a background pixel's color can change over time. **Figure II-2** shows how basic foreground segmentation techniques can fail when applied to background changes.

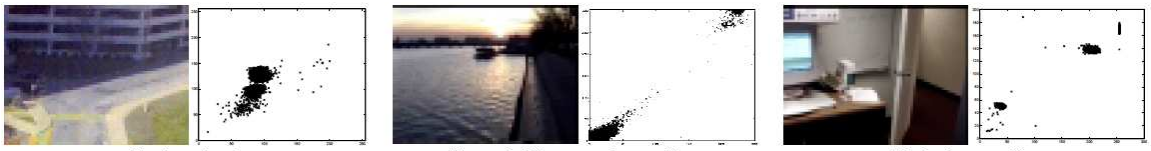


Figure II-1: Challenging backgrounds used to test foreground segmentation.

Figure II-1 shows backgrounds displaying multi-modal pixel distributions. A sample frame (left) is paired with a scatter-plot made using the red (horizontal) and green (vertical) values of a single pixel measured during the video feed. In the left pair, pixel variations result from the movement of a building's shadow. In the center pair, rippling water can appear very bright or very dark. In the right pair, a flickering computer monitor can produce dissimilar values. Figure copied from (Stauffer & Grimson, 1999).

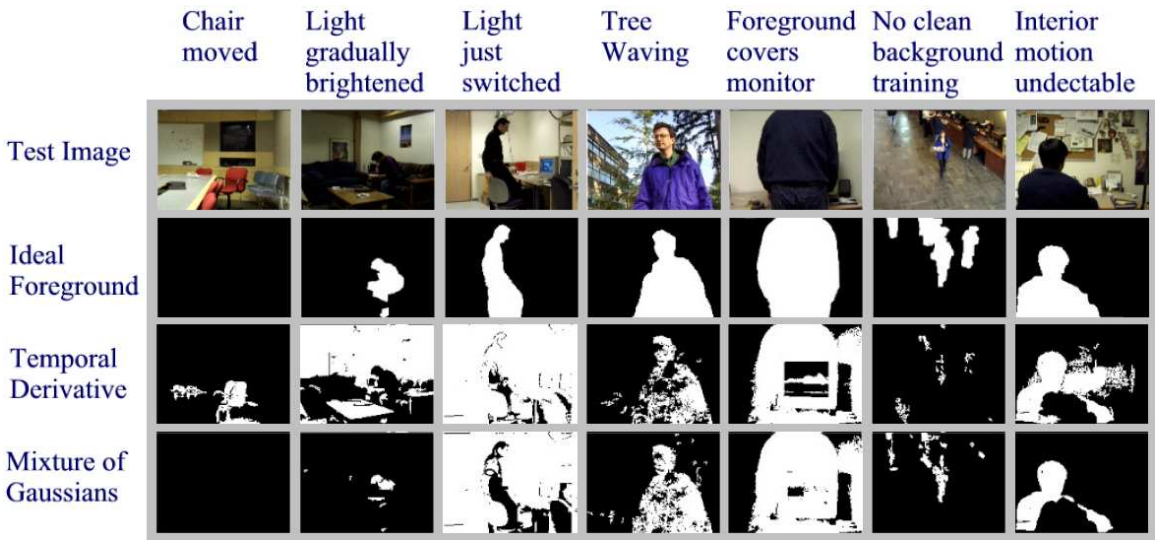


Figure II-2: Segmentation of challenging backgrounds.

Figure II-2 shows how foreground segmentation can fail when applied to non-static environments. Results are from two popular background modeling algorithms applied to seven environmental challenges. Top Row: a representative frame from the image sequence and a description of the environmental challenge. Second Row: a hand-selected baseline reference showing desirable segmentation. Third Row: image subtraction that uses the difference between the current frame and an initial frame. Forth Row: segmentation using a Mixture-of-Gaussian-based approach. Figure copied from (Toyama, Krumm, Brumitt, & Meyers, 1999).

Because of the popularity of background segmentation algorithms, a significant amount of research has been dedicated to handling background fluctuations using increasingly robust and adaptable background models. Strategies have applied Gaussian-models (Toyama, Krumm, Brumitt, & Meyers, 1999), mixtures of Gaussians (Stauffer & Grimson, 1999), histograms (Elgammal, Harwood, & Davis, 2000), and dozens of other techniques (Piccardi, 2004). Others have focused on augmenting the background modeling approach with additional features. Beymer (Beymer & Konolige, 1999) used a stereo depth-map to assign depth-layers to regions identified through background

segmentation. This modification allowed the objects to be separated from their shadows to some degree, and could differentiate between two foreground objects, even if one occluded the other (**Figure II-3**).

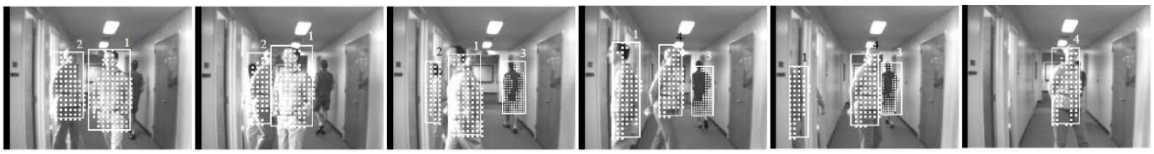


Figure II-3: Real-time stereo-based tracking proposed by Beymer.

Figure II-3 shows stereo-depth-map information being used to separate objects from their shadows while differentiating between multiple foreground objects. The six images were extracted from a video sequence involving multiple people walking through a hallway. Each person was assigned unique labels by the system (Beymer & Konolige, 1999).

II.i.2 Flow-Field-Based Segmentation

Because of the limitations imposed by traditional background-modeling algorithms, researchers have attempted to extend motion-based segmentation to moving platforms. Instead of measuring the absolute change in pixel values, these strategies estimate the motion of small image blocks between temporally adjacent frames and attempts to identify areas of motion that differ from the remainder of the scene. Sangi used particle filters to identify different moving layers (Sangi, Heikkila, & Silven, 2001). One particle filter tracked the most prominent motion, and additional filters identified increasingly less prominent layers. Particle filters were applied to blocks of pixels within the image. Block comparisons were done using the sum-of-square differences of contained pixels.

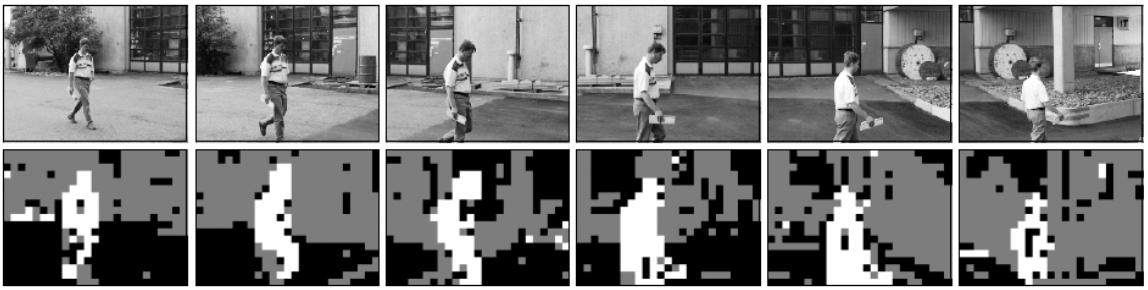


Figure II-4: Support regions from particle filter approach proposed by Sangi.

Figure II-4 shows motion-based foreground segmentation. Gray blocks best represent the primary motion, white blocks best represent the secondary motion, and black blocks are ambiguous (Sangi, Heikkila, & Silven, 2001).

II.ii Feature Detection

Background segmentation by itself is generally insufficient for tracking multiple objects over extended periods. In real-world scenarios, objects move in and out of the camera's field of view, become occluded, and can be lost due to detection errors. In these cases, a tracking system must have a mechanism to associate detected regions with those that were previously tracked. This requires a foreground model to be generated using information found in the segmented region. If the foreground model is sufficiently descriptive and efficient to implement, tracking can be achieved directly using the foreground model, thus eliminating the static-camera constraint. Features can include color, intensity, edges, corners, blobs, active contours (Baumberg & Hogg, 1996), primitive geometric shapes (Comaniciu, Ramesh, & Andmeer, 2003), articulated human shapes (Ning, Wang, Hu, & Tan, 2001), and many others.

II.ii.1 Color & Intensity Models

Color-based models are generally created using pixel values extracted directly from an image. These may be represented as simple means, Gaussians, mixture of Gaussians, or histogram distribution. The primary advantage of using color is that it can be rapidly extracted from an image, while being relatively stable through large changes in scale, viewpoint, and object transformation. The main disadvantage is that apparent color can fluctuate as illumination changes (Buluswar & Draper, 1998). Researchers have attempted to solve the color consistency problem, but algorithms tend to be computationally expensive, sensitive to choices of multiple parameter variables, have the potential for introducing color artifacts, can be affected by the presence or absence of colorful objects, and fail to model human color perception (Morel, Petro, & Sbert, 2009). In many applications, the color consistency problem can be solved by acquiring information over a range of lighting conditions, or by controlling the environment.

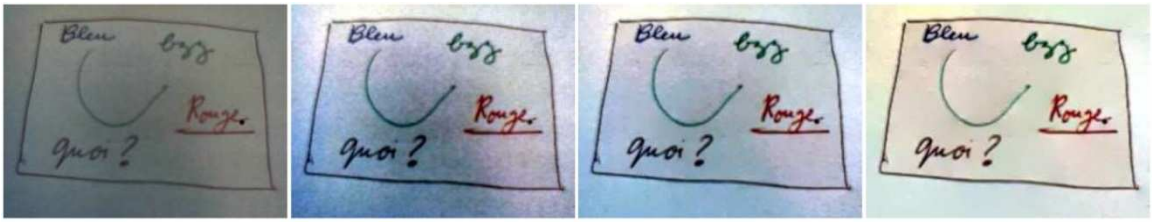


Figure II-5: The Retinex color consistency algorithm proposed by Morel.

Figure II-5 shows how color consistency can be used to estimate object coloration. The original image is on the left. The remaining images show the results using different contrast thresholds (Morel, Petro, & Sbert, 2009).

Despite the problems associated with color consistency, the ease and speed at which it can be extracted from an image makes color a useful feature for a variety of applications. In robotics, the visual identification and tracking of objects can be achieved relatively easily if those objects are painted with colors that are not found within the background. Foreground segmentation is then achieved by identifying and tracking clusters of pixels that match the known foreground color. In slightly more sophisticated applications, color information can be acquired during an acquisition period where an object is "introduced" to a robot. Color models can then be used to aid in subsequent tracking (Schlegel, Illmann, Jaberg, Schuster, & Worz, 1998).

Although color is sufficiently descriptive to facilitate foreground region detection, it generally is not possible to do the same using grayscale images unless additional spatial information is considered. Viola developed a system that could be trained to identify general object types in an image, in real time, using only the spatial distribution of light and dark patches (Viola & Jones, 2001). Viola recognized that the average intensity of a rectangular image region, of any size, could be determined using four references into an integral image array. Simple filters were constructed that used overlapping rectangles to detect adjacent regions of light and dark. Using large training sets, filter with high discriminative qualities were selected and cascaded in a way to optimize speed and classification robustness. Viola originally applied their algorithm to face-detection, but the algorithm has since been used on a variety of objects, and have even been adapted to incorporate color information (Okuma, Taleghani, Freitas, Little, & Lowe, 2004).

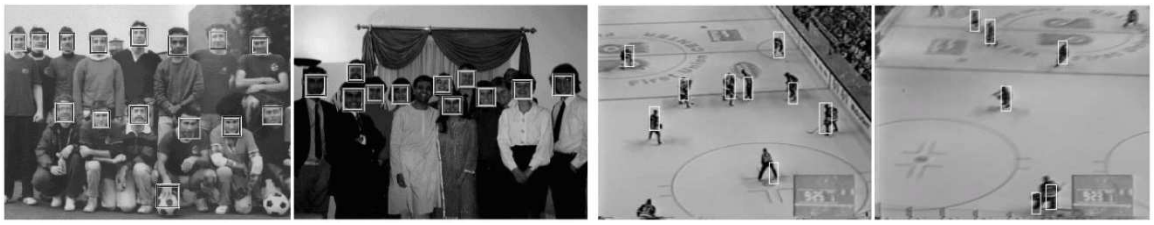


Figure II-6: AdaBoost object detection algorithm proposed by Viola.
Figure II-6 shows AdaBoost-detected regions. Left: the system was originally applied to facial recognition (Viola & Jones, 2001). Right: the system was extended to detect hockey players (Okuma, Taleghani, Freitas, Little, & Lowe, 2004).

II.ii.2 Line Detection

Edges are useful image features because they often correlate with meaningful image areas that include regions of occlusion, object boundaries, changes in surface orientation, changing materials, surface markings, etc. lines can also be used for 3-D reconstruction (Rothwell, Zisserman, Forsyth, & Mundy, 1992).

II.ii.2.1 Canny Edge Detection

One of most popular and effective edge detection algorithms is the Canny (Canny, 1986), which was designed to offer robust and accurate detection and localization of edges. The unique characteristic of the Canny is the way the algorithm applies a dual-threshold pixel tracing technique. One threshold is used to trace out strings of pixels that correspond to the most prominent gradients, while the second is used to extend the lines into sections of the image that contained a less prominent gradient. This allows ambiguous gradients to be ignored, except in places where they can be used to connect and extend the more prominent strings. The result is a robust collection of edges, which are stored as sequences of connected pixels. This particular representation is useful

because of its compactness, and because it minimizes the processing needed to subsequently identify line-segments, corners, and contours.

II.ii.2.2 Curve Linearization

The next step in simplifying edge information is to convert the strings of pixels into curves, line segments, and shapes. This provides data compression and can reveal features that might be useful for tracking, identification, and scene reconstruction. There are two primary strategies for identifying line features.

The simplest strategy for identifying line features is to directly convert strings of pixels into line segments. Ramer proposed a process that begins by attaching a line segment to the endpoints of an arbitrary curve (Ramer, 1972). The algorithm iteratively identifies the point on the line that has the greatest distance from the curve and subdivides the line so that each new segment is connected to that point. The process is repeated until all points on the lines are within a minimum distance of the curve.

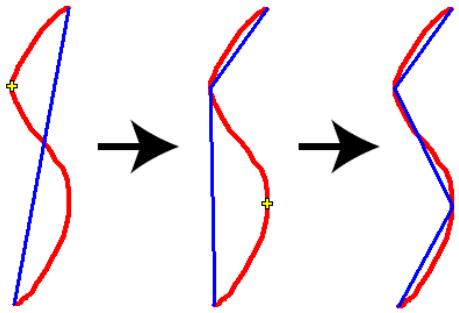


Figure II-7: Line-Fitting algorithm proposed by Ramer.

Figure II-7 shows line-segments that are fit to an arbitrary curve. After each iteration, an existing line-segment is divided so that the ends pass through the most distant point (Ramer, 1972).

II.ii.2.3 Hough Line Detection

In real-world environments, occlusion, noise, and detection errors can cause line segments to become disjointed or obscured. In these cases the Hough Transform algorithm is useful at identifying line segments that follow similar trajectories. In its most general implementation, each edge pixel casts votes for any line that could theoretically pass through that point. Line candidates that receive the most votes are analyzed to determine the extent of their match (Duda & Hart, 1972). This algorithm is useful in that it can be modified to detect circles, curves, or even arbitrary shapes (Ballard, 1981).

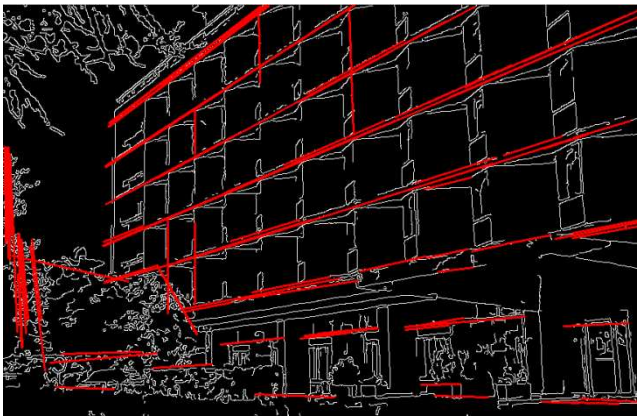


Figure II-8: Hough line detection algorithm proposed by Duda.

Figure II-8 shows Canny-detected edges in white with Hough-detected lines superimposed in red (Duda & Hart, 1972).

II.ii.3 Affine-Invariant Interest Point Detectors

Among the huge variety of feature-detection algorithms available, the Affine-Invariant-Detectors have been found to be extremely useful in a variety of computer vision

applications (Mikolajczyk, et al., 2005). The success of these algorithms lies in their ability to:

- Detect regions that are unique: allowing regions to be distinguished from other detected regions.
- Detect regions in a repeatable way: allowing the same region to be detected in subsequent frames despite changes in viewpoint, scale, orientation, blur, illumination, etc.
- Detect regions in a descriptive way: allowing features detected in one reference frame to be transformed into a second frame that differs in terms of viewpoint, scale, or orientation.

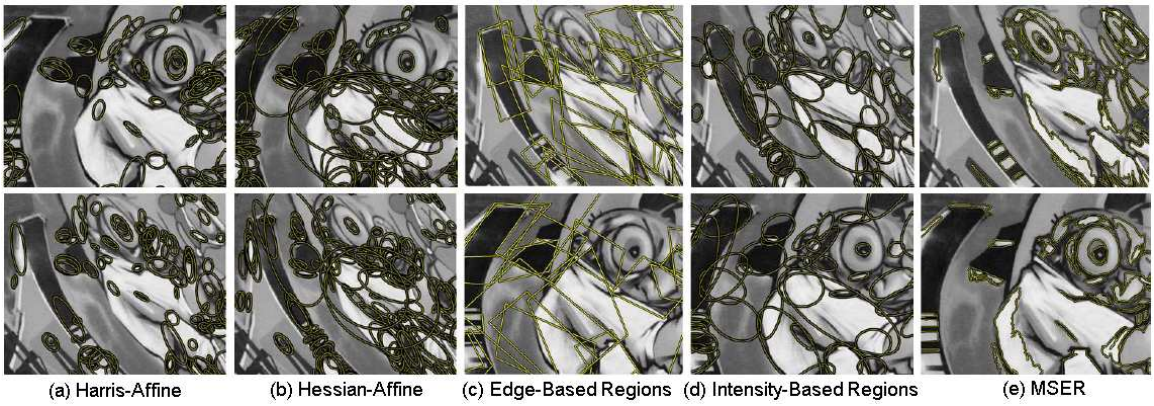


Figure II-9: Affine-Invariant Interest Point Detectors compared by Mikolajczyk.

Figure II-9 shows detected regions using a) Harris-Affine, b) Hessian-Affine, c) Edge-Based Regions, d) Intensity-Based Regions, and e) MSER. The listed algorithms are described in the following subsections. Images copied from (Mikolajczyk, et al., 2005).

II.ii.3.1 Corner-Based Detectors (e.g. Harris-Affine & Hessian-Affine)

Harris-Affine (Harris & Stephens, 1988) and Hessian-Affine (Mikolajczyk & Schmid, 2002) are two related detectors that identify descriptive points in an image. The Harris-Affine detector operates by taking average image intensities using a Gaussian-weighted kernel. With small shifts of this kernel, the filter responds to the underlying texture. Textureless regions yield small changes in all directions, linear textures yield responses in the direction perpendicular to the line, and corner-like features yield responses in all directions. Interest points are defined as those regions with locally-maximal responses in all directions (corners). The feature's orientation is defined as the direction corresponding to the largest eigenvector of the response, and the feature's shape is defined as an ellipse with dimensions corresponding to the ratio of the largest and the smallest eigenvalues. Feature-scale is determined by identifying the local maximum of the detector's response at different image resolutions.

The Hessian-Affine operates is computed in a similar way, but uses the Hessian matrix to determine point fitness. The Hessian detector responds most strongly to blobs and ridges. In both cases, an elliptical shape is assigned using the eigenvalues of the moment matrix, which is then normalized to be circular.

II.ii.3.2 Edge-Based Detectors

Edge-based detectors are used to identify regions associated with an edge. Edges are a desirable feature because they remain stable across viewpoints, scales, and illumination changes. Once edges are identified, points of maximum gradient are

identified along the edge. This significantly reducing the search space necessary to define affine regions (since the search is reduced from two-dimensions to one).

Tuytelaars described a technique that defines affine frames using the point of intersection ('p') between two lines ('l_1', 'l_2'), combined with stable endpoints ('p_1', 'p_2') found along the corresponding line-segments (Tuytelaars, Van Gool, D'haene, & Koch, 1999). Endpoints were selected in a way that maximized a similarity measurement within the region formed by the connecting lines.

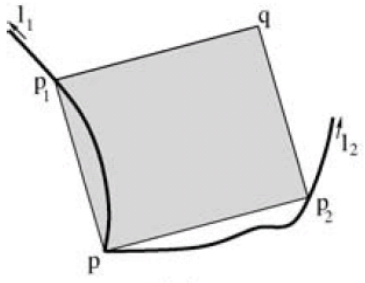


Figure II-10: Edge-based feature detection proposed by Tuytelaars.

Figure II-10 shows an example of an edge-based region. Parallelogram p_1-q-p_2-p is identified by finding stable points p_1 & p_2 (Tuytelaars, Van Gool, D'haene, & Koch, 1999).

II.ii.3.3 Region-Based Detectors (e.g. MSER)

Region-based detectors are generally used to identify image areas that share similar visual characteristic (similar intensity, color, or texture), while being bound by areas that are less similar. Ideally, the internal area will present enough of a contrast against the background to allow consistent and repeatable detection, however an algorithm should not be so discriminative that it leaves large portions of foreground objects without any detections at all.

Tuytelaars described a system that identifies ellipticial regions that are centered around intensity transition extrema (Tuytelaars & Van Gool, 2000). The size and shape of each ellipse is determined using a set of rays that are radially projected from the origin. Points along each ray are compared to the origin and an ellipse is constructed from the set of points that maximizes the gradient around the region's perimeter.

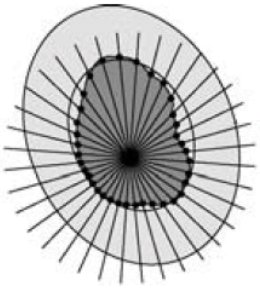


Figure II-11: Region-based feature detection proposed by Tuytelaars.

Figure II-11 shows an example of a region that is identified using points of intensity transition (Tuytelaars & Van Gool, 2000).

The MSER algorithm was developed by Matas to identify stable areas of light-on-dark, or dark-on-light, in grayscale images (Matas, Chum, Urban, & Pajdla, 2002). The primary advantage of this algorithm over previous region detectors was that it could identify all stable regions in an image in a single pass, and could do so efficiently (in nearly linear-time relative to the number of pixels in the image). The algorithm functions by applying a series of binary thresholds to the image. As the threshold value increases, areas of connected pixels grow and merge until every pixel has merged to become a single region. Region-sizes are measured after every increase in threshold, and those regions that display a relatively stable size through a wide range of thresholds are recorded as a maximally stable region. Since a single region may display stability at

multiple threshold-values during its growth, the resulting collection of detected regions will produce a hierarchical tree of nested MSERs.

To increase the number of detections and to improve image coverage, a logical improvement to the MSER algorithm is to additionally consider color information. Corso and Hager offered a simple extension into RGB space by applying the traditional algorithm to the grey-scale, red-green, and yellow-blue channels (Corso & Hager, 2005). Despite the increase in complexity Hager found their repeatability to be inferior to the original algorithm.

Forssen extended the algorithm into RGB color space by changing the criteria for region growth (Forssen, 2007). Instead of grouping pixels according to a global threshold, Forssen incrementally clustered pixels using the local color gradient. This process clusters pixels containing relatively homogeneous color and texture. Although Forssen claimed improvements in both stability and number of detections, his algorithm has some limitations. First, the algorithm fails to detect regions containing non-edge gradients (like those on curved surfaces or lightly-textured objects). This deterioration occurs because at the pixel level, gradients are indistinguishable from object boundaries. To limit this effect, Forssen applied multiple types of smoothing to his data. This improved stability of some regions, but at the expense of others. The second limitation resulted from Forssen's comparison of adjacent pixels to determine merge criteria. In most video feeds, the spatial correlation of color information is too high to offer reliable contrast, and MSER stability is greatly compromised. Forssen's response was to normalize edge weights in a way that ensured region growth occurred evenly across the

maximum threshold-iteration interval. This technique reduced missed detections, but resulted in a significantly greater number of unstable detections.

Figure II-12: Color MSER algorithm proposed by Forssen.

Figure II-12 compares detections from the traditional MSER algorithm (left) with Forssen's color based algorithm (right) (Forssen, 2007).

II.iii Region Classification

As mentioned in the previous section, object-models can be used to increase tracking precision, to reduce the search space, to resolve occlusion, and to provide information regarding an object's type, identity, and pose. Consequently, object modeling is used in almost every computer vision application. Depending on the constraints of the system and the demands of the application, object models may be general and incomplete (e.g. regions are specified only as being a pixel-cluster of minimum size (McKenna, Jabri, Duric, Wechsler, & Rosenfeld, 2000)), or they may be highly specified (e.g. articulated models of humans containing a constrained number of segments in a constrained set of configurations with each segment possessing a constrained size, shape, and color (Sigal, Bhatia, Roth, Black, & Isardy, 2004)). Models can include any number of features including a person's color (Bahadori, Grisetti, Iocchi, Leone, & Nardi, 2005),

outline (Cheung, Baker, & Kanade, 2003), facial details (Feris, Tian, & Hampapur, 2007), walking rhythm (Makihara, Sagawa, Mukaigawa, & Echigo, 2006), etc.

Building models from features usually involves three steps: features must be detected in a repeatable way; descriptive information about the feature must be quantized to reduce its dimensionality; and descriptors must be stored in a data structure that allows for efficient search and retrieval. In the sections below, we include descriptors and models that can be used as an extension to our version of the MSER algorithm. This will include color-based models, line-based models, and Affine Invariant Feature-based models.

II.iii.1 Color-Based Models

Color-based models are probably the simplest and most intuitive strategy for representing a foreground object. Information is acquired by iterating through the regions pixels and is stored using the average value, a Gaussian distribution, a mixture of Gaussians, or as a histogram.

The primary advantage of color-based models is their resistance to changes in viewpoint, scale, image resolution, or even complex transformations and deformations. These models have also been found to be efficient and relatively accurate for certain applications. Swain presented a system that used histogram representations to search for models of shirts, cereal boxes, and laundry detergent containers within a database of 66 objects. He found that the system returned the correct match 90% of the time (Swain & Ballard, 1991).

The primary disadvantage with using color-based representations is the extent that apparent color is dependent on light source, illumination angle, viewing angle, shadows, reflections, camera parameters, etc. Researchers have attempted to develop strategies to increase color consistency using alternate color spaces, lighting compensation strategies, or parametric modeling (Buluswar & Draper, 1998), but the problem is far from being solved. Another approach that improves color consistency is to model objects using the color-differences found between adjacent regions (Mindru, Tuytelaars, Gool, & Moons, 2004). Mindru combined affine-invariant spatial moments with color information to produce '*Generalized Color Moments*' and found that the combination provided improved classification, identification, and matching results.

Figure II-13: Affine-invariant spatial color moments proposed by Mindru.

Figure II-13 shows detections of color moments. Yellow ellipses show correspondences that were identified between frames (Mindru, Tuytelaars, Gool, & Moons, 2004).

II.iii.2 Line-Based Models

Before Affine Invariant Feature Descriptors were developed, line-based models were commonly used for object description and identification. Like with other modeling strategies, the concept behind efficient modeling is to identify features that remain

consistent, even when viewed from different angles, rotations, and scales. This eliminates the need to search a database for every possible transformation and makes it theoretically possible to have constant-time searches when using low-dimensional features. The affine transformation of a two-dimensional surface contains six-degrees of freedom, allowing the transformation to be defined using three non-collinear points. Such points might include corners, line end-points, curve maxima, curve minima, etc. These points can then be efficiently stored and searched using constant-time hash-based libraries.

Rothwell & Zisserman proposed a strategy for producing affine-invariant descriptors using four points defined by a concave or convex section of an object's contour (Rothwell, Zisserman, Forsyth, & Mundy, 1992). The contour was identified using the Canny edge-detection algorithm, followed by a grouping algorithm that connected disjointed edge segments. The section of the curve was then mapped to a unit square, with the four points corresponding to the four corners. Rothwell & Zisserman represented the shapes of each curve using the first and second degree moments of the area defined by the curve.

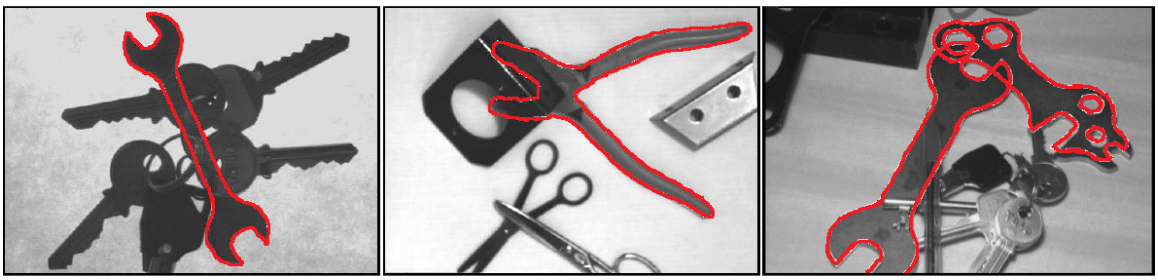


Figure II-14: Line-based classification proposed by Rothwell.

Figure II-14 shows detected regions using line-based models. Red contours are produced by the system to identify detected objects (Rothwell, Zisserman, Forsyth, & Mundy, 1992).

An additional advantage of using line-based representations is the potential for making deductions about an object's three-dimensional structure. Rothwell & Zisserman described an extension of their previous work that allowed objects to be grouped into generalized categories of three-dimensional objects.

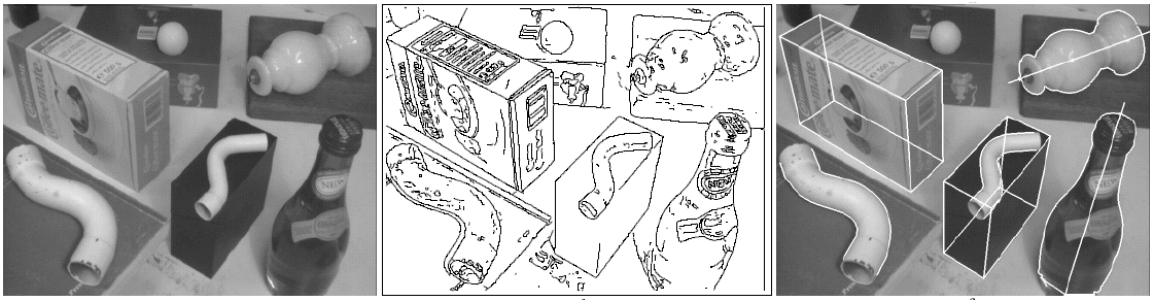


Figure II-15: Line-based detection of 3-D objects proposed by Rothwell.

Figure II-15 shows results from Rothwell & Zisserman's algorithm for identifying 3-dimensional structure (Rothwell, Zisserman, Forsyth, & Mundy, 1992).

II.iii.3 Affine Invariant Feature Descriptors

With the advent of Affine-Invariant Detectors, came the need to specify information about the detected region in a way that allows features to be reliably compared. Features needed to be specific enough to allow discrimination between large numbers of objects, while general enough to allow reliable detection despite changes in illumination, position, and scale. The SIFT descriptor was among the first that was really successful at satisfying these requirements and has become the benchmark against which all other descriptors are compared.

II.iii.3.1 SIFT & SURF

The Scale-invariant feature transform (SIFT) was designed to allow interest points to be described, stored, and retrieved in a reliable and efficient manner (Lowe, 1999). Although the descriptors can be applied to any stable interest points, Lowe demonstrated his approach using Scale-Invariant Keypoints. These keypoints were detected by finding the minima and maxima after processing an image using difference-of-Gaussian filters at multiple scales.

Once keypoints were detected, Lowe used the most prominent gradient to assign a characteristic direction and regions were normalized to a characteristic scale. Regions were divided into a 16x16 grid and gradient directions from each 4x4 group of cells were compiled into separate histograms, resulting in a 4x4 array of 8-bin histograms. The 4x4x8=128 element feature vector was then normalized to form the final descriptor. During object recognition, Lowe used a Best-Bin-First (BBF) data structure to identify the likely nearest neighbors. If three or more features matched in a spatially consistent way, an object match was reported.

Lowe's descriptors have been widely used for robot localization, panorama stitching, 3D scene modeling (Torr & Zisserman, 1999), and others. The descriptors have even been extended into the temporal domain for use in action recognition (Niebles, Wang, & Fei-Fei, 2008).

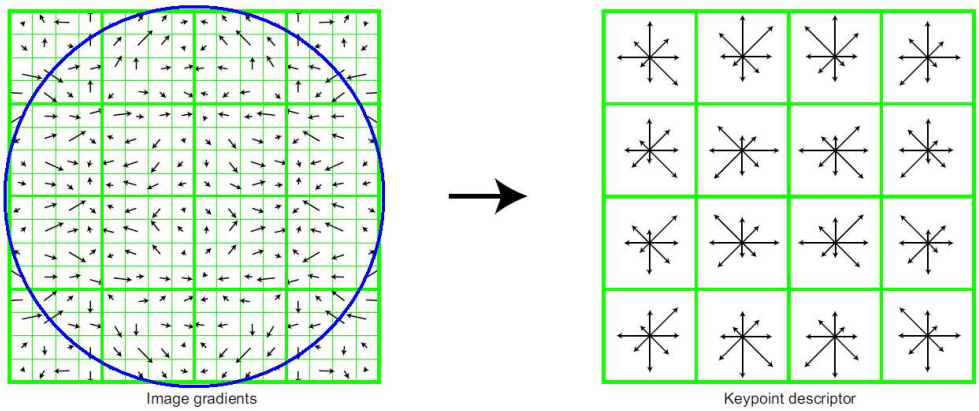


Figure II-16: Scale Invariant Feature Transform (SIFT) proposed by Lowe.

Figure II-16 shows SIFT feature computation. Vectors corresponding to maximum local image gradients are stored to a 16x16 grid (left). Magnitudes are weighted by a Gaussian distribution (blue circle) and compiled to a 4x4 grid of gradient histograms (right).

Speeded Up Robust Features (SURF) were developed as an optimized variant of the SIFT (Bay, Ess, Tuytelaars, & Gool, 2008). Instead of using image gradients, SURF features use Haar Wavelets, which can be approximated efficiently using box-filters and integral image arrays. The wavelet filters are applied across the image at multiple scales to identify clusters of dark pixels surrounded by light areas, or light clusters surrounded by dark areas. Regions showing maximal responses are considered stable features. Regions are assigned a characteristic direction using the distribution of filter responses and are normalized to a characteristic scale. Regions are divided into a 4x4 grid, with each grid cell being represented using a 4-bin histogram of wavelet responses in the dx, dy, |dx|, & |dy| directions. The resulting 4 x 4 x 4 = 64 element feature vector is normalized to form the final descriptor. Using Lowe's BBF search strategy, Bay claimed significant improvement in speed over the SIFT algorithm with comparable accuracy results.

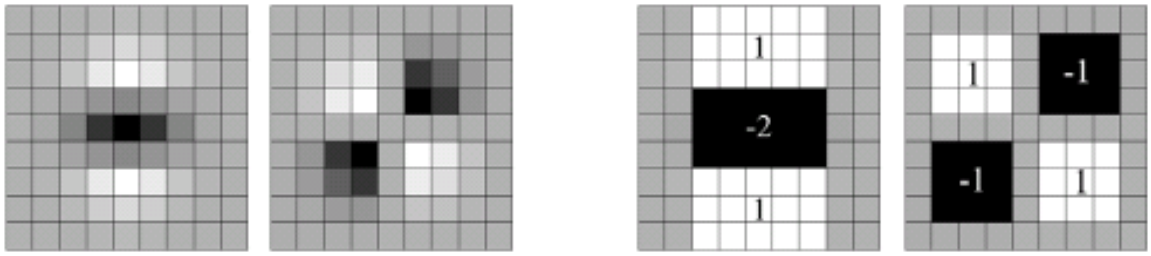


Figure II-17: Haar wavelet filters used in SURF algorithm proposed by Bay.

Figure II-17 shows how Haar wavelets (left) are approximated using box filters (right) (Bay, Ess, Tuytelaars, & Gool, 2008).

II.iii.3.2 Affine Frames

The Local Affine Frames (LAF) algorithm was developed to provide low dimensional affine-invariant descriptors for regions identified by the MSER detection algorithm (Obdrzalek & Matas, 2006). Compared to the SIFT descriptors (128 dimensions) and the SURF descriptors (64 dimensions), the LAF can be described using a modest 6 dimensions. This reduction allows for efficient search and comparison operations, which can approach constant-time complexity when using hash table data structures (Chum & Matas, 2006).

The principle behind LAF construction is to fully constrain the six degrees of freedom found in an affine transformation. This requires three non-collinear points which can be identified (or computed) from features (referred to as affine-covariant primitives) found within the region. Primitives are selected so their position on the object remains the same, even when viewed from slightly different angles. Examples include: center of gravity; covariance matrix; curvature minima and maxima; furthest point on contour; points along concavity; direction of linear segment; and many others.

Once detected, patches are normalized to a characteristic intensity, scale, and orientation. In one paper, Discrete Cosine Transformation (DCT) was used to produce descriptors (Obdrzalek & Matas, 2006). In a second paper, descriptors were constructed using the polar coordinates of the three characteristic points (Chum & Matas, 2006). The angle and offset of the central point was divided into 25 and 16 bins respectively, while the angles and offsets of the remaining points were divided into 25 and 6 bins respectively. This resulted in $9x10^6$ possible values for the descriptors, a search space that was managed using a hash table. As in the previous algorithms, object match was considered likely if multiple features were found to be spatially-consistent.

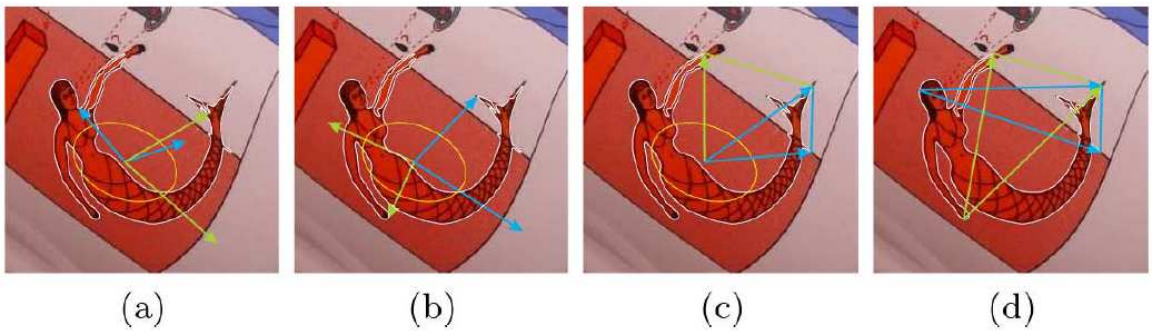


Figure II-18: Local Affine Frame (LAF) features proposed by Obdrzalek.

Figure II-18 shows how affine invariant frames can be defined using different feature types including center of gravity (a,b,c), covariant matrix (a,b), curvature minima (a), curvature maxima (b), concavity tangent (c,d), farthest point on contour (d). Obdrzalek's paper provided many other examples (Obdrzalek & Matas, 2006).

II.iv Region Tracking

Tracking involves the identification of a region's trajectory across frames. This can be done by tracing the path of a single point, or additional information can be extracted using increasingly complex representations: circles provide the ability to track size; ellipses can estimate orientation; multi-point models can provide information about affine transformations; contours make it possible to identify irregular deformations; articulated models can be used to track human movements; etc.

II.iv.1 Region-Based Tracking

Region-based tracking strategies are used to localize and match foreground regions across frames using the contained distribution of intensity, color, or texture. Because of the popularity of foreground segmentation techniques, region-based tracking is most commonly used to match foreground blobs between frames, while resolving situations where blobs appear to merge and split. In more complex applications, higher level models may be used to differentiate foreground blobs into their constituent parts. An example is Wren's PFinder algorithm, which uses color information to cluster pixels into visually similar regions, which are matched to an articulated model containing hands, arms, torso, legs, and feet (Wren, Azarbayejani, Darrell, & Pentland, 1997).

Figure II-19: PFinder segmentation for articulated models proposed by Wren.

Figure II-19 shows a segmented foreground region (center) being grouped into blobs and fit to an articulated human model (Wren, Azarbayejani, Darrell, & Pentland, 1997).

A common region-based tracking technique is to use elliptical or rectangular search templates that roughly match the dimensions of a target object. Templates may represent a general class of object, a specific object, or even a specific object in a specific orientation. During search, the template-kernel is scanned across the image at multiple scales. Features are compared against the kernel to produce fitness scores for the region at that location. Local maximum that exceed a minimum threshold are marked as a potential object candidate. Additional models may or may not be used to provide additional confirmation.

Schweitzer (Schweitzer, Bell, & Wu, 2002) developed a process that approximated image gradients using a linear summation of the rectangular features described by Viola and Jones (Viola & Jones, 2001). In addition to identifying general 'face' classes, Schweitzer's system could also identify specific faces in specific orientations. Matching speeds were found to be nearly 100 times faster than traditional template-based approaches.

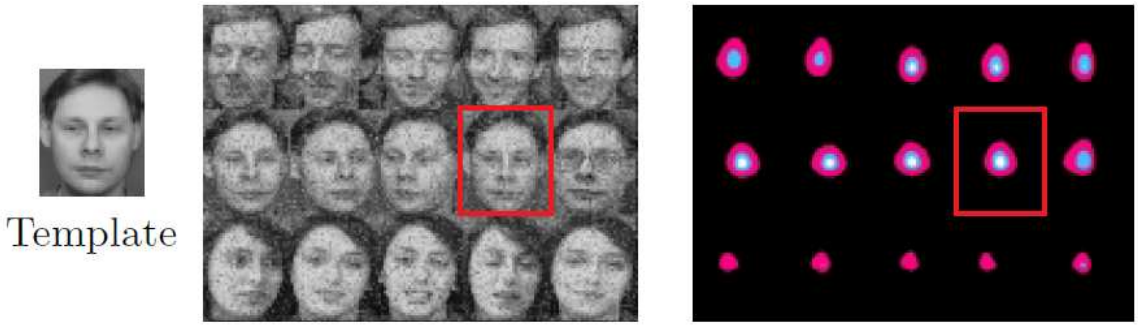


Figure II-20: An efficient template-matching algorithm proposed by Schweitzer.

Figure II-20 shows a template-based search for a face (left) in an image with added noise (center). The template response (right) revealed local maxima correctly centered at the location of every face in the image, with the global maximum correctly identifying the exact match (red rectangles) (Schweitzer, Bell, & Wu, 2002).

In addition to the computational complexity, another limitation with template-based matching is that the algorithm degrades rapidly when the query and target objects differ in terms of orientation or scale. One solution is to use features that are less sensitive to transformation.

Comaniciu discussed a system that uses color histograms to track non-rigid objects through partial occlusion and complex transformations (Comaniciu, Ramesh, & Andmeer, 2003). Comaniciu compiled three-dimensional color information into a 16x16x16 histogram using a weighted elliptical search kernel. Starting in the neighborhood of the object's previously known location; an iterative search is conducted to find a new position and scale that minimizes the difference between histograms. A Gaussian kernel allowed matches to be identified using gradient descent, greatly improving the efficiency of the search. Although Comaniciu's approach was shown to be robust to camera motion, partial occlusion, and clutter, it was not capable of identifying

objects solely using model information and was dependent on either manual initialization or background subtraction techniques.

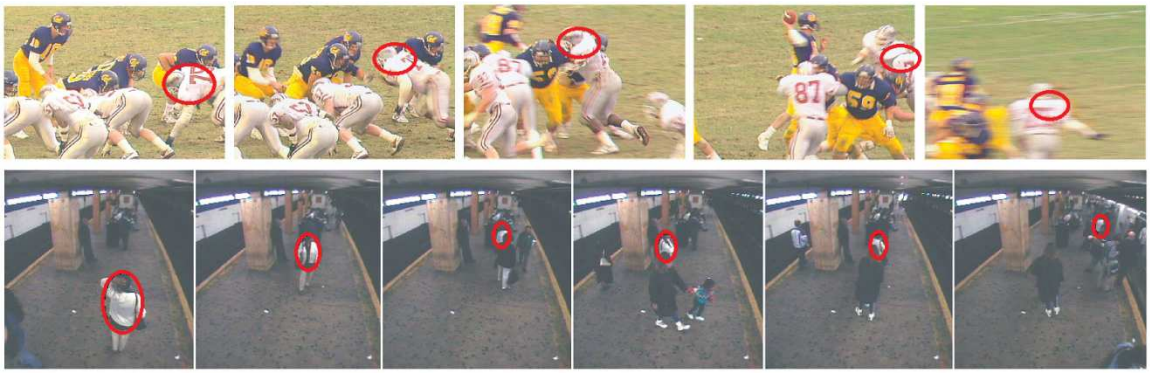


Figure II-21: A kernel-based tracking algorithm proposed by Comaniciu.

Figure II-21 illustrates kernel-based tracking of people in real-world scenes. Manually selected regions were successfully tracked using color histograms and displayed using colored ellipses (Comaniciu, Ramesh, & Andmeer, 2003).

II.iv.2 Contour-Based Tracking

Although region-based tracking techniques have been shown to be robust to complex transformations and occlusion; they generally offer inaccurate representations of the object's physical contours. This makes the algorithms unsuitable for the detection of subtle movements. If an application requires the identification of specific poses, gestures, or actions, it often becomes necessary to track the object's contour itself.

Kass was among the first to propose an algorithm that tracked the position of an object using the occlusion-boundary formed between the object and the background (Kass, Witkin, & Terzopoulos, 1988). Kass's algorithm used a flexible self-evolving structure called a 'snake', which contained a connected set of points placed near the

object's boundary in an initialization frame. During its evolution, each of the snake's points is shifted in the direction that reduces the snake's overall energy function. Points can reduce energy in one of three ways: 1) by moving to a location with a greater local image gradient; 2) by moving to a location that reduces the local curvature of the contour; 3) by moving to a location that places the point at an optimal (pre-defined) distance from adjacent points. The weight of each term can be adjusted to change the behavior of the snake. Evolution continues until a local minimum is achieved. By using the snake's position in each frame as initialization for the following frame, long-term tracking of the contour can be achieved.

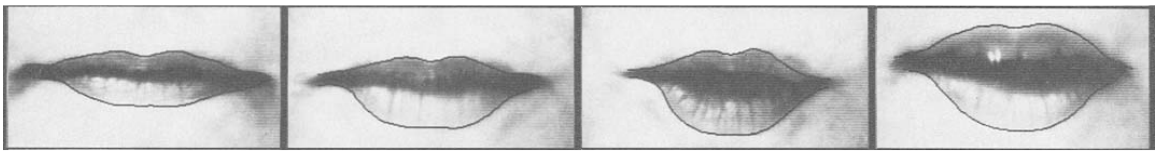


Figure II-22: Contour-based tracking proposed by Kass.

Figure II-22 illustrates the tracking of snakes surrounding a person's lips (Kass, Witkin, & Terzopoulos, 1988).

Major drawbacks to Kass's approach are that solutions are only found if they are within a relatively small range of the current contour. This increases the algorithm's dependence on accurate initialization, reduces its ability to handle large inter-frame image changes, and makes suboptimal convergence more likely. Long-term tracking is also problematic for the algorithm since there is no global information about the region that could be used to prevent localization errors from propagating to subsequent frames. To address these issues, researchers have developed techniques to incorporate both region and model information (Yilmaz, Li, & Shah, 2004).

Figure II-23: Contour tracking algorithm proposed by Yilmaz.

Figure II-23 illustrates how tracking results can be improved using color and texture models of foreground objects. Shape models are used to fill in missing contour information during occlusion (Yilmaz, Li, & Shah, 2004).

II.iv.3 SIFT & SURF Feature-Based Tracking

SIFT and SURF feature detectors and descriptors are among the most promising algorithms available for identifying objects. Not only can these algorithms support recognition using large databases, but when multiple features are detected on a single object, it can provide the precision necessary to determine an object's approximate orientation and distance from the camera. This makes it theoretically possible to track objects through a video feed by running the feature detection algorithm on every frame. When objects are rigid and textured, when images are of good resolution, and when there is no need to do computation on-line, both SIFT and SURF are well suited for a variety of tracking tasks including flow-field estimation, scene reconstruction, and many others (Torr & Zisserman, 1999). However, in real-world systems, tracking with high-dimensional features is often too computationally intensive to be used in real-time, especially when a dense representation of an object is desired. Possible solutions to the problem have been to use complex features to augment more conventional tracking algorithms (Zhou, Yuan, & Shi, 2009), or to match features between frames without

computing high-dimensional descriptors (Duy-Nguyen, Wei-Chao, Natasha, & Kari, 2009).

In Zhou's paper, SIFT feature tracking was combined with mean-shift tracking to improve overall robustness (Zhou, Yuan, & Shi, 2009). Zhou's algorithm was designed to track a single object through multiple frames, despite challenges that included: 1) dim lighting; 2) changing illumination; 3) occlusions from non-tracked people; and 4) rapid object movements. Regions were manually selected during the first frame of the image sequence and both SIFT features and a color histogram were computed using the information contained by that region. In subsequent frames, an expectation-maximization algorithm was used to iteratively identify the location that minimizes the color mean-shift, as well as differences between SIFT correspondences. Results showed significantly improved tracking accuracy over the individual approaches, with only a small decrease in frame-rate.

Duy-Nguyen suggested an approach that tracked objects using an optimized implementation of the SURF algorithm called SURF-Track (Duy-Nguyen, Wei-Chao, Natasha, & Kari, 2009). The system allowed interest points to be represented using both simple and complex descriptors. The simple descriptors could be computed and compared rapidly for real-time inter-frame tracking. Complex descriptors were used when increased discrimination was necessary for recognition. Using the same interest points for both functions allowed regions to be tracked without requiring an additional grouping algorithm to associate different feature-types. Testing showed that tracking objects with SURF-Track could be done 5-times faster than using SURF features alone.

II.iv.4 MSER-Based Tracking

When compared to other interest point detection algorithms, the MSER algorithm was found to offer a good balance between speed and accuracy (Mikolajczyk & Schmid, 2005). Donoser proposed a tracker that is specifically optimized to the MSER algorithm (Donoser & Bischof, 2006). Regions were tracked using size, grayscale value, center of mass, bounding box dimensions, and stability measurement. These features were selected because they offered reasonable tracking accuracy, required no additional processing to compute, and could be compared quickly. By using the location and intensity values obtained in previous frames, Donoser was able to apply the MSER algorithm to a targeted location in subsequent images to attained speeds that were 4 to 10 times faster than running a complete MSER search on every frame.

Figure II-24: MSER tracking algorithm proposed by Donoser.

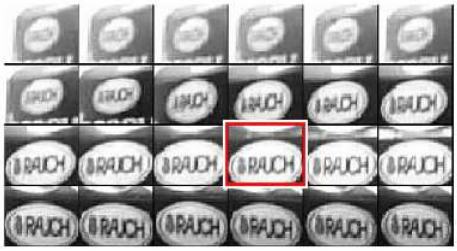
Figure II-24 illustrates the tracking of MSERs that correspond to license plate numbers. The right images show the tracking of a MSER corresponding to a person's face. For face tracking, a special grayscale color-space was created using the Mahanalobis distance of each pixel's (r-g) value from a distribution of known skin-color (r-g) values (Donoser & Bischof, 2006).

In a follow-up paper, Riemenschneider showed how the MSER tracking algorithm can increase the robustness of model creation and comparison (Riemenschneider, Donoser, & Bischof, 2008). In traditional recognition systems, objects

are identified by comparing a single view of an unidentified foreground region to a database of known objects. If the foreground is viewed from an angle or scale that differs from stored models, a match failure may occur. A naïve solution is to attempt foreground identification at every frame of observation, however given the computation necessary for many recognition algorithms, this strategy can be prohibitively expensive. Riemenschneider's solution was to use Donoser's tracking algorithm to maintain trajectories of unknown foreground objects until it is determined that the contained MSERs are at "optimal representation". Only then, is the region compared against stored models. A similar process is used during the model construction phase to identify the most robust views to record.

Two processes are used to determine the optimal MSER representations. First, since regions are tracked through multiple frames, it is possible to identify a set of MSERs that are both spatially and temporally stable. To satisfy the temporal stability requirement, it must be possible to track the region through a specified number of frames. The second process for finding optimal representation is to track a region until a maximum size is observed. This ensures that the object is at a position that is either at its closest approach to the camera or at an angle that is most directly facing the camera. After the process for finding optimal representation is complete, an object model will contain a set of stable MSER detections with observed confidence scores, as well as a description of the spatial relationships of those regions from different perspectives.

Using his tracking strategy on recognition tests, Riemenschneider found that tracking an object over multiple frames could produce recognition rates that were significantly better than those achieved using a single frame.



Objects	Full Tracking	Frontal Frame	Non-Frontal Frame
Eistee	100%	100%	25%
Geback	100%	50%	0%
Happyday	86%	75%	0%
Pringles	100%	0%	50%
Snack	100%	100%	100%
Total	97%	69%	42%

Figure II-25: MSER tracking algorithm proposed by Riemenschneider.

Figure II-25 illustrates how tracking can be improved by automatically identifying the "optimal representation" (red box) from a sequence of images (left). The table (right) shows recognition accuracy for different objects (1st column) using all available tracked frames (2nd column), only one frame from a frontal viewpoint (3rd column), and one frame from a non-frontal viewpoint (4th column) (Riemenschneider, Donoser, & Bischof, 2008).

II.v Behavior Description

Depending on the requirements of a system, behavior description may involve simple action recognition of a target individual (e.g. person is sitting, walking, running, carrying objects, etc.) or may require a higher level deduction about behavior examples.

II.v.1 Action Recognition

Advances into the study of human motion came from the invention of the rotoscope, which allowed sketches to be created from videos (Evans, 2001). Although it was intended to improve the realism of cartoon characters, it proved to be useful to

researchers as well, as it allowed articulated motion to be deconstructed and studied. Today, animation is still providing an impetus into the capture of motion. Modern animation studios can render animated scenes in real-time by tracking reflectors attached to actors. Similar systems are used to identify and categorize specific human movements with the goal of constructing a "language" of motion-primitives. For example, if movements such as forward-swing, back-swing, run-left, and run-right can be accurately identified, these primitives could allow a system to automatically describe a tennis match. A larger set of primitives could similarly be used to annotate football games, movies, or activities observed during surveillance.

Junejo described a technique for constructing a set of motion primitives using self-similarity matrices (SSM) (Junejo, Dexter, Laptev, & Perez, 2008). These matrices were found to be highly correlated for a given action, even when the action is made by different actors or observed from different viewpoints. SSM matrices were constructed using two strategies:

1) **Trajectory Based Matrix:** Multiple points from a person's body are tracked through a video sequence. A SSM matrix is constructed showing the point's average displacement between any two frames of the video sequence.

2) **Image Based Matrix:** A bounding box is drawn around the foreground for every video frame and the enclosed region is divided into a 5x7 grid. Gradient-directions corresponding to each cell are compiled into histograms. A SSM matrix

is constructed showing the combined difference in histogram values between any two frames of the video sequence.

Junejo demonstrated that both strategies yielded similar results and could provide discrimination between nearly a dozen actions.

Figure II-26: Action recognition algorithm proposed by Junejo.

Figure II-26 shows self-similarity matrices (left) being used for action recognition. Matrices were produced for each of nine different actions (center) and a confusion matrix was produced by comparing the recognition results (right) (Junejo, Dexter, Laptev, & Perez, 2008).

Duchenne describes a weekly-supervised system that could operate on real-world video (Duchenne, Laptev, Sivic, Bach, & Ponce, 2009). Actions (e.g. opening-door, smoking, drinking, etc.) were compiled using a computer search of movie sequences that contained those keywords within the corresponding movie script. MSER, Harris Laplace, and Hessian Laplace features were detected and tracked between frames. Those features that moved in respect to the background were segmented and compared to features extracted from similarly labeled sequences and clustered using K-means. Actions were defined using a set of "characteristic movements", which were movements found to be consistent between multiple instances. Duchenne demonstrated that his system could

provide discrimination between a small set of behaviors and could provide temporal and spatial localization of actions within a film.

Clip length (frames)	100	200	400	800	1600
Label accuracy	19%	43%	60%	75%	83%
Localization accuracy	74%	37%	18%	9%	5%

Figure II-27: Action recognition algorithm proposed by Duchenne.

Figure II-27 shows how MSER, Harris Laplace, and Hessian Laplace features being used for action recognition. Examples of characteristic features are displayed in yellow in the left images. A table showing label and localization accuracy for test sequences of different lengths system is shown on the right (Duchenne, Laptev, Sivic, Bach, & Ponce, 2009).

Another potential application for activity recognition algorithms is to automatically sort video sequences according to their content. Sivic proposed a system that would allow users to query a database of images using an exemplar image (Sivic & Zisserman, 2006). "Visual words" were SIFT descriptors that were indexed in a database using a strategy resembling Google's word search algorithm. Niebles described how the concept could be extended to video indexing (Niebles, Wang, & Fei-Fei, 2008). Niebles introduced an algorithm that extended the MSER, Harris, and Hessian feature detection algorithms into the temporal domain by identifying those features that also offered good temporal localization. By clustering each video sequence's according to its specific set of temporal features, Niebles produced a system that could automatically differentiate between running, walking, clapping, and waving behaviors, as well as between several figure skating moves including the camel-spin, the sit-spin, and the stand-spin.

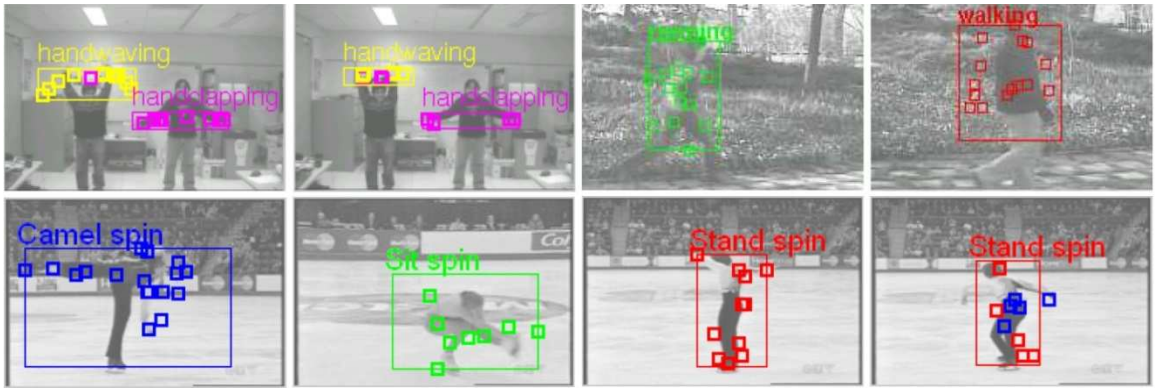


Figure II-28: Temporal feature-based recognition algorithm proposed by Niebles.

Figure II-28 shows temporal features (small square) being used for action recognition. Color assignments are made using the action label associated with the nearest feature match. The large squares display the action label that results from the contribution of all identified features (Niebles, Wang, & Fei-Fei, 2008).

II.v.2 Behavior Recognition

Behavior Recognition involves the high-level analysis of foreground identification, trajectories, and interactions. This may be used to label behaviors directly, or may produce judgments on the extent that behaving in a manner that is safe or unsafe, normal or abnormal, benign or malicious, etc.

Eng describes a scene-level detection algorithm that could potentially assist lifeguards by automatically detecting individuals that are drowning in a pool (Eng, Toh, Kam, Wang, & Yau, 2003). Swimmers were segmented from the pool background using a modified background modeling algorithm. Color homogeneity was used to resolve partial occlusions between swimmers. Descriptors were assigned to each swimmer that included: speed; angle; submersion depth; activity; and splash index. A neural network was trained to associate the descriptor to videos of people that were manually classified

as swimming, treading water, or drowning. The resulting network was shown to correctly classify 95.5% of training samples and 94.5% of test samples.

Sillito presented a partially-supervised system that was designed to assist surveillance operators by automatically identifying video segments containing pedestrians or vehicles that moved in ways that were inconsistent with typical observations (Sillito & Fisher, 2008). Such a system could theoretically reduce the overall workload of security agents by automatically discarding video sequences that did not appear to contain suspicious behaviors. Trajectories were approximated using cubic-spline-curves and were normalized to a fixed number of control points. Curves were classified using the samples that had been observed to that point. If a curve was considered anomalous, it was sent to an operator for verification. False detections were returned to the system so that they could be incorporated into the set of "normal" tracks. After the initial learning curve, false detections leveled off to about 20%. In both human and vehicle-tracking scenarios, the system did not miss a single instance of an anomalous behavior.

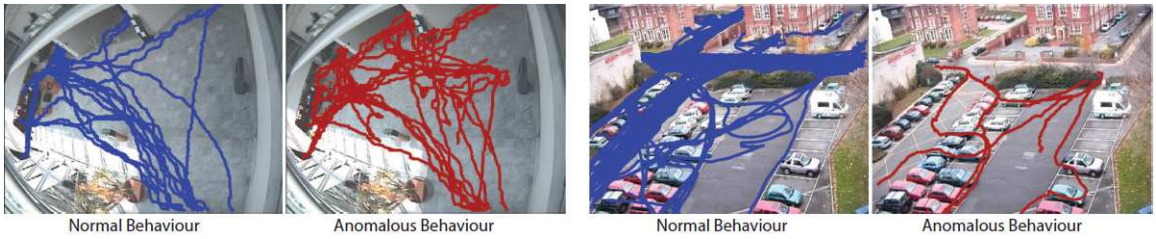


Figure II-29: Anomalous trajectory detection algorithm proposed by Sillito.

Figure II-29 shows the tracks of pedestrians (left) and cars (right), which are recognized by the system as being normal (blue) or anomalous (red) trajectories (Sillito & Fisher, 2008).

Xiang described a system that classified human behavior without requiring supervised training or object tracking (Xiang & Gong, 2006). Background modeling algorithms were used to identify, cluster, and segment foreground regions. Every time a background pixel was labeled as foreground, an event was recorded at the pixel's location. A moving object would therefore appear as a sequence of events occurring across the image. This sequence was modeled using a "Dynamically Multi-Linked Hidden Markov Model", developed by the researchers, and results were clustered using a graph-cut algorithm. The system was trained and tested using videos from three different settings including a roadway, an airport dock, and a small convenience store. For each scenario, the system generated a set of event clusters around the commonly observed patterns. Clusters were shown to represent distinct types of actions. For example, in the convenience store sequence, autonomously-identified events were consistent with: person entering or leaving; person browsing; person taking a can; person paying; and person shopkeeping.

Figure II-30: Action classification algorithm proposed by Xiang.

Figure II-30 shows a scenario (left) involving customers entering a store, browsing, taking a can of soda, paying the cashier, and leaving. The system segmented the scenario into 5 different event types. Events were assigned colors and their image position are shown using circles (center). Temporal information was shown using a graph displaying the relationship between events and image frames (right) (Xiang & Gong, 2006).

II.vi Automated Response

As research into automated surveillance progresses, it is likely that the focus will begin to shift from systems that can only sound alarms in response to identified malfeasance to systems that can actively deploy robots to pursue and interact with observed individuals. Before surveillance capabilities can be extended to robots, a variety of additional challenges must be addressed. Robots should possess the real-time capability to: 1) safely and reliably navigate past stationary and non-stationary obstacles; 2) build maps of the environment while determining their own position and the position of their targets; 3) Effectively engage with other robots and people.

II.vi.1 Robot Navigation

Robot control architectures have historically been categorized into deliberative, reactive, and deliberative-reactive hybrid navigation systems (Nakhaeinia, Tang, Mohd Noor, & Motlagh, 2011).

Deliberative navigation planning has traditionally been a top-down approach that assumes environments are completely and accurately represented by stored maps, while being free of non-static objects that could interfere with the robot's motion. Routes are planned by identifying optimal paths between mapped locations and are followed with inflexible precision. When map or odometry inaccuracies cause the planned path to diverge from a safe route, a robot must stop and formulate a new plan (assuming the robot can recognize the unknown obstacle or path divergence, accurately recalculate its position, appropriately update the map, and identify an alternative route).

Reactive navigation is a bottom-up architecture that bases control decisions on the local environment as it is perceived through the robot's sensors. Although this strategy allows robots to respond rapidly to dynamic and unknown conditions, it is not well suited for attaining complex goals.

Hybrid navigation provides the advantage of allowing a robot to formulate deliberate long-term plans; while also offering the safety and flexibility that comes from a robot's ability to react to unpredicted hazards. Systems usually separate the deliberative and reactive components into distinct modules, with the former having access to global map information and the later receiving sensor information. Data flow and fusion of module output depends on the specifics of the architecture (Nakhaeinia, Tang, Mohd Noor, & Motlagh, 2011).

II.vi.2 Mapping and Localization

Mapping and localization algorithms are necessary for allowing robots to track the position and movements of themselves, their team-members, and their targets. Generally, all robots in a system will coordinate their actions using the same global representation of the environment, which is often generated during an off-line mapping phase; but can be created (or updated) during on-line operation (Thrun, 2002). One of the most popular mapping strategies uses laser rangefinder information to iterative build a fine-grained grid of blocks that represent either occupiable space (open floor), unoccupiable space (walls or obstacles), or unexplored space. Kalman filters produce estimated positions of both the map components and of the robot. Accuracy is improved

by maintaining multiple hypotheses of the robot's position. Particle filter algorithms iteratively eliminate hypotheses that are least consistent with current observations, and replaced them with copies of those that are more probable. Expectation maximization algorithms have been effective for: connecting map sections produced by separate robots (to assemble global maps); connecting map sections produced by one robot (to close loops); or matching a robot's immediate measurements to those on a stored map (to localize the robot's current position) (Thrun, 2002).

II.vi.3 Real-World Surveillance Systems

Developing a surveillance system containing multiple mobile robots requires the robust and efficient operation of all system components related to the detection, tracking, and classification of objects of interest; compounded by the difficulties of operating these algorithms from different cameras with dissimilar viewpoints and non-static fields of view; combined with the challenges associated with robot control, mapping, localization, communication, and organization; all in a dynamic real-world environment with unpredictable lighting, spotty communication, unmapped obstacles, inaccurate sensors, etc.

Saptharishi developed an automated surveillance and reconnaissance system using multiple robots on ATV platforms (Saptharishi, et al., 2002). The system was specifically designed to avoid the costly and user-intensive requirements that was necessary in previous systems including: single platforms with human supervision, power-intensive sensors, detailed & accurate environmental maps, and reliable high bandwidth

communications with bulky relay stations. The system included two mobile and four stationary sentries. Foreground detection was accomplished using a color-based background subtraction algorithm that was modified to accept background-mosaics as input. This allowed background models to be produced using panning & tilting cameras as well as using those that were stationary. Segmented objects were normalized to 20 x 20 pixel grayscale silhouettes and were compared against support vectors of previously trained objects to determine if the region was a person, a group of people, a car, or an unknown object. The autonomous robot scout portion of the system was only shown in simulation. A global planner identified locations in the environment that were suitable for observation and assigned robots to those locations. Robots autonomously navigated toward the destinations, while avoiding static and moving obstacles.

Everett developed a partially supervised real-world surveillance system that used multiple robots to monitor warehouse inventory while identifying potential threats including intruders, fire, flood, etc. (Everett & Gage, 1999). Robots were equipped with: sonar for obstacle avoidance; a docking beacon detector for navigating toward a charging station; a RF-tag reader for tracking inventory; and bump sensors for collision avoidance. A semi-automated dispatch system would assign paths and instructions to an available robot. Robots would execute the assigned commands until completion or until an exception was identified. If obstacles were encountered, the information would be reported to the dispatcher, which would provide alternative routes. The system was designed to handle as many as 32 robots and was in operation for several years.

Carnegie developed a small fleet of autonomous security robots called MARVIN (Mobile Autonomous Robotic Vehicle for Indoor Navigation) that were modified to interact with people in a friendly and natural way (Carnegie, Prakash, Chitty, & Guy, 2004). The robots were equipped with speech recognition and speech synthesis capabilities; and were even provided a head-like appendage that could tilt and turn with the purpose of conveying non-verbal emotional states including happiness, sadness, tiredness, intimidation, submission, affirmative & negative nods, and others. MARVIN contained a laser rangefinder for navigation, multiple short-range "position sensitive devices (PSDs)" for obstacle avoidance, bump sensors for collision avoidance, and odometry sensors. In the security guard implementation, MARVIN was designed to scan its environment until a moving object was identified. If the object was identified as a human (using walking patterns that were identifiable using a laser rangefinder), MARVIN was programmed to approach the person, ask for their ID, and request a password. If a correct password was not provided, MARVIN would demand that the person leave the premises.

Chapter III Proposed Architecture

Surveillance-type applications may be among those that are least restricted by the inherent limitations of background modeling since they commonly have: cameras that are permanently affixed to solid structures; indoor environments that can be controlled; and objects of interest that will propel themselves across the visual field. Despite these accommodating conditions, background modeling algorithms are still not well suited for robust detection of foreground objects: fixed cameras can cause unrecoverable disruptions to a model if auto-focus or auto-adjustment kicks in; controlled environments can present rapid and unexpected changes; and there is no guarantee that self-propelled objects will maintain speeds that prevent them from assimilating into an adaptive background model. Given these limitations, alternatives to foreground segmentation would likely be welcome to the field of automated surveillance, and would certainly be welcome to other areas of computer vision.

As mentioned in the introduction, this dissertation proposes an automated surveillance system that has been designed from the bottom up using a novel set of image processing algorithms. One of the most noteworthy features of our architecture is that we have omitted traditional background modeling algorithms, thus eliminating the strict requirement for using static-cameras. In the traditional architecture, background segmentation algorithms are used to identify foreground regions. These regions are then described using a reduced set of features and tracked. In our architecture, low-level features are first detected throughout the image, and then independently tracked and

clustered. By identifying features that move independently of the background motion, regions can be segmented and identified as foreground. The remainder of this chapter will be dedicated to describing the details of our system. It will be divided into six subchapters, corresponding to the six sections of the '**Proposed Architecture Outline**'. These are as follows: **III.i Region & Feature Detection**; **III.ii Region Tracking**; **III.iii Foreground Segmentation**; **III.iv Foreground Classification**; **III.v Behavior Description**; and **III.vi Automated Response**. So far, the majority of our focus has been put on the lower levels of the architecture, with higher levels being used primarily for providing the demonstrations. This will likely be apparent in the disproportional level of detail found in the different subchapters.

III.i Region & Feature Detection

One of the most critical and challenging component of any computer vision system is the detection and tracking of low-level features. Even small detection errors can significantly alter the performance of routines further down the pipeline, which often requires the addition of cumbersome ad-hoc techniques. Compounding this challenge, low-level detection functions must process huge amounts of data, in real time, over extended periods. This data is frequently corrupted by the camera's sensors (e.g. CCD noise, poor resolution, motion blur, etc.), the environment (e.g. illumination irregularities, camera movement, shadows, reflections, etc.), and the objects of interest (e.g. out-of-plane rotation, deformation, occlusion, etc.). Recent interest-point detection algorithms including the Harris-Affine, Hessian-Affine, Salient Regions, SIFT, MSER, and others,

have shown great promise in their ability to detect object-features in a reliable and repeatable way, even when objects are shifted in their orientation, scale, illumination, blur, etc.

Despite the successes of modern feature detection algorithms, they still pose challenges for use in real-time surveillance applications:

1) Many algorithms are computationally intensive and cannot be run in real time on standard systems.

2) Since each feature detector typically responds to specific image features (e.g. corners, lines, blobs), it is often necessary to run multiple algorithms simultaneously, thus imposing further challenges for real-time operation.

3) Low-resolution imagery, combined with the significant variation in orientation, scale, posture, illumination, etc. of target objects makes the identification of high-value interest-points much less robust. This is compounded by the problem that, in many cases, interest point detection algorithms respond most strongly to the occlusion boundary between objects and their background. This is problematic for region comparison since there is no guarantee that the descriptors will remain unchanged as the object moves through the environment.

4) Although interest point models are good at identifying previously trained objects, they are not well suited to surveillance-type implementations that require the detection of untrained objects. Surveillance systems differ

from typical object recognition systems, in that they are expected to detect unknown foregrounds that have appeared against a known background.

5) Current interest-point detection algorithms frequently require computationally intensive model generation and storage that can only be executed as an off-line process. This strategy is less desirable for surveillance-type implementations, which are typically expected to train and store foreground models on-line, in real time. Real-time model generation is necessary to allow object tracking to resume after objects are lost during occlusion, after an object temporarily exits the frame, or when an object passes from the view of one camera to the next.

Mikolajczyk conducted a detailed comparison of the popular affine-invariant interest point detectors and concluded that the Maximally Stable Extremal Algorithm was among the best in terms of both speed and repeatability (Mikolajczyk, et al., 2005). Despite this success, Mikolajczyk concluded that inherent limitations of the algorithm would likely necessitate that it be combined with one or more complimentary feature detection techniques. Other researchers have reached similar conclusions when designing robust feature-based algorithms. Sivic found MSER detectors and Harris corner detector to offer a good representation of video frames (Sivic & Zisserman, 2006). It is primarily for these reasons that we have focused on offering our own improvements to the MSER algorithm for use in our computer vision system. Our primary goal was to produce a

variant of the MSER algorithm that could offer improved stability performance, could detect a wider range of affine covariant features, and could maintain real-time operation.

Our algorithm has been designed to detect maximally stable region features, edge features, corner features, and contour features, all within a synergistic framework that allows the stability characteristics of each detector to mutually reinforce the others. We have additionally designed our feature detection algorithm so that it can be used in the place of traditional background-foreground segmentation techniques. This extension could greatly simplify a general surveillance system, since the features detected during background modeling and foreground segmentation could be applied directly to foreground modeling and search.

The remainder of this section describes the algorithms used in our image processing architecture. For consistent illustration, processing steps will be shown as they are applied to the raw image shown in **Figure III-1**. This image was extracted from a sequence of images we refer to as our “Homework” sequence, which involved a person manipulating homework-related objects that include a book, a laptop, a computer mouse, and a soda bottle. The grayscale version of the image (right) is used for the Canny edge detection. The RGB image (left) is used as input for our other algorithms. Most images will be displayed using their actual size, as shown below (320x240). When it is necessary to show additional detail, displayed illustrations will be enlarged to double size (640x480).

Figure III-1: Unmodified input images from our '*Homework*' scenario.

Figure III-1 shows a screenshot containing unprocessed video frames. The color image (left) is used for our MSER algorithm. The grayscale image (right) is used for Canny edge-detection.

An overview of the processing steps involved in our approach is provided in the algorithm below, with details offered in the following sections.

Proposed MSER algorithm overview:
Edge Detection: Uses the Canny algorithm to detect edges
Line Detection: Uses the TINA toolkit to identify linear segments within the strings of Canny edges
Pixel Scoring: Calculate local texture gradient for all image pixels in the image
Sorting: Sort pixels using their gradient measurement
Masking: Mark pixels that are adjacent to Canny-detected lines as off-limits to 'blocking'
Complete Blocks: Process ordered pixels: If a pixel is completely surrounded by (non-line) pixels that have not yet been grouped into a block, form a new 3x3 block containing that pixel and its 8 neighbors
Incomplete Blocks: Process ordered pixels: If a pixel has not yet been grouped into a block, form a new block containing that pixel and any connected adjacent unplaced neighbors
Block Scoring: Compute color difference measurements between all blocks that share an edge
MSER detection: Apply MSER algorithm using pixel-blocks (instead of pixels) as the primary element
Region Expansion: Leaf-nodes in the MSER hierarchy are expanded by incrementally merging unassigned blocks in order of increasing gradients. Region information is propagated up the tree so that every cross-section of the MSER hierarchy represents a densely segmented image.
Region Pruning: Produce a sparse representation of the MSER hierarchy that contains the most stable regions.
Sub-Region Clustering: Group blocks within each region in preparation for the tracking phase

III.i.1 Edge & Line Detection

Edge detection is a popular computer vision tool that is used to identify image regions displaying abrupt changes in brightness. These areas are considered useful because of the frequency at which brightness discontinuities correlate with important world-features including depth discontinuities, changes in surface orientation, surface markings, changes in material, changes in illumination, etc.

Not only are line features robust and reliable, but it is also common for them to show the boundaries between foreground and background. In an ideal case, detected edges would form unbroken outlines around each image region, allowing objects to be cleanly segmented from the background. The resulting contour contains potentially useful and compact information about the object, however in real world video feeds, extracting complete contours can be difficult and error-prone. Contour edges are often interrupted by image artifacts or by regions where contrast is low between foreground and background. Modern edge-tracing algorithms are reasonably effective at connecting edge segments and removing stray detections, with post-processing algorithms offering further improvements (Loss, Bebis, Nicolescu, & Skurikhin, 2009), but there is still no guarantee that complete boundaries will be formed. Even if perfect boundaries can be guaranteed, additional algorithms are required to accomplish the actual foreground segmentation.

In contrast to edge detection, the MSER algorithm efficiently identifies fully enclosed regions, but it is more sensitive to the quality of the color-gradient. Theoretically, even if two regions have high gradients spanning all but one shared pixel, that pixel could create enough of a bridge between the regions to prevent them from

being represented as independent detections. This characteristic is particularly limiting when the algorithm is applied to real-world videos since the image gradient can be degraded by noise, movement-blur, shadows, reflections, etc. Additionally, even when color-gradient is of good quality, the MSER algorithm still cannot detect a well-localized perimeter. Since stability is computed as a global property of the region, that region can only be as stable as its weakest edge. Put simply, adjacent regions will grow toward each other until their separation is breached by a single connecting pixel. From that point forward, these regions will be treated as one, making it impossible to precisely identify pixels that further defined the separation. For illustration, consider a colored circle over a similarly colored background gradient, which has been blurred to obscure the precise location of the boundary (**Figure III-2** left).

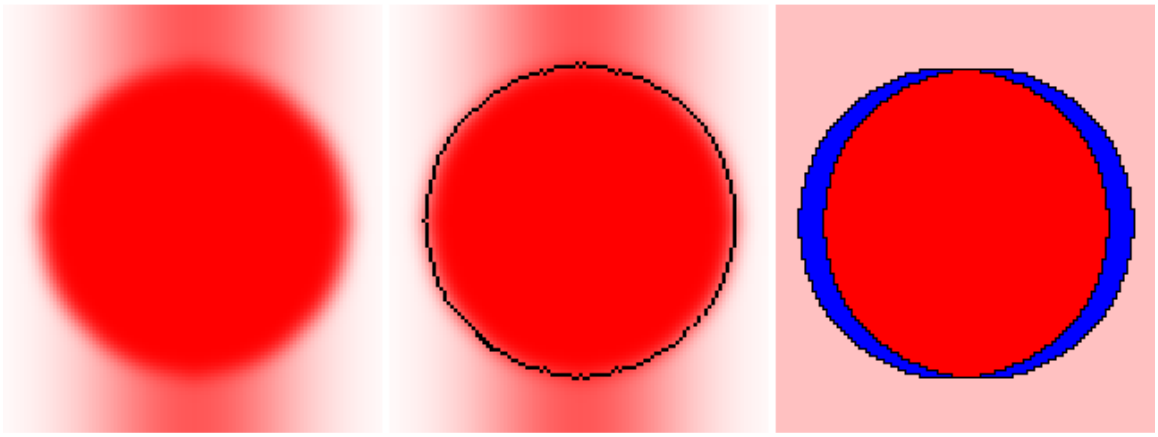


Figure III-2: Edge localization from edge-detection and the MSER algorithm.

Figure III-2 Illustrates the challenges associated with using a MSER algorithm to accurately localize region edges. The original image is shown on the left. Canny edge detection (black pixels) has been applied to the center image. An optimal detection of two MSERs (shown in pink and red) is shown at the right. Pixels shown in blue are not associated with either region since the two regions merge together before either region can be further expanded.

Though simplified, this image resembles a common real-world problem where portions of the foreground may partially blend into the background. The center image shows the input image after applying an edge detection algorithm. As can be seen, boundaries are identified with precision, though additional algorithms would be necessary to segment the foreground circle from the background. The right image shows the MSER algorithm at its optimal threshold value. For clarity in distinguishing between regions, pixels are colored using the average values of all pixels within the associated region. Blue areas represent pixels that contain gradients beyond the current threshold, so they have not yet been placed by the algorithm. Here, the gradient of the unplaced pixels is greater than the gradient corresponding to the top and bottom of the circle and the circle will merge with its background before any additional pixels can be placed. Although this algorithm does segment out two distinct regions, the best possible estimation of those regions will be missing those pixels marked in blue.

In this dissertation, we propose a framework that combines the MSER feature detector with edge detection, and does so in a way that preserves advantages from each. Our goal is to efficiently detect maximally stable regions that can either be represented using traditional MSER methods, or can be represented using a connected string of well-defined boundary pixels. Applying our algorithm to the sample image produces the results in **Figure III-3**. Like in the previous illustration, the left image shows the original input image, the center image shows the results of edge detection, and the right image shows the results of our MSER algorithm. In this example, our algorithm detected two regions: one corresponding to the circle; and one corresponding to the background. Since

the edge detection algorithm is used to confine evolution of the MSER, the resulting outline of our MSER almost exactly matches that of the edge detection algorithm.

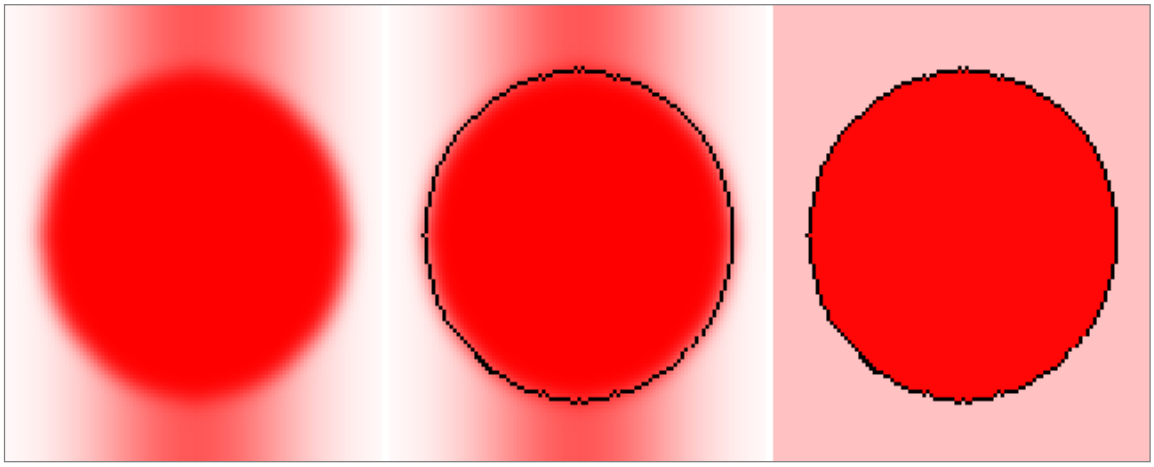


Figure III-3: Edge localization from edge-detection and the proposed algorithm.

Figure III-3 illustrates the proposed solution to the MSER edge localization problem. The original image is shown on the left. Canny edge detection (black pixels) has been applied to the center image. The right image shows the two MSERs detected by our system. Our multi-pass region expansion algorithm allows edges to be accurately localized.

Figure III-4 shows a real-world example of how our MSER algorithm can provide reasonable region segmentation. Screenshots were taken from the "homework" sequence of images and shows all regions detected by our algorithm at the same MSER threshold. Since region stability is determined by how long that region can survive incremental threshold increases, it is advantageous for useful regions (regions with high correspondence to real-world objects) to survive through higher thresholds. The left image shows the extent that merging has taken place without applying detected edges. The right image shows the extent of merging with edge detection applied as a preprocessing step.

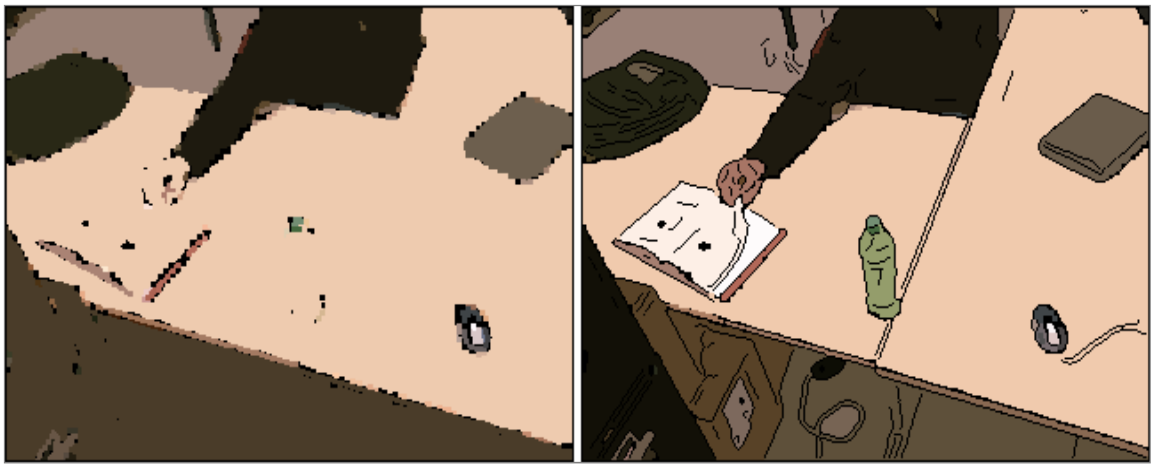


Figure III-4: MSER stabilization using edge detection.

Figure III-4 shows a real-world example of how an edge detection algorithm can stabilize MSER evolution. MSER evolution without edge constraints is shown on the left. MSER with edge constraints is shown on the right.

As can be seen, the detected edges have helped define boundaries between dissimilar regions, thus increasing the MSER threshold required before those regions merge. This also tends to improve the quality of the boundaries between regions (e.g. straight boundaries appear straighter).

III.i.1.1 Canny Edge Detection

The implementation of the Canny algorithm used in this dissertation was originally written by David Murray. Phil McLauchlan provided further improvements for use in the Horatio package of image processing algorithms (McLauchlan & Rahimi, 1992). We selected this version because of its superior speed and efficient data representation. For our implementation, we selected gradient thresholds that produced visually pleasing results using our table-top sequence of images (high_threshold = 5.0 &

low_threshold = 2.75). The same thresholds were used for all our sequences. In all cases, the Canny algorithm was applied to a gray-scale image that had been smoothed using a 3x3 Gaussian kernel.

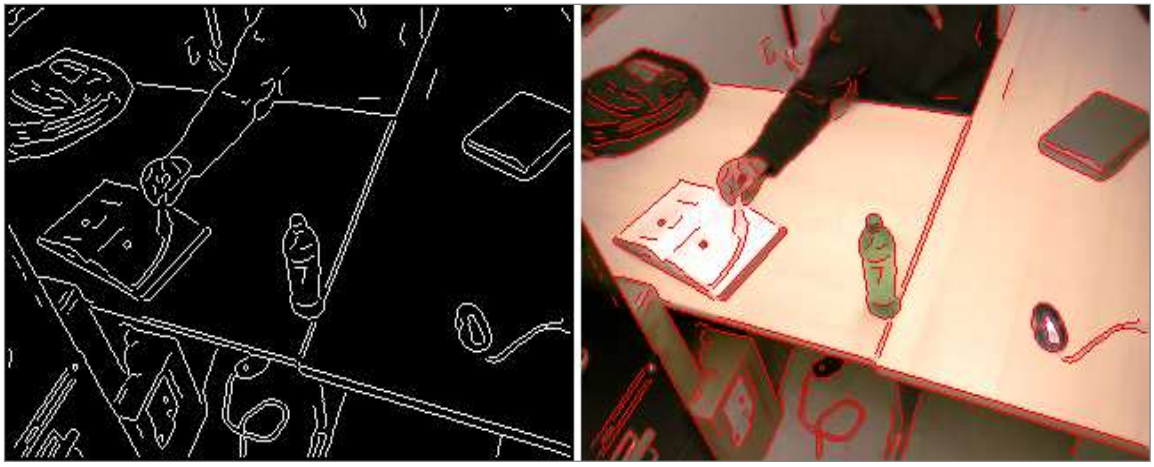


Figure III-5: Results from our application of the Canny algorithm.

Figure III-5 shows the results from our application of the Canny algorithm. Detected pixels only are shown on the left (white). Detected pixels superimposed over original image are shown on the right (red).

III.i.1.2 Edge Linearization

Since edges detected by the Canny algorithm are represented as ordered strings of edge-pixels, curve-fitting algorithms can be applied directly to the existing data-structures. Line and curve fitting is useful because it further compresses the stored data, can provide useful features for tracking and identification, can allow the computation of affine-invariant features such as corners or maximal curvature points, and can be used to interpret two and three-dimensional structure for object or scene reconstruction.

The line fitting algorithm used in this dissertation was part of the TINA image processing library (Pollard, Porrill, Thacker, & Bromiley, 1987). The algorithm identifies

straight segments within a string of edges provided by the Canny edge detector. TINA also offers a curve-fitting algorithm which was not used here.

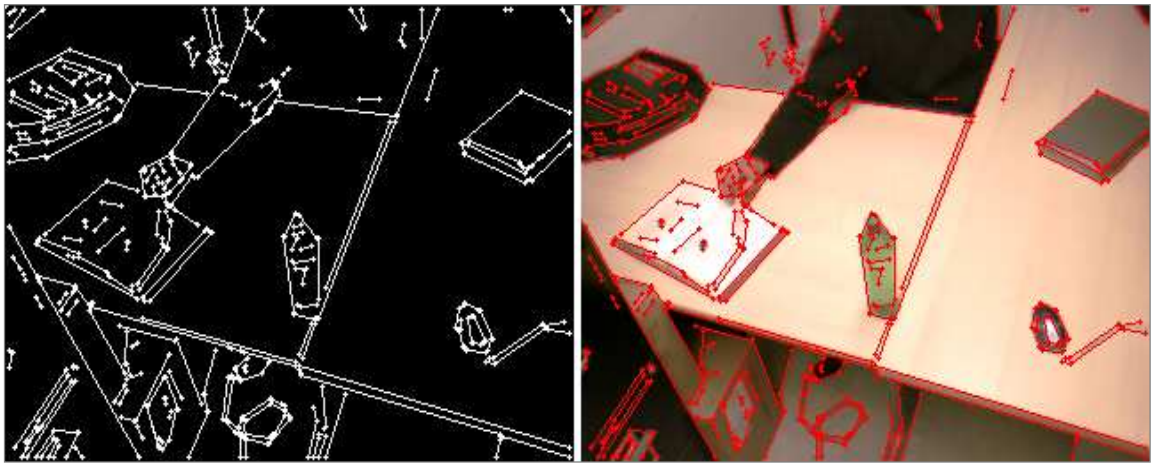


Figure III-6: Results from our application of the line-fitting algorithm

Figure III-6 shows the results of our application of the line-fitting algorithm. Detected lines and endpoints are shown on the left (white). Lines superimposed over original image are shown on the right (red).

III.i.2 Region Detection

As was mentioned previously, the Maximally Stable Extremal Region (MSER) algorithm was determined to be among the best in terms of computational speed and repeatability (Mikolajczyk, et al., 2005). The detected blob-like features also have properties that differ significantly from, and contrast nicely with, the line-segments, corners, and curve-maximals that are detected in other parts of this architecture. The MSER features are also useful, in that it is easy for humans to look at, and understand why features were detected, and what they correspond to in the real world. For example, there can be a reasonable expectation that a green bottle against a solid background will

be detected as a green bottle-shaped region. A final advantage of MSER detection is that unlike corners or edges, which may represent depth-discontinuities or borders between objects, the area inside MSER-detected region typically corresponds with a single object or a single part of an object. Therefore, when generating descriptors for these regions, it is possible to do so without inadvertently including background information into the model.

The MSER detector described in this dissertation was designed to provide a reliable and relatively dense set of region detections, which have perimeters that are primarily bound by detected lines and corners, while requiring a minimum amount of computation. This allows the flexibility of modeling regions using traditional region-based properties including the centroid, dimensions of a best-fit ellipse, color histogram of contained pixels, or texture of the contained region. It also allows contour-based models, line-segment reconstructions, or even models based on groupings of corners or curve-maxima.

III.i.2.1 Pixel-Blocking

Traditional MSER algorithms are designed to operate on individual pixels. Pixels are sorted according to their intensity value (lowest-to-highest or highest-to-lowest), and are added to an empty image in that order using to the following algorithm:

Pixel addition rules using the traditional MSER algorithm:

Pixels contain values ranging from 0 to 255
Process all pixels with value '0', followed by pixels with value '1', … , until all pixels are processed
    If the pixel is adjacent to one existing region
        Add pixel to existing region (increase region's size by one)
    If the pixel is adjacent to two or more existing regions
        Merge all adjacent regions into the adjacent region with the greatest area
        Add the pixel to the new region (the new region's size is the sum of all regions, plus one)
    Otherwise: There are no existing regions adjacent to the pixel
        Create a new region to contain the pixel (region has a size of one)

Once all pixels of a certain intensity value have been placed (e.g., after all pixels containing intensity of 'n', where 'n' is a value within the range of 0-255), a measurement of growth-rate is made for all existing regions:

$$growth\_rate = \frac{region\_area_{(i+w)} - region\_area_{(i-w)}}{region\_area_{(i)}}$$

i = iteration during which pixels of a specific intensity are processed
w = constant value that represents a range of iterations

If the growth rate exceeds a predefined threshold, it generally means that two collections of pixels have merged since the previous threshold. In this case, growth observations of the previous regions will cease and be stored for subsequent analysis. Growth information is then reinitialized for the new region. After all pixels have been processed, a hierarchy of regions will have been recorded, with each region being associated with an intensity-threshold of first appearance, and an intensity-threshold of the last observation. The optimum representation of this region is determined to be the collection of pixels that appeared during the threshold where the growth-rate was at its minimum. The final algorithm output detects either clusters of darker pixels that are surrounded by lighter

areas (if pixels were sorted lowest-to-highest), or clusters of lighter pixels surrounded by darker areas (if pixels were sorted highest-to-lowest). To increase the overall number of detections, the algorithm is often executed using both orderings.

An obvious extension of the traditional algorithm is to incorporate color information that is available in the image. This can be accomplished by applying the algorithm to each of the three individual color channels (Corso & Hager, 2005) (which often offers nominal gains while requiring three times the processing), or by applying the algorithm to the inter-pixel differences found across the image (Forssen, 2007). Using this technique, cluster elements (referred to as pixel “cracks”) are assigned values that are a function of the color-difference between adjoining pixels. Since both vertical and horizontal “crack” elements are needed to fully interconnect the image pixels, the number of elements needed is approximately equal to the number of pixels, times two.

Just like in the original algorithm, the elements are sorted and processed in order, from lowest value (connecting most similar adjoining pixels), to highest value (connecting least similar adjoining pixels). Regions grow when “crack” elements connect adjacent groups of pixels. Region sizes are monitored and recorded when they display a stable size over a range of thresholds.

Although the color-based algorithm offers a slightly higher number of detections, the resulting regions tend to be less stable than those in the original algorithm. This problem is largely due to the high color similarity between nearby pixels. Unlike the original MSER algorithm, which tends to evolve evenly through the entire range of intensities, the majority of merging in color version tends to occur during the first few

iterations of the algorithm. This makes it difficult to distinguish between stable regions and unstable regions. Forssen offered various normalization strategies to force region growth to occur over a greater number of thresholds, but these strategies inevitably increase sensitivity to pixel noise. Various smoothing techniques can be used to reduce pixel noise, but these are somewhat incompatible with solving the original problem of high spatial consistency.

Besides providing a limited gradient-to-noise ratio, running the color-MSER algorithm on individual pixel "cracks" can be computationally expensive and can also introduce certain artifacts. Specifically, because of the way digital images are captured and stored, thin features in the environment will be detected very strongly when they align with the pixel matrix and weakly when they appear at an angle to the matrix. To remove this bias, most MSER algorithms require an additional processing step to remove long thin regions. A similar, but more difficult artifact to identify is when thin strings of pixels act as bridges to connect adjacent regions.

The algorithm proposed in this dissertation attempts to reduce the shortcomings associated with processing individual pixels by applying a preprocessing step that groups pixels into slightly larger blocks. Block formation is achieved using a simple greedy algorithm that clusters pixels into groups ranging in size from 1 to 9 pixels (designed so each group will fit inside a 3x3 pixel square). Instead of spanning individual pixels, the basic element in our algorithm spans adjacent blocks. Here, the element weight is a function of the color-difference between blocks, where each block color is the average of

contained pixels. A MSER algorithm is then used to group block-elements, in the same way that the original algorithm grouped individual pixel-elements.

This blocking strategy is effective at both reducing pixel noise (through the averaging process), and increasing the variability in element weights (since the increased separation of blocks allows more of the gradient to be measured). These advantages are gained without having to balance normalization with smoothing, as is done in traditional algorithms. Additionally, the blocking process is biased toward finding regions that are at least 3-pixels thick. This almost completely eliminates long thin regions (or regions connected by thin pixel strings), and does so without requiring additional processing steps.

Speedup is achieved by reducing the number of iterations necessary for the more intensive MSER algorithm. For example, our “homework” sequence contains 76,800 pixels (in each 320 x 240 pixel image); the traditional color algorithm requires 153,040 elements to represent the vertical and horizontal connections (width x height – width – height). In contrast, our implementation requires an average of 14,649 blocks (5.24 times fewer than the original number of pixels), and 36,824 connection elements (4.17 times fewer). The observed speedup using our blocking algorithm is just over a factor of two.

Pixel-Blocking Algorithm

This preprocessing step was designed to produce a smaller input dataset for the MSER algorithm. It works by enlarging the basic elements of an image in a way that minimizes loss of texture information. This is achieved by using the underlying color

gradient to determine the optimal order to enlarge individual pixels into 3x3 square elements. Details will be provided in following sections.

III.i.2.1.1 Pixel Scoring

The first step the blocking algorithm is to assign a score to all image pixels using the local color gradient. For efficiency, the gradient is computed using the diagonals of the surrounding 3x3 pixel-square. Specifically, the score is set to the color difference between the top-left and bottom-right pixels, plus, the color difference between the top-right and bottom-left pixels. Score is then capped to an integer value between 0 & 253.

$$weight = \sqrt{(A_r - C_r)^2 + (A_b - C_b)^2 + \left(A_g - C_g\right)^2} + \sqrt{(B_r - D_r)^2 + (B_b - D_b)^2 + \left(B_g - D_g\right)}$$

$$weight = MAX(\, weight, 253\,)$$

$$A = \quad Top\ Left\ Pixel \qquad B = \quad Top\ Right\ Pixel$$
$$C = Bottom\ Left\ Pixel \qquad D = Bottom\ Right\ pixel$$
$$r = red\ channel \quad g = green\ channel \quad b = blue\ channel$$

The logic for using only the difference in gradient across the diagonal directions (opposed to diagonal, vertical, and horizontal) is that it minimizes the number of comparison operations without significantly reducing the sensitivity to the gradient. Since images are smoothed before processing, using all 8 surrounding pixels would be largely redundant. Diagonal differencing was selected over horizontal and vertical differencing because the larger separation along the diagonals maximizes the gradient measurement.

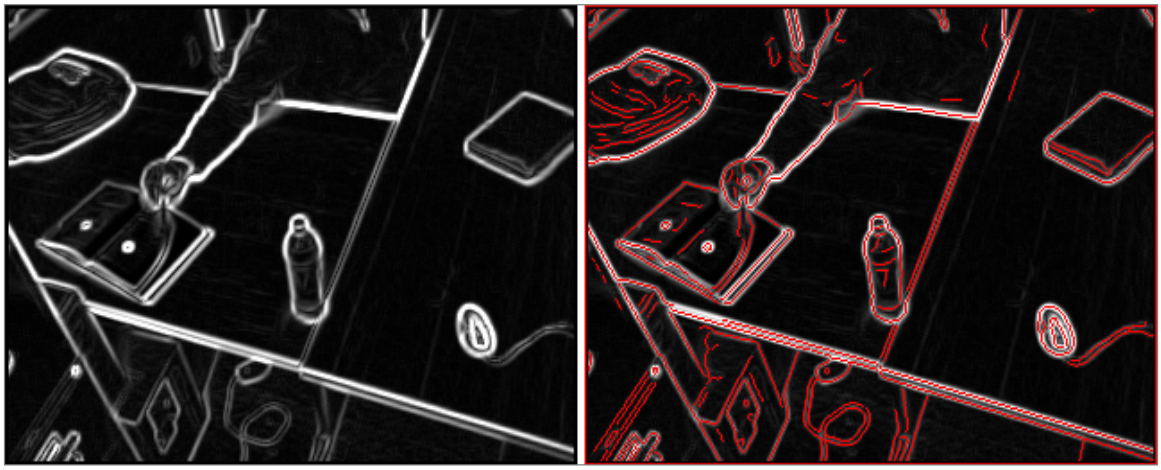


Figure III-7: Detected image gradients shown with corresponding lines.

Figure III-7 shows detected lines superimposed over the detected gradient. Grayscale pixel values represent the computed value of the color gradient at that location (left). Detected lines are added (red) to show the correlation in their position (right).

III.i.2.1.2 Pixel Sorting

Once texture values are assigned to pixels, pixels are sorted in order, from least textured to most. Since all pixels contain values within the range of 0 to 253, pixels can be sorted very efficiently, in linear time, using a counting sort algorithm.

III.i.2.1.3 Off-Limits Mask Initialization

The proposed grouping algorithm assigns pixels to 3x3 blocks on a first-come first-serve basis (described in the next section). To keep track of pixels that have already been placed, those positions are added to an off-limit mask. If the mask has already been filled for a specific pixel location, that pixel is ignored during the placement algorithm. An advantage of using the mask structure is that it allows other off-limit locations to be assigned. For example, the pixels around the image's outer perimeter are marked as off-

limits. This prevents regions from growing against the edge of the screen to simplify later stages of the architecture. Also, pixels corresponding to the Canny-detected lines are marked as off-limits to allow them to act as barriers to confine region growth.

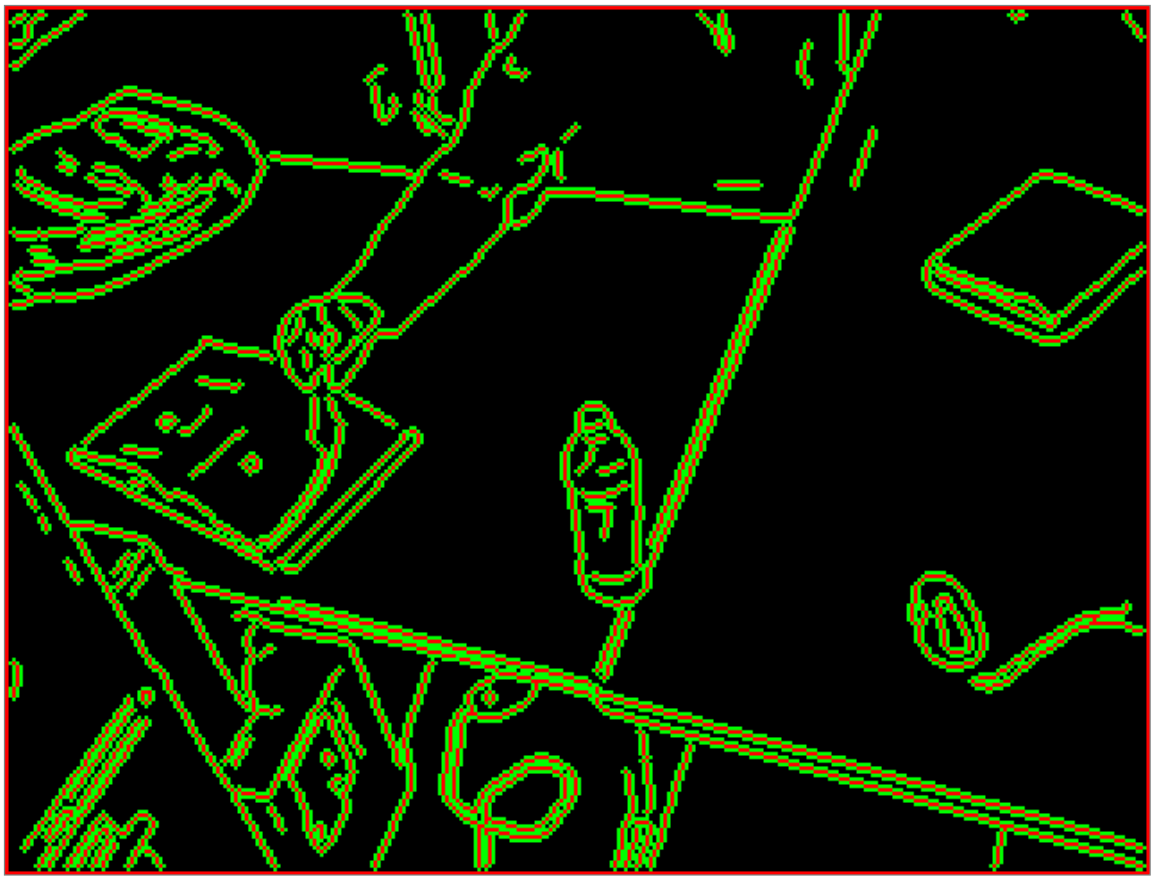


Figure III-8: Initialization mask for constraining subsequent region growth.

Figure III-8 shows an enlarged representation of the initialized mask. Red pixels represent those that are completely off-limits to region formation. Green pixels are adjacent to line pixels and are used as an optimization for pixel grouping (described in the following section).

III.i.2.1.4 Complete 3x3 Square Construction

After pixels have been scored and sorted, they are processed in order, starting with those of least texture, and progressing through those of greatest texture. In the first

stage, complete 3x3 pixel blocks are formed on a first-come first-served basis. For each processed pixel, the surrounding 3x3 pixel square is searched for pixels that have been previously placed, or pixels that belong to a line. If none are found, a new 3x3 block is formed and stored at that location. The corresponding pixels are then marked as off-limits.

It should be noted, that for efficient operation, it is not necessary to check all 9 pixels in the 3x3 square before recognizing that the region is available for placement. Since everything on the off-limit mask has a thickness of two or more pixels, it is sufficient to check only the center pixel and the 4 corner pixels. To allow this simplification, it was necessary to thicken the line-pixels (shown in green on **Figure III-8**). The blocking algorithm is shown below.

Algorithm for detecting complete 3x3 squares:

Process all pixels in order of texture, from least to greatest:
    If center the pixel is black and the 4 corner pixels are not red: The location is available.
    Otherwise: The location is not available.

    If the location is available:
        Make a new element containing the surrounding 3x3 square.
        Mark the corresponding 3x3 square on the off-limit mask red.

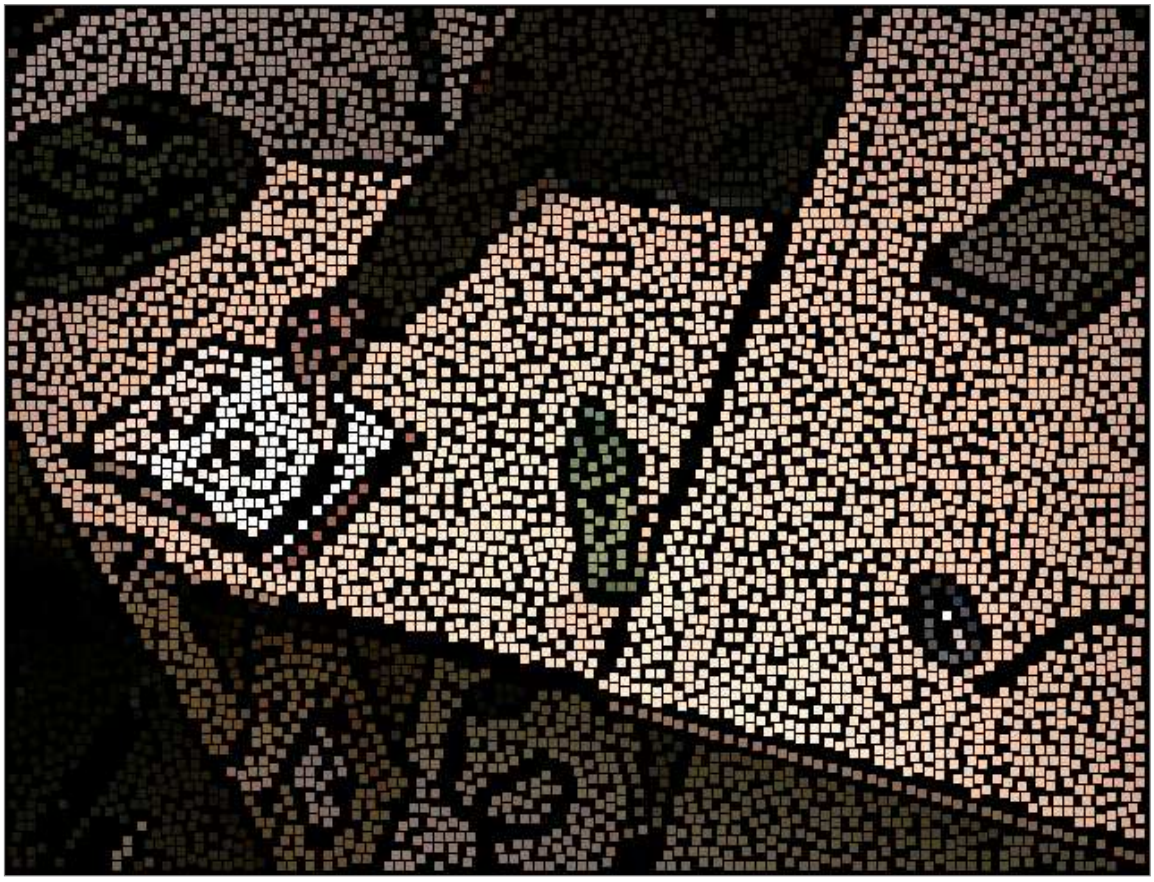


Figure III-9: Initial pass of the local pixel clustering process.

Figure III-9 shows the results after the first pass of our blocking algorithm. Pixels are grouped into 3x3 squares on a first-come-first-served basis. Resulting elements are displayed using the average color of the contained pixels.

III.i.2.1.5 Incomplete 3x3 Square Construction

After every possible complete square has been formed, a second pass is made through the sorted image pixels. The second pass identifies incomplete squares that can fill the remaining spaces. Depending on the availability of surrounding pixels, the incomplete square can contain as few as one pixel (just the center pixel), or may contain the center pixel, plus between 1 and 7 of the surrounding pixels (if the first-pass

algorithm functioned properly, it should be impossible to find a location with all 8 surrounding pixels available). To prevent fragmentation of blocks, corner pixels are only added if one of the adjacent edge pixels has also been added. The second pass of the blocking algorithm is shown below.

Algorithm for the second-pass clustering:

Process all pixels in order of texture, from least to greatest:
    If center pixel is black: The location is available.
    Otherwise: The location is not available.

    If the location is available:
        Begin a new region element that contains the center pixel.
        Add any available edge pixel (pixels immediately adjacent to the center).
        Add any available corner pixels (but only if they are adjacent to an edge pixel).
        Mark the corresponding pixels on the off-limit mask red.

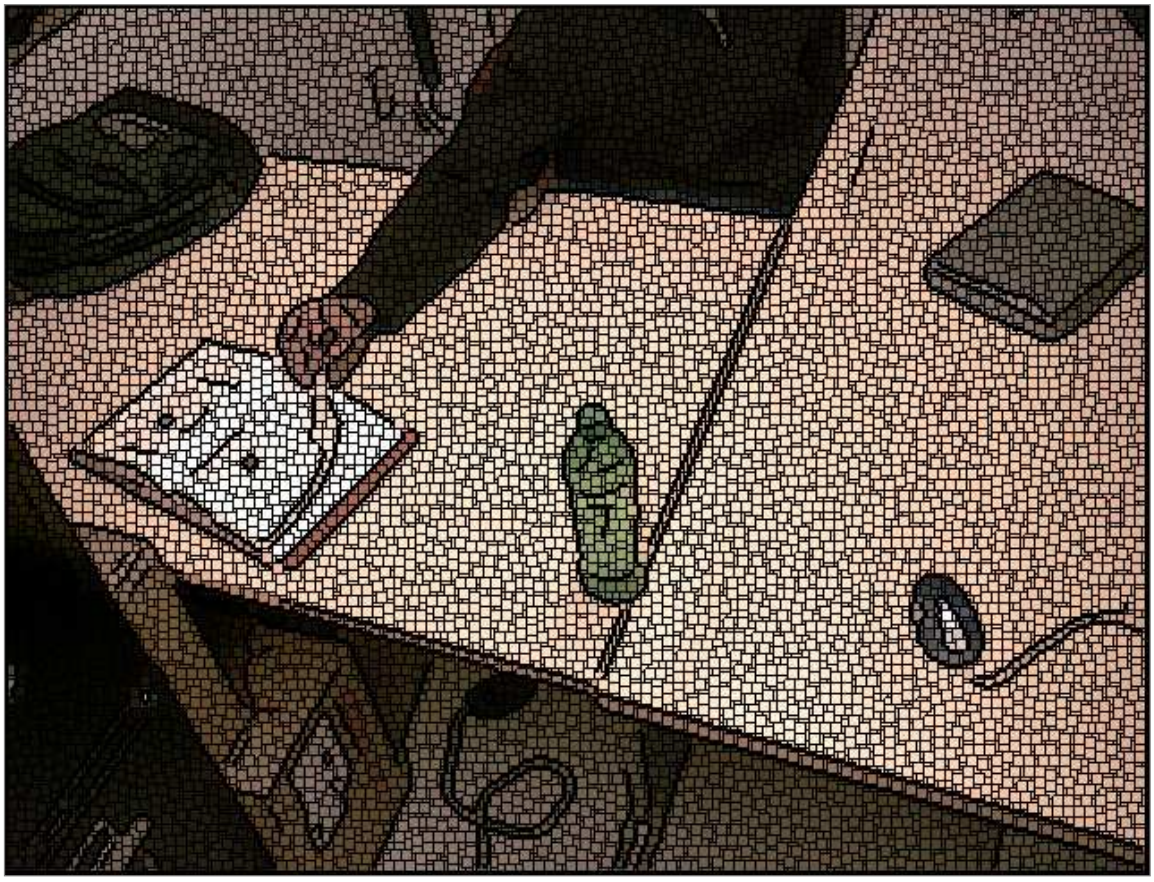


Figure III-10: Second pass of the local pixel clustering process.

Figure III-10 shows the result after the second pass of our clustering algorithm. Each cluster contain between 1 and 9 pixels.

III.i.2.1.6 Square Connection Scoring

The traditional color MSER algorithm is applied to the set of elements connecting adjacent image pixels. In the proposed algorithm, elements represent connections between adjacent blocks. In our algorithm, block connections are identified by checking block membership of the pixel at location (x,y), against the block membership of the pixel at location (x,y+1). If both pixels correspond to the same block, or if one of the

pixels corresponds to no block, that pixel is ignored. Otherwise a new connection is stored into an array.

After all vertical adjacency relationships are checked, the algorithm makes a second pass to identify horizontal relationships across pixels (x,y) and (x+1,y). We split the horizontal and vertical searches to simplify the removal of duplicate connections. Basically, the system only stores new connections when that connection contains at least one block that differs from the previously stored connection. Although this strategy won't catch every duplicate, it does catch most duplicates. Removing duplicates is done for efficiency and has no effect on the operation of subsequent algorithms.

After connections are identified, they are assigned a weight corresponding to the difference in average colors between the associated blocks. Connection weights range from 0 to 253, and can be sorted efficiently using the counting-sort algorithm. The equation below shows the computation used to determine each connection weight.

$$weight = \sqrt{(A_r - B_r)^2 + (A_b - B_b)^2 + \left(A_g - B_g\right)^2}$$

$$A = Region1 \quad B = Region2$$

$$r = red\ channel \quad g = green\ channel \quad b = blue\ channel$$

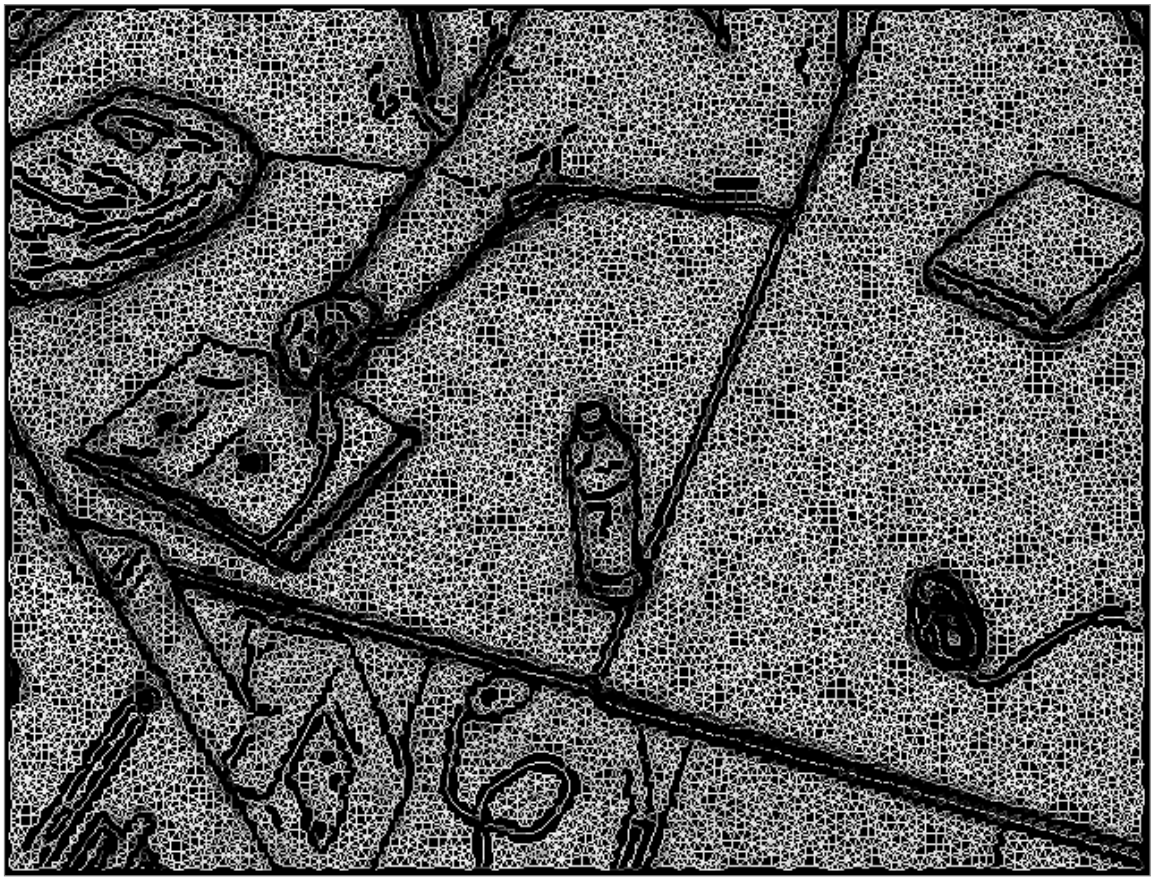


Figure III-11: Connection formation between local clusters.

Figure III-11 shows a visual representation of the connection weights between blocks. Connections are shown as gray lines. Smaller weights (greater similarities between adjacent blocks) are shown using lighter shades.

III.i.2.2 Region Detection

The original MSER algorithm was a type of watershed algorithm that translated pixel intensities into a three-dimensional surface contour, with dark areas represented depressions and light areas represented peaks. The MSER evolution process is akin to elevating a water-table below the contour. As the water begins to fill the lowest lying areas, the resulting pools are labeled as new regions. During this flooding process, the

surface area of each pool is measured at regular intervals. In depressions formed from shallow slopes, one can expect that the surface of the pools would increase rapidly. However, in depressions surrounded by very steep walls, the pool's surface would be expected to change little. It is these slowly changing pools that are treated as stable regions.

Keeping with the contour flooding analogy, it can be imagined that pools may be stable at some levels, but may potentially level out at higher levels. As the water fills beyond this point, expansion may increase, or pools may merge with adjacent pools. This may then be followed by additional periods of stability. This process continues until even the tallest peaks are submerged. At the end of the flooding process, one is left with a list of areas that demonstrated stability. It can be imagined, that even if the contour was changed slightly, the same set of regions should show stability during a second round of flooding. The same occurs with images. The theory is that in pictures taken from different angles, different distances, in different illuminations, or even using cameras of different qualities, the stable regions should be detected in the same places every time.

With the color-MSER, the flooded contour analogy still works, but instead of applying flooding to the original image, flooding is applied to the derivative of the image. So, instead of having peaks and depressions corresponding to light and dark areas, there is a flat landscape representing low texture regions, and higher levy features representing places where the color changes rapidly. To improve the continuity of these levies, our line detection algorithm figuratively constructs walls that reinforce the existing levy system. **Figure III-12** shows an example of how region evolution might occur for one

image pixel corresponding to the backpack found in our “Homework” sequence. In this example, the region containing the pixel displays alternating periods of slow and rapid growth. It is these periods of slow growth that suggests that a stable region has been found. As can be seen by this illustration, there are often multiple levels of nested detections that form a hierarchical tree. The root of the tree corresponds to the final threshold, a point where all regions have merged into one.

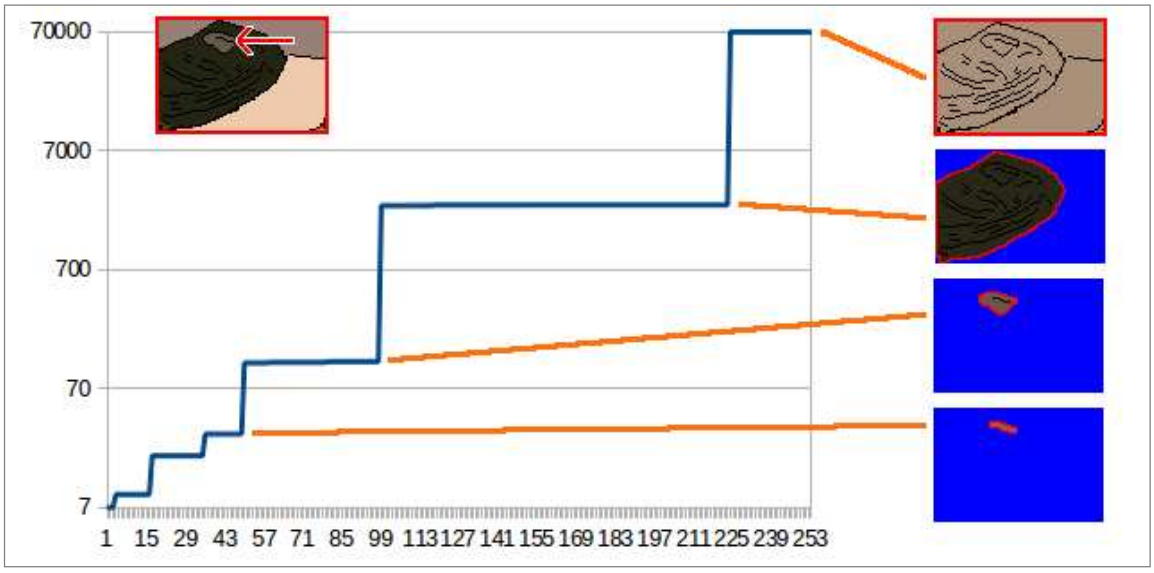


Figure III-12: MSER evolution history of one pixel.

Figure III-12 shows how region size varies with time. For compactness, only the top-left portion of the image is shown. The red arrow shows the location of the pixel being tracked. The y-axis of the graph shows the area (number of pixels) of the region that contains the given pixel. The x-axis shows the evolution threshold (corresponding to the water-table in the previous analogy). The blue line shows how the region size increases with increasing thresholds. The images on the right show stable regions. The first (bottom-right) corresponds to part of a patch on the backpack. The second shows a stable period involving the entire patch. The third image shows a detection of the whole backpack. The final image shows the end of the evolution process, where everything in the image has merged into a single region.

Region Detection Algorithm

The traditional algorithm operates on a set of elements that connect adjacent pixels. These elements are processed in order from the most similar pixel pairs, to the least. Our MSER algorithm operates in a similar way, but instead of operating on adjacent pixels, it operates on elements connecting adjacent pixel-blocks. Each element is assigned a weight related to the difference in color between the two associated blocks. Elements are sorted from lowest to highest weight and are processed in that order.

Algorithm for region detection:

Process all block edge elements in order, from least (0) to greatest (253):
    If the block is adjacent to one existing region
        Add block to existing region (increase region's size by one)
    If the block is adjacent to two or more existing regions
        Merge all adjacent regions into the adjacent region with the greatest area
        Add the block to the new region (the new region's size is the sum of all regions, plus one)
    Otherwise: There are no existing regions adjacent to the block
        Create a new region to contain the block (region has a size of one)

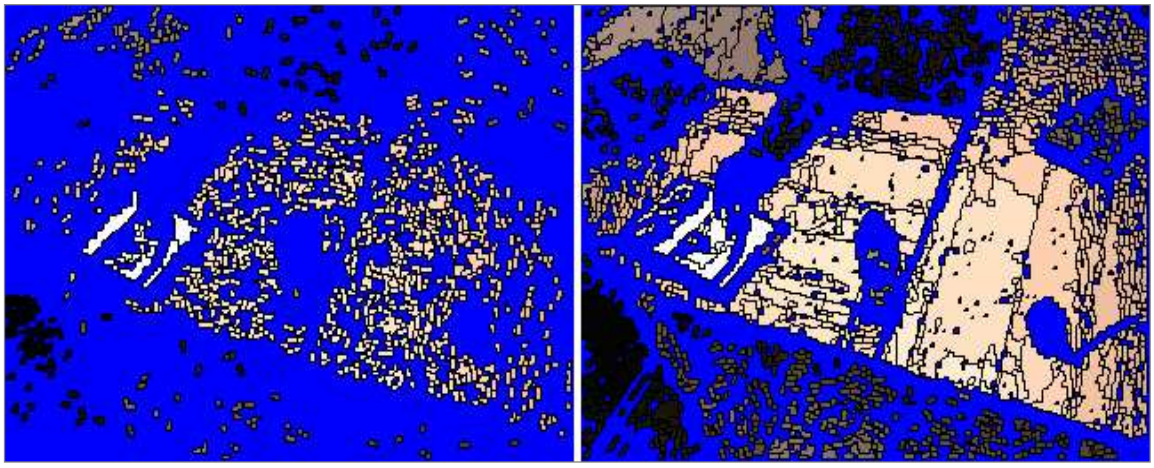


Figure III-13: Example of region growth taken at threshold one and four.

Figure III-13 illustrates the early stages of region evolution. Screenshots were taken after threshold 1 (left), and 4 (right) of the MSER clustering process. In the left image, the majority of blocks have not yet been placed, and those that have been placed are mostly small. In the right image, blocks corresponding to low texture portions of the image have already merged together to form larger regions, while more textured areas (e.g. the hand or mouse) are still absent. Pixels assigned to a region are colored using the region's average color. Unassigned pixels are shown in blue.

For efficient storage and sorting, element weights are restricted to the range 0-255. All elements of weight 0 are processed first, followed by weight 1, and so on. After all elements of a particular value are processed, existing regions are analyzed and their rate of growth is recorded. Regions are left to evolve as long as their rate-of-growth is declining. If an increase in growth-rate is observed, the state of the region before accelerated-growth is recorded as a new MSER candidate. The algorithm continues until all texture elements have been processed.

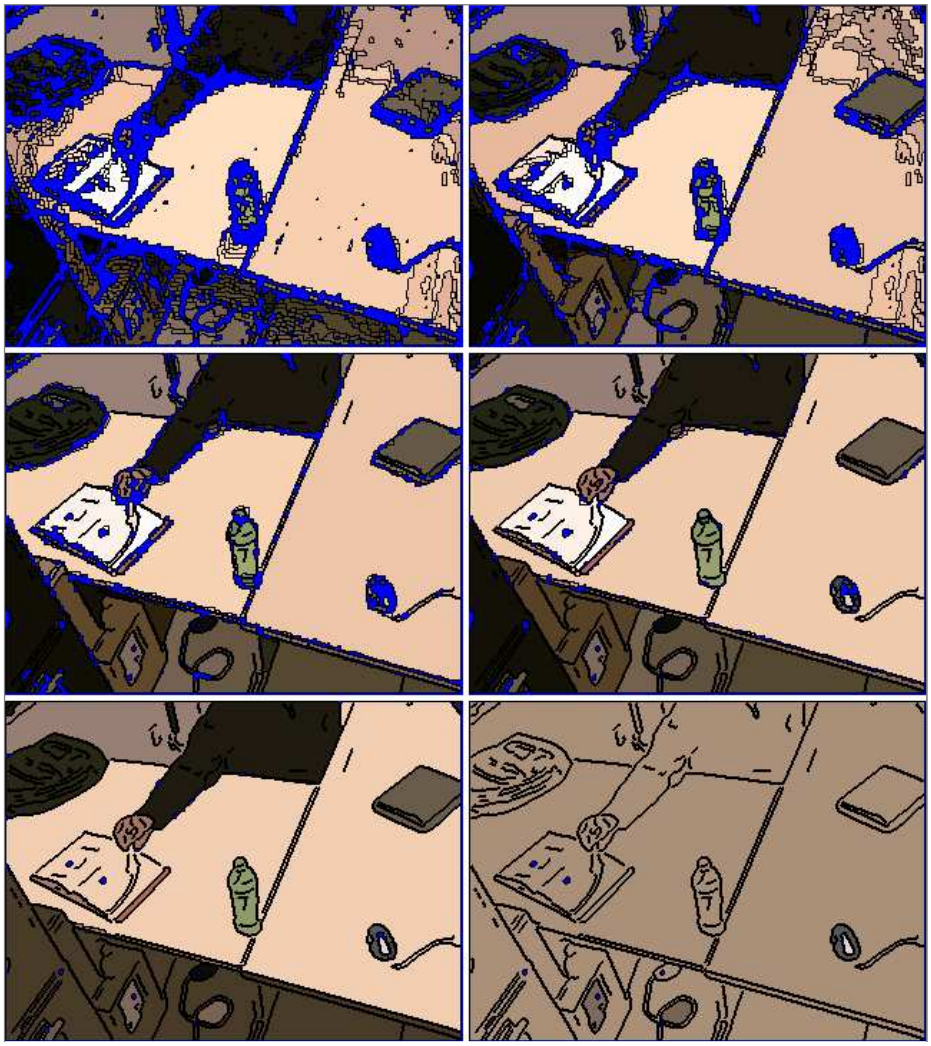


Figure III-14: Example of region growth taken at higher thresholds.

Figure III-14 shows screenshots of the MSER clustering process as they appear at higher thresholds. As thresholds increase, smaller regions merge into larger regions, creating a hierarchical collection of regions. The root of this structure is the region corresponding to the entire image, which is found after the highest threshold. From left to right, top to bottom, threshold values are 8, 16, 32, 64, 128, & 250.

The process of storing regions of incrementally larger sizes produces a hierarchical tree of formed regions, with tree-leaves representing initial clusters, and the tree-root comprising all pixels in the image. Since the criteria for selecting regions is very simple, this portion of the algorithm produces a very large number of detections.

However, because detections are stored in this hierarchical tree structure, traversing and culling portions of the tree can be done simply and relatively efficiently. This will be described in more detail in the next section.

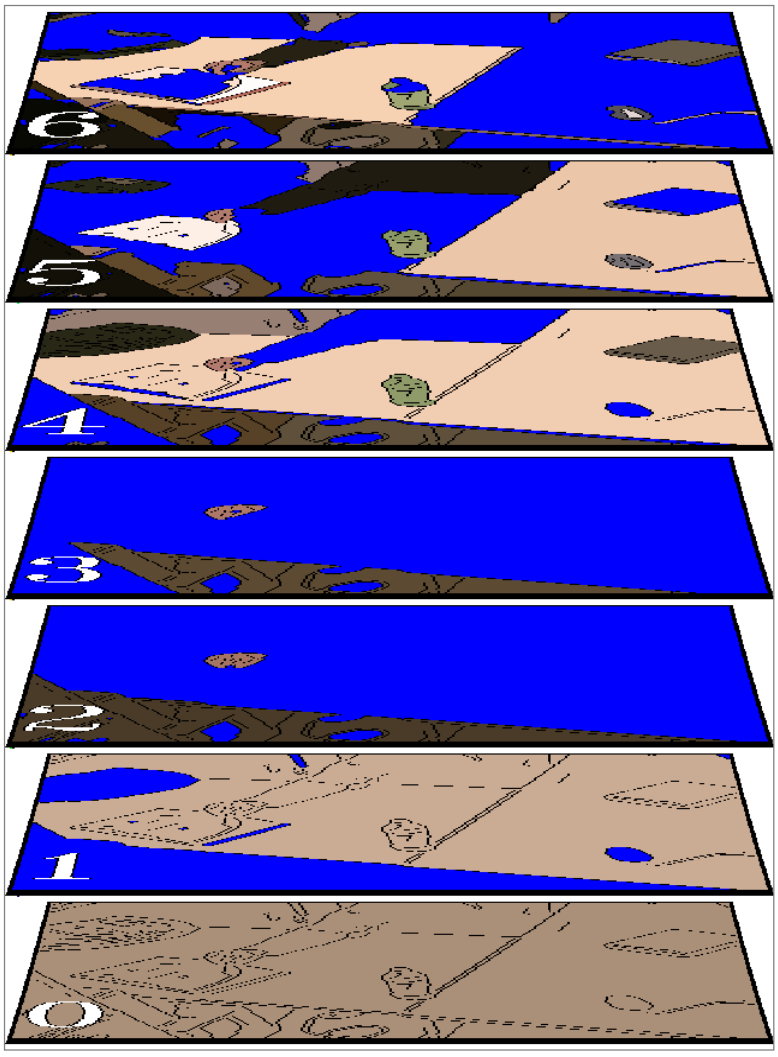


Figure III-15: Hierarchical organization of detected regions.

Figure III-15 illustrates the hierarchical nature of detected regions. Image 0 is the tree's root and contains a single region covering the entire image. Images 1-6 contain incrementally smaller detections, with the top image containing the first regions to form, which are the tree leaves. These images can be represented as a sparse collection of the most stable regions (shown), or as a dense set containing complete image segmentation at every tree level. Blue areas represent those that do not correspond to a stable region.

III.i.2.3 Region Expansion

In the Edge & Line Detection section, we argued that Canny edge detection offers better line localization than the MSER algorithm. Making use of the improved contour precision required an additional modification to the MSER algorithm, which we call region expansion. **Figure III-16**, shows the results of the proposed algorithm (right) against the traditional algorithm (left).

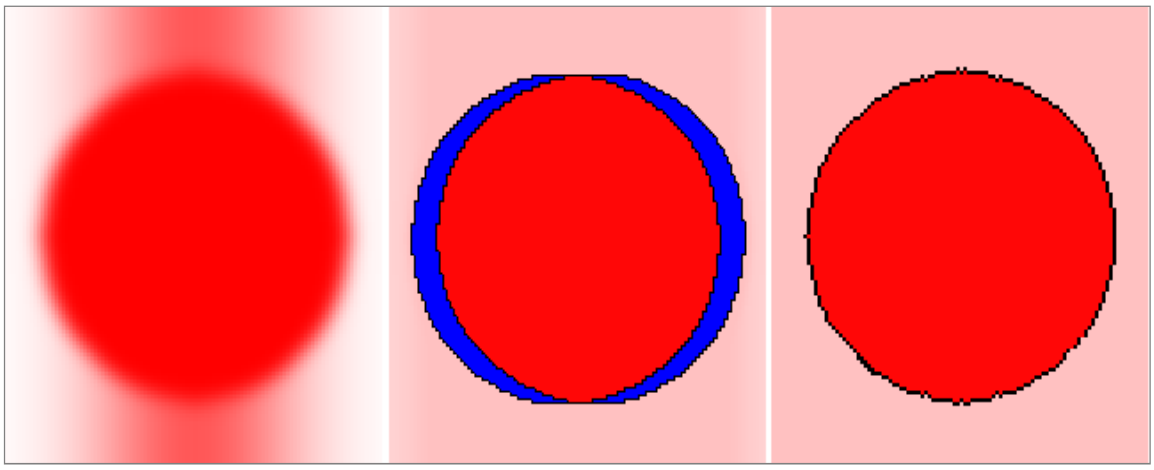


Figure III-16: Example of how region expansion can improve edge localization.

Figure III-16 compares edge location using the traditional and proposed MSER algorithm. The left image shows the unprocessed image. The center image shows two regions obtained after ideal MSER detection (the centered red ellipse appears as one region, and the pink background is the second). The blue pixels represent those cannot be assigned to either region (since the gradients of the unplaced pixels are greater than the gradient separating the circle from the background). The right image shows the results of our algorithm after region expansion has been applied.

Region Expansion Algorithm

As has been mentioned, our algorithm incorporates the Canny edge detector to guide the region clustering process and to offer more precise localization of region edges.

Even with edges guiding MSER evolution, it is still possible that gaps in the contour will allow regions to merge before higher texture pixels have been added. To maximize the extent that clustering can take place and to maximize the precision of our region-contours, we apply a two-phase grouping process. The first phase was explained in the previous section. This produces a hierarchical tree of MSER detections. In the second phase, the leaves from the hierarchical tree are used as seeds. Pixel blocks that could not be assigned during the first phase are then added on a first-come first-serve basis during the second phase. This merging process expands the leaves of the MSER hierarchy to the point where every pixel in the image is contained by exactly one leaf-region. Since the MSER hierarchy preserves relationships between leaf regions and all regions containing those leafs, region filling that occurs at the leaf level can easily be propagated up the tree, thus filling out regions at every level. After propagation, every region will be expanded to their maximum amount (without overlapping adjacent regions), and any cross-section of the hierarchical MSER tree will produce a fully segmented representation of the image. It should be noted that although dense image segmentation might be useful for some applications, it is not necessary for our particular application. To minimize the number of regions we model and track, we prefer to use only the most stable subset of regions from those we have available. Therefore, subsequent representations of our MSER tree will show a sparse set of regions.

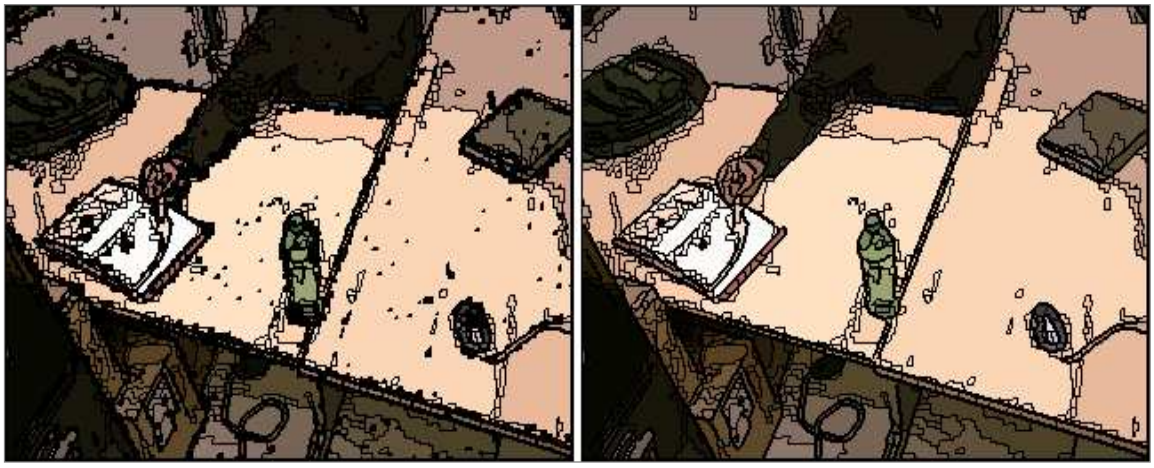


Figure III-17: Example of region expansion.

Figure III-17 illustrates the results of region expansion using our algorithm. The left image contains only leaf regions. These are regions that were detected in the initial stage of MSER evolution and cannot be subdivided into smaller regions. Regions contain small holes and rough edges due to the fact that higher textured pixels were not added to the image until after the leaf regions merged. The right image shows leaf-regions after these additional pixels have been placed. It should be noted, that at this point of our algorithm, there is obvious over-segmentation at the lower levels of the hierarchical tree. This occurs because we maintain very lax criteria for selecting an initial set of regions. We use higher-level algorithms to remove regions that are less stable. This process will be explained in more detail in the next section.

III.i.2.4 Region Pruning

The primary goal of traditional MSER detection is to identify a set of regions that can be detected with high probability, regardless of viewpoint changes, changes in depth, lighting intensity, or even changes in camera focus. To test the fitness of a detection algorithm, they are often put through a test suite that measures the ability for detections to be repeated against images containing these variations. Consequently many of the detection algorithms are designed specifically to perform well on this type of test and one way to improve results is to apply various pruning algorithms that reduce the overall number of detections while preserving those that are most stable. When dealing with the

MSER, these steps are often based on low-level features related to the region's physical dimensions or characteristics measured during MSER formation. Typical criteria for culling regions are as follows:

1) **Growth Duration Threshold:** Regions must exist for a predefined number of iterations during the MSER evolution process.

2) **Size Threshold:** Regions must be larger than a predefined size.

3) **Thickness Threshold:** Regions must be greater than a certain thickness (usually one or two pixels), as very thin structures are generally only stable when aligned with the image pixel.

4) **Relative Region Stability:** If multiple regions of similar size and location are found to be nested within one another, only the most stable one within a specified range of sizes is preserved.

5) **Absolute Stability Threshold:** Regions must have a stability measurement that is above a predefined value.

Depending on the particular application, a possible disadvantage to using test suites to tune detection algorithms is that the reduction in total detections may also reduce the extent that detections cover an image. In some applications, an object of interest may not have sufficient texture to contain any strongly detectable features. In these cases, it might be more useful to have low stability features, rather than having no features at all. Our algorithm attempts to increase image coverage through modifications that make the

region detectors behave in a way that more resembles image segmentation. To this end, our pruning operations are tuned to identify stable regions that favor image coverage over simple stability. Below is a list of modifications that we have applied to the typical pruning algorithms:

1) **Growth Duration Threshold:** We remove this requirement, primarily because we find over-detection desirable in the early stages of pruning.

2) **Size Threshold:** We use a minimum region size of 32 pixels. This significantly reduces the total number of detected regions, while preserving features that are large enough to track.

3) **Thickness Threshold:** We remove this requirement. The preprocessing step we use to cluster pixels into blocks eliminates sensitivity to this particular artifact.

4) **Relative Region Stability:** We compare the stability measurement of each region against those that are larger or smaller. Within a range of 10% larger or smaller, only the most stable region is preserved.

5) **Absolute Stability Threshold:** Traditionally this threshold is applied to a score based on the region's rate of growth. We instead use our own perimeter-based stability measurement.

In summary, our algorithm eliminates two of the five common pruning strategies that are implemented in similar algorithms. We also modify one of the stability-based pruning mechanisms so that it takes advantage of the edge features we have available

from our Canny detection algorithm. Our strategy is to eliminate regions that have low overall stability by removing regions that display low-quality contours. Since contour quality is directly related to the region's stability, this is not an unreasonable approach to take. This approach provides the additional advantage of maximizing the extent that edges are present on detected regions. These edges are used to construct lines and corners, which aid description, tracking, and modeling of each region.

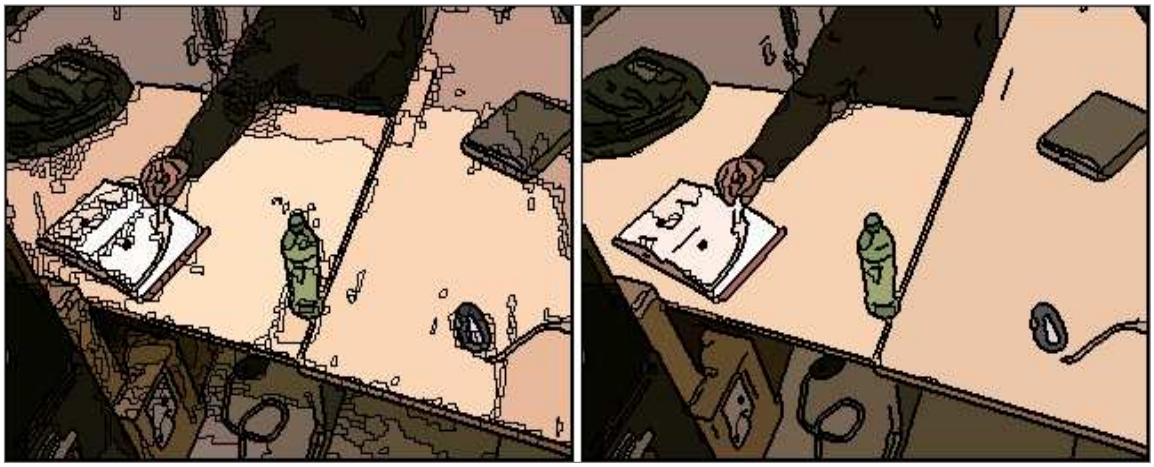


Figure III-18: Example of region culling.

Figure III-18 shows the results of our culling operation. The left image contains the set of regions before our culling operations. Here, there are detections related to shadows, reflections, faint textures, image noise, and other less stable information. The right image contains the regions after culling. The minimum size requirement accounts for the largest reduction in the overall number of regions. Our edge-based stability threshold removes the second-greatest number of detections. These will be described further in the following sections.

Region Pruning Algorithm

As was mentioned in *Region Expansion* (Section III.i-III.i.2.3), a unique aspect of our MSER algorithm is that we apply a second round of clustering that fills gaps within

regions and helps maximize the precision of region contours. Since our pruning algorithms rely on accurate region sizes and contours, we delay pruning until after the *Region Expansion* phase. Pruning is then applied directly to the hierarchical tree structure that is constructed using the parent-child relationships of our detected regions.

III.i.2.4.1 Size Threshold

The MSER algorithm functions by identifying periods of stability that emerge during the iterative expansion of image regions. Since expansion is measured as a proportion of current size (instead of using the absolute number of added pixels), small regions are more effected by noise and small textural irregularities. This creates a problem where statistical fluctuations in growth cause small regions to be incorrectly identified as stable. This is particularly problematic when using our blocking algorithm, because of the increased size of our block-element (over individual pixels). Since our algorithm uses line-based stability as our primary fitness measure (described in *Perimeter Completeness* (Section III.i-III.i.2.4.3)), we do not consider the imprecision in traditional MSER stability measurements to be a significant drawback, especially given the performance benefits allowed by our use of pixel blocks. We do however still find it necessary to remove the large number of small regions that occur.

The typical strategy for reducing inaccuracies in stability measurements is to ignore smaller regions and regions that have not had sufficient time to display a pattern of growth. In our algorithm, regions are only considered if they have existed for more than

two merging iterations and if their size exceeds 32 pixels. This value is selected as it provides a reasonable amount of detail to resolve the objects found in our experiments.

III.i.2.4.2 Relative Region Stability

Since the MSER algorithm is specifically designed to detect regions that are nested within other regions, it is common for detections to appear, which are nearly identical in terms of size, shape, and location. Since these duplicate detections provide little additional value, it is desirable to remove them. For our implementation, we have elected to remove all but the most stable. Each of our regions contains a stability measurement, which is inversely related to the rate of growth over a predefined number of MSER iterations. Here we use a window of 4 iterations.

$$Stability = 1 - \frac{count_{(i)} - count_{(i-window)}}{count_{(i)}}$$

$$\begin{aligned} window &= number\ of\ growth\ iterations\ over\ which\ region\ size\ is\ measured \\ count_{(i)} &= region\ pixels\ during\ the\ final\ iteration\ of\ MSER\ evolution \\ count_{(i-win)} &= region\ pixels\ at\ 'win'iterations\ prior\ to\ the\ final\ iteration \end{aligned}$$

The stability score is used to find local maxima in the MSER hierarchy. Regions are removed from consideration if their score is less than that of a parent (grandparent, etc.) or child (grandchild, etc.) of similar size. In our implementation, a region is considered to have a similar size if the contained number of pixels is within the range of 10% larger or smaller. The culling operation is applied using an algorithm that traverses the MSER Hierarchy tree, and operates as is shown below. In the tabletop sequence of images, this culling operation removes about 15% - 20% of the total number of regions.

```
Algorithm for culling regions of similar size:

Set maximal flags in all regions to true.
For each region:
    Iterate through the region's ancestors (parent, grandparent, great-grandparent, etc.)
        If the ancestor's size is larger than 110% of the region's size:
            Break out of loop (stop checking ancestors)

        If the ancestor has a score that is better than the region's score:
            Set the region's maximal flag to false
        Otherwise:
            Set the ancestor's maximal flag to false

Remove all regions that have maximal flag set
```

It should be noted that this culling operation is largely unnecessary for the implementation of our algorithm. If this operation was not applied, the regions that would have been removed in this step would instead be removed by the line-based culling operation shown in the next section. The primary reason for using this operation is because it requires minimal computation and has the effect of significantly reducing the height of the MSER hierarchy tree. Since the efficiency of subsequent algorithms is more computationally intensive and is a function of tree-height, this step was added primarily to improve overall efficiency.

III.i.2.4.3 Perimeter Completeness

In keeping with our overall strategy of unifying edge detection and region detection, one of our primary culling algorithms is designed to remove unstable regions by removing those that have low quality contours. Since region stability is ultimately a function of the bounding contour, contour quality is something the color MSER algorithm measures anyway. However, where traditional stability measurements make

their determinations using the weakest link of the boundary, our contour-based measurements increases the extent that the entire boundary is considered.

In *Region Expansion* (Section III.i-III.i.2.3), we described a second-pass grouping algorithm that merges unplaced high-texture pixels into detected regions. The primary reason for this additional growth-phase is to expand each region's perimeter to the maximum extent. After doing so, portions of the perimeter (that are not touching the screen borders) will either touch edges detected by the Canny, or they will touch adjacent regions. We base our culling operations off the assumption that region boundaries containing Canny-detected edges are more reliable than those lacking detected edges. To quantify this premise, we assign a score to each region that is based on the ratio of pixels that touch Canny-edges, to total number of perimeter pixels (excluding pixels around the image borders).

$$score = \frac{(pixels_{canny\_edges})}{(\mathrm{p}ixels_{total\_perimeter} - pixels_{canny\_edges} - pixels_{image\_border})}$$

We apply a simple threshold for removing less stable contours. If the number of Canny-detected edge-pixels in the contour is less than half the total number of pixels, the region is removed.

Although this fitness score is reasonably straight forward, counting the number of edge, and non-edge pixels in the perimeter requires a significant number of computational iterations. Because regions are nested within other regions, each edge pixel can be included in 'h' number of different regions, where 'h' is the height of the

hierarchical tree. It is this reason that we use the less computational *Relative Region Stability* (Section III.i-III.i.2.4.2) culling operation first.

III.i.2.5 Sub-Region Clustering

Once stable regions have been detected, it is useful to describe those regions using a reduced set of features. Our complete feature-set will be described in *Feature Representation* (Section III.i-III.i.3), though here we describe an algorithm that will specifically allow our color-related features to be efficiently computed.

The simplest way to represent a region's coloration is to assign a value, which is the average of all pixels contained by the region. Though fast and compact, this value may not adequately represent the region, especially if that region contains pixels of varied color. Alternately, a color histogram is better suited at describing the entire composition of the region, but requires considerably more computation and storage. Both strategies are useful in their ability to produce models that remain consistent, even if the region is translated, rotated, or deformed. There are times, however, when it might be desirable to create models that record the spatial distribution of colors in a compact way, as this information could be useful in identifying the orientation of a region. This representation usually requires a sparse sampling of pixels across the region, or the extraction of averages taken from small groups of pixels.

Regardless of the model-type being constructed, a choice must generally be made between constructing the model during MSER evolution, or delaying the modeling process until after stable regions have been identified. If the modeling operation is done

during region formation (bottom-up approach), overhead is required to store and combine the models during the many merging events that take place. This is especially prohibitive when models are of high dimensionality. If the modeling operation is delayed (top-down approach), then a second pass must usually be made through each region's pixels. This again might be computationally intensive, especially in places with high region-nesting, as modeling the regions will require every pixel to be processed multiple times (once for every level in the MSER hierarchy).

Since color models are an essential part of our architecture, we have attempted to optimize efficiency by combining both bottom-up and top-down approaches. The models themselves are constructed after all regions have been identified and after less stable regions have been pruned. However, as the basic building blocks for these models, we use color averages that represent small clusters of similarly-colored pixels (clusters contain 12-18 pixels). Clusters are formed using a single-pass preprocessing step that resembles the greedy algorithms described in previous sections.

Sub-Region Clustering Algorithm

In *Pixel-Blocking* (Section III.i-III.i.2.1), a greedy algorithm was used to combine neighboring pixels into blocks containing between 1 and 9 pixels. Adjacent clusters were then linked by a set of connections with values corresponding to the color differences between blocks. These connections were sorted and processed to form stable regions. Our sub-region clustering algorithm uses the same set of connections, processed in the same order, but instead of allowing sub-regions to grow indefinitely, they are limited to a size

that provides the desired cluster sizes. For our algorithm, we set the maximum sub-region size to 18 pixels. This process fragments each region into a set of sub-regions, which can be used to provide detail about the distribution of color across each region without requiring individual pixels to be stored and processed. This algorithm only needs to be executed once on the leaf-level of the MSER pyramid, and are subsequently propagated up the tree.

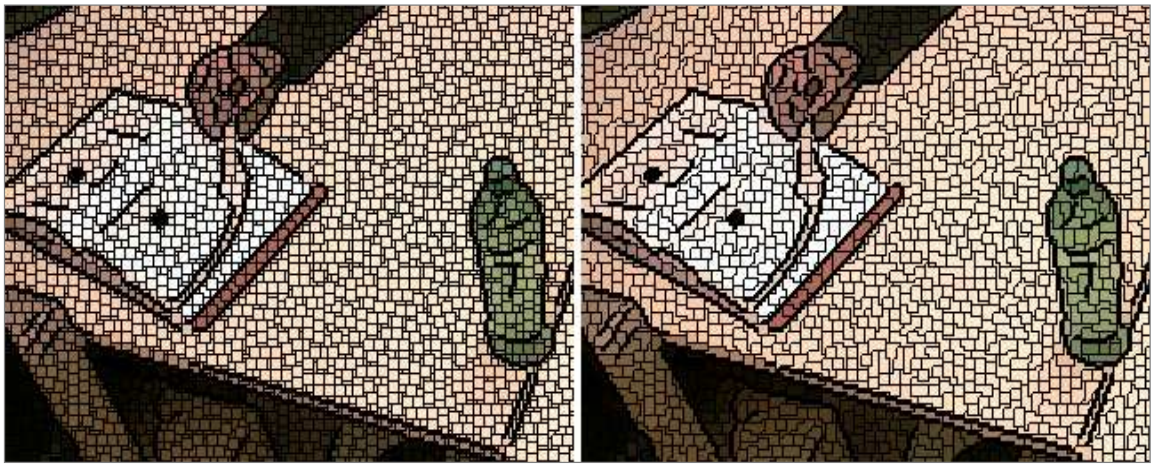


Figure III-19: Example of secondary clustering.

Figure III-19 shows the result of our sub-region clustering algorithm. The image on the left shows our original blocking algorithm. The image on the right shows the effects of the secondary clustering algorithm. These clusters are larger and less varied in size, while still preserving the boundaries between regions.

III.i.3 Feature Representation

Traditionally, computer vision algorithms are designed to operate on a specific feature set, and a specific toy problem is used to validate the approach. If an edge-based algorithm is used, then the input dataset will likely contain objects that display prominent edges. Color-based algorithms are often applied to video-feeds containing colorful

foreground objects. Contour-based algorithms often require stationary cameras and background modeling, or backgrounds devoid of any texture. Feature-detection algorithms usually involve objects with prominent surface patterns. Although many of the accommodations made for these algorithms may be reasonable in certain contexts, they all detract from the real-world applicability of the algorithm.

The primary focus of this dissertation is to present a computer vision algorithm that synergistically combines multiple feature detectors into a single real-time architecture. The idea is to provide an all-purpose algorithm that could seamlessly handle real-world environments. The previous section provides the motivation and details behind our region detection algorithm. The current section describes some features that can be efficiently extracted from these regions. For the most part, these features are unique to our algorithm and cannot be easily extracted from regions detected by traditional algorithms. This is not an exhaustive list, but does include features used in our demonstrations.

III.i.3.1 Ellipse Pyramid

One of the simplest ways to represent a region is using the centroid, orientation, and dimensions of the best-fit ellipse (computed using the mean and variance of the contained pixels). The color is generally represented using the average RGB intensities.

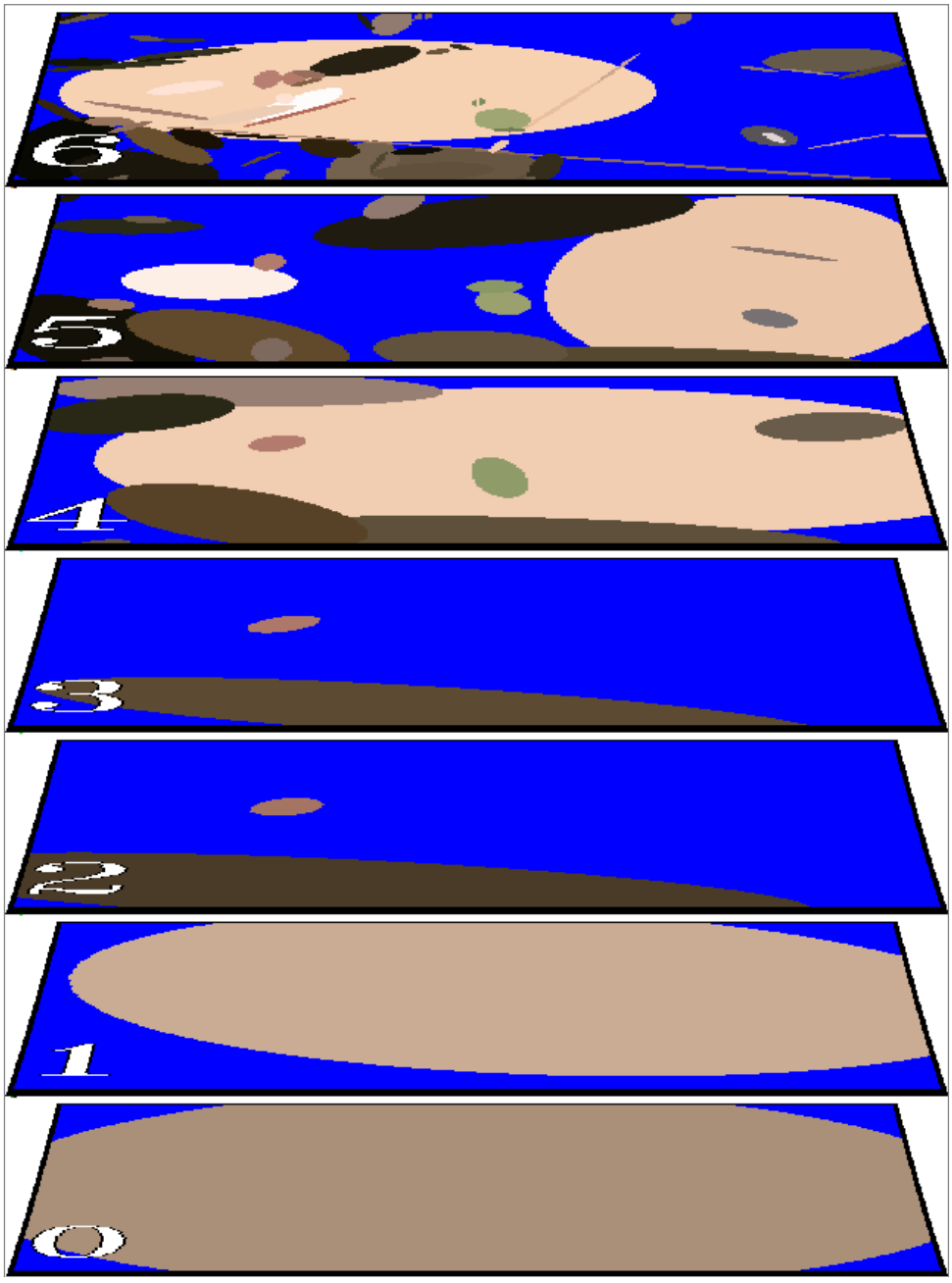


Figure III-20: Hierarchical organization of elliptical features.

Figure III-20 shows the set of ellipses corresponding to the detections in our "homework" scenario. The images are arranged to show the hierarchical arrangement of the regions. The top image shows the ellipses corresponding to the stable leaf regions. Images further down the stack show increasingly larger regions resulting from incremental merging. Stable regions are represented by the second-degree moments of their contained pixels.

III.i.3.2 Mixture-of-Gaussian

One of the simplest strategies for modeling a region's color is to use a single value that represents the average color of all contained pixels. Because of its simplicity, this representation offers little information about the distribution or variability in colors. Additionally, averages tend to be affected by outlier values, which can be caused by illumination, shadows, reflections, or the presence of background pixels that may have found their way into the foreground model.

A slightly more descriptive color model involves the combination of one or more Gaussian distributions. For our demonstrations, we model regions using a two-Gaussian distribution. Since the MSER algorithm is designed to identify regions that are relatively homogeneous, we believe that two Gaussians are sufficient to model the coloration while being more resistant to outliers.

A mixture of Gaussian model is usually constructed by iteratively adding region pixels to the Gaussian in the model that most represents that pixel's color. Our only modification to this algorithm is that we instead us the average values extracted from the *Sub-Region Clustering* process (Section III.i-III.i.2.5) described previously.

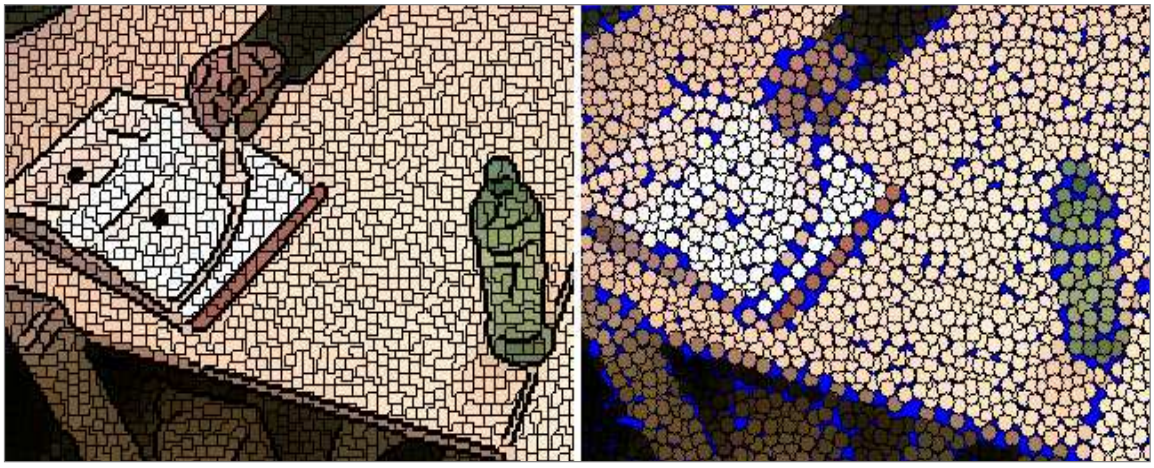


Figure III-21: Example of pixel clusters being efficiently represented as circles.

Figure III-21 illustrates how the information pertaining to locations and colors of individual pixels (left) is reduced to locations and colors of pixel-clusters (right). This simplification significantly reduces the amount of information to be processed and stored. Original pixel clusters (left). Simplified representation (right) preserves important information about distribution of colors in the image.

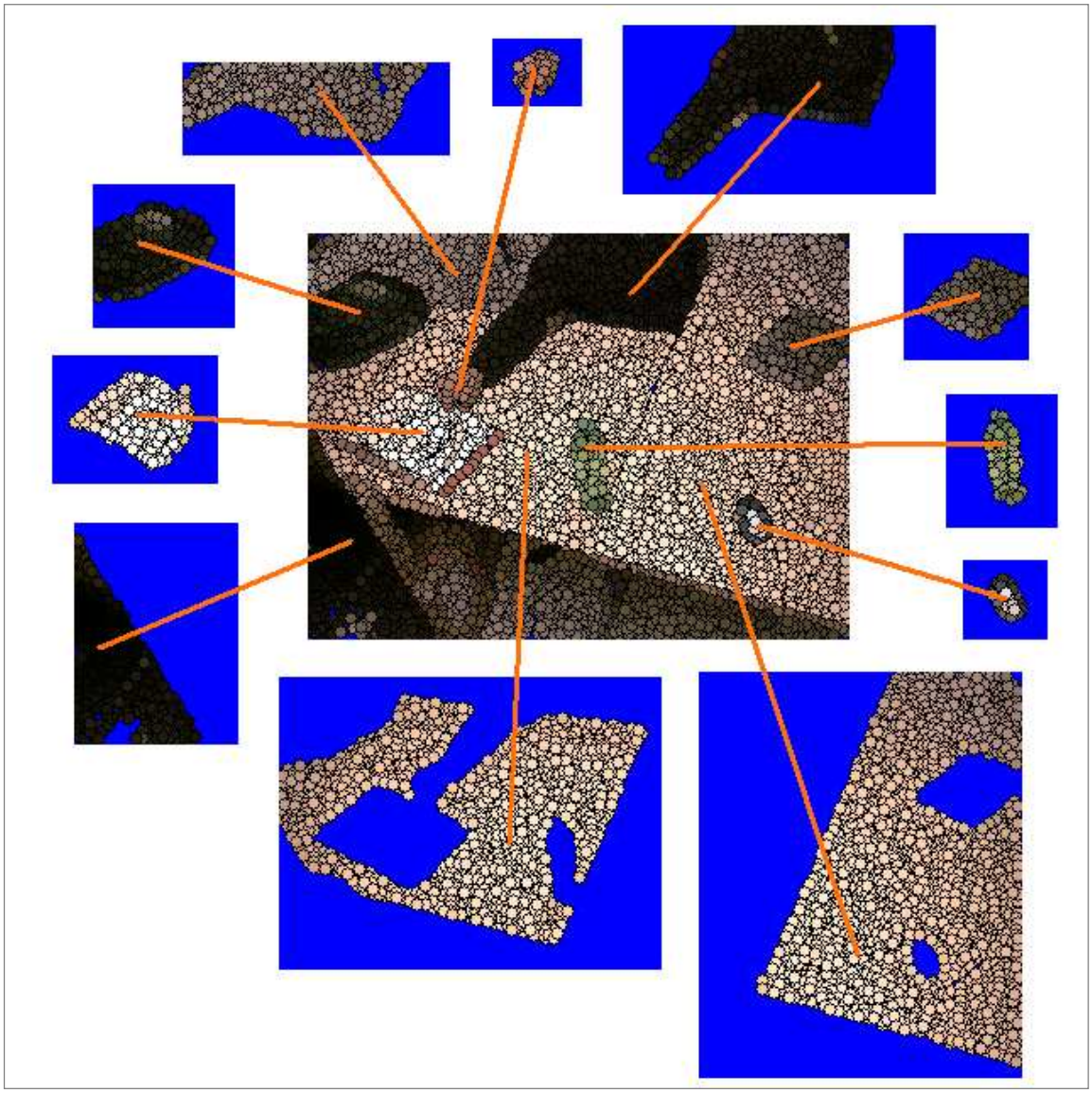


Figure III-22: Examples of regions detected from the '*Homework*' scenario.

Figure III-22 shows how different regions appear after our algorithm has extracted them from the "homework" image, and after grouping individual pixels into clusters. The central picture shows the clusters from all regions (smaller regions printed over large regions). The surrounding pictures show the clusters contained by eleven individual regions.

III.i.3.3 Perimeter Tracing

Background modeling is one of the most popular foreground segmentation tools used in computer vision. As output, the algorithm generally yields a binary image, with every pixel in the foreground being assigned one value, and background pixels being assigned a second. Although the resulting foreground silhouette lacks the original texture and color information, researchers have found that this representation provides sufficient information for object recognition (Stein & Hebert, 2005), pose recognition (Cheung, Baker, & Kanade, 2003), and even gait (Makihara, Sagawa, Mukaigawa, & Echigo, 2006) and behavior recognition (Eng, Toh, Kam, Wang, & Yau, 2003). Because of the popularity and utility of contour-based modeling algorithms, and because of the ease at which contours can be extracted from the regions produced by our algorithm, it is a reasonable extension for us to provide this capability.

Perimeter Tracing Algorithm

Traditional contour extraction algorithms are usually applied to the binary image that results from foreground-background segmentation (Pavlidis, 1982). The algorithm begins by stepping across every pixel in the image until a new foreground region is identified (one that has not been outlined in a previous step). When a foreground pixel is located, that pixel is labeled as the first point in the contour. From there, the algorithm iterates around the remainder of the contour, storing each new pixel in the order that it is discovered, until all border pixels have been identified and stored. The algorithm then resumes its search for additional foreground regions.

Our contour tracing algorithm is similar to the traditional approach, but instead of processing a binary image, we process images that are densely populated with multiple regions. To make this dense representation possible, each pixel contains an integer value, which is the unique index of the region that occupies that location. To accommodate region nesting, a separate image is required for every level in the MSER pyramid. Region images ('RI') are constructed and contours are traced one pyramid level at a time using the following algorithm.

Algorithm for identifying perimeter pixels:

For all levels $L_{l=0} \rightarrow L_{l=total}$ in the MSER pyramid
    Reset all pixels in the image 'RI to zero
    For all regions $R_{r=0} \rightarrow R_{r=total}$ in level L_l
        Initialize the perimeter seed (r_x,r_y) to the upper-left coordinate of the image
        For all pixels $P_{p=0} \rightarrow P_{p=total}$ in region R_r
            Add the unique index of region R_r to image 'RI' at the location of pixel P_p
            If coordinates of pixel P_p is lower-right of seed (s_x,s_y) then set seed to coordinates of P_p

    For all regions $R_{r=0} \rightarrow R_{r=total}$ in level L_l
        Pixels in 'RI' matching the index of R_r are 'foreground'. Other values are 'background'.
        Starting from seed (r_x,r_y), iterate around the region of 'foreground' pixels
            Record each encountered perimeter pixel to region R_r

As each contour pixel is identified, a determination is made as to whether or not that pixel corresponds to a Canny-detected edge. If it does, then the corresponding line information is stored along with the perimeter information.

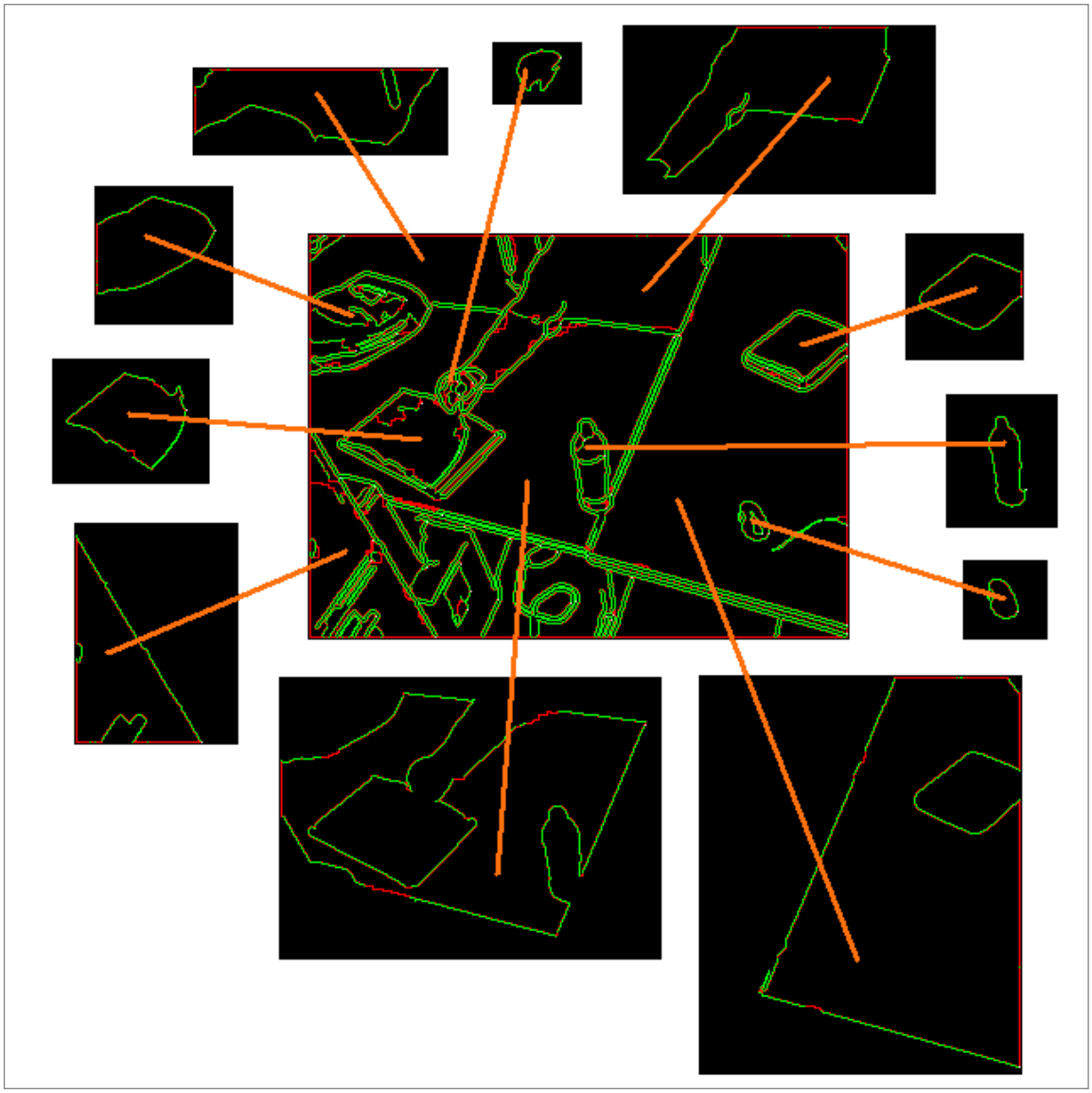


Figure III-23: Examples of extracted perimeters.

Figure III-23 shows the results of our perimeter tracing algorithm. The center picture shows the contours from all regions. The surrounding picture shows the contour of eleven individual regions. Green pixels represent those that are adjacent to edges detected by the Canny algorithm. Red pixels represent those that are not adjacent to detected edges.

III.i.3.4 Perimeter Lines & Corners

Once a contour has been extracted, it is sometimes useful to further compress the information by identifying lines and interest points. Interest points are generally identified as being corners or maxima in contour curvature. Most algorithm require additional algorithms to identify these features, but since lines have already been extracted in the initial phases of our region detection algorithm, identifying lines and corners in the contour can be done much more simply.

Perimeter Lines & Corners Algorithm

As was mentioned in the Perimeter Tracing section, after each perimeter pixel is identified, any associated line information is stored along with the perimeter. This greatly simplifies the linearization of the contour, since an algorithm only needs to identify the subset of existing lines that corresponds to each region. This can be done relatively easily by stepping through the contour and recording any associated lines. Along with the lines, new endpoints can be identified by finding the two most distant contour points along the line. The subsequent identification of corners can be done by stepping through the stored lines and storing the points of intersection between adjacent lines. If adjacent lines are separated by a gap, the corner is assigned to the point on the contour that is half-way between the endpoints of the corresponding lines. To reduce the total number of corners detected, we use the following corner reduction criteria:

1) **A corner's corresponding lines should not be parallel:** Since nearly-parallel lines provide corners that are unstable in terms of both repeatability and localization, it is not unreasonable to discard these. We require a minimum angle difference of 5 degrees.

2) **Only one corner is recorded for any given line-pair:** Since corners are identified using transitions between lines along the contour and not by using the actual point intersection between the lines, multiple detections are possible. Additional detections will occur close to the initial detection and provide no additional value. It is therefore reasonable to discard them. For simplicity, we discard all but the first transition point, though one could also select the point closest to the theoretical intersection of the lines.

3) **The endpoints of a corner's corresponding lines must be sufficiently close together:** When gaps are found between adjacent lines, it is usually means that the gradient at those locations were not prominent enough to be detected as edges. To reduce the number of unstable corner detections, it is not unreasonable to ignore corners detected at these locations. We select a maximum distance threshold of 16-pixels. This distance is measured by stepping around the perimeter from one line to the other.

4) **A corner must be sufficiently far away from adjacent corners:** Since real-world corners are often imperfect, there may be several small “corners” detected on either side of the actual corner. These offer no additional value and it is

reasonable to discard them. We select a minimum distance threshold of 4-pixels. Again, for simplicity, we give priority to the first corner detected, though a better solution would likely be to select the most prominent corner in each grouping.

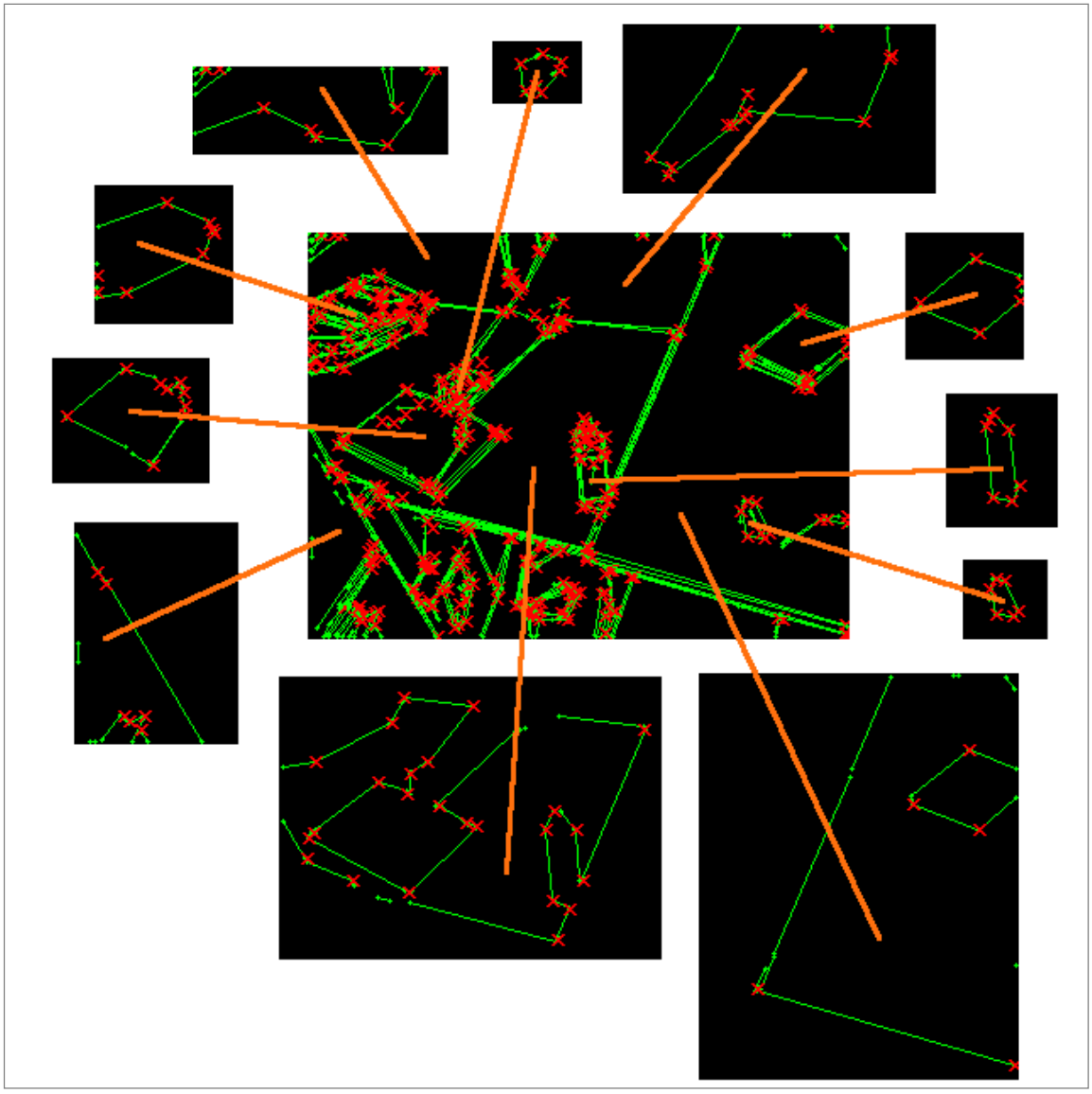


Figure III-24: Examples of line and corner detection.

Figure III-24 shows the results of our perimeter lines & corners algorithm. The center picture shows the lines and corners from all regions. The surrounding pictures show the lines and corners of eleven individual regions. Lines are shown in green, line end-points are shown as small green circles, and corners are shown using a red 'x'.

III.i.3.5 Color-Perimeter

Although contours can be useful in identifying and tracking objects, the overall descriptive value can be relatively low, especially when used on regions that undergo deformation or complex transformations. As mentioned in *Mixture-of-Gaussian* (Section III.i-III.i.3.2), color models may be better suited to track non-rigid objects, but that too is limited in their ability to distinguish regions from those that may be similarly colored. For illustration, consider the problems involved with tracking a hand as it moves through a scene. Not only will the texture and shape change significantly during reaching, grasping, or manipulation of objects, but even the measured color will fluctuate as the hand moves through bright areas and shadows. Given the variability of the hand itself, a reliable model would likely require additional information from the scene. Many approaches attempt to track the hand as being part of an entire articulated model of the human, but this introduces additional problems, especially when the entire person is not visible in the scene.

In simple scenes, fully articulated human models may be overly complex and inflexible, while simple region models are still insufficient in their complexity. Our approach attempts to balance both complexity and flexibility by using adjacency information to construct simple multi-region models. In the simplest application, an articulated model could be constructed from two physically connected regions, each with known approximate coloration. In more complex applications, models could be formed from multiple regions that display a history of being in proximity with one another. To

maintain generalizability, our system might ultimately allow the automatic detection of models by gradually acquiring adjacency information over multiple video frames.

Adjacency models are constructed using the small clusters of similarly colored pixels described in *Sub-Region Clustering* (Section III.i-III.i.2.5). By pairing each cluster on the inside of a region's perimeter to a cluster found on the outside of the perimeter, a set of color-pairs is formed. Although we currently use these pairs only for tracking, we hope to eventually use them as the basic unit for building color models. When identifying the pattern of regions using a single frame, it is not possible to identify which regions are adjacent because they are physically attached, and which are adjacent because one object is touching or occluding another. We hope that by tracking our contour features over time, it would be possible to probabilistically differentiate between these types of adjacencies, thus allowing the automatic construction of increasingly complex color models.

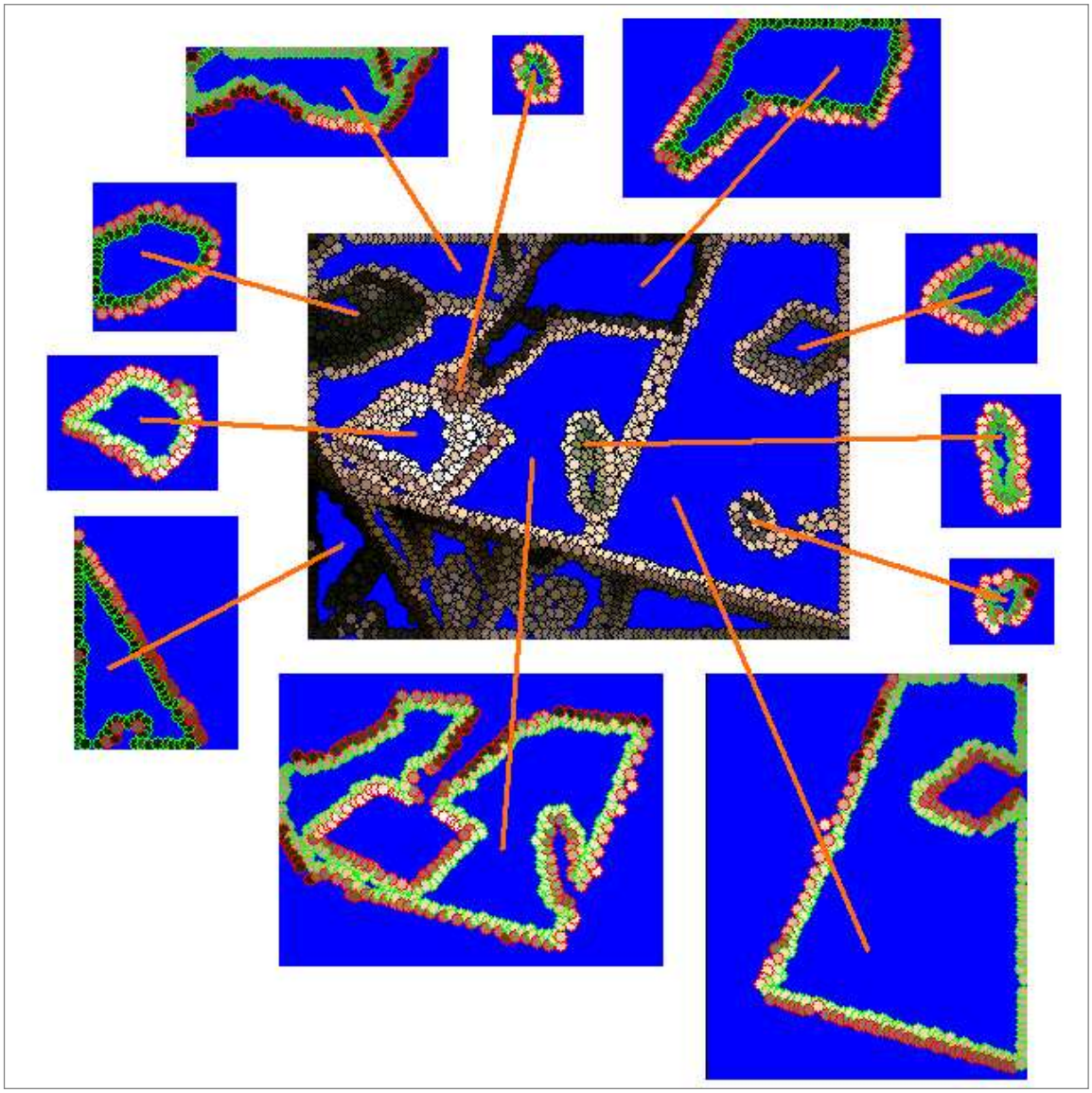


Figure III-25: Example of color-perimeter representation.

Figure III-25 shows results from our color-perimeter algorithm. The center picture shows the perimeters from all detected regions. The surrounding pictures show the color-perimeter formed around eleven individual regions. Circles are shaded using the cluster's average color. Green borders identify clusters that are on the internal edge of the perimeter. Red identifies clusters lying on the outer edge. Every inner cluster is paired with one outer cluster.

Color-Perimeter Algorithm

The naïve approach to identifying adjacent clusters would be to use cluster links established in the MSER portion of the algorithm. By iterating through these links while storing only those that form unique cluster pairs. Though simple, there are two problems with this approach. First, this requires links to be present for every possible adjacency. This is problematic because the current MSER algorithm does not store adjacency links when clusters are separated by Canny edges. Although it would be relatively easy to modify the MSER algorithm to facilitate this strategy, it would add computational cost, and still wouldn't eliminate the second shortcoming. The second problem with the naïve approach is that when long thin sub-regions form, they tend to stretch out along the boundaries between regions. Not only does this reduce the connectedness between regions, but the color of these sub-regions tend to be darkened, muddy, and dissimilar to the colors found in either individual region (since colors of the two region tend to blend toward their boundary). Forming cluster pairs using only direct adjacency information would primarily connect the elongated clusters of pixels on the interior of a region boundary to those on the exterior of the boundary, thus reducing their overall descriptive value.

III.i.3.5.1 Color-Pair Assignment

The idea behind *Color-Pair Assignment* is to form a network of regularly spaced links between adjacent regions, where each link connects a pixel cluster from one region to a cluster in the second region. To form the network, every cluster from every region

must be linked to all nearby clusters outside that region. Our algorithm links cells that have centers within a max-distance of 6 pixels from each other. The linkage is done using a two-dimensional bin-based data structure, which is populated by adding each cluster to the bin that lies closest to that cluster's centroid coordinates. The size of each bin is a function of the max-distance between pairs, which in this case produces bins that are 6x6 pixels. The remainder of the algorithm is as follows:

1) Assign clusters to bins using the coordinates of the cluster-centers (the average of the contained pixels).

2) Copy the contents of each bin to the 8 surrounding bins. This can be done efficiently by first copying the contents of all bins to the 2 laterally adjacent bins, and then doing the same to the 2 vertically adjacent bins.

3) Since each bin now contains all clusters centered within that bin, as well as all clusters centered within max-distance of that bin, parings can be made by linking every cluster within a bin, to every other cluster within the same bin. Because this strategy could potentially link clusters that are separated by more than max-distance, any clusters with separation exceeding this threshold are ignored.

4) Store each linked-pair to each of the associated regions. This is done at the leaf-level first, and then pairs are added to increasingly higher levels of the MSER hierarchy.

III.i.3.5.2 Color-Pair Reduction

A consequence of using a search-based pair assignment algorithm over an adjacency-based algorithm is that it provides a comparatively dense network of connections. Although this might offer advantages in certain applications, it can also increase the computational expense of subsequent algorithms without offering substantial improvements in accuracy. The purpose of our *Color-Pair Reduction* algorithm is to reduce the set of cluster pairs stored for each region. This reduction is achieved by removing some of the pairs that have clusters extending deeper into each region. As an example, consider a set of interior clusters 'A' & 'B', which are connected to exterior clusters 'a' & 'b'. Before reduction, the set of interconnections might include: 'A-a', 'B-b', A-b', and 'B-a', while after, we would be left with only 'A-a', and 'B-b'. Average reduction is typically over 50%. In the 'Homework' scenario screenshot example, we identify 13278 perimeter pairs from all detected regions, and our pair reduction algorithm reduces this to 5625 pairs; offering a reduction of approximately 57%.

Our pair reduction algorithm uses the contour detected in *Perimeter Tracing* (Section III.i-III.i.3.3) to identify a reduced set of connections. The algorithm is as follows:

1) Store all pairs to a 2-dimensional matrix of bins. This allows a near constant-time bin-based search of the image. We use a bin-size of 3 pixels, which is half the max-distance threshold mentioned in the *Color-Pair Assignment* (Section III.i-III.i.3.5.1) algorithm. Pairs are stored according to the point that is half-way between the two cluster centers. Each pair is stored to the 4 bins that lie closest to

that point. Searching for the closest cluster involves accessing the bin that lies closest to the search point, and checking all clusters stored in that bin.

2) Increment around the perimeter of each region, and for each perimeter-point, find the closest cluster-pair to that point. Cluster-pair distance is measured as the squared distance from the perimeter-point to the first cluster in the pair, added to the squared distance from the perimeter point to the second cluster.

3) If the current perimeter-point is closer to the cluster-pair than every other perimeter-point, store that point to the cluster-pair.

4) If, a cluster-pair does not have a perimeter-point stored to it after all perimeter-points have been processed, it means that there was another cluster-pair found that lies closer to the perimeter than this one. These pairs can therefore be removed as being invalid.

5) A linear-time counting sort is used to order the valid cluster-pairs so that they can be accessed in the same order as the sequence of points around the perimeter.

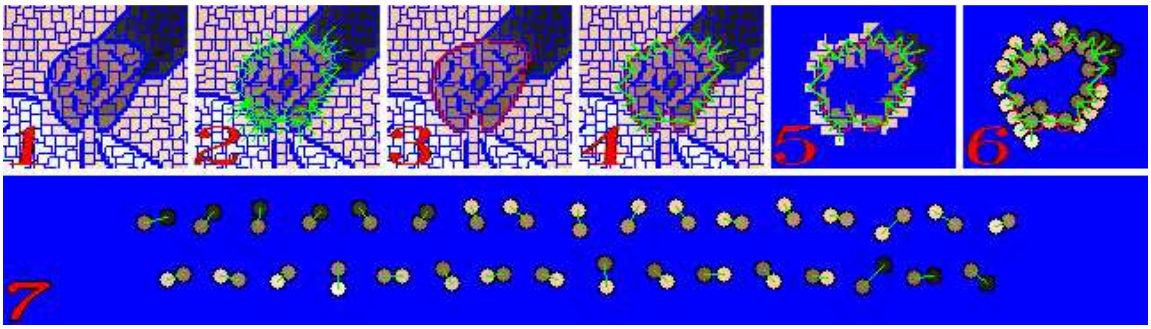


Figure III-26: Summary of algorithms used to produce perimeter cluster-pairs.

Figure III-26 summarizes the stages of the proposed *Color-Pair Assignment* (Section III.i-III.i.3.5.1) and *Color-Pair Reduction* (Section III.i-III.i.3.5.2) algorithms. **Image-1:** detected sub-regions. **Image-2:** results of our *Color-Pair Assignment* algorithm, with cluster-pair connections marked using green lines. **Image-3:** results of our *Perimeter Tracing* (Section III.i-III.i.3.3) algorithm, with the hand's perimeter shown in red. **Image-4:** cluster pairs that remain after our *Color-Pair Reduction* algorithm. **Image-5:** clusters that are associated with inter-region links. **Image-6:** the same set of reduced pairs, with clusters represented as circles. **Image-7:** all individual cluster pairs, printed in same order that they appear when iterating counter-clockwise around the perimeter, starting at the top-right pixel in the region.

III.ii Region Tracking

There are two basic approaches to tracking features across frames in a video sequence. The first is to use algorithms that are able to identify highly unique, high-dimensional features within a region. Ideally, these are features would be highly specific to the target object, yet general enough to allow matching despite small changes in illumination, orientation, and scale. This approach is especially effective when objects of interest are rigid, flat, and display prominent and unique texture patterns, making it ideally suited for tracking items containing logos and commercial artwork (Mikolajczyk & Schmid, 2005). Another useful characteristic with using high-dimensional features is that tracking and object identification can be achieved simultaneously. A disadvantage is

the cost associated with processing complex features. In most cases, storing features to a searchable database can only be done as an off-line process, and even queries can be too computational to be used on a frame-by-frame basis.

The second strategy for feature matching is to apply search algorithms to a large number of simple, unreliable, low-dimensional features, with the expectation that their combined contribution will provide the necessary accuracy. This approach may be favorable when image resolution is bad, when target objects are nondescript, or when an object's appearance can change rapidly enough to necessitate tracking between every frame. Since surveillance-type videos often present these challenges, we focused on tracking low-dimensional features.

Our tracking algorithm requires regions to be detected in every video frame. Inter-frame associations are then identified using simple ubiquitous features. In order to provide a large number of samples for both visual and temporal modeling, our tracking algorithms are designed to track region positions, while also tracking observable changes in the contours. This requires small sections of the region's perimeter to be independently and efficiently tracked, even when those sections lack usable textures. This is accomplished by dividing the contour into simple elements. These elements, referred to as 'cluster-pairs' (or 'color-pairs'), were detailed in *Region & Feature Detection* (Subchapter III.i). Cluster-pairs are defined using color information found on the inner and outer areas near the contour, as well as using a directional component extracted from the local gradient. We find that the simplicity of these elements allows them to be efficiently constructed and compared, while their descriptive qualities provide reasonable

tracking results. This allows the perimeters of all regions in the image to be detected and tracked between every frame, in real time. The specific cluster-pair features used are as follows:

1) **The two associated colors:** The average color extracted from a cluster of pixels on the internal edge of the region's perimeter, and the color from a cluster of pixels on the external edge of the perimeter

2) **x-y coordinates within the image:** The point where the line extending between the two clusters intersects the region's perimeter.

3) **Perimeter-Normal:** If a line-segment is present at the point where the cluster-pair intersects the perimeter, the line-segment's perpendicular (pointing away from the region's interior) is used as the perimeter-normal. Otherwise, if there is a detectable gradient at that location, the direction of maximum gradient (away from the region) is used. If the gradient is not measurable, the normal is defined as the angle of the line extending from the inner cluster to the outer cluster.

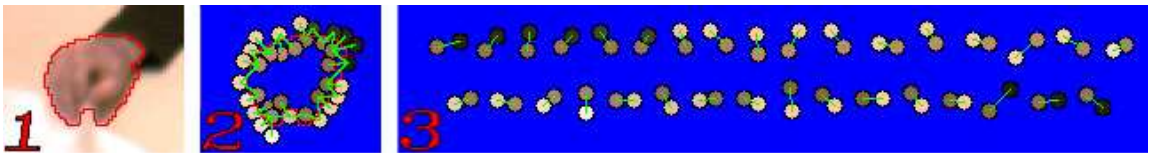


Figure III-27: Example of the set of cluster-pairs that make up a region.

Figure III-27 shows an example of the cluster-pairs that were detected around the region corresponding to a subject's hand. These features are the basic element used in our tracking algorithm. **Image-1:** the detected perimeter as a red line. **Image-2:** the set of corresponding perimeter cluster-pairs that were produced by our algorithm. The two associated colors are represented as two circles, with one inside, and one other outside the region's perimeter. **Image-3:** the same cluster-pairs extracted from their relative positions.

Region Tracking Algorithm

Our tracking algorithm is divided into three main sections. *Cluster Matching* (Section III.ii-III.ii.1) describes how our cluster-pairs are stored in a searchable data-structure and how features in one frame are matched to those in the next. Initial matching is a one-to-many operation with probability scores and motion estimates accompanying each candidate match. A vector flow-field is then computed by extracting local maximum from the spatially averaged estimates. In *Cluster Tracking* (Section III.ii-III.ii.2), flow-field information is used to reduce the one-to-many cluster matching to a one-to-one match. In *Region Tracking* (Section III.ii-III.ii.2), cluster associations are used to match regions across multiple frames.

III.ii.1 Cluster Matching

The primary motivation behind this algorithm is to provide a mechanism by which hundreds of detected regions can simultaneously be tracked across all video frames, in real time, using a standard processor. Tracking regions at this frequency affords several advantages that aren't available using slower algorithms. A high frame-rate increases the probability that tracking can be maintained, it increases the number of observations available for constructing appearance models, and it provides better resolution for temporal models. For these reasons, our algorithms have been designed primarily to offer operational efficiency.

III.ii.1.1 Searchable Bin Construction

At the base of our search algorithm, is a simple bin-based data structure that allows our pixel-cluster features to be stored and searched using minimal computation. Since features are tracked every frame, it is reasonable to assume that a given feature can be found within the proximity of its location in the next frame. Therefore, we apply the common strategy of limiting searches to a predefined distance from the original location. Increasing the size of the search window allows the algorithm to handle greater movement between frames, but adds to computational costs. Our algorithm uses a hierarchy of search windows in an attempt to balance the flexibility of accommodating more extensive searches, while maintaining efficiency in areas where motion is limited. A search is first executed using the finest resolution, and if a suitable match is not found, the search is repeated for increasingly larger areas until a predefined maximum is reached. The sizes of our search window are selected specifically to allow their efficient construction, which in the case of our algorithm, requires each window to provide three-times the search radius of the previous window. In the finest resolution, features are matched if target location is 1-pixel or less from the query location. The next resolution can identify features that are within 3-pixels or less, followed by 9-pixels, and finally 27-pixels. Our algorithm allows the addition of larger windows, as long as they follow the same multiple, but for our applications, a maximum range of 27-pixels has been deemed sufficient.

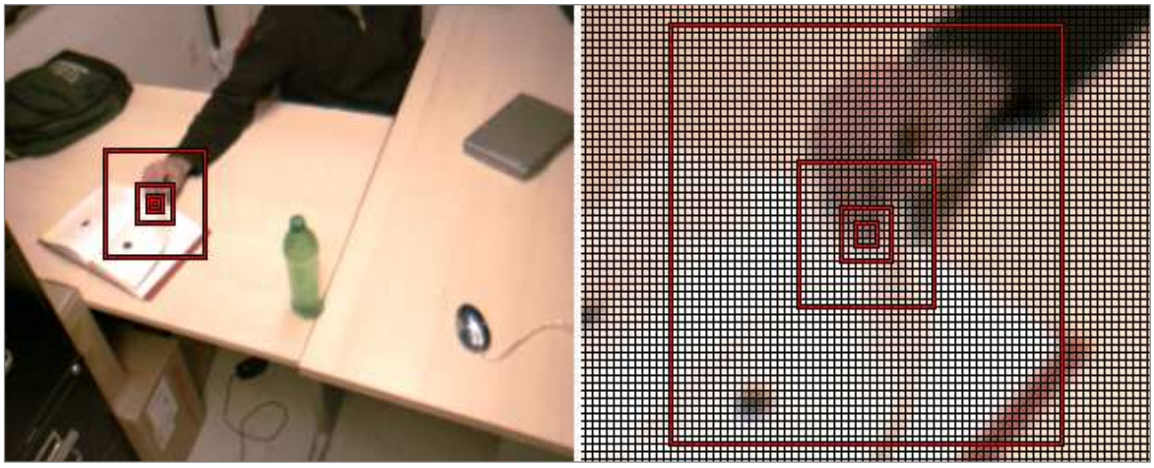


Figure III-28: Example of the search window hierarchy.

Figure III-28 provides an illustration of the search windows that would be used for tracking a pixel-cluster that originates at the end of a subject's hand. Left image: search windows (red) at normal resolution. Right image: zoomed-in representation of the windows (each pixel is outlined in black). In our implementation, Chebyshev distance is used, which generates square search windows). Euclidean distance can be applied just as easily to create circular search windows.

To construct the hierarchical search window data structure, cluster-pairs are added to the appropriate bin of the data structure's first level using the x-y coordinates of each cluster-pair's as indices. The pair's id is added to the bin's linked list and propagated up:

1) **For every bin, the contents of the 8 adjacent bins (2 vertical, 2 horizontal, and 4 diagonal) are copied to the center location:** This allows bins within 1-pixel of the original location to be accessed by referencing only the center bin. It should be noted that copying the contents of adjacent bins can be done more efficiently if the two-dimensional space is separated into two one-dimensional spaces. First copies of clusters from the horizontally adjacent bins are added to each target bin. Then, the process is repeated using vertically adjacent bins.

2) **To propagate information to higher levels, the contents of every third bin (i.e. 1, 4, 7, 10, ...) is copied to a corresponding bin in the subsequent level:** Since bins contains clusters from each adjacent bin, all clusters are represented exactly once in every third bin. This also means that every level in the structure has approximately 9-times fewer bins than the level below it.

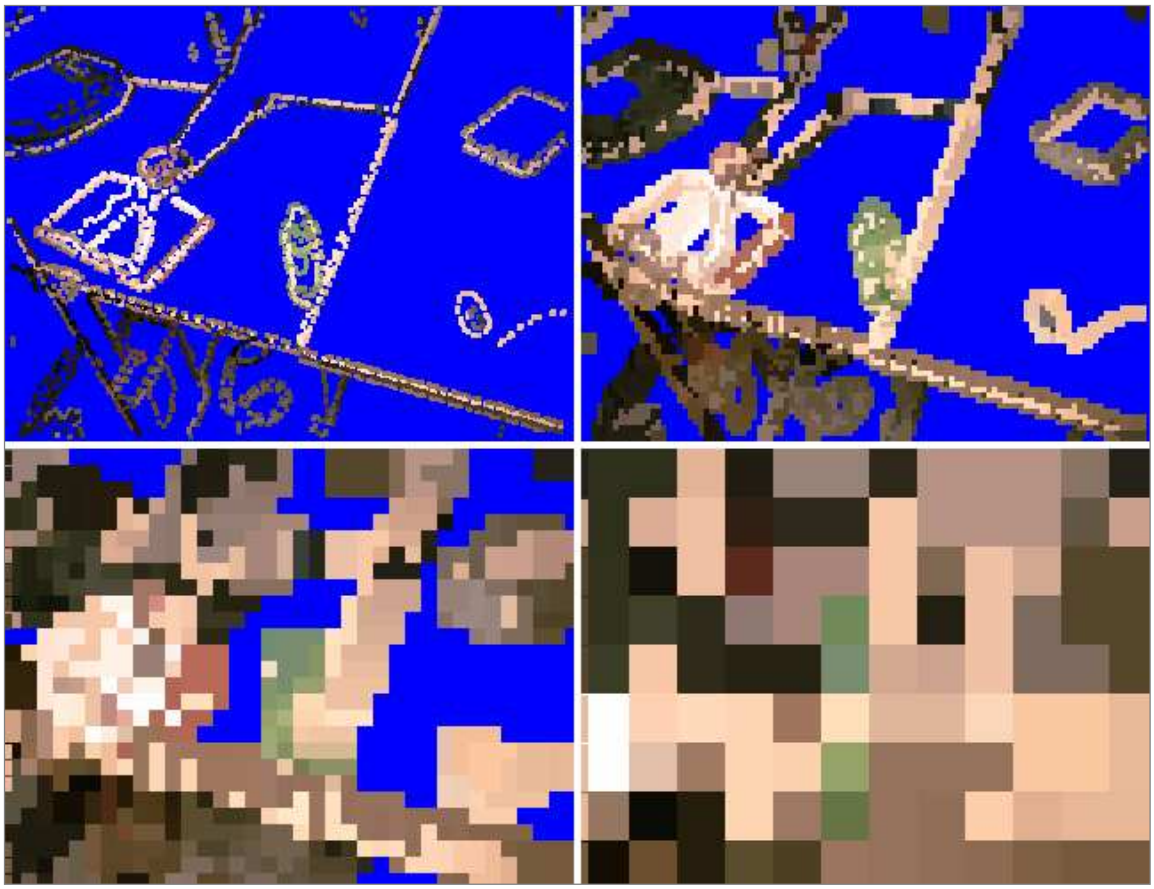


Figure III-29: Example of the multi-level search being applied an image.

Figure III-29 provides a rough visualization of the multi-level search data structure as applied to an image. Top-Left: 1-pixel radius search structure. Top-Right: 3-pixel radius. Bottom-Left: 9-pixel radius. Bottom-Right: 27-pixel radius search structure. Squares in the images correspond to the position of the search-matrix bin and are colored using one of the more prominent colors found in that bin. The size corresponds to the represented search range. Since bins are overlapping, they partially occlude adjacent bins, making them appear smaller than they actually are.

III.ii.1.2 Match Estimation

Every cluster-pair detected in frame ‘n-1’ is used as a query to identify zero or more matching cluster-pairs in frame ‘n’. When searching for a match, the coordinates of the query cluster-pair is used to index the nearest bin in the search data structure, starting at the finest resolution and the coordinates, perimeter-normal, and colors of the query are compared to every element in the bin. In our algorithm, the maximum allowed distance between the query and match is determined by the search window size (1 pixel, 3 pixels, 9 pixels, & 27 pixels); maximum allowable difference between normals is ‘22.5 degrees’; and the maximum allowable difference between colors is ‘2’ for clusters on the perimeter interior and ‘4’ for clusters on the perimeter exterior. The difference in color is computed using the length of the three-dimensional vector separating the query color from the match color (in RGB color space with values between 0 and 255). If one or more matches are found to be sufficiently close to the query, those matches are stored for the *Flow-Field Estimation* step in the algorithm below (Section III.ii-III.ii.1.3). If a match is not found, the search is repeated using a larger search window. Increasingly larger windows are used until a match is found, or until the largest window has been used.

It should be mentioned, that depending on the actual pixel location of the query and target clusters, the above binning algorithm returns all matches that are within the radius specified by the search level, but can also return matches as far away as 2-times the specified distance. This is a consequence of the smaller number of bins and coarser coverage at higher levels. To maintain the search specifications, matches outside the range guaranteed by the bin are omitted from the returned results.

Although the binning algorithm dramatically reduces the number of comparisons necessary for searches, it still requires an exhaustive comparison of all elements found in the local bin. At higher bin resolutions, this can require a large number of comparisons. To reduce this overhead, we made two modifications. The first was mentioned in the color-perimeter section. There, we described a technique that reduced redundant information in the perimeter by selecting a more representative subset of cluster-pairs. Our experiments show that this reduces the number of elements contained in each bin to less than half the original number.

The second optimization is a modification to the binning process itself. Our region detection algorithm is designed specifically to produce detections that densely cover the image. This means that every point along a contour is also contained by a contour of an adjacent region, and every cluster-pair in one region is matched to an inverse cluster-pair in another (with swapped cluster colors, and a normal that is rotated 180 degrees). Using a simple bin-based search, both elements would be placed in the same bin (since they share the same coordinates), and the extra match couldn't be discarded until after the search algorithm compared the element's normal against the query. To eliminate this common false match, we chose to additionally divide the bins using the perimeter normal. Specifically, we divide the entire range of angles into 8 bins and maintain a separate multi-level search data structure for each angle-bin. The only modification made to the search algorithm is that the search must be conducted in the bin corresponding to the normal of the query cluster-pair, as well as to the two adjacent angle-bins.

III.ii.1.3 Flow-Field Estimation

As described above, the proposed tracking architecture matches each element in frame ‘n-1’ to zero or more elements in frame ‘n’. Although individual matches are highly inaccurate, errors can be reduced by combining the contribution of multiple matches. This is achieved using a simple voting process that is applied to local accumulators distributed across the image. In our application, each accumulator covers an area that is 8x8 pixels, meaning that the 320x240 pixel image is divided into 40x30 patches.

Since features that produce a large number of matches are less likely to be reliable predictors of movement, we reduce the weight of each vote, by dividing it by the total number of candidates matched to the same query element. Additionally, we weight each vote by the degree of similarity between the query and candidate-match. Here, the best candidate receives no reduction in its weight, while the other candidates receive a weight reduction that is proportional to the comparative match dissimilarity. The match-score is computed using the equation below. The difference in color is computed using the length of the three-dimensional vector separating the query color from the match color (in RGB color space with values between 0 and 255). The max values are the predefined thresholds used for matching (if any of the values exceed the corresponding threshold, the match is discarded). In our application, the color threshold of the interior cluster is ‘2’, the threshold of the exterior cluster is ‘4’, and the angle threshold is ‘22.5 degrees’.

$$score = \frac{color\_diff_1}{color\_max_1} * \frac{color\_diff_2}{color\_max_2} * \frac{angle\_diff}{angle\_max}$$

$$color\_diff_1 = interior\ cluster: ABS\left(color_{query} - color_{match}\right)$$
$$color\_diff_2 = exterior\ cluster: ABS\left(color_{query} - color_{match}\right)$$
$$angle\_diff = normal: ABS(\text{angle}_{\text{query}} - \text{angle}_{\text{match}})$$
$$\text{color\_max}_1 = \text{interior color threshold}$$
$$\text{color\_max}_2 = \text{exterior color threshold}$$
$$\text{angle\_max} = \text{angle threshold}$$

Once the score for each candidate is computed, the value is added to the accumulator at the image patch closest to that feature. The specific accumulator bin is determined by the motion inferred by the element match. As an optimization, we chose to separate the movement accumulator into x and y components. Therefore, instead allowing features to vote for a candidate 2-dimensional vector in 2-dimensional space, we use the x-component of the vector to place a vote in the 1-dimensional space corresponding to the x-axis, and use the y-component to vote for the y-direction. This reduces the number of voting bins that must be accessed, while also reducing the potential for noise.

After votes are applied to all image patches, we attempt to further reduce noise, by applying a 3-dimensional Gaussian smoothing algorithm across the matrix of image patches. This smoothing is applied across the one-dimension of the x-component accumulator, across the one-dimension of the y-component accumulator, and both accumulators are smoothed across the two dimensions of the image. After smoothing, the most prominent component of the x and y accumulators are identified and stored as a movement vector. This is done for every patch in the image, which produces a vector

flow-field for the image. Each vector in the flow field is represented by the typical direction component, but also by a strength component. The vector strength is a function of the number of nearby cluster-pairs that had motion components similar to that that was found to be most prominent in the image patch.

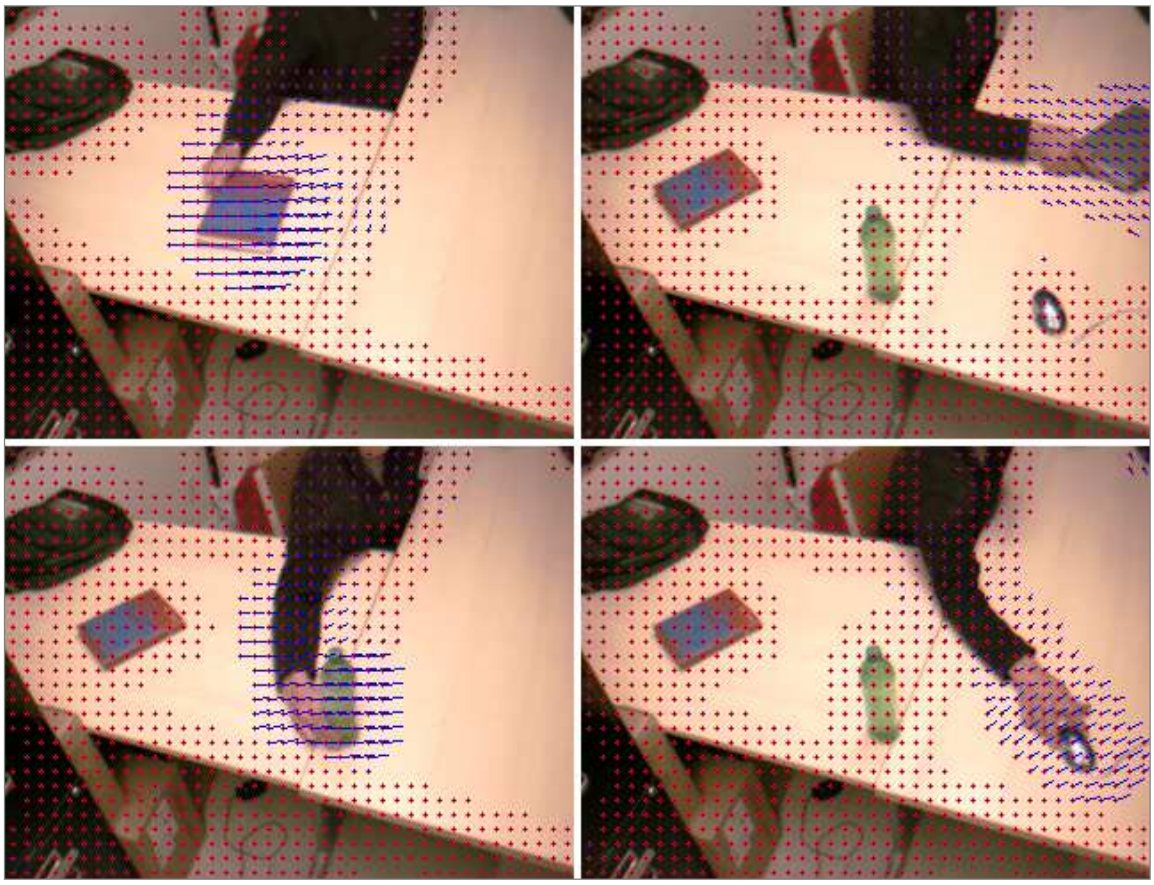


Figure III-30: Examples of detected flow fields.

Figure III-30 provides an illustration of the flow fields found during four different snapshots of the 'homework' sequence. Screen shots were taken from the '*Homework*' sequence. Subject is moving a book, a bottle, a laptop, and a mouse. A red '+' is placed in the center image patches that is above a minimum strength threshold (here there threshold is set to zero). Blue lines represent the most prominent motion vector found in those patches.

III.ii.2 Cluster Tracking

In the previous section, we described how our searchable data structure is used to identify cluster-pairs in frame 'n' that match queries from frame 'n-1'. For every query, a set of candidate cluster-pairs are returned. Although this provides sufficient information to estimate a flow-field, certain applications might require a more precise tracking of clusters across successive frames. When dealing with non-rigid moving objects, for example, additional information about the shape and motion of the object could be revealed by tracking different portions of the contour through a one-to-one relationship. The following sub-sections describes how the information obtained during flow-field generation is used to produce different relationships between cluster-pairs in frame 'n-1', and frame 'n'.

III.ii.2.1 One to One Matching

For our implementation, we use the fastest, and least restrictive technique, which is to link every cluster-pair in frame 'n' to at most one cluster pair in frame 'n-1' without enforcing an exclusive match-pair (meaning that more than one pair in frame 'n' may link to the same pair in frame 'n-1'). This allows a single motion vector to be easily computed for each cluster in the current frame, and simplifies the subsequent matching of regions across frames, but complicates the long-term tracking of single clusters over multiple frames.

The algorithm used for reducing the set of candidate pairs down to a single pair is relatively simple. In the previous section, a local flow-field was calculated using the

consensus of the candidate matches between frame 'n-1' and frame 'n'. The best candidate paring is then determined to be the one that produces an offset that is closest to the flow-field at that location. If there is a tie between two or more candidates, then the candidate that has the closest absolute distance from the cluster is selected.

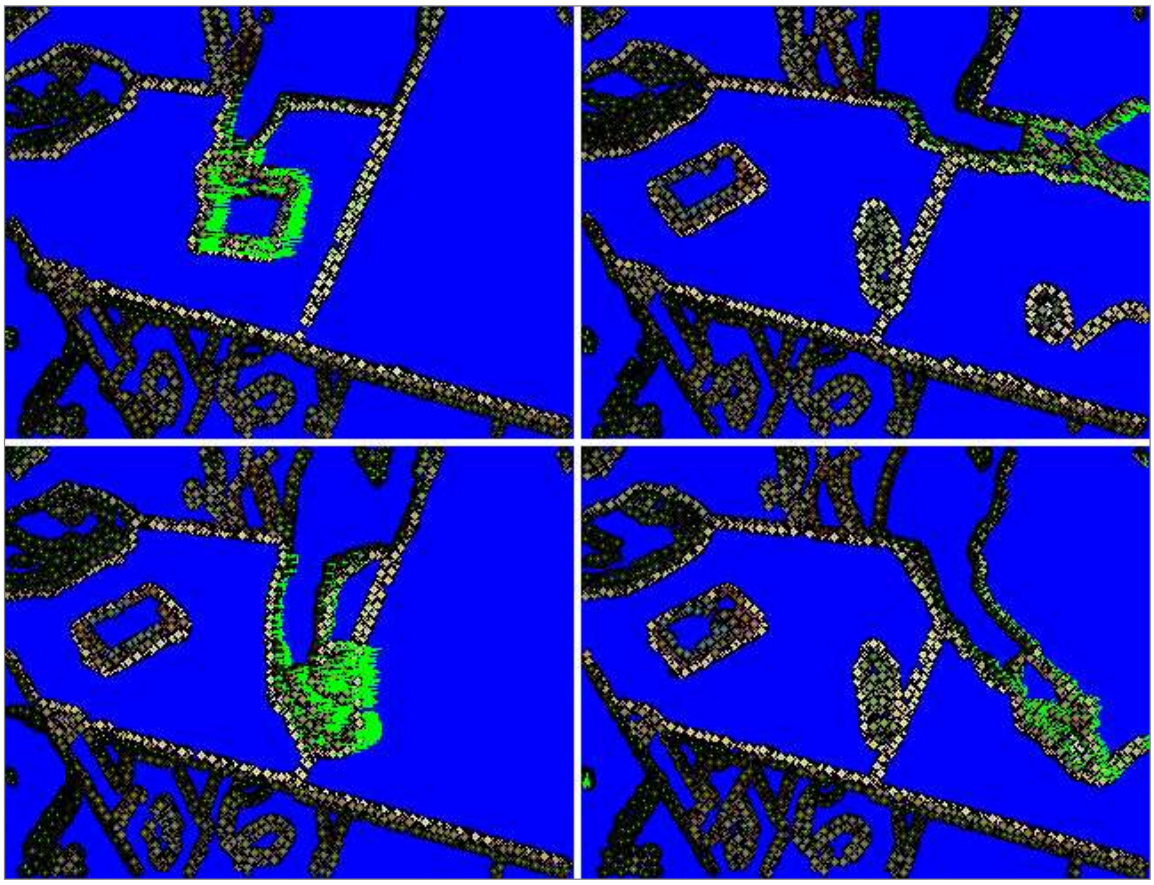


Figure III-31: Examples of matches identified between cluster pairs.

Figure III-31 illustrates identified matches between cluster pairs. Clusters from the current frame are displayed as circles. Green lines extend from the cluster in the current frame, to the location of the corresponding cluster from the previous frame. When no motion has occurred, the green line appears as a dot.

III.ii.2.2 Exclusive One to One Matching

Although our algorithm is one of the more simple and efficient ones to implement, it is not necessarily the best, and we have experimenting with a slightly more sophisticated strategy. In some cases, it might be desirable to track a single cluster over multiple frames. This type of tracking could be simplified if there was a true one-to-one paring of clusters between frames. To achieve one-to-one paring, we implemented a greedy algorithm similar to those described previously. A score is assigned to all candidate matches, which is a function of the difference between the pair's offset and the offset predicted by the flow-field. A linear-time counting sort is used to order matches from best to worst fit, which for efficiency, can be applied in a single pass to all matches in the entire image. Matches are processed on a first-come first-serve basis, and are only considered if clusters from both frames have not been assigned to a previous match. The result is an exclusive one-to-one match between cluster-pairs across frames. A disadvantage of this approach is that it produces a slightly less dense set of matches. The first-come first-serve component of this approach may cause cluster-pairs to be out-competed for their closest match, causing them to pair with more distant matches, thus potentially increasing the amount of noise from spurious matches. It is because of these reasons that we use the less restrictive matching strategy described previously.

III.ii.2.3 Constrained One to One Matching

The final matching strategy that will be discussed is one that uses higher-level information available after associations have been made between regions of adjacent

frames. The advantage of using region associations is that regions provide ordering constraints that are not otherwise available. For example, if a region contour in frame ‘n-1’ contains a sequence of features A, B, C, and D, then it should be expected that the ordering should remain the same along the same contour in frame ‘n’. This is a reasonable assumption since a rigid transformation will never change the relative feature ordering. Consequently, restricting feature matches to those that preserve ordering across frames can reduce the spurious matches mentioned previously in *Exclusive One to One Matching* (Section III.ii-III.ii.2.2), while also offering more descriptive contour matching. Instead of simply tracking a single region across frames, sections of the contour itself can be tracked, thus providing insight into transformations and deformations of involved.

The algorithm used to enforce ordering constraints is similar to that described previously. All cluster matches are scored according to how well they match the flow-field, are sorted using a linear-time counting sort, and are processed in order, starting with those that represent motion most consistent with the flow-field. Matches are only considered when neither of the contained clusters-pairs have been previously assigned, and when the ordering of the clusters in frame 'n' are consistent with the ordering of the associated clusters in frame 'n-1'. Ordering is determined by checking the relative placement of the clusters being considered against adjacent clusters that have previously been processed and assigned. For example, consider a contour containing points 1a, 2a, 3a, 4a, & 5a, which is being matched with a second contour containing points 1b, 2b, 3b, 4b, & 5b. If previous iterations have already mapped 2a to 3b, and 4a to 5b, then on the next step, matches containing valid ordering would include: 3a to 4b, or 3a to 5b, while

any other mappings would be considered out of order, and therefore invalid. It should be noted that the last point on the contour wraps around to the first point, so 1a would be considered to fall immediately after 5a.

The best-case computational complexity for constructing an ordered array of associations is O(n·log(n)), where 'n' is the number of features in the contour perimeter. This complexity results from the construction of the binary tree used in the sorting algorithm, and best case occurs when clusters are processed in an order that creates a perfectly balanced tree. Worst case is $O(n^2)$, which occurs in the degenerate case where features are processed in the same order that they fall around the region's perimeter.

The algorithm is applied to features identified in frame ‘n’, and proceeds as follows:

1) Features are indexed in a clockwise fashion when iterating around the contour. Features are stored in the binary tree using the index from frame ‘n-1’ (a), but only if the associated index in frame ‘n’ (b) maintains a consistent ordering.

2) Features are processed in an order that depends on the similarity between the measured offset, and the offset predicted by the flow-field.

3) The first two features processed from a region are accepted automatically. The index ‘a’ of these features is used to establish the range for the first level in the binary tree. In the above example, where the initial correspondences are 2a-to-3b, and 4a-to-5b, the left branch would contain the range 2a(3b) → 4a(5b), and the right branch would contain the range 4a(5b) → 2a(3b).

4) The index ‘a’ from subsequent features is used to identify the appropriate branch to follow through the tree, until a leaf is found. In the example, if the correspondence 3a-to-4b is processed, the left branch would be explored.

5) Once a leaf in the tree is found, the associated index ‘b’ is compared to the associated index ‘b’ of the leaf range. Again, in our example, the initial level is also the leaf, so 4b should fall after the small end of the range (e.g. 3b), and before the large end of the range (e.g. 5b). Since this is the case, the leaf will be split into two branches, and another level will be added to the tree. In the above example, the left branch will still contain the range 2a(3b) → 4a(5b), while the left leaf on the left branch will contain the range 2a(3b) → 3a(4b), and the right leaf on the left branch will contain the range 3a(4b) → 4a(5b).

6) If the associated index ‘b’ falls out of the associated range, the feature is discarded.

After all candidate element matches have been processed, the resulting subset of matches will maintain consistent orderings around the perimeter of both regions.

III.ii.3 Region Tracking

Region tracking involves the matching of detected regions from one frame in a video feed to the next. This allows a trajectory to be identified for each region of interest,

which can facilitate the identification, modeling, and tracking of objects that move in relation to the environment.

Our region-tracking algorithm is implemented using the information provided by our *One to One Matching* perimeter cluster-tracking algorithm (Section III.ii-III.ii.2.1), which determined the cluster from frame 'n-1' that best matches each cluster in frame 'n'. After establishing the cluster assignments, determining the best region match is a matter of finding the region in frame 'n-1' that has the most clusters in common with the region in frame ‘n’. Specifically, the match score is the square of the number of elements the two regions have in common, divided by the product of the number of elements in the first region and the number of elements in the second region.

$$\text{score} = \frac{|\text{elements}_1 \cup \text{elemetns}_2|^2}{|\text{elements}_1| * |\text{elemetns}_2|}$$

Regions are matched using a enforced one-to-one relationship, meaning that a region in frame ‘n-1’ can be matched to at most one region in frame ‘n’, and a region in frame ‘n’ can be matched to at most one region in frame ‘n-1’. This strategy simplifies the tracking of a single region through multiple frames.

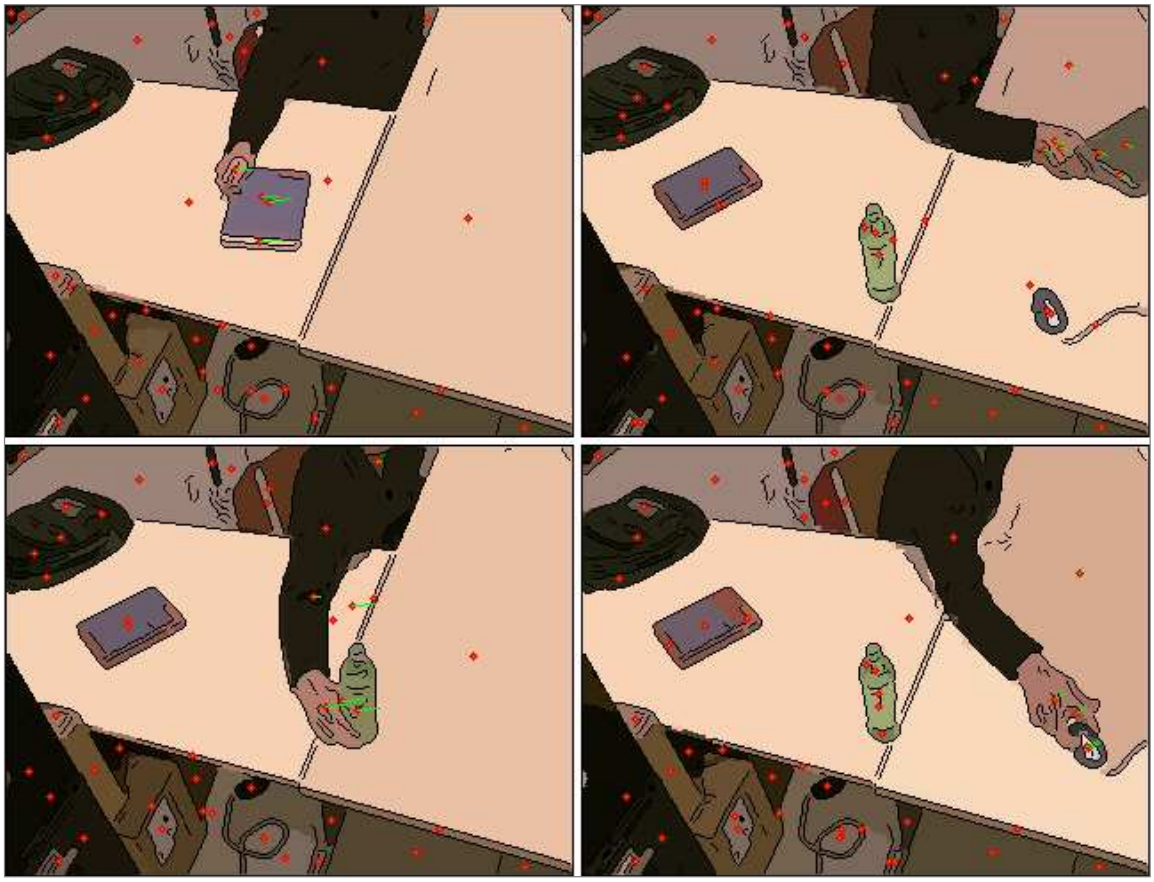


Figure III-32: Examples of estimated region motion.

Figure III-32 illustrates the motion that is estimated after matching regions in frame 'n-1' to those regions in frame 'n'. Red circles are displayed at the centers of their corresponding regions. Green lines extend from the region in the previous fame to the region in the current frame. If no motion has been observed between, the green line will appear as a dot.

III.iii Foreground Segmentation

One of the advantages of the proposed feature detection algorithm is that it is a flexible multi-purpose architecture that can be used in conjunction with existing algorithms. This is useful in cases where certain algorithms are more or less suited for specific environmental conditions. Furthermore, our architecture allows transitions to

occur between approaches if environmental changes are observed. For example, during periods when a camera remains motionless, a background subtraction algorithm can be applied to increase the accuracy of foreground segmentation. If motion is detected, the system could ignore background subtraction information and segment objects according to tracking information and foreground models that have been created up to that point. If none are available, the system could resort to the least accurate segmentation techniques, which involve the identification of motion differences between foreground and background.

III.iii.1 Background Subtraction

In surveillance systems, foreground segmentation is almost always achieved using one of the background subtraction algorithms. Although these algorithm are dependent on a considerable number of assumptions (e.g. fixed cameras, unchanging backgrounds, foreground objects that remain in motion, minimal occlusion, etc.), the disadvantages are offset by the fact that foreground regions can be detected accurately, even when no prior information is available about the objects of interest. The disadvantages are further reduced in applications where the assumptions do not place additional constraints on the system. For example, it is not uncommon to have existing surveillance systems that contain only fixed cameras monitoring indoor environments with controlled lighting and relatively stable backgrounds. In these cases background subtraction would likely be the optimal choice. Given the unequalled accuracy of background subtraction in a limited set of applications, we chose to include this algorithm as one potential component to our

architecture. Background subtraction could be applied throughout a video sequence or may be switched on and off, depending on whether or not the majority of the scene appears static.

Traditionally, background subtraction is applied as the first phase in foreground segmentation to identify clusters of pixels that differ from reference frames. Even when the necessary accommodations have been made to allow the background modeling portion of the algorithm to function reliably, there are still numerous problems associated with the detection of foreground objects:

1) **Shadows:** An object's shadow may be detected as part of the object. As an object moves through the environment, this shadow can change rapidly and unpredictably, making it difficult to acquire consistent shape models of the object. Color models may similarly be corrupted as pixels from shadowed portions of the background may be added to the foreground models.

2) **Reflections:** Portions of an object may be reflected off nearby surfaces, producing problems similar to those created by shadows.

3) **Incomplete Detections:** When portions of the background contain colors similar to the foreground object, they may appear as holes in the foreground detection. These features may cause irregularities in a contour or produce corner features that aren't actually present in the object.

4) **Merged Detections:** When multiple foreground objects exist in a scene, it is often difficult to identify them as separate objects when they touch or occlude one another.

Although some of these issues can be resolved during the segmentation process (Stauffer & Grimson, 1999), it is much more common to apply post processing steps, often requiring complicated ad-hoc rules.

The proposed *Background Subtraction* architecture differs from traditional approaches in that we do not consider background subtraction information until after the image has already been segmented and until after region motion information has been extracted. We then determine the likelihood that any individual region is part of the foreground by looking at the proportion of pixels within the region that differ from the reference frames. After foreground regions are identified, they can be naively connected to produce detections that resemble the traditional approach or color, texture, shape, motion, or history information can be used to make more educated guesses about how the regions should be linked.

Even when our background subtraction algorithm is applied naively, it still offers certain advantages over the traditional approach:

1) **Shadows:** In indoor environments, shadows generally do not contain the sharp, well defined boundaries that can occur outdoors. Office and home environment often use multiple or diffuse lighting sources that produce shadows as illumination changes across surfaces. This property reduces the extent that our

algorithm will detect shadows as stable features, and in many of these cases shadows can be completely ignored at the feature level while requiring minimal additional processing. If, for example, only a small portion of a wall or floor is covered by a shadow, the region corresponding to that surface may contain enough non-shadowed pixels to prevent it from being detected as foreground. Even if the shadowing becomes extensive enough to cause the entire region to be detected as a foreground candidate, our algorithm can still recognize that the region's edges have not changed from their known background positions, suggesting that the detection is not reliable.

2) **Reflections:** Unless surfaces are highly polished, reflections tend to be diffuse features that are not reliably detected by our segmentation algorithm. This allows them to be ignored in the same way that shadows are.

3) **Incomplete Detections:** Our algorithm detects regions before determining if they are part of the foreground. Therefore, pixels within those regions will not appear as holes if background subtraction cannot differentiate them from the background pixels.

4) **Merged Detections:** Since merging foreground regions occurs at a higher level in our architecture, there is much more information available to allow educated decisions about physical relationships. Even if higher level model information is not available, the boundaries around detected regions often correspond to the boundaries between objects.

Background Subtraction Algorithm

Most background subtraction algorithms produce as output a binary image of the same size as the video feed. Foreground pixels are displayed using a positive value and background pixels are represented using a null value. More sophisticated systems may apply clustering algorithms to eliminate small isolated detections while fusing together areas with a denser array of detections (Toyama, Krumm, Brumitt, & Meyers, 1999). However, since our algorithm achieves similar results using the foreground density within detected regions, we chose to use only the unprocessed output. Similarly, we found that our algorithm is effective at eliminating many of the inaccuracies produced by background subtraction algorithms, allowing even some of the simplest strategies to be effective.

For our demonstrations, background models were accumulated using the first twenty frames of a video sequence. During initialization, the grayscale intensity values measured at each pixel location were modeled using a Gaussian distribution. During subsequent frames, pixels were compared to the corresponding models and were marked as foreground if their values were more than two standard deviations from the known distribution. There was no adaptation made to the original models during the course of our trails. For efficiency, our clustering algorithm was applied to small blocks of pixels, and not to individual pixels. This required a separate operation to compile the pixel information into blocks, where blocks were labeled as foreground if at least half of the contained pixels were labeled foreground. Similarly, regions were labeled as foreground candidates if at least half of the contained blocks were labeled foreground.

In the motivation above, it was suggested that the output of our background subtraction algorithm could be considered in conjunction with the results of our tracking algorithm. Specifics will be detailed in the *Feature-Based Background Subtraction* section below.

III.iii.2 Feature-Based Background Subtraction

Traditional background subtraction algorithms have several limitations, many of which have been mentioned previously. Limitations pertaining to this section are as follows:

1) **Camera Motion Effects:** Background subtraction algorithms are usually implemented by comparing each pixel to its corresponding reference pixel model. If an image contains a texture or gradient that causes most pixels in a region to be dissimilar from nearby pixels, then any movement that shifts the region by that amount will cause false detections. Some researchers have designed algorithms that reduce these effects by allowing comparisons to take place within a small range of pixels (Elgammal, Harwood, & Davis, 2000); however the computational cost of applying the differencing algorithm to all nearby pixels does not make it a practical solution for all but the tiniest movements.

2) **Adaptation Efficiency:** If a shift in the image does occur, it is often detected when the proportion of pixels being detected as foreground exceeds some threshold. Pixel models for the entire image are then thrown away and an entirely

new set of models are trained using the new background. This is problematic because of the computational cost and time associated with retraining all models, and because the tracking of objects will be interrupted during retraining. To make matters worse, this process could potentially train foreground objects into the background models. When objects resume their motion, two foregrounds would then appear. One corresponding to the moving object, and one corresponding to the background that was concealed by the object during retraining.

3) **Acquired Information:** Adaptation is usually done in a very simplistic manner. When illumination or position changes do occur, the system will attempt to maintain the existing tracking information until a point is reached when everything must be reset. There is no way for a system to use the observed changes to recover the existing models and the information cannot be used to better understand the environment.

The proposed *Feature-Based Background Subtraction* architecture differs from the existing background subtraction algorithm in that we incorporate the contour features that were explained in *Region Tracking* (Subchapter III.ii). These features were useful because their simplicity allowed them to be stored and searched efficiently in real time, while providing dense enough coverage to reveal detailed and complex movements. Although this algorithm was designed to track regions from one frame to the next, in situations where the background is relatively static, tracking could also be useful in determining the extent that regions are part of the original background. In this way, our

tracking algorithm can be used as a background subtraction algorithm. During a background model acquisition period, perimeter edges from detected regions are stored to a reference search structure. Region edges from subsequent frames are compared to the reference edges. If a significant amount of the perimeter cannot be matched, then there is a good chance that the region is new to the video sequence. Advantages of using our edge features for background subtraction instead of the traditional pixel-based strategy are as follows:

1) **Camera Motion Effects:** The proposed feature-based background subtraction algorithm is designed to use efficient search-based comparison. This means that small image movements would have almost no effects on the accuracy of the system.

2) **Adaptation Efficiency:** The proposed system can facilitate more intelligent adaptation. If a shift in the image is detected, instead of building entirely new background models, it would be possible to simply update the position of the existing reference features. Features could similarly be adjusted to accommodate global changes in illumination.

3) **Acquired Information:** The proposed system can intelligently monitor background information. Not only can the system identify that a change has occurred, but can often identify and store the type, direction, and extent of change. This may be useful to better understand the environment. For example, if it becomes apparent that the background no longer matches the stored models, the

system may be able to identify if the camera moved (features match to a different location in the image), if the lighting changed (features match to the same location, but to a different RGB value), if something has occluded part of the frame (features match in a portion of the frame), or if something more significant occurred (no match can be made to any existing features).

Feature-Based Background Subtraction Algorithm

For our demonstrations, background models were accumulated using the first twenty frames of a video sequence. During initialization, regions were detected in each image using the algorithm described in *Foreground Segmentation* (Subchapter III.iii) and simple features were identified around the perimeters of each region. As was detailed in *Region Tracking* (Subchapter III.ii), features were described using: 1) The RGB color of the nearest sub-regions found on the interior and exterior edges of the detected region; 2) The x-y coordinate of the feature in the image; 3) The normal of the perimeter at the location of the feature. Features were stored to a searchable data structure and duplicate features were removed to increase search speed (features were considered duplicates if they contained the same color combinations and fell into the same bin at the lowest level of the search structure.

During runtime, regions and contour features were detected in every frame of the video sequence. Features were matched to both the previous frame (as was detailed in *Region Tracking* (Subchapter III.ii)), and to the features stored from the reference frames.

If a feature could not be matched to a feature with similar position, angle, and color from the reference frame, that feature was labeled as a foreground feature. If a match could be identified, the feature was labeled as a background feature. Each region was assigned a foreground score that corresponded to the ratio of foreground features contained by the region, divided by the total number of features. Regions are labeled as foreground candidates if at least 75% of their contained features were labeled as foreground candidates.

At the end of the iteration, the average color and displacement of background features are compared to the reference features. If a consistent global change has been observed, information can be updated in to a new searchable data structure.

In its simplest implementation, foreground and background feature information are used to identify those regions that appear after the initialization phase of a video sequence. It is possible, however, to use the available information to identify structural patterns in the scene. For example, if two foreground regions share the same color-pair feature for multiple frames, then a possible interpretation is that the regions are physically connected. Conversely, if a color-pair feature is frequently found between a foreground and a background region, then chances are good that that feature represents an occlusion boundary. More details about the interpreted structure will be presented in *Foreground Classification* (Subchapter III.iii).

III.iii.3 Motion Segmentation

As has been illustrated above, foreground segmentation is a non-trivial task that presents numerous problems, even when video is taken from a relatively stationary camera. When the camera is in motion, the task becomes significantly more challenging. If foreground regions can be identified before camera motion begins, tracking algorithms can be used to maintain the position of those objects in subsequent frames. Tracking can be improved if the object can be recognized by its characteristic color, texture, or shape information. These strategies will be described further in *Foreground Classification* (Subchapter III.iv). In the absence of additional information, one of the few remaining options for identifying foreground objects against a moving background is to apply motion segmentation techniques that cluster features that move together in a consistent way. This involves two major challenges: 1) Optical flow within a scene must be identified and described using a matrix of motion vectors. 2) Vectors corresponding to foreground objects must be differentiated from those corresponding to the background.

Identifying optical flow requires that small regions be matched between video frames. This can be done at regularly spaced intervals across the image to produce a dense mapping of pixels, or matching can be accomplished between a sparse set of features. The first strategy is more conducive to image segmentation since it assigns motion estimates to every pixel (or block of pixels), thus ensuring the foreground can be fully represented. However, producing dense flow fields is highly computationally intensive, making real-time operation nearly impossible. Plus, producing movement

vectors at every location could be detrimental in areas where texture is low and matching is unreliable.

The purpose of the proposed architecture is to compute an optical flow field using a sparse set of features, while still allowing dense motion-based segmentation. To do this, we use our feature tracking algorithm to identify motion in the higher texture areas surround detected region perimeters. This information is then used to estimate the motion of the low texture areas within each region. Regions that display similar motion characteristics are then clustered together.

Motion Segmentation Algorithm

Unfortunately, in realistic environments, using motion information to distinguish between foreground and background objects is not a well-defined problem. Occlusion and the aperture problem can produce ambiguities in the measurement of optical flow, while a camera's movement through a complex environment containing large depth discontinuities can produce highly variable background movements. Combine this with an articulated foreground object, and accurate motion segmentation can become nearly impossible. Despite these problems, the proposed algorithm can still be useful by increase the system's robustness to certain environments. Specifically, it is assumed that foreground features occupy less than half the screen, that the background does not contain large depth discontinuities, and that camera motion will be largely translational to increase the extent that background features will produce reasonably smooth optical flow.

In *Region Tracking* (Subchapter III.ii), we described how optical flow-field estimations are computed and how regions are tracked across multiple frames. The results of these two algorithms are combined in a way that allows foreground regions to be identified when they display motion that differs from the background. When their motion is not distinct, tracking is used to preserve the foreground assignments. The first step in motion segmentation is to estimate the most prominent background motion. This is done by compiling the translation values of all image features into a two-dimensional histogram of possible ‘x’ and ‘y’ image offsets. The offset vector showing the highest frequency is used to approximate the apparent motion of the background.

As was described in the tracking section, the motion of each segmented region is estimated using the measured offsets of all features in the region’s perimeter. In cases where different sections of the perimeter display different motion characteristics, the region is assigned as many as three of the most prominent values. For additional values to be stored, they must represent a local maximum in the offset histogram and at least 10% of the features in the region must display a similar offset. Motion segmentation is achieved by comparing each region’s possible motion offset to the approximation of the background motion. If none of the possible offsets match, the region is labeled as a foreground candidate. Before a candidate region’s foreground status can be verified, it must satisfy the following criteria: 1) the region must be tracked for at least five frames; 2) the region must be labeled as a foreground candidate for each of those five frames; 3) the average motion over those five frames must resemble the motion for each of the individual frames. Once a region is verified as being foreground, that label will be

preserved for as long as the region can be tracked. This allows the foreground classification to be preserved, even after the region stops moving.

III.iv Foreground Classification

In the automated surveillance architecture, foreground classification is the step that bridges low-level feature information with high-level comprehension about the events in a scene. Not only can an effective classifier provide understanding about ‘who’ is present, and ‘what’ kind of objects are being handled, but it can also be essential for the robust operation of foreground segmentation and tracking. If a region has been detected as foreground with a low degree of certainty, classification can help determine if it is a true foreground object or part of the known background. Similarly, if the tracker fails or if objects disappear from the scene, an effective classifier can resume the trajectory of that object once it returns.

Classification is divided into Model-Construction and Model-Identification phases. During Model-Construction, a set of features that uniquely represent objects of interest are identified and stored into a searchable database, along with a meaningful label. This may be a manual process that is guided by a user (Lowe, 1999), it could be an automated process (Ranzato, Huang, Boureau, & LeCun, 2007), or a mixture of the two (Li & Wang, 2003). When involving a large number of models, this is often done as a highly computational off-line process (Sivic & Zisserman, 2006), though simple models can be stored at runtime (Riemenschneider, Donoser, & Bischof, 2008). During Model-Identification, image features are identified and compared against those in the stored

models. If features are highly specific to an object, they may be individually used as identifiers. Otherwise collections of features may be required for positive identification.

As with most computer vision systems, foreground classification algorithms are selected for their suitability to handle the requirements of the task, the environmental conditions, the sensor types, the available processing power, and other system-specific considerations. Although the goal of our architecture has been to provide a general and robust system, our initial verification was done using a set of simple sequences that offered a few challenges but also allowed for certain simplifications. Modeling challenges included: possibility of a non-stationary camera; highly deformable foreground regions; simultaneous tracking of multiple objects; occlusions between foreground objects; low resolution; limited texture; and limited number of training iterations. Characteristics that allowed certain simplifications were: good color saturation; limited number of target objects; distinct coloration of foreground objects; and the availability of manual off-line training for initial runs.

Almost our entire automated surveillance architecture was built from the ground up by a single programmer. Because of the number of components involved in this system, and the amount of time required to construct robust low-level algorithms, we found it necessary to limit the complexity of the high-level algorithms to ensure that our demonstrations could be completed. Because of the modularity of our architecture, more advanced algorithm can be inserted to extend the system for operation in more complex environments. In *Region & Feature Detection* (Subchapter III.i), we listed a variety of color, shape, contour, and texture features that are all detected within our architecture.

Although each of these features could be used to construct foreground models, our initial demonstrations were simplified to use basic color models. To simplify things further, models were trained off-line using manual user input. We intend to increase the extent of automation in subsequent trials.

III.iv.1 User-Assisted Mixture-of-Gaussian Models

Our first approach to object modeling was developed around a software package created by Charles Bouman (Bouman, Shapiro, Cook, Atkins, & Cheng, 1997). This package was designed to model data using an unspecified number of Gaussian distributions. More specifically, we used the package to classify regions using their color distribution. Foreground models were created offline, as a user-guided process. Background models were acquired automatically at the beginning of each trial. At runtime, color information from detected regions was used to query the database of stored models. If the best match occurred with a background model, that query region would be labeled as background, and ignored. Otherwise, the region was labeled using the specific name that was associated with the matching foreground model. However, since there is no guarantee that a single segmented region would exactly correspond to the entire object of interest, it was necessary to develop a technique to efficiently test different combinations of adjacent regions in a way that maintained real time operation. Specifics will be provided in the following sub-sections.

III.iv.1.1 Model Construction

The Bouman modeling software was originally tested using raw pixel information, and although it proved to be an effective classifier, the processing time that was required to model and classify large regions was not conducive to real-time operation. To improve the speed of operation, we took several steps to reduce the amount of redundancies in the system. First, instead of building models using individual pixels, we clustered similarly colored pixels and built models using only the cluster averages. Second, instead of using Bouman's strategy of computing the Gaussian parameters from an array of values, we modified the system to allow parameters to be estimated from a histogram of values. This increased the efficiency that regions could be tested within multiple models.

The pixel clusters used in our modeling application were produced during the region detection portion of our algorithm. These clusters ranged from 1 to 18 pixels in size, depending on the local texture gradient. The average cluster size was around 9 pixels. Cluster values were computed using the average of the contained pixels and were compiled into a three-dimensional histogram, in RGB color space, with each color channel containing 64 divisions. It should be noted, that even though this three-dimensional histogram contained 262,144 bins (64 x 64 x 64), we were able to ignore any bins containing null values by using a list to record the indices of accessed bins. After the histogram was complete, Gaussian models could be constructed using the value of each bin, multiplied by the number of occurrences within that bin.

III.iv.1.2 Foreground Modeling

When our system starts, it automatically loads a file that provides the known models and their associated name. If a file cannot be located the user would is given an opportunity to create one. This file can be generated using a live feed or existing video. In the case of a video-feed, the user is given the opportunity to fast-forward until the object of interest is present. Otherwise, the user must make sure that the object to be modeled appears on the video monitor. The user is then asked to manually click on the segmented regions that are associated with the desired object. Selected regions are highlighted accordingly. Once regions are selected, the user enters the name of the object, followed by the enter key. The system converts the regions into a color histogram, and then into a Bauman model. It stores the model with the provided name and returns control to the user, who can repeat the process as many times as is necessary to model the foreground objects.

III.iv.1.3 Background Modeling

New background models are produced by the system every time the program is executed. This increases the system's ability to identify foreground objects in novel environments. When the system is started, the segmentation algorithm described in *Region & Feature Detection* (Subchapter III.i) is applied to detect stable background regions. Each region is converted into a separate Bouman model using the process described in the *Model Construction* (Section III.iv-III.iv.1.1) above. During storage, models are provided the name background, followed by a unique number. During

runtime, any model with the prefix background will be ignored by the system as being unimportant.

Because of the large number of background regions detected within an image, the described modeling algorithm produces an unnecessarily large number of background models. To reduce this number, we take each of the models and apply the classification operation to it. Any models that are misclassified are simply assigned the name of the model it mapped to, and the original model is removed from the system. This process is repeated until all models produce correct classifications. To further reduce the number of background models, a confusion matrix (based on the match score) is created between all models. If the amount of confusion between two models is above a predefined threshold, those models are merged into a single model. Using the typical 240x320 image, it takes less than a second of processing to produce the set of background models.

III.iv.1.4 Foreground Identification

Our foreground identification algorithm can be used in conjunction with background subtraction algorithms, but does not require them. When used with background subtraction, foreground identification is only applied to regions that are detected as foreground. This improves computation speed and reduces the number of false detections. When used without background subtraction, foreground identification is applied to all regions and the Bouman models are used to determine if a particular region is foreground or background. In our initial tests we combined the two strategies. We began each trial using simple background subtraction and took advantage of the added

accuracy for as long as segmentation could be achieved. If changes in lighting or camera position occurred, the background subtraction algorithm would produce increasing numbers of false detections, causing an increased reliance on model-based segmentation. This allowed the transition between approaches to occur naturally, without requiring any actual change to the program's operation. It should be mentioned, that in our initial trials, we made no effort to update the Bouman models of the background, so these too would be expected to degrade. We found, however, that the Bouman models were much more resistant to change (especially to changes in position) and persisted much longer than did background subtraction models. We tested this using both changes in lighting (trash can fire sequence), and using changes in camera position (interactive robot sequences).

Foreground identification was achieved using the following algorithm:

Algorithm for foreground identification:

Create a sorted list of potential foreground regions
For each region:
    Make a list of potential foreground regions that are adjacent to the region, and have a size that is smaller than the region (region adjacencies are identified during *Color-Pair Assignment* (Section III.i.3.5.1), as described in *Region & Feature Detection (*Subchapter III.i)) (if two regions have the same size, the region with the smaller index is treated as the smaller one).

Process regions in sorted order, from largest to smallest:
    Use the Bouman classifier to classify each region.
    If the region classifies as background:
        Label the region background and continue.
    If the region classifies as foreground:
        Combine region with an untested adjacent region and re-apply the Bouman classifier
        If the combined region does not produce an improved foreground score:
            Discard the new addition to the region, and continue
        If the combined region produces an improved foreground score:
            Keep the new addition to the region, merge the adjacency lists, and continue

Our foreground detection algorithm is useful at classifying foreground regions, even if they are touching or are occluded by other foreground regions. This is useful in scenarios where one foreground object is a person, who is holding or moving other foreground objects.

III.v Behavior Description

Once objects in a scene are identified and tracked, it is often useful for a system to assign high-level meanings to the low-level observations. Although a record of coordinates that represents a region's position through multiple frames might be useful information to a computer, a human user would likely prefer that the information be presented in a meaningful way. For example, it might be better for the system to say that person 'Bob' had a notebook in front of him, and reached multiple times toward his laptop. It might be better still for the computer to simply deduce that Bob was probably working on his computer. In settings where security is an issue and where rapid response times are critical, a sufficiently complex system might even try to anticipate the actions of people within the scene. If an unknown person reached toward Bob's laptop, it would be reasonable for the system to conclude that the person could be interested in stealing the computer or that he may have other nefarious intent. In either case, it would be appropriate for the system to alert security.

Describing observable behavior is usually a matter of matching interactions and patterns of motion to those of existing models. Unlike in the previous stages, behavior description is not as often used for resolving ambiguities from lower architectural levels,

and isn't as effective at filling in missing information. This increases the importance that the assignments and measurements made by other parts of the architecture are accurate. If a localization error has occurred in one or more video frames, it can change the perceived behavior of the object. Similarly, misclassification of the foreground can completely change the interpretation of the scene. In our application, the only tools we use to improve the reliability of our detection is to average the underlying data. Before we classify behavior, we apply a smoothing algorithm to reduce abnormalities in the object's trajectory and we combine the results of the object classifier over multiple frames. Only after the same analysis is made consistently, does our system assign a classification to the behavior.

Another limitation with behavior classifiers is that the systems are usually only able to recognize a limited set of behaviors that are trained (or programmed) into the system. Our system is no different in this respect. We provided recognition for a limited number of object-types and a limited number of observable interactions. Unique behavior-labels are assigned to pairs of objects, depending on the observed interaction between them. To simplify things further, we classify objects as being either active or passive, and only attempted to classify pairs containing an active object. Active objects are those that move under their own power. These include people (or in some scenarios, people's hands), animals, robots, etc. Passive objects include items like tools, food, books, furniture, etc.

The observable interactions are based on the relative changes in position between the objects in the object-pair. The simplest interaction-types are listed below. These can be used alone, or can be combined to produce more complex interaction-types:

1) **Convergence:** The distance between two objects is decreasing
2) **Divergence:** The distance between two objects is increasing
3) **Moving Together:** Two objects are moving with similar trajectories
4) **Stopping Together:** Two objects are stationary

The object-types, the interaction-types, and the behavior-labels are all scenario-specific and specific details will be provided in *Experimental Validation & Results* (Chapter IV). Examples can be found in **Table 1**.

Table 1: Example labeling of interaction sequences.

Behavior-labels are based on object-1-type, object-2-type, and a sequence of one or more interaction-types. Sequence ordering is represented using the '→' symbol. Heavy-black lines separate different test scenarios.

Object-1-Type	Object-2-Type	Interaction-Type	Behavior-Label
Person	Person	Convergence → StopTogether → Divergence	Meeting
Person	Person	Convergence → Divergence (with change in relative orientation)	Passing
Person	Person	MoveTogether	Following
Person	Bag	MoveTogether	Carrying
Person	Bag	MoveTogether → Divergence	Dropping
Person	Bag	Convergence → MoveTogether	Taking
Hand	Item	Convergence	Reaching

As mentioned in the introductory paragraph, it is sometimes useful for the system to make its own deduction about what is occurring in a video (or what might occur). Making this kind of determination often requires information about the identification, position, and interaction of objects, as well as the overall context of the scene. Context is important to resolve actions that might otherwise be ambiguous. For example, if a person reaches for a glass while sitting at a table containing plates of food, it would be reasonable to conclude that the person will be drinking from the glass. However, if a person is standing over the table while holding a tray of empty plates, it might be better to conclude that the person is cleaning up. A sufficiently robust system may be able to determine context on its own, though in our systems, we have contextual information supplied by the user. Examples of the context-based conclusions we use in our scenarios can be found in **Table 2**.

Table 2: Examples of context-based conclusions.

All interactions involve one person and one item. Deductions are based on the item, the context, and the behavior-label. The heavy-black line separates different test scenarios.

Item	Context	Behavior-Label	Deduction
Bag	Airport	Dropping	If bag is left unattended: Security Threat
Bag	Office	Taking	If person is unknown: Security Threat
Book	Office	Reaching	Reading
Glass	Office	Reaching	Drinking
Laptop	Office	Reaching	If person is unknown: Security Threat

When developing the set of interactions, we experimented with both hard-coded and trained techniques. Details will be provided in the following subsections.

III.v.1 Hard-Coded Behavior Descriptor

Many of our surveillance scenarios involve the tracking of a subject's hand across a table containing multiple objects. In these instances, the only interaction that interests us is when the person reaches toward an object. Since the reach interaction is relatively straightforward, we chose to manually specify the criteria that define the action. This allows us to eliminate any potential training errors, increase the transparency of operation, and eliminate quirks that we found to occur with our more sophisticated HMM-based strategy.

We define our reach action using only the relative position of the hand, compared to detected objects. If the hand is closer to one object than it is to all the others, and if the distance to that object is within a predefined threshold for a predefined number of frames, then the action is considered a reach. Otherwise, the system indicates that no action has taken place.

There are several ways to define distance in this type of system. The measurement can be made using image-coordinates in pixel-based units, or a system could be calibrated to allow the image to be mapped into physical world, providing measurements in meters and centimeters. Since our algorithm only requires relative distances, and since our objects appear at depths that are within the same order of magnitude, we found the simpler strategy to be sufficient.

There is also the question of whether distances should be measured between the objects' apparent center of mass, or whether distance should be measured from the outer edge of one object to the outer edge of the second. Unless all objects are small, round,

and similarly sized, measuring distances between centroids may not produce the desired results (larger objects will appear further away). The alternative, however, can also be problematic, since finding the shortest separation between the outer edges of two objects is not a task that can be done using a small number of operations. We chose to balance efficiency and accuracy by representing objects using the second-order moments of their pixel distribution (a best-fit ellipse). Ellipses can be constructed efficiently during a region's construction, and provides a reasonable approximation of a region's perimeter. Region separation can then be computed using the distance between the nearest edges of the two ellipses (measured along the line that extends from one region's centroid to the other). If one region occludes the other, the distance becomes negative and represents the extent of overlap.

Hard-Coded Behavior Descriptors Algorithm

During every video frame that an active object is present, a separation distance is computed between that region, and every other foreground region in the image. Separation is measured as the distance (in pixels) between the nearest edges of the ellipses representing the two regions. The region that presents the shortest distance is labeled as an interaction candidate if the separation is smaller than the average radius of the ellipse representing the active region. If the interaction lasts for five or more consecutive frames, then the system obtains the corresponding behavior-labels, makes the corresponding deductions, and initiates any programmed response. All labels and responses are manually specified during the model construction phase.

III.v.2 HMM-Based Behavior Descriptors

Like many of the high-level components of our system, our *Hard-Coded Behavior Descriptor* algorithm (Section III.v-III.v.1) was written as a placeholder to provide the minimum functionality necessary to operate a complete automated surveillance system. Although the algorithm worked well in our simple scenarios, we recognize that it lacks the flexibility and automation necessary for an intelligent system. In this subsection, we present an algorithm that can potentially allow a robot to learn interaction-types by observing people engaging in those interactions. Once learned, the robot could recognize subsequent displays of the interaction, and even engage humans in a way that mimics the learned interaction. The algorithm described in this subsection was not created by the author of this dissertation, but was included to demonstrate how the system can be extended to operate on more complex problems. A more detailed description of the HMM can be found in R. Kelley's paper (Kelley, King, Tavakkoli, Nicolescu, Nicolescu, & Bebis, 2008).

III.vi Automated Response

The primary purpose of an automated surveillance system is to reduce the amount of time it takes for authorities to respond to criminal activity. One approach has been to increase the amount of interaction that occurs between humans and the computer. Systems have been designed to alert authorities every time an unusual event is perceived, with the expectation that humans will respond by telling the system if the event was a genuine threat or a false call. In the case of a false call, the system could learn to

disregard a similar event in the future (Sillito & Fisher, 2008). This approach might be useful during development and may reduce response times in settings with a large number of dedicated security staff, but this does relatively little to assist authorities in the recognition, pursuit, or apprehension of perpetrators. Without requiring additional hardware, a system may be able to zoom in a camera to take higher resolution photographs or communicate with other cameras to map the path of the fleeing individual. However, if the intent is to develop a fully automated system, the architecture should also possess the capabilities to solve minor problems or even engage criminals on its own. The inclusion of robots to a system is therefore a logical extension.

Many of our scenarios involve interaction with one or more robots. This is primarily what motivated us to develop a system that could be used on moving platforms. Using the same system on both the static and robot-mounted cameras allows models to be transferrable from one to the other. In cases where a theft is observed, this makes it possible for models that had been created of the suspect to be transferred to a patrol robot, which could use the information to approach or pursue the person. Alternatively, a stationary robot could act as the surveillance camera until it becomes clear that a person is in need of assistance. The robot could then spring into action and assist the human without losing track of its existing models of the visual scene.

We used several robots to validate our approach; however the one that was used most frequently was the Pioneer 3DX mobile robot. This was equipped with a SICK LMS-200 laser rangefinder, front and rear sonar, and a pan-tilt-zoom camera. For robot control we use the Player/Stage/Gazebo robot device interface (Gerkey, Vaughan, &

Howard, 2003). The following sections provide details about our robot control, mapping, and localization algorithms. Most of our robot controls are hard-coded behaviors, though to keep with the format of previous subchapters, we also provide details on an approach that could increase automation by allowing robots to learn behaviors on their own. Using this approach, a programmer would only need to define desirable and undesirable outcomes and the system would use simulation to train physical behaviors.

III.vi.1 Obstacle Avoidance Behavior Fusion

The mobile robots used in our experiments were supplied with an obstacle avoidance behavior and a waypoint seeking behavior. A simple fuzzy system was used to fuse the output of these behaviors. The obstacle avoidance behavior used laser range-finder information to identify objects that were within the path of the robot. The output of the behavior is the turn angle and speed necessary to avoid the closest of those objects. The waypoint seeking behavior points the robot in the direction that would bring it closer to a target. The recommended speed produced by the planner is always the maximum.

The fuzzy system fuses the two behaviors in a way that takes the nearest obstacle into consideration. If an obstacle is dangerously close to the robot, the obstacle avoidance behavior is executed exclusively (Obstacle avoidance weight = 1, Waypoint seeking weight = 0). If the nearest obstacle is reasonably far from the robot, the waypoint seeking behavior is executed with full weight. If the object falls within the fuzzy region between near and far, the output angle and speed are a function of the fuzzy weights corresponding to the distance activation function as shown in **Figure III-33**, followed by

the related equations. This fuzzy system was used in all trials that involved waypoint seeking (including both static and dynamic waypoints).

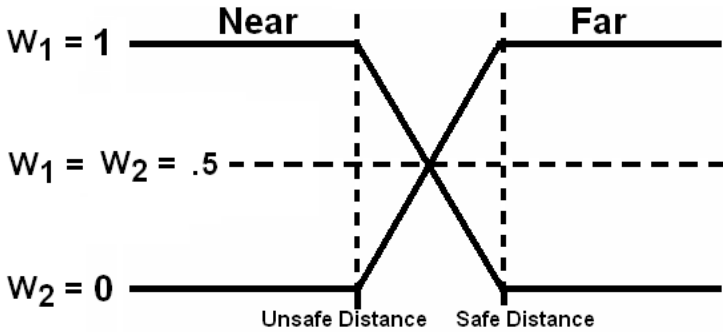


Figure III-33: Distance-dependent activation function.

$$OutputAngle = A_1 * w_1 + A_2 * w_2$$

A_1 = Recommended angle to avoid object
A_2 = Recommended angle to face waypoint
w_1 & w_2 = Values of the activation function

$$OutputSpeed = S_1 * w_1 + S_2 * w_2$$

S_1 = Recommended speed to avoid object
S_2 = Recommended speed to face waypoint
w_1 & w_2 = Values of the activation function

III.vi.2 Mapping & Navigation

A considerable amount of work has been done in the field of robot mapping and a considerable amount of software is available to allow robot mapping and navigation. The software used in this thesis is available as part of the "Simple Mapping Utilities (pmap)" from the Player software (Howard, 2004).

The pmap package provides several utilities for laser-based mapping in 2D environments. The components are as follows:

1) **Laser-stabilized odometry (lodo):** This library maintains a record of recent laser readings and will make corrections to the robot's odometry readings such that the current laser readings will be most consistent with previously obtained readings.

2) **Particle-filter-based-mapping (pmap):** This library maintains a particle filter representing a set of approximately 200 maps, which are produced by adding Gaussian noise to the predicted measurements. Maps that deviate significantly from expected measurements are removed from consideration and replaced with copies of more consistent maps in the set.

3) **Relaxation over local constraints (rmap):** This provides a post-processing step that uses an "iterated closed point algorithm" to refine a map so that there is increased consistency between repeated observations of the same features. This step is supposed to provide a solution to the "loop closing" problem, allowing a robot to provide an accurate correspondence when it returns to a location that it has already mapped.

4) **Occupancy grid mapping (omap):** This converts the laser readings taken during the mapping process to be converted into an occupancy grid.

Though effective in simulated environments and in small real-world environments, the Player mapping software performed poorly in the hallways used in our

experiments. Two major problems were encountered. First, the hallways were too long to be accurately mapped by the software. Odometry errors accumulated quickly and straight halls were mapped as being gently curved. Second, the loop in the environment was too big for the rmap portion of the algorithm to produce accurate correspondences.

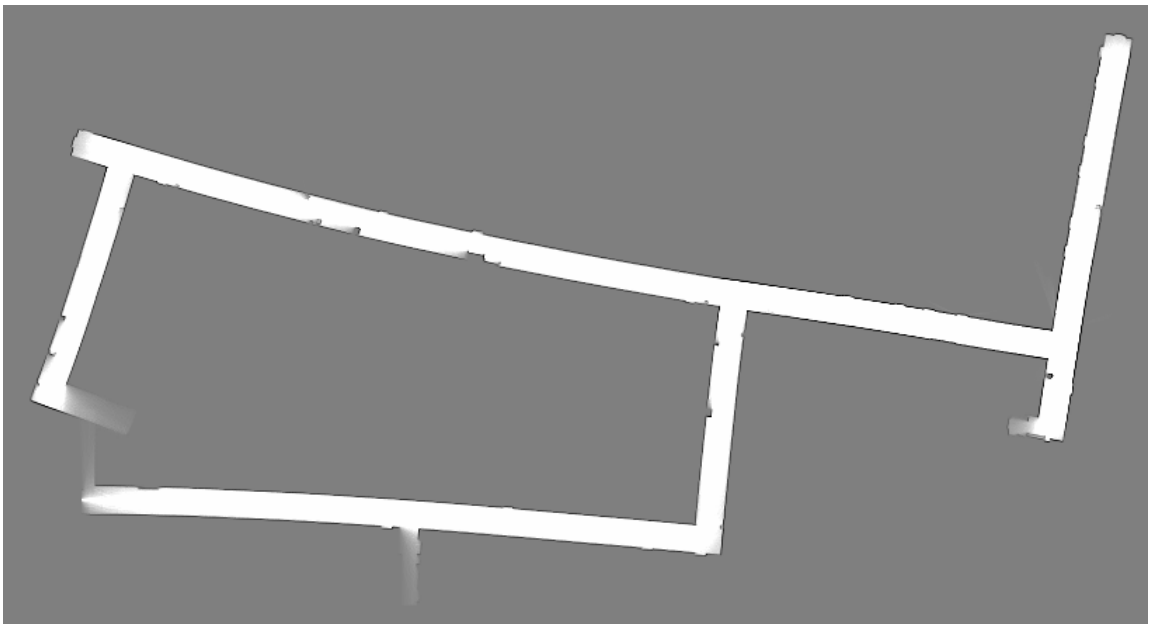


Figure III-34: Original mapping result using a particle filter with 200 particles.

Figure III-34 shows the map recorded by Player's software before modification. This map was made using 200 particles in the particle filter. Odometry errors resulted in curved corridors and a failure to close the loop.

The inaccuracies produced by the Player software were too large for the map to be useful in actual trials. Plus, maintaining 200 separate maps in the particle filter was too slow to be implemented on the robot in real time. In response to these problems, we developed an algorithm that improved the speed and accuracy of the results by making a minor assumption regarding the architecture of the building. This assumption was that most walls would be straight, parallel, and would meet at right angles. Since the vast

majority of modern buildings meet this criteria, this assumption should not be considered unreasonable. If a building did not meet the criteria, the mapping system would automatically resort to building a traditional map.

This assumption was applied as a secondary filter on the map particles maintained by player. As the maps were generated, a line-detection algorithm was used on each map to identify regions where wall measurements corresponded to linear regions. When linear regions were located, they were used to judge the accuracy of subsequent measurements. Subsequent measurements that were consistent with the observed linear regions were assigned higher accuracy readings, increasing the probability that the maps containing those measurements would be maintained.

A simple algorithm was used to implement this linearity. Linear regions were identified within an 8x8 meter area surrounding the robot on each map and tallies were made of the number of predicted wall-segments that were consistent with the linear portions. If the number of consistent measurements in the best of all maps became significantly greater (beyond a predefined threshold) than the number of consistent measurements in the worst of all maps, the worst map was replaced by a copy of the best map. Although this algorithm should function for lines intersecting at any angle, the line-detection algorithm was simplified and the accuracy was improved using the added assumption that lines could only intersect at right angles.

Figure III-35: Modified mapping results using a particle filter with 20 particles.

Figure III-35 shows the generated map using the modified Player software. This map was generated using only 20 particles (one tenth of the number of particles used in the previous map). This reduction in tracked particles allowed mapping to take place in real-time on a robot. It should be noted that if this algorithm is applied to regions containing no linear segments, it will default to the standard mapping algorithm, though performance will degrade since significantly fewer particles are used. Corridors appear straight and loop closure was successful.

III.vi.3 Target Seeking & Following

During all scenarios involving robot responses, robots are provided a map of their environment. Using Player-provided functions, the robots use this map to estimate their own location and orientation, as well as the location of other objects in the environment. Waypoints represent destinations that a robot will actively seek out. We allowed targets to be selected either manually or automatically, depending on the experimental demands. Manual waypoints are selected using a graphical user interface that allowed users to

apply a mouse-click to a desired location on the map. Robots patrolling the environment communicate among themselves to determine the robot (or robots) that are best suited for the task (based on robot-type and proximity). Those robots then converge on the destination.

Figure III-36: Example showing multiple robots seeking multiple waypoints.

Figure III-36 contains a screenshot showing multiple robots converging on targets. Waypoints are shown as white plus-signs inside colored circles. Robots appear as red or green varieties. Red and green lines connect robots to their destination waypoint. The pictures scattered around the image are videos taken from different parts of the hallway. Colored cylinders visible in these images correspond to waypoints on the map.

In addition to static waypoints, our software also allows waypoints to be assigned automatically by our behavior recognition software. For example, if our system

determines that a person has engaged in questionable activity, then a waypoint is created at the location of that person. As the person moves, the coordinates of the waypoint is adjusted accordingly. This adjustment can be made by the primary detection system or by a robot that is in pursuit.

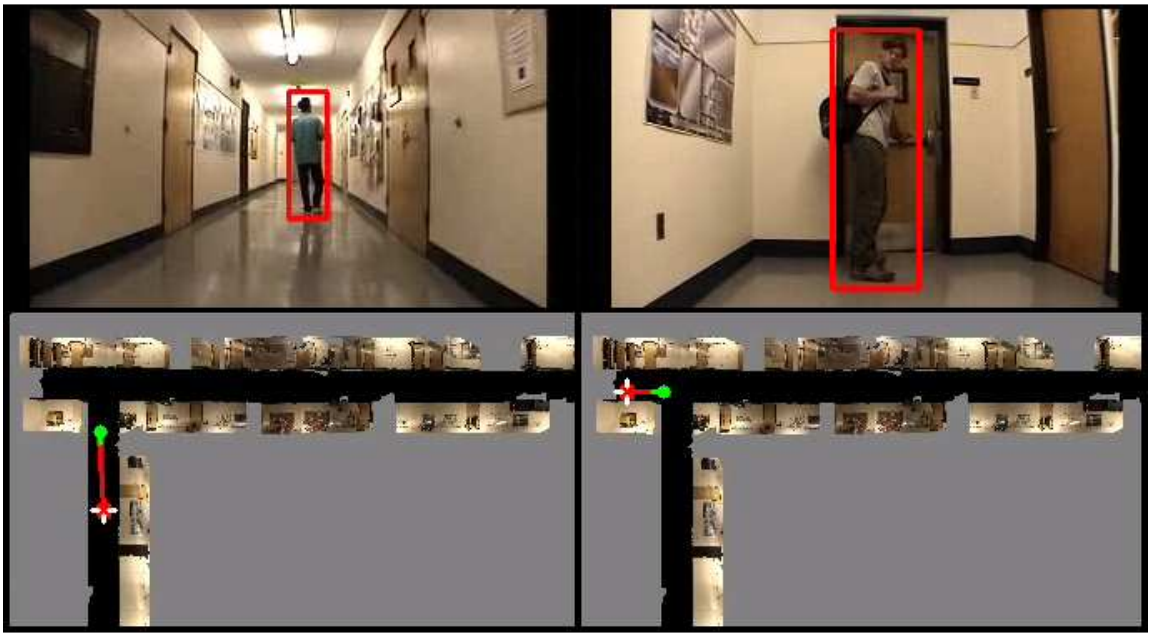


Figure III-37: Examples of a robot pursuing a person.

Figure III-37 shows screenshots from two pursuits. The pursuing robot estimates the subject's position within its own field of view and draws a red rectangle around the estimation (top images). The robot also estimates the subject's position on the global map and displays it using a red and white symbol (bottom images). The robot's own position is estimated and drawn on the map using a green symbol.

Target Seeking & Following Algorithm

The player software provides a wave-front planning algorithm that can be used to guide robots to a specific destination. The algorithm operates by dividing the known map into a matrix of square regions, which are assigned values that corresponds to the

minimum distance from that location to the target. The algorithm begins by assigning the value ‘0’ to the square that represents the destination. Squares that are immediately adjacent to the initial square are assigned a value of ‘1’, the next set of adjacent squares are assigned a value of ‘2’, and so on. This process continues until there are no more adjacent squares to be filled. When complete, only those squares that are reachable from the destination will contain a value. At runtime, the wave-front planner will iteratively advance the robot toward the adjacent square that contains the lowest value until the destination is reached.

When seeking stationary targets in an unchanging environment, the wave-front planner is guaranteed to identify the shortest path (within a given map resolution). The main disadvantage to the algorithm is that it is generally not possible to make local changes to the plan. If an obstruction is identified or if a target moves, a new wave-front must be generated for the entire map. Depending on the resolution and size of the map, frequent re-planning can interfere with real-time operation.

In our scenarios, we wanted to make it possible for robots to track and pursue moving objects in real time. Our goal was to have one or more robots converge on a person of interest using coordinates supplied by our system. If movement was observed, we wanted their position to be updated in real time. If one of our robots was in pursuit, that robot should also be able to continually update the position of the person to allow other robots to intercept. Because of the amount of time required for the traditional wave-front planner algorithm to operate in our environment (and because the Player version

was not designed to allow any re-planning at all), we decided to write our own version of the planner.

Our planner was optimized specifically for a hallway environment (characterized by long narrow interconnected corridors). In this environment, robots are relatively constrained to large-scale motion in one dimension (the direction of the corridor), though on a small-scale, complex maneuvering may still be necessary to avoid objects. Using this assumption, it is possible to greatly simplify the way a map is represented. Instead of representing every square decimeter of traversable space, the map can be reduced to a simple connected graph. The graph is designed so that nodes appear at intersections and at the ends of hallways. Additionally, nodes are selected so that there is a direct line of sight between every node, and all nodes directly connected to that node. For our scenarios, the simplification was significant. Using a matrix of .125 square-meter blocks to represent the floor-plan, our map required around 34,000 individual elements. Using a connected graph, the same floor-plan only required 8 nodes (shown in **Figure III-38**). Although this representation is particularly well suited for hallway floor-plans, it should be noted that this type of representation can be used for any environment. In addition to reducing storage and processing demands, a graph also increases the flexibility of the system. If a robot is attempting to navigate to one of the nodes and encounters an obstruction, it is relatively simple to eliminate the connection from the graph, while searching the remaining graph for the next-shortest route. The main disadvantage is that a shortest path will not be guaranteed, although depending on how the graph is constructed, the difference may be insignificant.

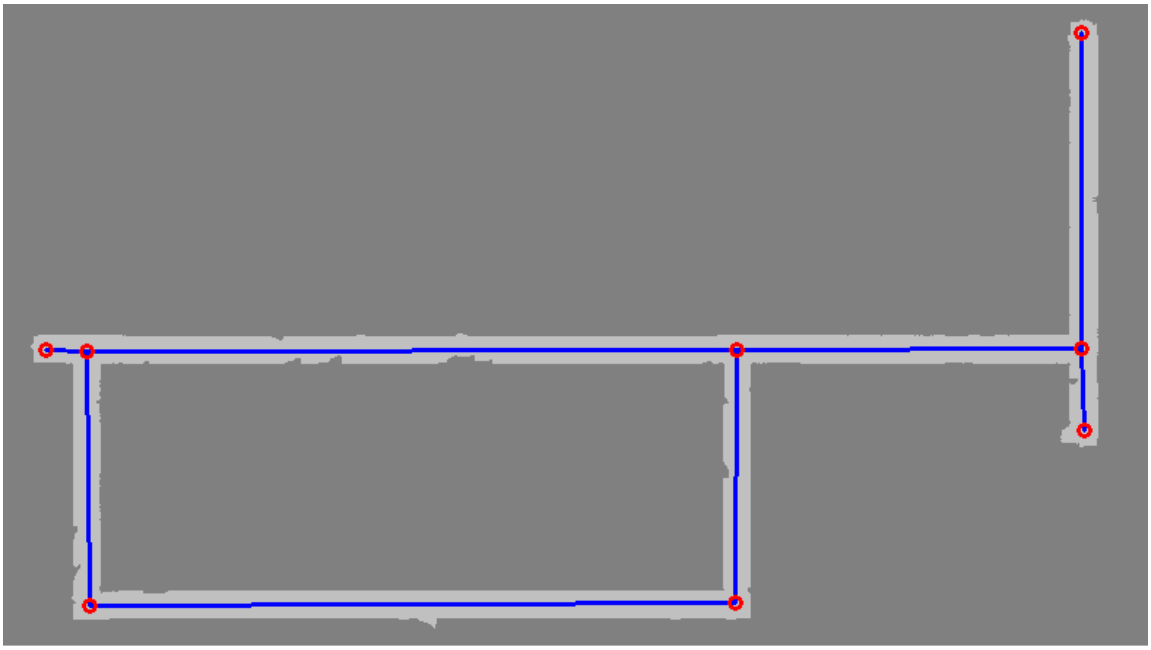


Figure III-38: Example graph representation of the hallway environment.

Figure III-38 shows the graph that is produced for our hallway environment. Graph nodes are represented using red circles. Links are represented using blue lines.

The following sub-sections will provide details on how we constructed the graphs and how we used the graph for waypoint seeking.

III.vi.3.1 Map Graph Construction

The idea behind our graph construction algorithm was to identify a graph containing a set of nodes that represent points within a given map that correspond to intersections, dead-ends, and hallway regions of high curvature. The results of our graph algorithm were shown in the previous section (**Figure III-38**). Since our environment primarily consisted of narrow interconnected corridors, we found that acceptable points could be extracted from a skeletalized representation of the original map. Image

skeletalization involves the iterative erosion of pixels from a region's perimeter (as shown in **Figure III-39)**, until the width is reduced to a single pixel. The graph-node detection algorithm is summarized after the image.

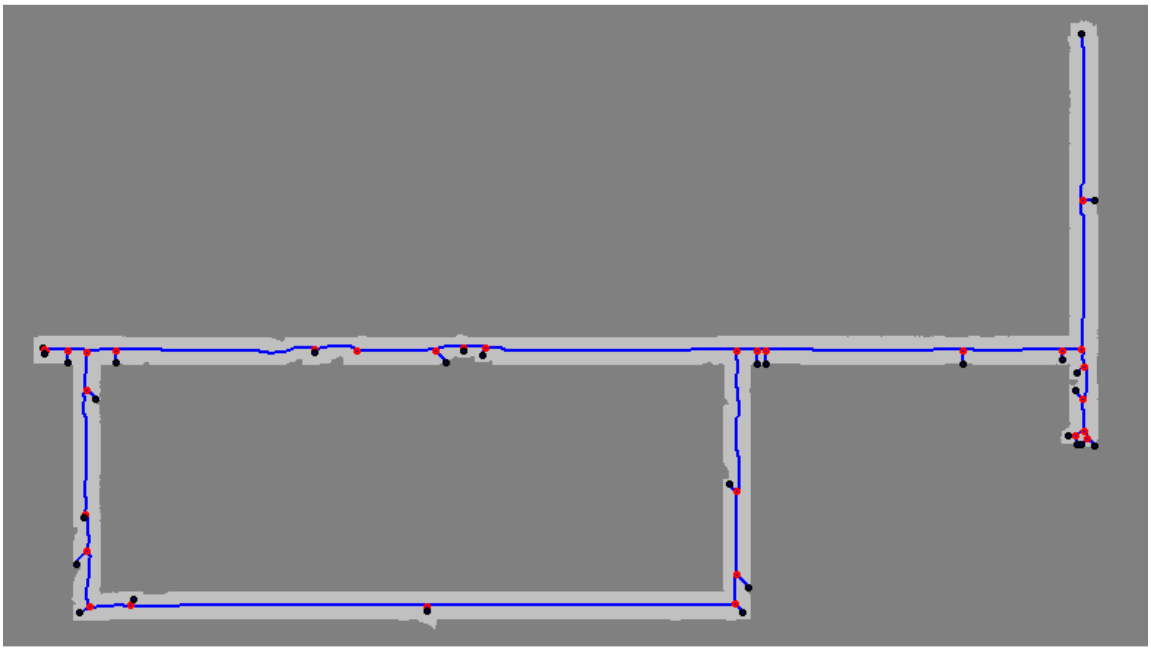


Figure III-39: Results of our skeletalization algorithm.

Figure III-39 shows a skeletalized version of the hallway environment. Blue lines represent the skeleton. Red circles represent nodes found at skeleton intersections. Black circles represent nodes found at skeleton end-points. Here, the majority of the black nodes are found at the ends of very short skeleton extensions (short blue lines). Almost all of these are culled from our final graph because they either fall too close to a wall or too close to the base of the extension. After the black nodes are removed, most of the corresponding red nodes are subsequently removed because they do not represent either intersections or endpoints.

Graph Node detection algorithm:

1) **Skeletize Map:** Reduce a map to its minimum skeletal structure. In **Figure III-39**, the skeleton is shown in shown in blue.

2) **Identify Nodes:** Check every pixel that lies along the map skeleton. If a pixel is connected to a skeletal pixel on only one side, that pixel is marked as an end-point-node. If a pixel is connected to three or four skeletal pixels, that pixel is marked as an intersection-point-node. Otherwise, the pixel will be connected to two skeletal pixels and will be ignored. In **Figure III-39**, end nodes are shown using black circles and intersection nodes are shown using red circles.

3) **Cull Nodes Near Walls:** A characteristic of the skeletization algorithm is that small narrow bumps will be detected as extensions to the skeleton. We use a simple threshold to eliminate any nodes that fell so close to a wall, that they could not be safely visited by a robot (threshold is $1/4^{th}$ of the width of the hallway). In **Figure III-39**, these unsafe locations appear as black circles immediately adjacent to walls.

4) **Cull Nodes Near Other Nodes:** By the same principle as that mentioned in (3), wide irregularities will produce shorter extensions that have endpoints closer to the base of the extension. We find that very short extensions provide little additional information about the environment, so we use the same distance threshold mentioned in (3) to also eliminate nodes related to these features.

5) **Cull Nodes on Straight Paths:** Under certain conditions, we do not require that the graph be minimized (a graph is minimized after the removal of nodes that are connected to exactly two other nodes). However, we do want to remove unnecessary nodes if their two connecting nodes are within line-of-sight of each

other. Two nodes are considered to be within line-of-sight, if one could step along the entire connecting line that spans between them, without leaving the portions of the map accessible by a robot (the light-gray portion).

6) **Subdivide Curve Paths:** We wanted every node to be within line-of-sight of each of its connecting nodes. If two nodes were found that did not satisfy this criterion, then an additional node was added along the skeleton path that spanned the two nodes. If the criterion could be satisfied by adding only one node, then the node was added to the point that maximized visibility to both existing nodes (if an additional node was added between existing nodes ‘A’ and ‘B’, and there was a region where both ‘A’ and ‘B’ was visible, then the point of ‘maximum visibility’ was determined to be half-way between the point where only ‘A’ was visible, and the point where only ‘B’ was visible). If the criterion could not be satisfied by adding only one node, then a node was added to the point on the skeleton that fell halfway between the two nodes. Each new segment was then recursively tested.

7) **Create a Map-to-Graph Translation Array:** After a graph has been generated and minimized, we create a fast-access array that allows robots in the environment to identify the graph node that is closest to its position on the map. This array contains the same dimensions as the original occupancy map and is produced at the same resolution. At the final stage of initialization, for each location on the map, we store a reference to the nearest graph node (with distances measured only across traversable space). A representation of our Map-

to-Graph translation array is shown in **Figure III-40.** Specifics about how this information is used to locate and track waypoints will be provided in the following section.

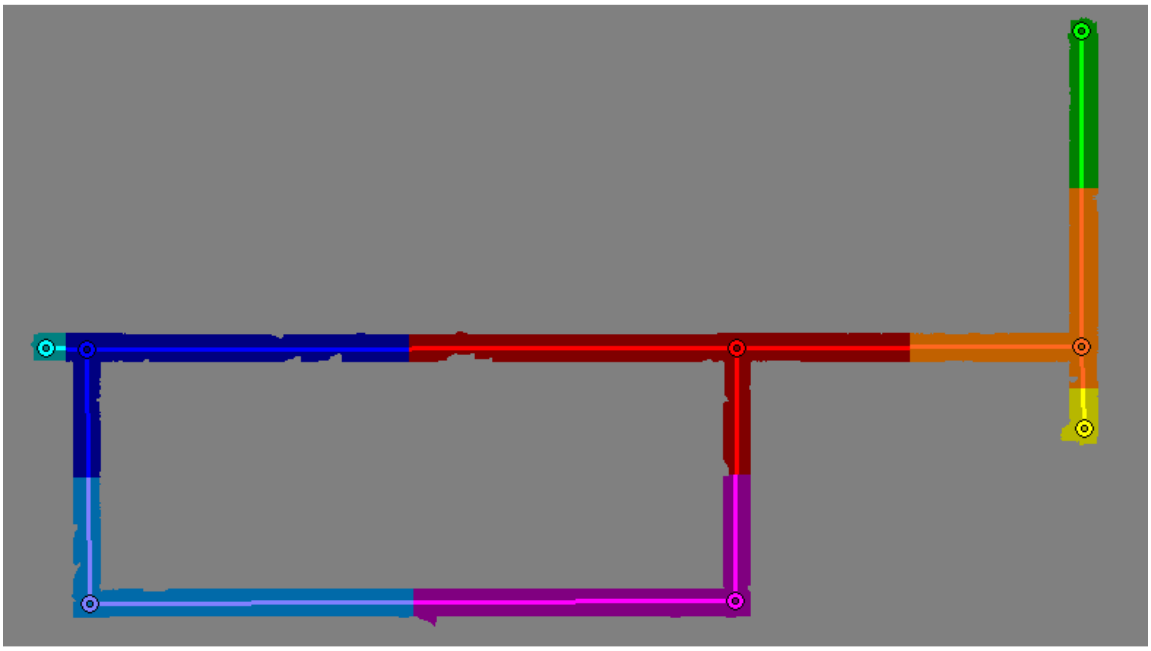


Figure III-40: A representation of our map-to-graph translation array.

Figure III-40 shows how a robot can efficiently find the nearest graph node by using its location as a query. Graph nodes are represented by colored circles, with corresponding regions being illustrated using a similar color. Lines between nodes are added only to help visualize the underlying graph.

III.vi.3.2 Dynamic Waypoint Seeking

We used waypoints in two different types of experiments. Manual waypoints are useful for moving robots from on point to another. A user can select a location in the environment using a GUI, and one or more robots will converge on that point. These waypoints are represented only by coordinates on the map, and are not necessarily related

to physical objects. Once the robots reach their destination, the waypoint is removed and the robots wait for additional commands.

Dynamic waypoints are assigned automatically during the activity recognition phase of our experiment. These waypoints differ in that they are physical objects that a robot can locate in its environment using its sensors. Once the robot has identified the object, it uses its localization information and the measured distance and angle from the object to continually update the position of the waypoint. The robot then makes the necessary adjustments in its orientation to maintain its approach. Even if the person disappears around a corner, the robot will still converge on the last known location of the waypoint until visual contact can be regained. The alternating adjustment of waypoint position and travel direction produces a realistic looking chase.

The algorithm for pursuing the waypoint is relatively simple when using the graph described in the previous section. The robot uses the Map-to-Graph Translation Array to identify the graph-node that is closest to its position in the environment (This is referred to as the primary-node. This is always considered to be within the robot's line-of-sight.). It then goes through the adjacency list of that node and identifies the set of connected nodes that are within the robot's line-of-sight (connected-nodes). As was mentioned previously, nodes are generated using the constraint that each node is within the line-of-sight of their adjacent nodes. This is done to simplify the way that a robot determines if a node is within its own line-of-sight. Basically, if the angle between the connected-node and the primary-node is similar to the angle between the connected-node and the robot (allowable error is within the width of the hallway), then the connected-node is

determined to be in the line-of-sight of the robot. The robot then stores the list of all node-pairs that are within its line-of-sight (all node-pairs will contain the primary-node and one of the connected-nodes).

Once the robot establishes its own position among the nodes, it repeats the test on the waypoint. If the waypoint returns one of the same node-pairs as the robot, then the waypoint is considered to be within the robot's line-of-sight and the robot will approach the waypoint directly (even if there is some error in this assumption, it will be close enough that the robot's obstacle avoidance routine will ensure that a safe path is followed). If the waypoint is not within the robot's line-of-sight, it will conduct a recursive search of the graph to identify the sequence of nodes that produces the shortest distance to the destination. The robot will then be directed toward the first node in that sequence. Since the graph search can be done extremely quickly, it is conducted at every time step. It should be noted that our definition of visibility ensures that before any node in the sequence is actually reached; subsequent nodes will always become visible. This ensures that the robot will not attempt to position itself at the exact location of each intermediate waypoint before continuing on to the next, thus producing a more natural and efficient path through the environment.

III.vi.4 Behavior Acquisition Using Genetically-Evolved NN

Designing robots that autonomously respond to dynamic environments has been such a difficult challenge that it was largely ignored until relatively recently. In solving this problem many researchers have largely abandoned the traditional techniques of rigid

plans and hard-coded controls and replace them with sensor-dependent, highly reactive behaviors. This offers the robot a distinct advantage over planning-based strategies, which usually require perfect world models and perfect localization techniques for safe locomotion. Instead, reactive robots only need the ability to sense and respond to their immediate surroundings. If these surroundings change, the robot can adapt its behaviors to the new situation.

Typically, the design of a reactive control algorithm is accomplished using a set of simple behaviors, which are each designed to accomplishing a very specific goal. Such behaviors could include avoiding obstacles, wall following, seeking out landmarks, seeking out open spaces, seeking out other robots, etc. Depending on the design strategy, robots may switch between its various behaviors, or it may use a weighting scheme to determine the extent that each behavior determines its actions. Although this approach to robot control is much more flexible than traditional methods, it still requires a considerable amount of hard coding and heuristics. Researchers must engage in the laborious process of separately coding each of the simple behaviors and then determine an appropriate way to fuse these behaviors, given certain environmental conditions. Each of these steps can be time-consuming, error prone, and subject to researcher biases and impatience.

In this section, we discuss how a genetically evolved neural network could take the idea of reactive control a step further to eliminate the laborious process of coding and fusing behaviors. Instead of coding the robot to react to its environment, a genetic algorithm could allow a naïve robot to be placed in an environment, where it would react

to its own interaction with the environment. A robot could attempt to fine-tune behaviors that prove effective, while abandoning those that are less effective. If a researcher wanted the robot to learn a new task, they would simply specify the feedback criteria. For example, to train a robot to follow another robot, a researcher would only need to define the desired distance the robot is expected to maintain, along with a numeric reward scale that the robot could use to define its own success. The robot would then train itself the behaviors necessary to identify the target in the environment, orient itself in the right direction, and pursue the target at a safe speed. We will show that this approach is effective at training a robot such tasks as safely moving about its environment, waypoint seeking, strategy building, pursuit, cooperation, and evasion. This technique can even make it possible for robots that have learned one skill to incorporate this knowledge into the learning additional skills.

Another interesting thing with this approach is its resemblance to the evolution of biological systems. To emphasize this similarity, we will show how different fitness criteria can allow the emergence of life-like behaviors. We will discuss five different robot interaction scenarios including individual competition, team competition, predator-prey relationships, mutualistic relationships, and parasite-host interaction.

In the following sub-sections, we will discuss the simulation environment, the learning algorithm, and some results from our trials.

III.vi.4.1 Simulation for Behavior Acquisition

Although the current implementation is entirely in simulation, the simulation has been made to closely resemble a real environment and it is believed that robots trained in the simulation should fair reasonably well in the real world. The map used for the simulation was generated from laser data acquired by driving a robot through the halls of the computer science building. The drivable area is about 90 by 50 meters and is represented at a scale of 1 pixel per .125 meters. This map is shown as figure 1. The rough color texture is only used to aid the researcher in location identification and is not currently used by the robot.

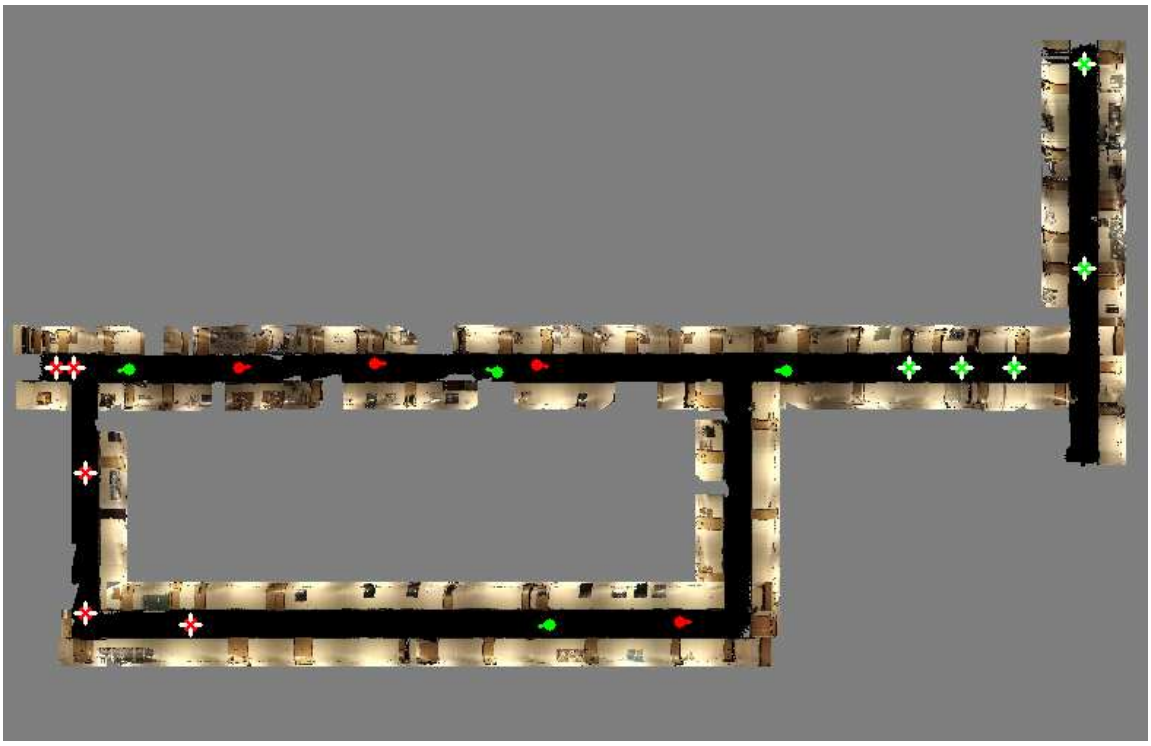


Figure III-41: Simulated robot environment.

Figure III-41 contains a screenshot of the system in action. A description of the symbols are provided in the following section.

III.vi.4.1.1 Map Entities

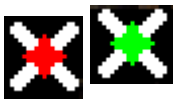
Since the primary goal is to train robots to operate in a dynamic environment. The simulation had to contain a sufficiently dynamic quality. This was achieved by simultaneously operating and training eight robots within the same simulation. Each robot was considered an obstacle to every other robot and created a very unpredictable setting. Graphic representations of the map's components are as follows.

Living robots can be one of two types represented by red and green. In heterogeneous populations, the colors represent different specialties or tasks. In homogenous populations, colors may represent robot teams. Robots view their environment through a set of sensors (explained below) and respond to the environment by manipulating their position.

Dead robots are represented as hollow circles and are incapable of movement, though these robots are still considered obstacles to other robots. A robot dies when it hits a wall or another robot.

Waypoints represent goals to the robot. If a robot gets sufficiently close to the center of the waypoint, that goal is considered attained. In this simulation, red waypoints are goals to green robots and green waypoints are goals to red robots.

If the goal represented by a waypoint can be satisfied. Then the satisfaction of that goal will cause the waypoint to deactivate and turn dark.



Some waypoints represent complicated goals, which require the visitation of multiple robots. An ‘X’ represents a waypoint that has only partially been satisfied.

III.vi.4.1.2 Robot Sensors (26 total)

The robots are only aware of locally accessible information. They do not have any global knowledge of the map and do not know the locations of waypoints or other robots unless those objects are sufficiently close and are in the robot’s line of sight. If an object is just around a corner, the robot will be unaware of its presence. The robot contains sensors to detect its own speed, the location of obstacles, the location waypoints, and the location of other robots. Sensor specifics are shown below.

- **1 Speed Sensor:** The robot receives an accurate measure of its own speed. Due to the imposition of momentum, this speed will not necessarily equal the speed commands being sent by the robot.
    - Provides: Measurement of speed (meters/sec)
- **7 Laser Sensors**: For computationally efficiency, the laser used in simulation provides less information than a real-world laser. In the real-

world, the laser takes up to 361 measurements with 180.5 degrees coverage, providing an angular resolution of 0.5 degrees.

  - Activated by walls or other robots
  - Provides: Straight-line distance (meters)
  - Total Coverage: 180 degrees in front of robot
  - Resolution: 30 degrees
  - Range: 8 meters.

- **4 Team Robot Sensors**: This low-resolution device will not provide an accurate angle to an object, but will provide an accurate distance. The angular resolution is only good enough to determine if the object is in front, behind, left, or right.
  - Activated by the presence of same-colored robots
  - Provides: Straight-line distance (meters)
  - Total Coverage: Overlapping coverage of 360 degrees
  - Resolution: 120 degrees
  - Range: 24 meters

- **4 Opponent Robot Sensors:**
  - Activated by the presence of opposite-colored robots
  - Specification match the sensor above

- **4 Team Waypoint Sensors:**
  - Activated by the presence of same-colored waypoints

- Specifications match the sensor above

- **4 Opponent Waypoint Sensors:**
  - Activated by the presence of opposite-colored waypoints
  - Specifications match the sensor above

- **2 Internal Memory Sensors**: The conventional reactive-architecture does not contain any internal state. Robots only respond to their immediate surroundings. Though this can produce effective behaviors, it also limits what a robot can accomplish. In an attempt to offer robots some ability to store current state information, these memory sensors were added. These sensors simply report the values of two output neurons, producing a limited amount of recursion in the neural network.
  - Always active
  - Provides: A capped approximation of two output values
  - Range: Values are capped to a range of -10.0 to 10.0

III.vi.4.1.3 Robot Output Commands (10 total)

For every iteration of the simulation, robots receives information from their available sensors, they processes the data, and produce an output, which controls their steering and drive mechanisms. The robots are only able to issue a limited number of commands to their motors. These are as follows.

- 5 Steering Angles:
    - -15° (right)
    - -5° (right)
    - 0° (straight)
    - 5° (left)
    - 15° (left)
- 3 Speed Velocities:
    - 0.02 meters/sec
    - 0.1 meters/sec
    - 0.5 meters/sec
- 2 Internal Memory Outputs: Recursively fed to the 2 internal memory sensors

In an attempt to increase the simulation's realism, commands to the motors do not generate instantaneous changes. Instead, the motors will respond by the amount that is half-way between the robot current state and the requested state. For instance, if the robot is traveling at 0.1 meters/sec and it receives a command to go 0.5 meters/sec, the resulting speed will be .3 meters/sec. After the command has been issued to the motors, the robot's position will be updated on the map. If the movement placed the robot onto a wall or another obstacle, that robot is considered wrecked and will be disabled for the remainder of the trial.

III.vi.4.2 Genetically-Evolved NN Algorithm

Robot-learning was accomplished using the "Neuroevolution with enforced sub-populations" (ESP) algorithm. Gomez demonstrated that this algorithm works well for general learning tasks (Gomez & Miikkulainen, 1999), and Bryant successfully applied it to a multi-agent computer game that required agents to compete for resources on a 331-cell playing grid (Bryant & Miikkulainen, 2003). The algorithm contains a simple feed-forward neural network containing an input layer, an output layer, and a single hidden layer of neurons. Each neuron produces a floating-point output, which is the summation of its inputs multiplied by its input-weights. The network is trained by systematically adjusting these weights. Weights are modified using a genetic algorithm that treats the set of weights as a single-dimensional chromosome. The genetic algorithm maintains populations of these chromosomes, and modifies the values using mutation and crossover functions. The fitness for each chromosome is determined by evaluating the fitness of the neural network of which it is participating. Chromosomes corresponding to high-fitness networks have a higher probability of surviving to subsequent generations, thus preserving the set of weights that most benefited the neural network.

The novel feature of the ESP algorithm is that each network is made up of several chromosome sub-populations, which are evolved independently of one another. The number of sub-populations is determined by the number of hidden cells in the network. In the example shown in figure 2, there are six hidden cells, and therefore six separate populations that make up this network. The number of alleles in each chromosome is determined by the number of input weights entering the hidden cell, plus the number of

output weights exiting the cell. To evaluate a member of the chromosome population, the represented weights are applied to the hidden cell that corresponds to its population. Five more chromosomes are randomly selected from their respective populations and the network is evaluated. The resulting fitness is assigned to every individual that participated in the network. This process is repeated until every member of each population has been evaluated a sufficient number of times to estimate its contribution.

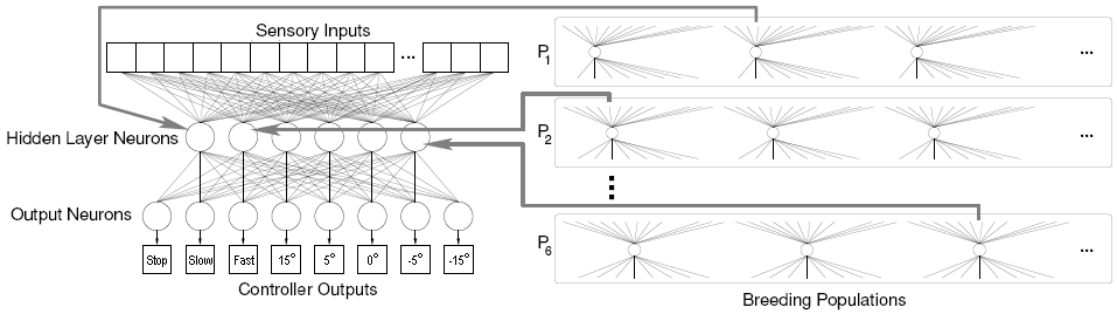


Figure III-42: Representation of hidden cell breeding populations.

Figure III-42 shows that each cell in the hidden layer is taken from a population of cels that are separately trained and scored based on their ability to function with other randomly selected cells. Image is a modified version of a graphic in (Bryant & Miikkulainen, 2003).

III.vi.4.2.1 Application of ESP to robot control

Each robot controller consisted of a fully-connected three-layer neural network similar to the one described above. The network contained 26 input neurons corresponding to the 26 input sensors, 10 output neurons corresponding to the 10 output commands, and 6 hidden neurons. Additionally, the network contained bias neurons in the input and hidden layers that propagated a value of one (multiplied by the corresponding weights) to the downstream neurons.

There were 37 weights associated with each hidden neuron, containing weights from the 26 input neurons, 1 input-bias neuron, and 10 output neurons. These weights were stored as a 37-allele long, one-dimensional chromosome. New robots were given weights that were initialized to random values between -1 and +1 such that there was a high likelihood of values close to zero and an exponentially lower likelihood that values would be close to ±1. Mutation could change the weights from their original values but a restriction was applied that capped the ranges between -1.0 and +1.0. Each of the hidden neuron chromosomes was maintained in a separate sub-population. Six sub-populations contained chromosomes from the 6 hidden neurons and a seventh sub-population contained the chromosome associated with the hidden bias-neuron. For the current implementation, each population contained 51 individuals. This number was selected somewhat arbitrarily to produce sufficiently long learning-curve graphs in a tractable amount of time. Bryant suggested that better results could be obtained using much larger populations (500+ individuals) (Bryant & Miikkulainen, 2003) but this has not yet been investigated for the current project.

During each learning cycle, a neural network was assembled by randomly selecting a set of chromosomes from each of the 7 hidden-cell populations. The network was evaluated and the resulting fitness value was given to each chromosome that participated. This process was repeated for another randomly selected set of chromosomes until all chromosomes got a chance to participate in 10 networks. The chromosomes final fitness is the average of all 10 evaluation opportunities.

III.vi.4.2.2 Evaluation

The fitness for each neural network was determined by using it as a controller for one robot simulation trial, which included 7500 robot-movement iterations. During each iteration, input neurons received the values of the robot's sensors and the network's output neurons directed the robot's motors. At the start of the trail, the robot was placed in a random location of the environment, along with seven other robots and as many as 10 waypoints. The robot was then driven using the neural network until it crashed or until the trial-duration elapsed. This process was repeated multiple times from multiple starting points until all hidden cells had the opportunity to be evaluated 10 times, for each of the 51 individuals in the population. Consequently, 510 trials were executed for every generation of individuals. Throughout each trial, fitness points were assigned to or deleted from each robot. Whether the robot gained or lost points depended on whether its behavior matched that, which was expected by the researcher. Details about each of the specific training environments will be described shortly.

III.vi.4.2.3 Mating

After the set of evaluation trials was completed, individuals from each sub-population were mated using an elitism breeding strategy. The one individual with the highest fitness continued to the next generation unchanged. 24 breeding pairs were then selected to produce 24 pairs of offspring for the next generation. Selection was determined using a normalized roulette-wheel strategy. The roulette-wheel provided high-fitness individuals with a high probability of participating in a breeding pair, while

low-fitness individuals were given a low probability. Individuals were bred with a 75% probability using single-point crossover. Mutations occurred with a 1% probability on the resulting offspring.

To further introduce genetic variability, two “outsider” individuals were added to the subsequent population. Depending on the training phase, these individual were either modified versions of the most fit individuals (modified, in such a way that every allele was changed a small amount from the original), or they were extracted from the population of another robot of the same type. This genetic sharing was used as a mechanism to allow individuals to either evolve independently, or to evolve as a team. This was necessary because of the way robots were trained. Before robots were placed into a complex learning scenario, they were first taught basing obstacle avoidance behaviors. During this initial training, robots were not allowed to share genetic information with the hope that they would each develop their own strategies and personalities. For all subsequent scenarios, robots on the same team (represented by the same color), were allowed to share their best population members with others on the same team. It was believed that this would help reintroduce beneficial genetic variability into populations that had already converged during the initial obstacle avoidance training.

III.vi.4.3 Experimentation

Robots were placed in six different training scenarios to demonstrate their ability to autonomously learn a variety of tasks. These scenarios were Obstacle-Avoidance,

Individual Waypoint-Seeking, Team Waypoint-Seeking, Predator-Prey Interaction, Mutualistic Pursue-Pursued Interaction, and Parasitic Pursue-Pursued Interaction.

A population of 51 robots was trained in each scenario for between 100-300 generations. The robots placed in any given scenario were physically identical to those placed in any other scenario. All robots possessed the same set of sensors and produced the same output control, with one exception: A few of the scenarios required a diverse group of individuals. This was accomplished by simply regulating the top speed of one group of individuals to be slightly slower than other robots in the scenario. When going into the scenario, the modified individuals had only been trained on their non-modified equipment and had to adapt to the modifications on the fly.

Although the robots were physically identical, they all had different learning backgrounds. The only scenario that was run on randomly initialized robots was the Obstacle-Avoidance scenario. All other scenarios were run on robots, which had already learned basic obstacle-avoidance techniques.

Scenarios were analyzed qualitatively by visually observing the behavior of the robots at different stages in training and quantitatively by observing the resulting learning graph. The specifics for each scenario are as follows:

III.vi.4.3.1 Obstacle Avoidance Scenario

The Obstacle Avoidance Scenario was intended to teach robots safe navigation. Robots placed in this scenario had no prior driving experience and contained neural networks

with randomly initialized weights. The robots populations that were trained in this scenario were used as seed populations for all subsequent scenarios.

Setup:

Four green robots, four red robots, five green waypoints, and five red waypoints were placed at random locations across the simulation environment.

Fitness Evaluation:

Robots were encouraged to drive fast and to explore a large portion of the map. The fitness function was (Fitness = DistanceTraveled x AreaExplored). It should be noted that the robots were not explicitly punished for crashing, though crashing indirectly reduced their fitness by preventing them from explore further. Robots did not receive any reward for crossing waypoints. The waypoints were simply placed in the environment so that the robots would get used to their presence.

Qualitative Evaluation:

When the robots were first exposed to the map, about 60% drove directly into a wall, 30% sat in one place and spun, and 10% drove at least a short distance. Surprisingly, a select few preformed quite well on their first attempt. These robots usually drove parallel to a nearby wall, and would even follow it around corners. Most robots drove quite slowly.

After twenty generations, the robots seemed to get used to their environment. Their mastery of following walls was good but appeared unable to avoid other robots.

Robots at this point seemed to have learned that they could get more points by driving fast. Spinning behaviors were replaced by small irregular-circle behaviors, which apparently allowed them to log more miles.

After 100 generations, the robots had almost completely mastered wall-avoidance. They still had occasional collisions with each other but appeared to be "aware" of each other's presence. Robots would drive at top speed most of the time but would frequently slow down when approaching corners, dead ends, or other robots.

Quantitative Evaluation:

Robots displayed a rather linear learning curve that may have leveled off by the hundredth iteration. It is unknown what would have happened in future generations.

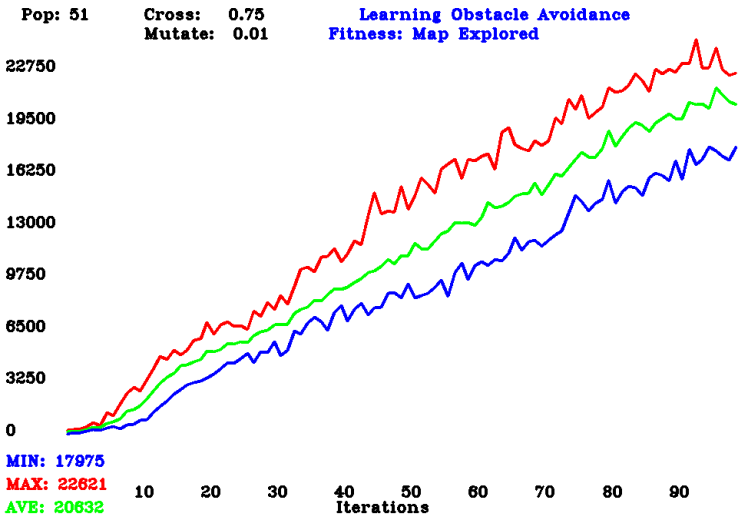


Figure III-43: Obstacle Avoidance learning rates.

III.vi.4.3.2 Individual Waypoint-Seeking Scenario

The Waypoint-Seeking Scenario demonstrates that robots that are previously trained in an obstacle avoidance behavior, could be further trained to engage in a task.

Setup:

The environment was divided into three sections. One section contained only red robots and red waypoints. A second section contained only green robots and green waypoints. The third section acted to create an open area between the other two.

Fitness Evaluation:

Robots were encouraged to find all waypoints of the opposite color as quickly as possible. Every robot that drove sufficiently close to the appropriate waypoint was awarded an increase in fitness that was proportional to the amount of time left in that trial. When a robot discovered all five waypoints, its accumulated fitness value was doubled and it was removed from the map.

Qualitative Evaluation:

As an accidental consequence of the waypoint application algorithm, all waypoints tended to appear in the centers of the corridors. Consequently, the robots learned rather quickly to drive rapidly around the map, while maintaining this position. Although this strategy was effective at finding waypoints, it was also effective at crashing into other robots.

After about 100 iterations, the robots seemed to be exploring the map very systematically. Robots stayed to the center of the corridors when approaching waypoints and moved to the left side of the corridors when approaching other robots. Most robots could very quickly eliminate all waypoints.

Quantitative Evaluation:

The obstacle avoidance learning somewhat prepared the robots for this new task. On the first generation, they were already exploring and finding waypoints reasonably well. For some reason, their performance dropped steadily for the first fifteen generations. It is hypothesized that this decrease was a result of the new influx of genetic material from teammates or it may have been a consequence of learning to move toward the centers of corridors to find waypoints. After 15 generations, the robots improved for a while, stabilized, and then continued further until the session ended.

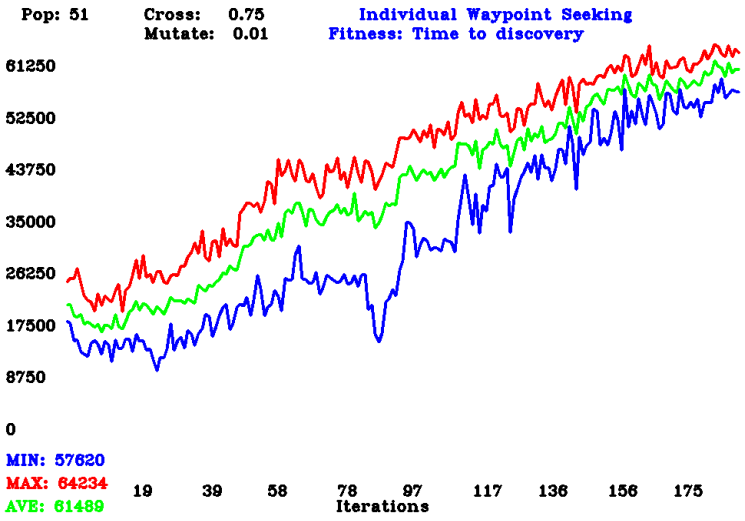


Figure III-44: Individual Waypoint Seeking learning rates.

III.vi.4.3.3 Group Waypoint-Seeking Scenario:

The Group Waypoint-Seeking Scenario demonstrates strategy development. Robots do not directly gain an advantage from their actions but do directly suffer losses. This creates a situation where robots are conflicted between the selfless desire to accumulate points for their team and the selfish desire to be safe and have others accumulate points for them.

Setup:

The environment was divided into three sections and object were placed the same way that they were in the Individual Waypoint-Seeking scenario.

Fitness Evaluation:

Robots teams were encouraged to find all waypoints of the opposite color as quickly as possible. If any robot on a team discovers a waypoint, every living member of that team is rewarded with an increase in fitness that is proportional to the amount of time left in that trial, while every member (living or dead) on the opposite team receives a penalty equal to half the amount.

Qualitative Evaluation:

Like in the previous experiment, early stages of this game were quite chaotic. After a hundred, or so iterations, however, the robots really appeared to develop a strategy. As mentioned, robots received a stiff penalty for dying, so most moved quite carefully when around other robots. Robots also seemed to hover around their own

waypoints with the possible dual-goal of protecting their own life while blocking their waypoint from the opposite team. When robots would see an opening toward the opponent's waypoint, they would race at full speed to intercept the waypoint.

Quantitative Evaluation:

As in the previous scenario, Performance dropped steeply early in the trial. In this case, the drop is likely due to the robots selfish desire to get more points by driving slower and living longer. This behavior was harmful to the team because waypoints discovered later in the trial were worth fewer points. The green team was the first to recover from the drop and dominated the field for about 45 generations. The red team then gained control and dominated through the remainder of the run. Following the initial drop in fitness, combined fitness for the two teams increased steadily through the experiment and did eventually exceed the initial values.

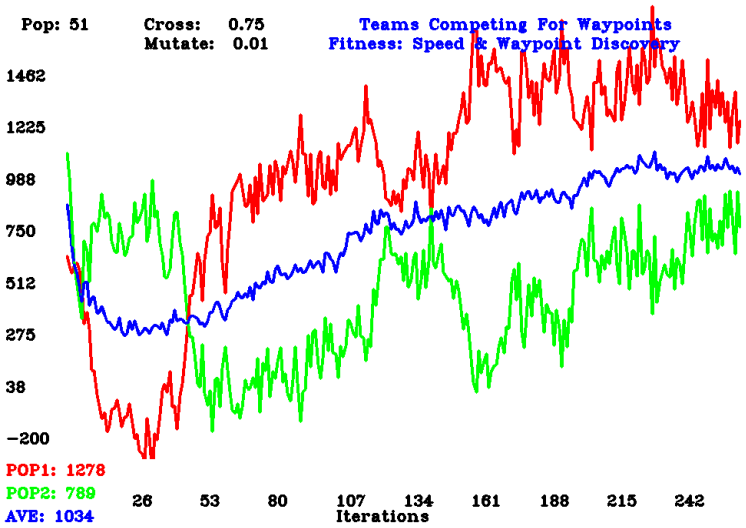


Figure III-45: Team Waypoint Competition learning rates.

III.vi.4.3.4 Predator-Prey Scenario:

The Predator-Prey Scenario **d**emonstrates the acquisition of pursuit and evasion strategies. Here, a small group of predatory robots must capture individuals from a fast group of prey robots. Prey robots must develop strategies of survival.

Setup:

The environment was divided into three sections. Two predator robots were placed on one side of the map and six prey robots were placed at the opposite side. For this scenario, prey robots were not destroyed if they crashed into each other and predator robots were unaffected by any robot. Robots were still required to avoid walls.

Fitness Evaluation:

Predator teams were encouraged to quickly seek out and destroy prey robots while prey robots were simply encouraged to survive. Predator robots that intercepted prey robots were rewarded with an increase in fitness that is proportional to the amount of time left in that trial. Prey robots were assigned a fitness that was equal to the number of iterations they survived. All robots received a fitness of zero if they crashed into a wall.

Qualitative Evaluation:

It didn't take long for the prey robots to discover that they were prey. After only a few generations, they were starting to show dodging and fleeting behaviors. It is unclear when or if the majority of predators learned that they were predators. Many just appear to drive around at top speeds. In later generations, some predators were found to reverse

direction if a prey was following them and many would risk getting close to a wall if a prey was trapped there.

Quantitative Evaluation:

After the initial generations, the learning curves for both predators and prey remained quite flat. This is most certainly due to the fact that both predators and prey were getting better at their job at about the same rate.

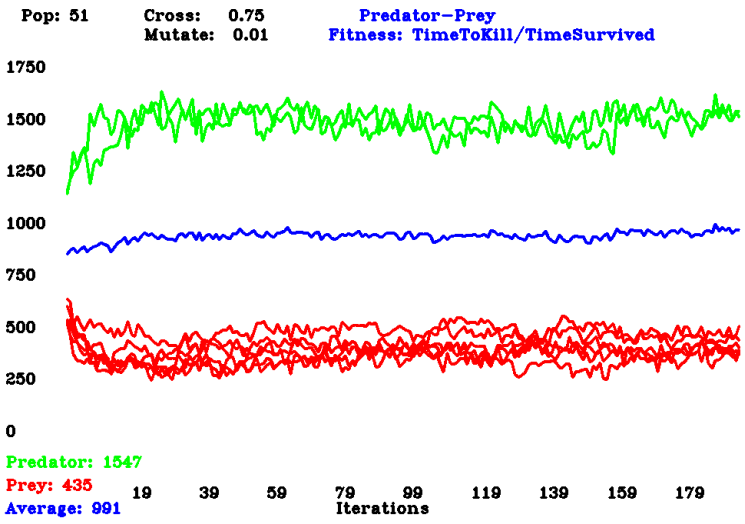


Figure III-46: Predator-Prey learning rates.

III.vi.4.3.5 Mutualism Scenario:

The Mutualism Scenario demonstrates cooperation. Two dissimilar groups of robots must work together to accumulate points.

Setup:

The environment was divided into three sections. Two grouper robots were placed on one side of the map and six groupie robots were placed at the opposite side. For this scenario, robots were not destroyed if they crashed into each other but they were still required to avoid walls.

Fitness Evaluation:

Groupers were encouraged to have groupie robots following them. Groupie robots were encouraged to follow grouper robots. If the distance between the following robot and followed robot was between 2 and 6 meters and if the groupie was facing the grouper, both robots received a point for the iteration. Grouper robots could receive multiple points if they were pursued by multiple groupie robots.

Qualitative Evaluation:

Within 20 iterations, groupies were regularly observed attempting to pursue their groupers, though they often failed at the task. Over dozens of iterations, the groupers appeared to be assisting the groupies by reducing their speed. By the hundredth iteration, robots were behaving very productively. Groupies formed nice tight clusters behind groupers and groupers all drove at their minimum allowable velocity to facilitate the collection of groupies and when groupers were not pursued by a groupie, it would frequently sit in one spot and spin until a groupie approached. The grouper would then move in the opposite direction of the groupie. After two hundred iterations, groupers learned that they could make the most points if they stuck together. Even if a grouper possessed all the groupies, it would still speed up to catch the other grouper.

Quantitative Evaluation:

Groupers and Groupies steadily increased their fitness throughout training.

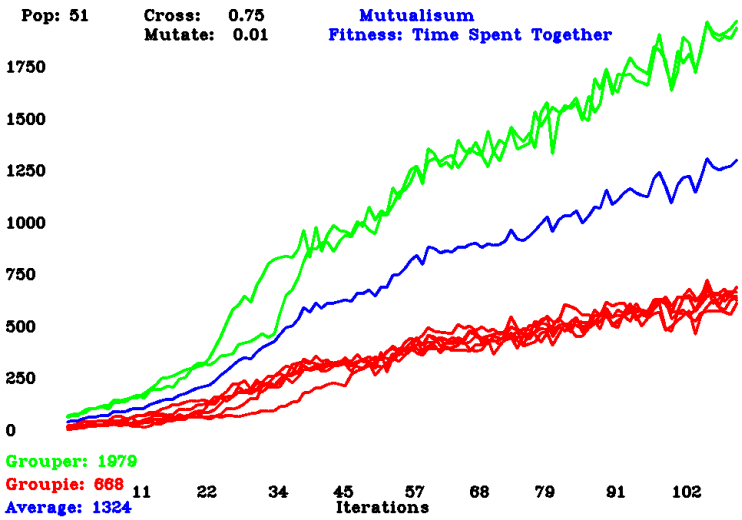


Figure III-47: Mutualism learning rates.

III.vi.4.3.6 Parasite-Host Scenario:

The Parasite-Host Scenario demonstrates pursuit and evasion. Similar to the predator-prey scenario but here, robots are not competing against each other and may be able to cooperate to achieve their goal.

Setup:

The environment was divided into three sections. Two host robots were placed on one side of the map and six parasite robots were placed at the opposite side. For this scenario, robots were not destroyed if they crashed into each other but they were still

required to avoid walls. The host robots used in this scenario had only been trained on the obstacle avoidance task. The parasite robots had additionally been trained as groupies under the previous scenario. Although the parasite fitness conditions are slightly different than the groupie conditions, they were similar enough to give the parasites an advantage.

Fitness Evaluation:

Hosts were encouraged to evade the parasites. Parasite robots were encouraged to remain in the proximity of host robots. If the distance between the parasite robot and host robot was less than 3 meters, the parasites received a point for the iteration. The host robot automatically received a point during every iteration but lost a sixth of a point for every parasite it had acquired for that iteration.

Qualitative Evaluation:

The robots used in these experiments have a drive setting for slow speeds, but not one to allow them to stop. Early in the training, hosts exploited this fact and were to loose many of their parasites by entering into a spin behavior. This behavior essentially halted the forward movement of the host and parasites continued past. It didn’t take long before parasites adapted to this behavior. In response, parasites started displaying a circling behavior around the host. When this happened, hosts completely stopped with the spin behavior and began a new behavior, where they would move at minimum speed when not infected and then when they were infected, they would seek out their teammate in an apparent attempt to attract the parasites to him.

Quantitative Evaluation:

Throughout the game, hosts steadily lost their advantage over the parasites.

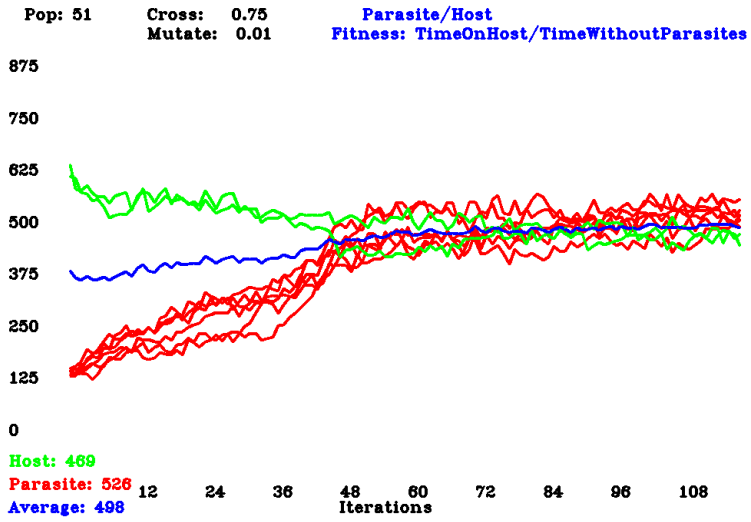


Figure III-48: Parasite-Host learning rates.

III.vi.4.4 Discussion & Future Work

Even in simulation, the presented approach demonstrates that ESP can be a powerful and flexible learning tool. In a matter of a few hundred generations using 51 individuals, behaviors can emerge that are both complex, productive, and have a strong similarity to natural animal behaviors. It is rather remarkable that neural networks containing only 26 input cells, 6 hidden cells, and 10 output cells can produce actions that go beyond basic reactive behaviors and almost appear to be carefully planned out strategies as was shown in the team waypoint seeking tasks. The network even managed to overcome inadequacies that a professional programmer would have had a very hard time overcoming. For instance, the resolution of the opponent robot detection device was

extremely poor. It really only indicated if the opponent was in front of, beside, or behind the robot. Yet somehow, the pursued robot was able to dodge and avoid the predator as if it had perfect knowledge of the predator's location.

In addition to being robust enough to learn complex tasks, the ESP algorithm also proved to be incredibly flexible. One network with no modifications could be trained to handle a large variety of different task. This could potentially be the ideal tool for robot development. By simply defining a set of appropriate criteria, a researcher could let a robot teach itself how to complete a task. If a simulated world was sufficiently similar to the real world, a robot that was given a task could simply "think" through the request until it was sufficiently confident that it could actually achieve it. The robot would then proceed in the real world until it ran into an unexpected situation that required more internal processing.

Most research in the field of neural evolution has been done in simulated environments, many of which are far simpler than the environment presented here. This work brings a real-world application for neuroevolution well into the realm of possibilities. Everything in the simulation has been designed so that it would have a real-world counterpart and most of the behaviors have already been implemented by this researcher using non-neural network approaches including obstacle avoidance, pursuit behavior, and waypoint seeking. There would still be a number of hurdles to be overcome before this extension into a real environment is possible but it is certainly well within reach.

Chapter IV Experimental Validation & Results

We have incorporated various components of our algorithms into full functioning systems, presented here with a brief description of the details, results, and images. Most of our scenarios begin with the automated monitoring of an individual as that person interacts with different objects. The system then attempts to identify the person's actions (Section Chapter IV-IV.i.1). In cases where the identified behaviors require robot interaction, the related robot response routine is initiated. The response may be in the form of aid provided by a helper robot (Section Chapter IV-IV.i.2), or a patrol robot may be summoned to investigate the person's presence or engage in pursuit (Section Chapter IV-IV.i.3). In our final section, we propose a multi-robot architecture that allows patrolling robots to autonomously communicate between themselves for the purpose of responding to events in a way that minimizes the distance traveled by robots capable of handling the task (Section Chapter IV-IV.i.4).

IV.i.1 Action Recognition

This section contains examples of our basic action recognition systems. Images from three different scenarios are shown. In two of the scenarios, a robot was positioned so that it could view a tabletop that a person was using while they engage in activities that were related to eating or doing homework (**Figure IV-1**). In the third scenario, the robot watched a person who sat on a couch and extinguished a trashcan fire. In all scenarios, the system tracked a subject's hand as that person reached toward and moved

various objects. Object models and action labels were assigned manually during an off-line process described in **Chapter III** and are listed in a table found in each of the following sub-sections.

Figure IV-1: Setup used in our tabletop demonstrations.

Figure IV-1 shows the robot (lower left) looking down at the table using its red stereo camera (top left) while the user manipulates objects on the table's surface.

For display purposes, we show the environment from the robot's point of view and detected objects are highlighted by outlining the regions using color-coded ellipses. The system also provides a list of all objects that are visible to the system. When the hand approaches one of the objects, the hand is considered to be "interacting" with that object (shown on the image using a blue connecting line). The names of interacting objects are

highlighted with color and appear to rise from the list. The system also provides a label describing the predicted action below the object name. **Figure IV-2** shows an example screenshot from our homework scenario. Here, a person is reaching toward an open laptop while the system predicts that the likely action will be that the user intends to "operate" the laptop.

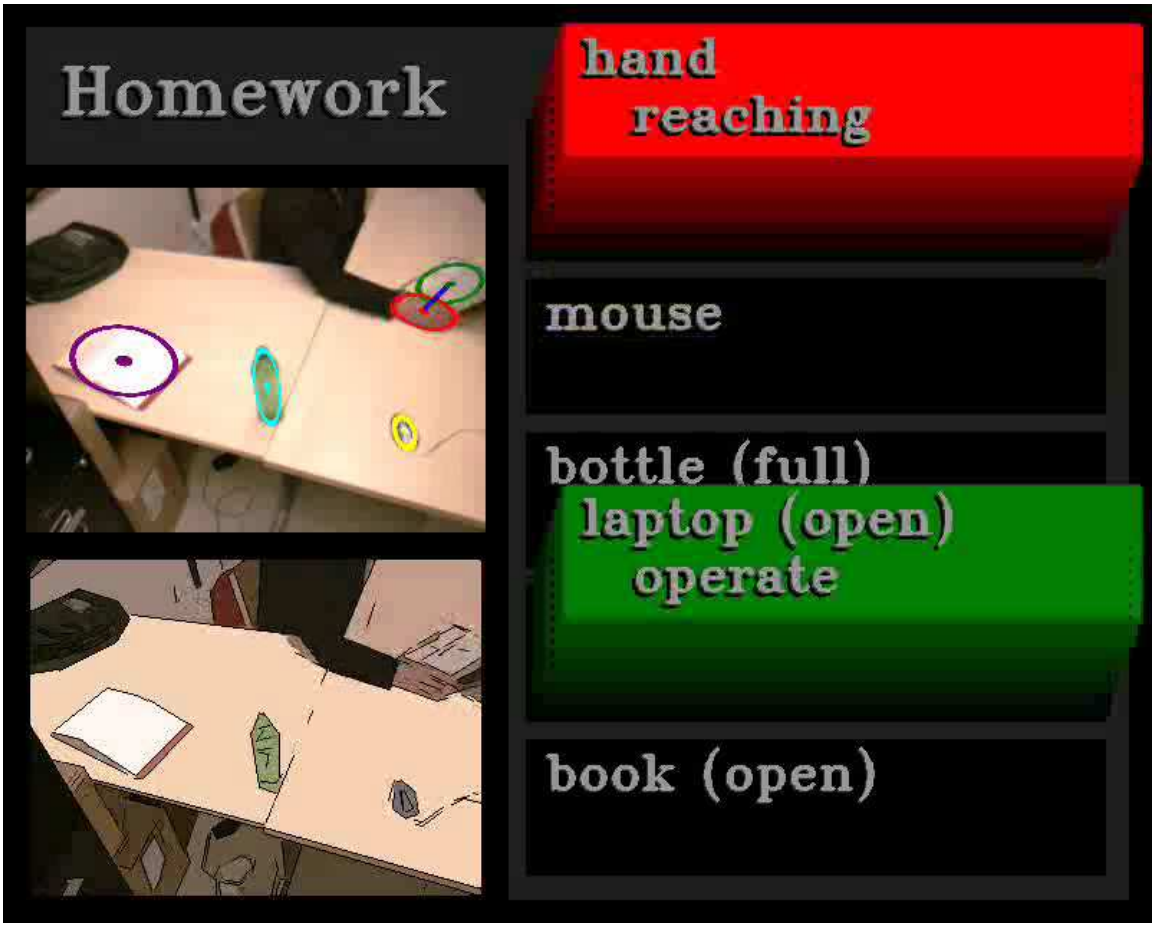


Figure IV-2: Screenshot taken during a tabletop scenario trial.

Figure IV-2 shows the system during its operation. The top-left image shows the robot's point of view, with detected regions outlined using ellipses. The blue line between the hand and laptop suggest that an "interaction" has been detected between those two objects. The lower-left image displays the results of our segmentation algorithm. The list on the right shows recognized objects that are detected within the robot's field of view. If objects are found that are part of an interaction, those names are shown to rise out of the

list. The predicted action is shown below the highlighted object and suggests that the user will “operate” the laptop.

In our videos, the output resembles the screenshot shown in **Figure IV-2**. However, to provide more compact illustrations, subsequent images in this paper have been edited to show only the system’s view of the environment and the name of the current object of interaction. This configuration allows us to fit six screenshots in a single image.

IV.i.1.1 Homework Scenario

In the ‘*Homework*’ scenario the robot watched a table as a person reached toward and moved four homework related objects: a book; a laptop; a mouse; and a bottle. For two of the objects, changes in the object’s state could be detected (the book and laptop can be “open” or “closed”), while the remaining two objects were found in only one state. The recognized objects, states, and action labels are summarized in **Table 3**. The corresponding screenshots are shown in **Figure IV-3**.

Table 3: Programmed reactions to recognized objects in the '*Homework*' scenario.

Numbers and colors in the left column correspond to the relevant frames and objects from **Figure IV-3**.

	Object (State)	Predicted Action
	Hand	
1	Book (Closed)	Open
2	Book (Open)	Read
3	Laptop (Closed)	Boot
4	Laptop (Open)	Operate
5	Mouse	Click
6	Bottle	Drink

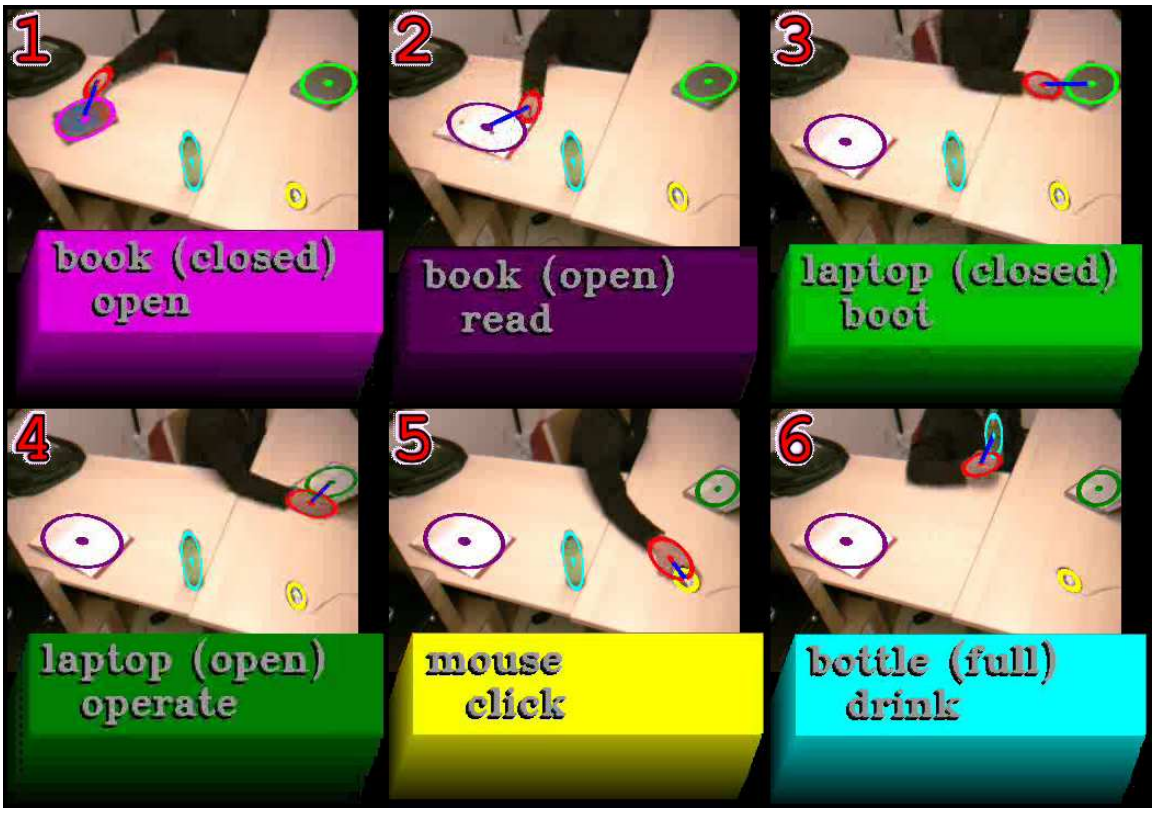


Figure IV-3: Screenshots taken from our '*Homework*' scenario.

Figure IV-3 shows the state of the system at six points during the scenario: 1) User reaches toward a book. 2) User opens the book. 3) User Reaches toward a laptop. 4) User opens the laptop. 5) User moves a mouse. 6) User takes a drink from the bottle.

IV.i.1.2 Eating Scenario

In the '*Eating*' scenario, the robot watched a table as a person reached toward and moved three food related objects: a pitcher; a glass; and a plate. For two of the objects, changes in the object's state could be detected (the glass and plate can be "empty" or "full"), while the remaining object was found in only one state. The recognized objects, states, and action labels are summarized in **Table 4**. The corresponding screenshots are shown in **Figure IV-4**.

Table 4: Programmed reactions to recognized objects in the '*Eating*' scenario.
Numbers and colors in the left column correspond to the relevant frames and objects from **Figure IV-4**.

	Object (State)	Predicted Action
	Hand	
1	Glass (Empty)	(none)
4	Glass (Full)	Drink
2,3	Pitcher	Pour
5	Plate (Full)	Eat
6	Plate (Empty)	Throw-Away

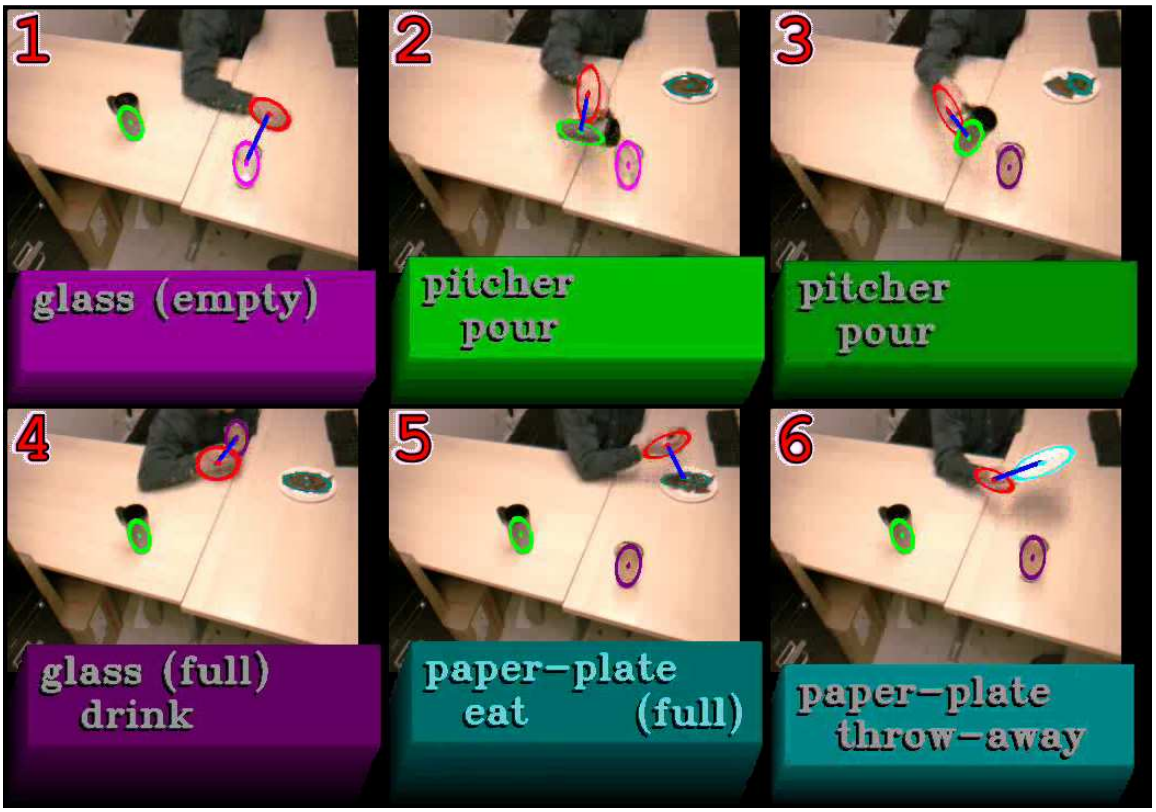


Figure IV-4: Screenshots taken from our '*Eating*' scenario.

Figure IV-4 shows the state of the system at six points during the scenario: 1) User has just placed an empty glass. 2) User begins pouring the pitcher into the glass. 3) User has finished filling the glass. 4) User drinks from the glass. 5) User eats from the plate. 6) User picks up empty plate.

IV.i.1.3 Fire Scenario

In the '*Fire*' scenario, the robot watched a couch as a person reached toward and moved two objects (a bag of chips, and a fire extinguisher). Both objects could be found in only one state. The recognized objects, states, and action labels are summarized in **Table 5**. The corresponding screenshots are shown in **Figure IV-5**.

Table 5: Programmed reactions to recognized objects in the '*Fire*' scenario.

Numbers and colors in the left column correspond to the relevant frames and objects from **Figure IV-5**.

	Object (State)	Predicted Action
	Hand	
4,5	Extinguisher	Extinguish
1,6	Snack	Eat

Figure IV-5: Screenshots taken from our '*Fire*' scenario.

Figure IV-5 shows the state of the system at six points during the scenario: 1) User reaches into the bag of chips. 2) User eats the chip. 3) User notices that a trashcan fire has started. 4) User reaches for the fire extinguisher. 5) User extinguishes the fire. 6) User reaches into the bag of chips.

IV.i.2 Robot Interaction

The examples in this section are intended to show how our architecture could be used by a service robot to visually monitor human actions and provide assistance when needed. In our demonstrations, we used the Nao robot, which is a small bipedal robot with an onboard camera and processor (**Figure IV-6**). The robot stood on the table in front of a user and watched that person as they engaged in homework and eating related behaviors. When the robot identified known objects or actions, the robot provided commentary and assistance.

Figure IV-6: An image of the Nao Robot crouching on a table.

IV.i.2.1 Homework Scenario

In the '*Homework*' scenario, the robot watched a table as a person reached toward and moved three homework related objects: a book; a laptop; and a paper. For two of the objects, changes in the object's state could be detected (the book and laptop can be "open" or "closed"), while the remaining object was found in only one state. The recognized objects, states, and robot responses are summarized in **Table 6**. The corresponding screenshots are shown in **Figure IV-7**.

Table 6: Programmed responses to recognized objects in the '*Homework*' scenario.
Numbers and colors in the left column correspond to the relevant frames and objects from Figure IV-7.

	Object (State)	Robot Response
	Hand	
1	Book (Closed)	"Hey, I know that book. It is about robots."
2	Book (Open)	"Are you going to read for a long time?... OK I will be quiet then."
3	Laptop (Closed)	"I see you need to start your computer."
4	Laptop (Open)	"I will get some rest while you are typing."
	Laptop (Closed)	"OH, you are done working... Excellent."
5 6	Paper	"Do you need a pen for your writing?" (Hands user a pen) "Here you go."

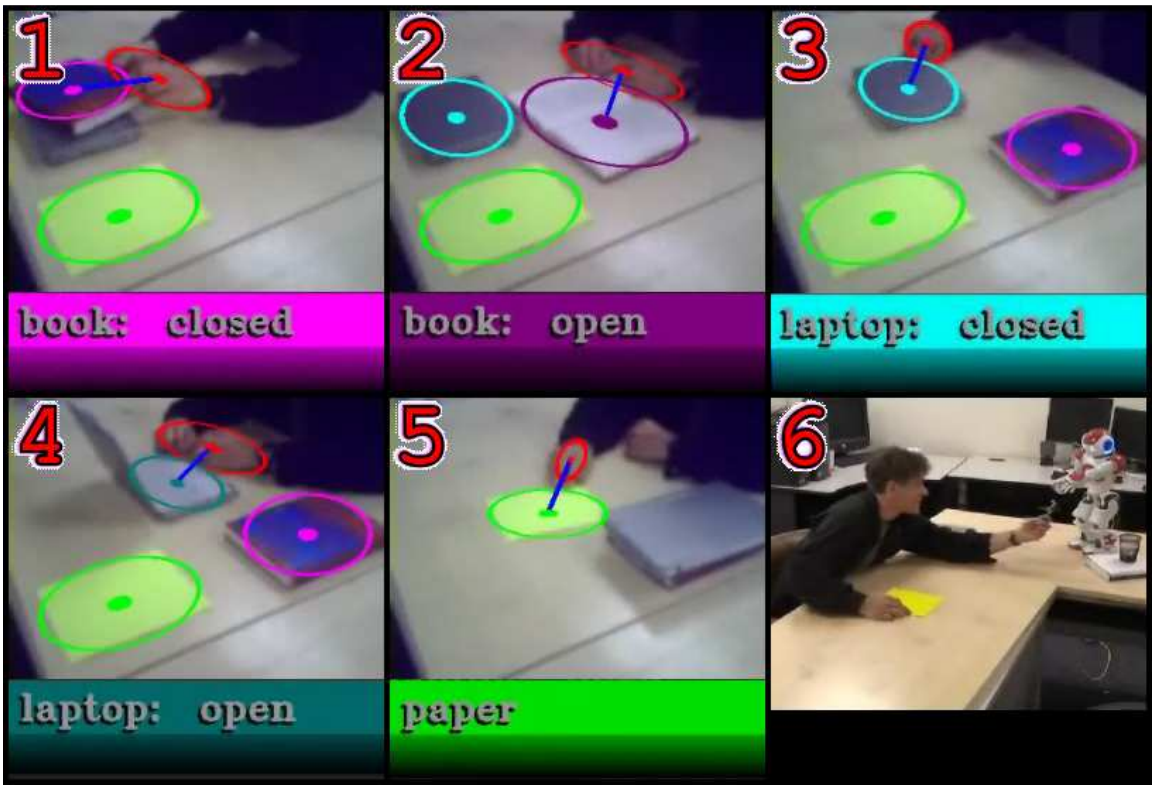


Figure IV-7: Screenshots taken from our Nao '*Homework*' scenario.

Figure IV-7 shows the state of the system at six points during the scenario: 1) User reaches for a book. 2) User opens the book. 3) User reaches for a laptop. 4) User opens the laptop. 5) User reaches for a paper. 6) Robot recognizes that the user has reached for the paper and hands user a pen.

IV.i.2.2 Eating Scenario

In the '*Eating*' scenario, the robot watched a table as a person reached toward and moved three food related objects: a bottle; a glass; and a plate. For two of the objects, changes in the object's state could be detected (the glass and plate can be "empty" or "full"), while the remaining object was found in only one state. The recognized objects, states, and robot responses are summarized in **Table 7**. The corresponding screenshots are shown in **Figure IV-8**.

Table 7: Programmed responses to Recognized objects in the '*Eating*' scenario.

Numbers and colors correspond to the relevant frames and objects from **Figure IV-8**.

	Object (State)	Robot Response
	Hand	
	Bottle	"Do you have a glass for your drink?"
	Glass (Empty)	(none)
4	Glass (Full)	"Be careful. You do not want to spill on the table."
	Glass (Full)	"Soda makes cavities."
1 2	Plate (Full)	"I see it is time for lunch... Would you like a fork?" (Hands user a Fork) "Here you go."
	Plate (Full)	"That food you are eating looks yummy."
5 6	Plate (Empty)	"Do you want me to throw away the plate?" (Takes plate from user, turns, walks, and drops it off the table)

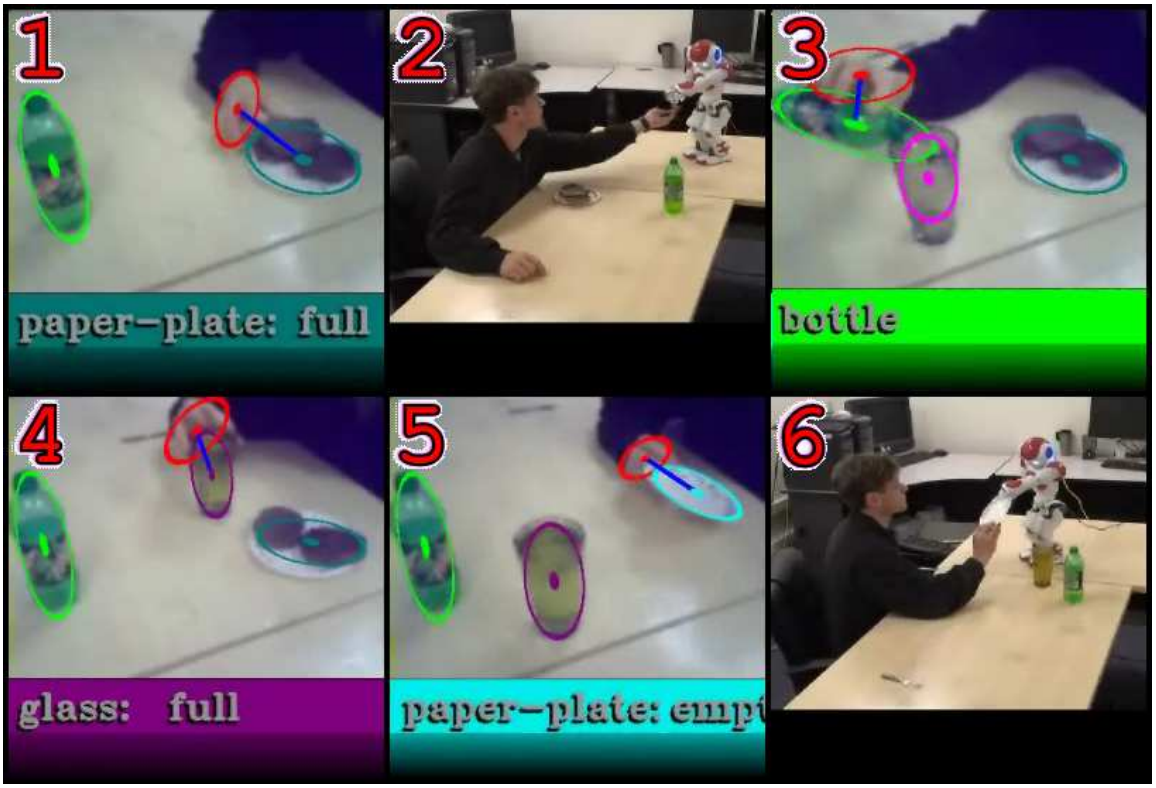


Figure IV-8: Screenshots taken from our '*Eating*' scenario.

Figure IV-8 shows the state of the system at six points during the scenario: 1) User reaches for a full plate. 2) Robot recognizes that the user has reached for a full plate and

hands user a fork. 3) User pours a bottle into a glass. 4) User drinks from the glass. 5) User reaches for an empty plate. 6) The robot recognizes that the user has reached for the empty plate and takes the plate from the user.

IV.i.3 Robot Pursuit

In these examples, the surveillance system was designed to recognize when people interacted with luggage in suspicious ways. If the system identified a suspicious event, it would dispatch a chase robot to pursue the perpetrator. Our specific scenarios involved the detection of theft activities and when a person might be abandoning a bag. For both surveillance and chase, we used the robot shown in **Figure IV-9**.

Figure IV-9: The robot used for activity detection and chase.

Figure IV-9 shows the Pioneer 3DX mobile robot equipped with a camera and SICK laser rangefinder running the Player/Stage/Gazebo robot device interface (Gerkey, Vaughan, & Howard, 2003).

IV.i.3.1 Theft Scenario

In the first part of the '*Theft*' scenario, the robot watched a room as a person entered and set his bag on the table. As long the bag remained within control of the original person, the system would passively monitor the room. If, however, an unknown person acquired the bag, the system would signal that a theft has occurred (**Figure IV-10**).

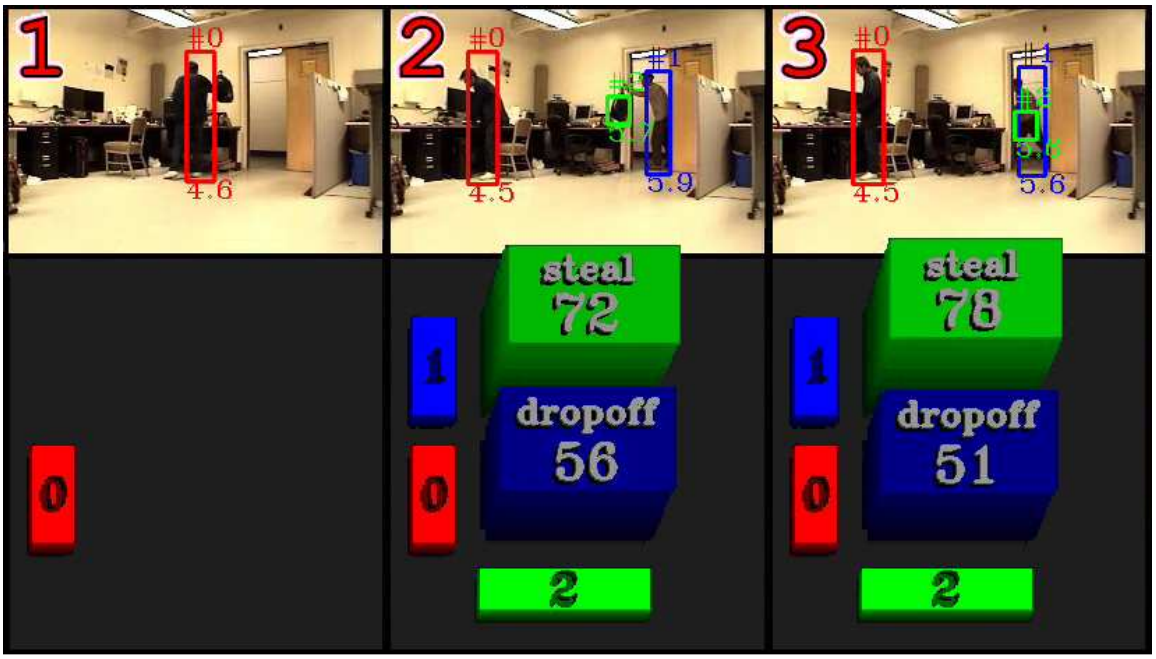


Figure IV-10: Screenshots taken from our '*Theft*' scenario.

Figure IV-10 shows the state of the system at six points during the scenario: 1) User places a bag on the table. 2) A second person enters the room and grasps the bag. 3) The second person takes the bag out of the room. The matrix on the bottom shows the interactions between the bag (green-2) and each of the two people (red-0 & blue-1). The system has correctly identified that person '0' has dropped off the bag ('*drop-off*'), and has created an alert after person '1' has taken possession ('*steal*').

In the second part of the '*Theft*' scenario, a patrol robot was summoned to pursue the suspect that stole the bag (**Figure IV-11)**. Once the robot identified the person, it

displayed a rectangle around the target and updated the person's location on the map. If the person was not currently in view, the robot drove toward the person's last known location. Otherwise, the robot drove toward the person. The chase continued until the person exited through a door.

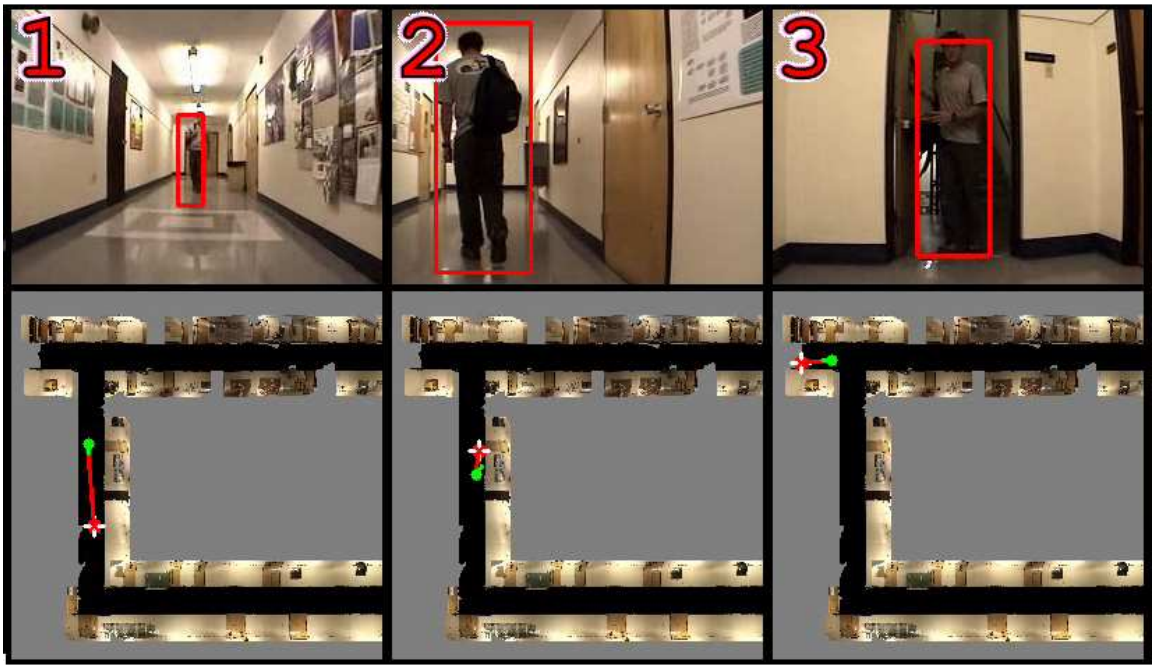


Figure IV-11: Screenshots taken from our '*Theft*' chase sequence.

Figure IV-11 shows the state of the system at six points during the scenario: 1) The thief has just exited a room with the stolen bag and the system summons the nearby patrolling robot to that location. The robot identifies the person in the hall as the suspect and begins updating the person's position on the map as the person walks toward the robot. 2) The suspect has just passed the robot and the robot has turned to follow. 3) The suspect heads for the nearest exit and the robot, being unable to open the door, stops at that location.

IV.i.3.2 Abandoned Bag Scenario

In the first part of the '*Abandoned Bag*' scenario, the robot watched the hallway as a person entered and set his bag on the floor. As long the bag remained within control

of the person, the system would passively monitor the hall. If, however, the person left the bag, the system would signal that an '*abandoned bag*' has occurred (**Figure IV-12)**.

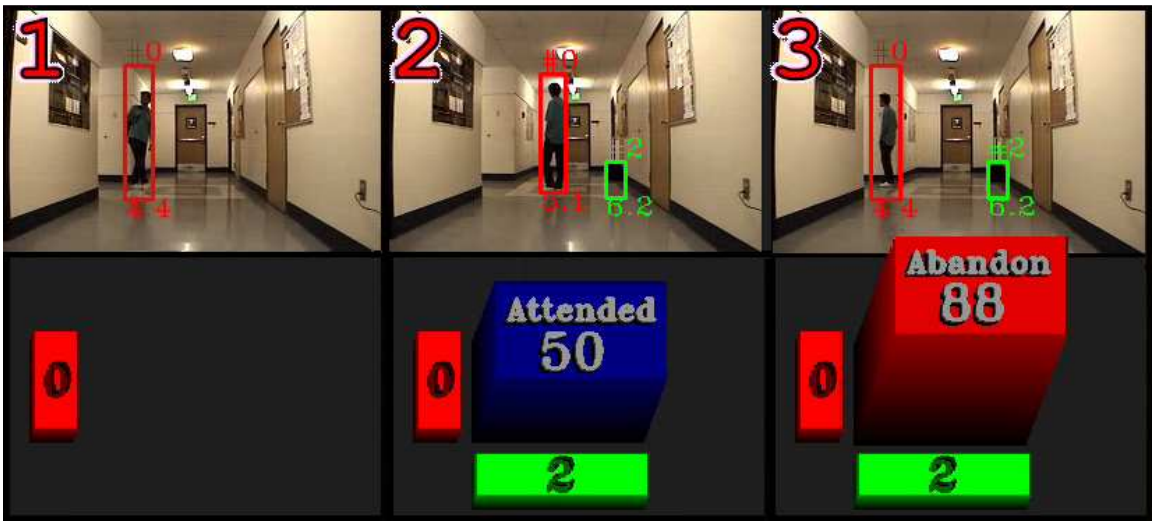


Figure IV-12: Screenshots taken from our '*Abandoned Bag*' scenario.

Figure IV-12 shows the state of the system at six points during the scenario: 1) User the robot's field of view. 2) The person places the bag on the floor and remains close to that location. 3) The person begins to walk away from the bag. The matrix on the bottom shows the interactions between the bag (green-2) and the person (red-0). When the person is close to the bag, the system correctly identifies that the bag is '*Attended*'. When the person begins to walk away from the bag, the system correctly identifies that the bag has been '*abandoned*'.

In the second part of the '*Abandoned Bag*' scenario, the robot that was watching the hall initiated pursuit of the person who abandoned the bag (**Figure IV-13)**. If the person was not currently in view, the robot drove toward the person's last known location. Otherwise, the robot drove toward the person. The chase continued until the person exited through a door.

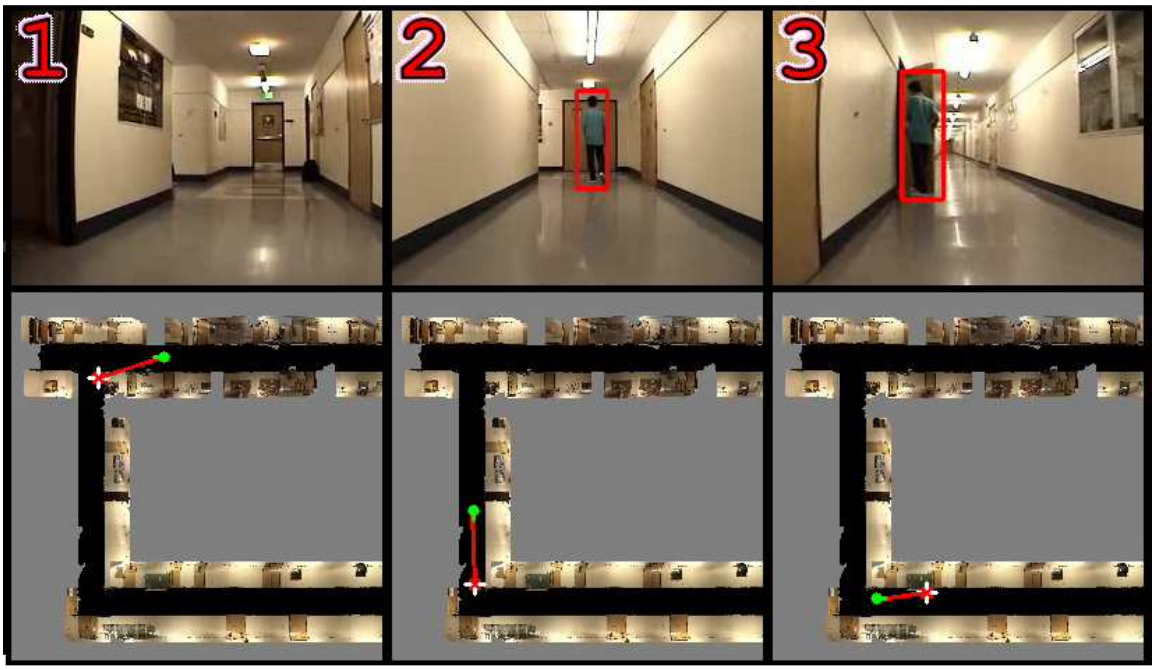


Figure IV-13: Screenshots taken from our '*Abandoned Bag*' chase sequence.

Figure IV-13 shows the state of the system at six points during the scenario: 1) A person has just abandoned a bag and walked out of the robot's field of view. The robot records the person's last know location to the map and begins to drive to that location. 2) Once the person is in the robot's field of view, it draws a rectangle around the suspect and updates the waypoint during the pursuit. 3) The suspect heads through an unlocked door and the robot, being unable to open the door, stops at that location.

IV.i.4 Automated Multi-Robot Dispatch

There are many applications where a single robot would be insufficient to complete a desired task. The task may require information or equipment that cannot be carried by a single robot, or the task may be such that multiple robots must work together to complete it. In a two-robot scenario, it might be possible for a single human user to select the properly equipped robots, direct the robots to the job's location, and guide them through the task. However, as the number of robots and the number of available tools

increases, the problem of selecting and directing robots becomes increasingly impractical unless the robots are capable of handling part of the job for themselves. Even a limited amount of autonomy can still greatly simplify the organization of a robot team. If robots are endowed with awareness of their own capabilities and can communicate with other robots, then a group of robots could decide for itself the best way to distribute a task. Furthermore, if robots are provided with knowledge of their environment and of their current position, then they could direct themselves to the tasks location.

This scenario demonstrates the extent that our robots can autonomously localize, self-organize, and respond to events in an optimal way. Although our architecture is capable of autonomously directing robots to emergency situations (shown in the previous set of demonstrations), we chose to use a GUI to allow us to control the occurrence of multiple simultaneous events. The system was validated using four Pioneer 3DX robots in the corridors of the second floor of the SEM building at UNR. Each robot was provided a copy of the floor map shown in **Figure III-35**. During the experiment a user would select a series of red, green, and red-green waypoints at different locations in the environment by applying mouse-clicks to the GUI. The system would then broadcast the waypoint's position to all robots within range. Each robot that received the information would determine if it was available to handle the task (not currently engaged in another task) and if it was equipped to handle the task (their color matched the color of the requested waypoint). If the robot was able to accept the task, it would broadcast a signal to the other robots to indicate its intent and the required cost of its actions (in this case, cost was a function of distance from the robot to the waypoint). If another robot could

accomplish the task more efficiently, then that robot would broadcast its own bid to its teammates. Once the robots discovered their best candidate, that candidate would communicate the information to the GUI and would begin executing the task.

At the start of the experiment all robots were localized by placing them on a predefined starting point in the environment. Then, over the period of 14 minutes, 11 different waypoints were issued to the robots. In every instance, the correct (correct color and closest distance) robots responded to the requested waypoints. All the robots made it to their waypoints successfully and no robot collisions occurred. Two screen shots of the GUI with superimposed images from video feeds are shown in **Figure IV-14**.

Figure IV-14: Screenshots from the multi-robot experiment.

Figure IV-14 shows the state of the system at two points during the scenario: Robots appear as red and green circles. Waypoints appear as circles centered on white crosses. Red, green, and red-green lines are drawn between robots and their respective waypoint. Each red waypoint (i.e. 3) will require the attention of one red robot, each green waypoint (i.e. 4 & 5) will require the attention of one green robot, and each red-green waypoint (i.e. 1 & 2) will require the attention of one red and one green robot. The images on the East, West, and South portion of the map are captured from video cameras placed in the hallway. The image on the North portion of the map was captured from a camera that was mounted on one of the green robots (i.e. 6).

Chapter V Conclusion

Despite the fact that modern surveillance systems have become increasingly sophisticated in terms of video quality, and their ability to detect basic intrusion events, they are still limited by the extent that humans can monitor the video feeds. Consequently, there is significant motivation for researchers to produce intelligent monitoring systems that automatically detects and responds to various events. This dissertation discussed several novel algorithms that could be useful for producing such a system. Our system was designed from the bottom up to use a novel set of image processing algorithms. We described a unique dual-pass region detection algorithm that could segment an image while simultaneously producing a hierarchy of MSER detections. Our region-set offered several unique advantages including fast operation, dense coverage, and temporal stability. It also facilitated efficient computation of additional features including line segments, corners, contours, color histograms, and others. Our region tracking could track all image regions through every frame of the video feed, in real time. This was accomplished by matching sub-regions around each region's perimeter to those in subsequent frames. Foreground segmentation was achieved by identifying regions that display consistent motion that differs from the background. Regions were modeled and classified using simple color histograms compiled from the contained pixel-clusters. Simple pre-defined interactions were identified between people and objects using proximity measurements and relative motion vectors. If the interaction

information suggests that a theft or other threat might be occurring, the system was able to dispatch one or more automated robots to investigate or pursue an individual.

We found that our system offered certain advantages over traditional systems, including the ability to detect, track, and identify highly deformable object from non-stationary cameras, in real time using a standard laptop. We successfully tested our system in a number of real-world activity-recognition scenarios.

References

Bahadori, S., Grisetti, G., Iocchi, L., Leone, G., & Nardi, d. (2005). Real-time tracking of multiple people through stereo vision. *Proc. of IEE International Workshop on Intelligent Environments.*

Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition, 13(2),* 111-122.

Baumberg, A., & Hogg, D. C. (1996). Learning deformable models for tracking the human body. *Motion-Based Recognition,* (pp. 39-60).

Bay, H., Ess, A., Tuytelaars, T., & Gool, L. (2008). SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU), 110(3),* 346-359.

Beymer, D. (1991). Finding junctions using the image gradient. *International Conference Computer Vision and Pattern Recognition,* (pp. 720-721).

Beymer, D., & Konolige, K. (1999). Real-time tracking of multiple people using stereo. *SRI VSAM Project: Extra Sets of Eyes.*

Bouman, C. A., Shapiro, M., Cook, G. W., Atkins, C. B., & Cheng, H. (1997). Cluster: An unsupervised algorithm for modeling Gaussian mixtures.

Bryant, B. D., & Miikkulainen, R. (2003). Neuroevolution for Adaptive Teams. *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003),* (pp. 2194-2201). Piscataway, NJ.

Buluswar, S. D., & Draper, B. A. (1998). Color Recognition in Outdoor Images. *ICCV '98 Proceedings of the Sixth International Conference on Computer Vision,* (pp. 171-177).

Buluswar, S., & Draper, D. (1998). Color recognition in outdoor scenes by non-parametric learning. *International Conference on Computer Visoin.*

Canny, J. (1986). A Computational Approach To Edge Detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*(8(6)), 679-698.

Carnegie, D. A., Prakash, A., Chitty, C., & Guy, B. (2004). A human-like semi-autonomous Mobile Security Robot. *2nd International Conference on Autonomous Robots and Agents.*

Cheung, G., Baker, S., & Kanade, T. (2003). Shape-from-silhouette for articulated objects and its use for human body kinematics estimation and motion capture.

Chum, O., & Matas, J. (2006). Geometric Hashing with Local Afine Frames. *CVPR.*

Comaniciu, D., Ramesh, V., & Andmeer, P. (2003). Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 25,* 564-575.

Corso, J. J., & Hager, G. D. (2005). Coherent regions for concise and stable image description. *CVPR.*

Donoser, M., & Bischof, H. (2006). Efficient Maximally Stable Extremal Region (MSER) Tracking. *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR), 1,* 553-560.

Duchenne, O., Laptev, I., Sivic, J., Bach, F., & Ponce, J. (2009). Automatic annotation of human actions in video. *Proceedings of the International Conference On Computer Vision (ICCV09),* (pp. 1-8). Kyoto, Japan.

Duda, R. O., & Hart, P. E. (1972). Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Comm. ACM, 15*, 11-15.

Duy-Nguyen, T., Wei-Chao, C., Natasha, G., & Kari, P. (2009). SURFTrac: Efficient Tracking and Continuous Object Recognition using Local Feature Descriptors. *CVPR.*

Elgammal, A., Harwood, D., & Davis, L. (2000). Non-parametric model for background subtraction. *European Conference on Computer Vision (ECCV), 1843*, pp. 751–767.

Eng, H. L., Toh, K. A., Kam, A. H., Wang, J., & Yau, W. Y. (2003). An automatic drowning detection surveillance system for challenging outdoor pool environments. *Proceedings of the Ninth IEEE International Conference on Computer Vision*, *2*. Washington, DC, USA.

Espina, M., Grech, R., Jager, D., Remagnino, P., Iocchi, L., Marchetti, L., et al. (2011). Multi-Robot Teams for Environmental Monitoring. *Innovations in Defence Support Systems – Intelligent Paradigms in Security, Springer-Verlag*, 183-209.

Evans, N. W. (2001). Follow the bouncing ball: The technical innovation of max fleischer. *Digital Media FX - The Power of Imagination*.

Everett, H., & Gage, D. W. (1999). From Laboratory to Warehouse: Security Robots Meet the Real World. *International Journal of Robotics Research, Special Issueon Field and Service Robotics, 18(7)*, 760-768.

Feris, R., Tian, Y. L., & Hampapur, A. (2007). Capturing people in surveillance video.

Fieguth, P., & Terzopoulos, D. (1997). Color-based tracking of heads and other mobile objects at video frame rates. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 21-27).

Forssen, P. (2007). Maximally stable colour regions for recognition and matching. *CVPR.*

Forssen, P., & Lowe, D. G. (2007). Shape Descriptors for Maximally Stable Extremal Regions. *International Conference on Computer Vision (ICCV).*

Gerkey, B., Vaughan, R. T., & Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. *Proc., the 11th International Conference on Advanced Robotics*, (pp. 317-323).

Gomez, F., & Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuroevolution. *Proceedings of the 16th International Joint Conference on Artificial Intelligence.*

Grewe, L., & Kak, A. (1995). Interactive learning of a multi-attribute hash table classifier for fast object recognition. *Computer Vision and Image Understanding, 61(3)*, 387-416.

Harris, C., & Stephens, M. (1988). A combined corner and edge detector. *Alvey Vision Conference*, (pp. 147-151).

Howard, A. (2004). Simple mapping utilities (pmap). University of Southern California. Retrieved from http://www-robotics.usc.edu/~ahoward/pmap/index.html

Ivanov, Y., & Boblic, A. (2000). Recognition of visual activities and interactions by stochastic parsing. *IEEE Trans. Pattern Anal. Machine Intelligence, 22*, 852-872.

Junejo, I. N., Dexter, E., Laptev, I., & Perez, P. (2008). Cross-view action recognition from temporal self-similarities. *ECCV.*

Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active Contour Models. *International Journal of Computer Vision, 1*.

Kelley, R., King, C., Ambardekar, A., Nicolescu, M., Nicolescu, M., & Tavakkoli, A. (2010). Integrating Context into Intent Recognition Systems. *Proceedings of the International Conference on Informatics in Control, Automation and Robotics*, (pp. 315-320). Funchal, Madeira, Portugal.

Kelley, R., King, C., Tavakkoli, A., Nicolescu, M., Nicolescu, M., & Bebis, G. (2008). An Architecture for Understanding Intent Using a Novel Hidden Markov Formulation. *International Journal of Humanoid Robotics - Special Issue on Cognitive Humanoid Robots*, 203-224.

Kelley, R., King, C., Tavakkoli, A., Nicolescu, M., Nicolescu, M., & Bebis, G. (2008). An Architecture for Understanding Intent Using a Novel Hidden Markov Formulation. *International Journal of Humanoid Robotics - Special Issue on Cognitive Humanoid Robots, 5(2)*, 203-224.

Kelley, R., Tavakkoli, A., King, C., Ambardekar, A., Nicolescu, M., & Nicolescu, M. (2012). Context-Based Bayesian Intent Recognition. *IEEE Transactions on Autonomous Mental Development - Special Issue on Biologically-Inspired Human-Robot Interactions, 4(3)*, 215-225.

Kelley, R., Tavakkoli, A., King, C., Ambardekar, A., Wigand, L., Nicolescu, M., et al. (2013). Intent Recognition for Human-Robot Interaction. *Plan, Activity, and Intent Recognition, Elsevier*.

Kelley, R., Tavakkoli, A., King, C., Nicolescu, M., & Nicolescu, M. (2010). Understanding Activities and Intentions for Human-Robot Interaction. *Human-Robot Interaction*, 288-305.

Kelley, R., Tavakkoli, A., King, C., Nicolescu, M., Nicolescu, M., & Bebis, G. (2008). Understanding Human Intentions via Hidden Markov Models in Autonomous Mobile Robots. *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, (pp. 367-374). Amsterdam, Netherlands.

King, C., Palathingal, X., Nicolescu, M., & Nicolescu, M. (2007). A Vision-Based Architecture for Long-Term Human-Robot Interaction. *Proceedings of the International Conference on Human-Computer Interaction*, (pp. 1-6). Chamonix, France.

King, C., Palathingal, X., Nicolescu, M., & Nicolescu, M. (2007). A Control Architecture for Long-Term Autonomy of Robotic Assistants. *Proceedings of the International Symposium on Visual Computing*, (pp. 375-384). Lake Tahoe, Nevada.

King, C., Palathingal, X., Nicolescu, M., & Nicolescu, M. (2009). A Flexible Control Architecture for Extended Autonomy of Robotic Assistants. *Journal of Physical Agents, 3(2)*, 59-69.

King, C., Valera, M., Grech, R., Mullen, R., Remagnino, P., Iocchi, L., et al. (2012). Multi-Robot and Multi-Camera Patrolling. *Handbook on Soft Computing for Video Surveillance*, 255–286.

Langridge, D. (1982). Curve encoding and detection of discontinuities. *Computer Graphics Image Processing, 20*, 58-71.

Li, J., & Wang, J. Z. (2003). Automatic Linguistic Indexing of Pictures by a statistical modeling approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1075-1088 .

Lindeberg, T. (1988). Feature detectionwith automatic scale selection. *International Journal of Computer Vision, 30(2)*, 79-116.

Loss, L., Bebis, G., Nicolescu, M., & Skurikhin, A. (2009). An Iterative Multi-Scale Tensor Voting Scheme for Perceptual Grouping of Natural Shapes in Cluttered Backgrounds. *Computer Vision and Image Understanding, Elsevier*, 126-149.

Lowe, D. (1999). Object recognition from local scale-invariant features. *Proceedings of the 7th International Conference on Computer Vision*, (pp. 1150–1157). Kerkyra, Greece.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal on Computer Vision, 60(2)*, 91-110.

Makihara, Y., Sagawa, R., Mukaigawa, Y., & Echigo, T. (2006). Gait Recognition using a View Transformation. *ECCV*.

Matas, J., Chum, O., Urban, M., & Pajdla, T. (2002). Robust wide-baseline stereo from maximally stable extremal regions. *Proceedings of the British Machine Vision Conference*, (pp. 384–393). Cardiff, UK.

McKenna, S. J., Jabri, S., Duric, Z., Wechsler, H., & Rosenfeld, A. (2000). Tracking groups of people. *Computer Vision and Image Understanding*, 42-56.

McLauchlan, P., & Rahimi, A. (1992). Horatio: a computer vision and image processing library.

Mikolajczyk, K., & Schmid, C. (2002). An affine invariant interest point detector. *Proceedings of the 7th European Conference on Computer Vision.* Copenhagen, Denmark.

Mikolajczyk, K., & Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1615-1630.

Mikolajczyk, M., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., et al. (2005). A Comparison of Affine Region Detectors. *International Journal of Computer Vision, 65(1/2)*, 43-72.

Mindru, F., Tuytelaars, T., Gool, L. V., & Moons, T. (2004). Moment invariants for recognition under changing viewpoint and illumination. *Computer Vision and Image Understanding, 94*, 3-27.

Morel, J. M., Petro, A. B., & Sbert, C. (2009). Fast implementation of color constancy algorithms. *Proc. SPIE.*

Nakhaeinia, D., Tang, S. H., Mohd Noor, S. B., & Motlagh, O. (2011). A review of control architectures for autonomous. *International Journal of the Physical Sciences, 6(2)*, 169-174.

Niebles, J. C., Wang, H., & Fei-Fei, L. (2008). Unsupervised learning of human action categories using spatial-temporal words. *International Journal of Computer Vision, 79(3)*, 299–318.

Ning, H. Z., Wang, L., Hu, W. M., & Tan, T. N. (2001). Articulated model based people tracking using motion models. *Proceedings International Conference of Multi-Model Interfaces*, (pp. 115-120).

Nowak, E., Jurie, F., & Triggs, B. (2006). Sampling strategies for bag-of-features image classification. *Proceedings of the European Conference on Computer Vision*, (pp. 490-503).

Obdrzalek, S., & Matas, J. (2006). Object Recognition Using Local Affine Frames. *Toward Category-Level Object Recognition, LNCS*, 83-104.

Okuma, K., Taleghani, A., Freitas, N. D., Little, J. J., & Lowe, D. G. (2004). A boosted particle filter: Multitarget detection and tracking.

Oliver, N., Rosario, B., & Pentland, A. (2000). A Bayesian computer vision system for modeling human interactions. *IEEE Trans. Pattern Analysis and Machine Intelligence, 22*, 831-843.

Pavlidis, T. (1982). Bilevel Pictures. In *Algorithms for Graphics and Image Processing.* (pp. 143-146).

Piccardi, M. (2004). Background subtraction techniques: A review. *IEEE International Conference on Systems, Man and Cybernetics*.

Pollard, S., Porrill, J., Thacker, N., & Bromiley, P. (1987). TINA: Image Analysis Toolkit. *AIVRU, University of Sheffield.*

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*(1(3)), 244-256.

Ranzato, M., Huang, F. J., Boureau, Y. L., & LeCun, Y. (2007). Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 1-8).

Riemenschneider, H., Donoser, M., & Bischof, H. (2008). Robust Online Object Learning and Recognition by MSER Tracking. *Computer Vision Winter Workshop.* Moravske Toplice, Slovenia.

Rothwell, C. A., Zisserman, A., Forsyth, D. A., & Mundy, J. L. (1992). Canonical Frames for Planar Object Recognition. *ECCV*, (pp. 757-772).

Sangi, P., Heikkila, J., & Silven, O. (2001). Extracting motion components from image sequences using particle filters. *In Scandinavian Conference on Image Analysis.* Bergen, Norway.

Saptharishi, M., Oliver, C. S., Diehl, C. P., Bhat, K. S., Dozlan, J. M., Trebi-Ollennu, A., et al. (2002). Distributed Surveillance and Reconnaissance Using Multiple Autonomous ATVs: CyberScout. *IEEE Transactions on Robotics and Automation: Special Issue on Multi-Robot Systems*.

Schlegel, C., Illmann, J., Jaberg, H., Schuster, M., & Worz, R. (1998). Visionbased person tracking with a mobile robot. *British Machine Vision Conference*, (pp. 418-427).

Schweitzer, H., Bell, J. W., & Wu, F. (2002). Very fast template matching. *European Conference on Computer Vision (ECCV)*, (pp. 358-372).

Sigal, L., Bhatia, S., Roth, S., Black, M. J., & Isardy, M. (2004). Tracking loose-limbed people.

Sillito, R. R., & Fisher, R. B. (2008). Semi-supervised learning for anomalous trajectory detection. *Proceedings BMVC*, (pp. 1035-1044).

Sivic, J., & Zisserman, A. (2006). Video Google: Efficient visual search of videos. *Toward Category-Level Object Recognition, 4170*, 127-144.

Stauffer, C., & Grimson, W. (1999). Adaptive background mixture models for real-time tracking. *Institute of electrical engineers (IEEE), CVPR, 2*, pp. 246–252.

Steffens, J., Elagin, E., & Neven, H. (1988). Person spotter-fast and robust system for human detection, tracking and recognition. *Proc. IEEE International Conference of Automatic Face and Gesture Recognition*, 516-521.

Stein, A., & Hebert, M. (2005). Incorporating Background Invariance into Feature-Based Object Recognition. *Application of Computer Vision.*

Swain, M., & Ballard, D. (1991). Indexing via color histograms. *In-tern. Journal of Computer Vision, 7*, 11-32.

Tao, H., Sawhney, H., & Kumar, R. (2002). Object tracking with bayesian estimation of dynamic layer representations. *IEEE Trans. Pattern Analysis and Machine Intelligence*.

Tavakkoli, A., Kelley, R., King, C., Nicolescu, M., Nicolescu, M., & Bebis, G. (2007). A Vision-Based Architecture for Intent Recognition. *Proceedings of the International Symposium on Visual Computing*, (pp. 173-182). Lake Tahoe, Nevada.

Tavakkoli, A., Kelley, R., King, C., Nicolescu, M., Nicolescu, M., & Bebis, G. (2008). A Visual Tracking Framework for Intent Recognition in Videos. *Proceedings of the International Symposium on Visual Computing*, (pp. 450-459). Las Vegas, Nevada.

Thrun, S. (2002). Robotic Mapping: A Survey.

Torr, P. H., & Zisserman, A. (1999). Feature based methods for structure and motion estimation. *Vision Algorithms: Theory and Practice* (pp. 278-295). Springer-Verlag.

Toyama, K., Krumm, J., Brumitt, B., & Meyers, B. (1999). Wallflower: Principles and practice of background maintenance. 255-261.

Tuytelaars, T., & Van Gool, L. (2000). Wide baseline stereo matching based on local, affinely invariant regions. *Proceedings of the 11th British Machine Vision Conference*, (pp. 412–425). Bristol, UK.

Tuytelaars, T., Van Gool, L., D'haene, L., & Koch, R. (1999). Matching of affinely invariant regions for visual servoing. *International Conference of Robotics and Automation (IRCA).*

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *IEEE Conference on Computer Vision and Pattern Recognition.*

Viola, P., Jones, M., & Snow, D. (2003). Detecting pedestrians using patterns of motion and appearance. *IEEE International Conference on Computer Vision (ICCV)*, (pp. 734-741).

Wren, C. R., Azarbayejani, A., Darrell, T., & Pentland, A. P. (1997). Pfinder: real-time tracking of the human body. *IEEE Trans. Pattern Anal. Machine Intell, 19*, 780-785.

Xiang, T., & Gong, S. (2006). Beyond tracking: Modelling activity and understanding behaviour. *International Journal of Computer Vison, 67(1)*, 21-51.

Yilmaz, A., Li, X., & Shah, M. (2004). Contour-Based Object Tracking with Occlusion Handling in Video Acquired Using Mobile Cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(11).*

Zhou, H., Yuan, Y., & Shi, C. (2009). Object tracking using SIFT features and mean shift. *Computer Vision and Image Understanding, 113,* 345-352.

Zhu, S., & Yuille, A. (1996). Region Competition: Unifying Snakes, Region Growing, and Bayes/MDL for Multiband Image Segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence, 18(9),* 884-900.