IMAGE SEGMENTATION AND OBJECT TRACKING FOR A MICRO AIR VEHICLE

By

TED L. BELSER II

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2006

ACKNOWLEDGMENTS

I thank Dr. Dapeng Wu for his role as my supervisory committee chair. I thank Drs. Michael Nechyba and Eric Schwartz for the semesters of challenging coursework, which no doubt increases the value of my degree. I thank the AVCAAF team for giving me access to their data and research papers. I thank the Intel Corp. for making available the Open Source Computer Vision (OpenCV) library.

I thank my family and friends for their support while attending the University of Florida. I thank Aimée Baum for being my best friend and for putting up with the late nights of work required to finish this thesis.

TABLE OF CONTENTS

page

LIST OF TABLES

Table page

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

IMAGE SEGMENTATION AND OBJECT TRACKING FOR
A MICRO AIR VEHICLE

By

Ted L. Belser II

May 2006

Chair: Dapeng Oliver Wu
Major Department: Electrical and Computer Engineering

This thesis describes a system that can perform object tracking in video produced by a camera mounted on a micro air vehicle (MAV). The goal of the system is to identify and track an object in full motion video while running in real-time on modest hardware (in this case a Pentium III running at 800Mhz with 512 MB RAM).

To achieve this goal, two vision processing algorithms are coupled. A graph-based segmentation algorithm is used to identify individual objects in the image by discriminating between regions of similar color and texture. A pyramidal implementation of the Lucas-Kanade feature tracker is used to track features in the video. Running at a lower frequency than the tracking algorithm, the segmentation algorithm labels the features according to the corresponding object. By tracking the labeled features, the Lucas-Kanade feature tracker tracks the objects in the video.

Analysis and experimentation show that the pyramidal implementation of the Lucas-Kanade is both efficient and robust. The system performance however is

dominated by the performance of the segmentation algorithm. The segmentation algorithm, while capable of meeting the functional requirements of the system, requires two to three times more processing power than the feature tracking algorithm requires.

The system described in this thesis is capable of meeting the requirements for object tracking on a MAV platform. The analysis suggests that the pyramidal implementation of the Lucas-Kanade is an essential component of the MAV platform due to its efficiency and robust performance. The analysis also suggests a direction for improvement. While the segmentation algorithm was able to fulfill the requirements, it did so at a high computational cost. One possible direction for future work is to improve the performance of the segmentation process.

CHAPTER 1
INTRODUCTION

Problem Definition

In its mission statement the AVCAAF (Active vision for Control of Agile Autonomous Flight) group at the University of Florida's Machine Intelligence Laboratory describes the potential missions of a MAV (Micro Air Vehicle). The potential missions include search and rescue, moving-target tracking, immediate bomb damage assessment, and identification and localization of interesting ground structures. To achieve this mission statement the MAV platform utilizes a number of instruments. As is evident in the group's name, the AVCAAF is focused on vision processing systems for flight vehicle control. The primary instrument for vision-based control is the video camera. The video camera coupled with a computer running sophisticated vision processing algorithms forms a versatile system capable of performing functions such as automatic attitude control, object recognition and object tracking. Examples of attitude control and object recognition applications are discussed in AVCAAF's research papers [1, 2, 3 and 4]. Object tracking however is not discussed in these papers and is the focus of this thesis.

To track an object it must first be identified. Object identification solutions are not trivial and can arguably be called a central problem in computer vision. The problem of what makes an object an object was once a question for philosophers; but in computer vision, it is a question for engineers. In many ways, engineers have not answered this question. Our systems are capable of identifying specific, narrowly defined classes of

objects, but there is no general solution to object identification. This paper does not attempt to solve this problem; however understanding this problem helps to put the problem of object identification and tracking into context.

In image processing, segmentation is the partitioning of a digital image into multiple regions according to a criterion. Segmentation can be applied to identify objects in a scene if a suitable segmentation criterion can be defined to identify objects of interest. In their paper "Intelligent Missions for MAVs: Visual Contexts for Control, Tracking and Recognition"[2] Todorovic and Nechyba discuss an object segmentation algorithm. While this algorithm performs object segmentation efficiently, it is computationally excessive to use the same algorithm for object tracking. Once an object is acquired, the tracking problem has constraints such as a spatial and temporal locality that reduce the complexity of the tracking problem. Furthermore, an object can be expected to maintain its shape and appearance in a sequence of frames. Given these constraints the problem is to design and implement an object tracking system that meets the following requirements:

- The system should locate objects of interest
- The system should track the object(s) of interest through sequential frames in video
- The system should run in real-time on standard hardware (Pentium III 800 MHz)

Approach

Three Processes to Achieve Object Tracking

The approach taken to solve this problem is to divide the tracking task into three processes. The first process identifies and enumerates objects in the image. The second process identifies significant features on each the objects. The third process is correspondence of each feature between adjacent frames in a video sequence. Object tracking is possible by defining which objects own which features and the tracking those

features over a sequence of frames. This method is described in the remaining chapters. First, each of the processes used are described in detail. Chapter 2 describes the segmentation process. Chapter 3 describes the feature extraction process and the feature tracking process. The system organization and implementation is discussed in Chapter 4. Chapter 5 is an evaluation of the system's performance including limitations of the system and how the parameters of the individual processes govern the performance of the system as a whole. Finally, in Chapter 6 suggestions for future research are proposed.

Assumptions

The segmentation, feature extraction and feature tracking processes do not need to run with equal effort. The feature tracking process should run frequently in order to accurately track features from frame to frame. While the segmentation and feature extraction processes may run less frequently. By making reasonable assumptions, the segmentation and feature extraction processes can occur at a frequency significantly less than the frame rate of the video sequence.

The essential mechanism of the feature tracking process is correspondence. Correspondence identifies the offset between a specific feature in two frames. These two frames may represent images from cameras that differ in time and/or space. In the case of a moving camera, the two frames differ in time and space. By using a correspondence mechanism, feature tracking is possible. By associating features with objects it is therefore possible to track an entire object. A few assumptions however must be made.

1. An object to be tracked has identifiable features.
2. These features can be tracked by a correspondence mechanism over a sequence of frames.
3. Each feature instance belongs to only one object during the period of time in which the tracking occurs.

CHAPTER 2
A GRAPH-BASED SEGMENTATION ALGORITHM

Functional and Performance Requirements for Segmentation

In [5] a graph-based image segmentation algorithm is defined. The segmentation algorithm is designed to meet the following requirements:

1. The algorithm should capture perceptually different regions of an image.
2. The algorithm should account for local as well as global information while partitioning the image regions.
3. The algorithm should be efficient, running in time nearly linear in the number of image pixels.

Graph-Based Segmentation

In Felzenszwalb and Huttenlocher [5] a graph-based approach is presented. An image is represented as an undirected graph $G = (V, E)$. V represents the vertices of the graph and corresponds one-to-one with the pixels in the image. E represents the edges between the pixels. In the actual implementation, E represents every adjacency in the four-connected adjacency of every pixel. Each edge in E is given a non-negative weight $w((v_i, v_j))$ that is a measure of the dissimilarity between the pixels belonging to vertices v_i and v_j. In the implementation, this difference is the distance between the color values of v_i and v_j in the RGB color space. The objective is to produce a segmentation S composed of components C. Each component is defined by a set of edges $E' \subseteq E$ between vertices representing pixels of low dissimilarity as defined by a comparison predicate.

The Comparison Predicate

Requirement 2 above describes a need to take into account global and local information when forming a component in the segmentation. The comparison predicate described by Felzenszwalb and Huttenlocher [5] is designed to meet this requirement. The predicate is designed to measure dissimilarity between components relative to the internal dissimilarity within each component. This technique is capable of identifying regions that have little internal dissimilarity while also identifying regions where there is great internal dissimilarity.

The predicate is defined in terms of two quantities, the *minimum internal distance* of two components C_1 and C_2 and the *difference* between components C_1 and C_2. The *minimum internal distance* captures the global information of the two components and forms the basis on which the predicate decides if the two components are actually different. The minimum internal difference is defined as

$$Int_{\min}(C_1, C_2) = \min\left(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)\right),$$

where the internal difference *Int* is defined as the largest weight in the minimum spanning tree of the component, $MST(C, E)$,

$$Int(C) = \max w(e) \text{ for } e \in MST(C, E)$$

The difference between the components C_1 and C_2 is the minimum weight in the edges connecting the two components,

$$Diff(C_1, C_2) = \min w((v_i, v_j)). \text{ for } v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E$$

By comparing the minimum internal difference of the components with the smaller of the two internal differences, a predicate $D(C_1, C_2)$ can be defined. If true, edge

$(v_i, v_j), v_i \in C_1, v_j \in C_2$ forms a boundary between components C_1 and C_2, otherwise C_1 and C_2 are the same component.

$D(C_1, C_2) =$ **True** if $Diff(C_1, C_2) > Int_{\min}(C_1, C_2)$; **False** otherwise

This formulation has only one parameter, *k*. The parameter *k* is coupled with the system by way of the threshold function τ. The threshold function $\tau(C)$ defines by how much the differences between the components must exceed the lesser of the two component internal differences. A higher value decreases the likelihood that two components will be declared different and therefore encourages larger components. The function τ also serves to minimize error due to small component sizes in the early stages of the computation. Specifically, it scales a constant *k* by the inverse of the component size:

$$\tau(C) = \frac{k}{|C|}$$

The parameter *k* determines granularity of the segmentation. Larger values of *k* produce larger components and therefore fewer components per image. Smaller values of *k* produce smaller components and therefore more components per image.

The Segmentation Algorithm

Felzenszwalb and Huttenlocher [5] apply the predicate function using the following algorithm:

4. Calculate the weight for all edges in *E*.
5. Sort *E* into non-decreasing order by edge weight resulting in the sequence $(o_1, \ldots, o_m)$
6. Assign all vertices in V one-to-one with components $(C_1, \ldots, C_r)$ so that each vertex belongs to its own component.
7. for *i* = 1 to *m* do the following

a. Let $o_i = (v_{i,} v_j), v_i \in C_i, v_j \in C_j$

b. if C_i and C_j are different components AND $D(C_i, C_j)$ is false, then merge C_i and C_j; otherwise do nothing.

8. Return $S = (C_1, \ldots, C_n)$ where n is the number of components remaining.

This algorithm runs in $O(m \log m)$ time where m denotes the number of edges in E.

Qualitative Analysis

Figure 2-1 shows an image and its segmentation using this technique. Regions of little dissimilarity, such as the dark region to the left of the butterfly, are properly segmented. More interesting is the left side of the leaf on which the butterfly sits. This region shows dissimilarity in the form of dark and light ridges formed by the veins in the leaf. This region is segmented as a single component despite large internal dissimilarities.

A B

Figure 2-1. Graph-based Image Segmentation Results. A) Original Image. B) Segmented image labeled in randomly chosen colors.

A close inspection of the upper wing reveals much smaller speckles of white between the larger dots of white in the. The dark speckled background of the wing is segmented as a single component. Also evident from this picture is that the algorithm does not favor an orientation; it is capable of identifying regions of any orientation. This

is an important requirement for the MAV platform where the image orientation changes with the pitch and roll of the aircraft.

Parameters of the Segmentation Algorithm

The author of [5] also published C++ language code implementing this segmentation algorithm. This implementation accepts the following parameters:

- $\boldsymbol{I_{size}}$ – This quantity represents the size of the input image. As discussed above, this algorithm performs within $O(I_{size} \log I_{size})$ time. This parameter is the only parameter that affects performance.

- $\boldsymbol{k}$ – This quantity represents the threshold used to perform the segmentation. It affects the size resulting components and therefore the total number of components. Larger values of *k* produce larger components and therefore fewer components. Smaller values of *k* produce smaller components and therefore more components. This parameter does not affect the algorithms performance.

- $\boldsymbol{C_{min\text{-}size}}$ – This quantity defines the minimum size of the components produced by the segmentation. Any component less than the minimum size are merged with an adjacent component. This process is performed after the segmentation is completed. This parameter can improve the performance if its value is 1 in which case it can be ignored.

CHAPTER 3
THE LUCAS KANADE FEATURE TRACKING ALGORITHM

The Lucas Kanade Correspondence Algorithm

Lucas and Kanade [6] describe an algorithm for registering like features in a pair of stereo images. The registration algorithm attempts to locate a feature identified in one image with the corresponding feature in another image. Despite stereo vision being the motivation for the algorithm, there are other applications for this technique. Tracking motion between frames of a single motionless camera is simply the correspondence of features through time. In the case of a moving camera, feature tracking is the correspondence of features in images differing in time and space. The two situations are analogous with the stereo application in that the problem is finding the offset of a feature in one image to the feature in the second image.

The algorithm solves the following problem. Two images that are separated by space or time or both (within a small amount of time or space) have corresponding features. These features exists somewhere in space relative to the camera. As the camera moves, its position relative to these features changes. This change in position is reflected as a movement of the feature on the image plane of the camera. This algorithm identifies the movement (the offset) of a feature in two sequential images.

Lucas and Kanade describe this more formally [6], define the feature of interest in image *A* as the vector $\bar{x}$ and define the same feature in the image *B* as $\bar{x} + \bar{h}$. The problem is to find the vector $\bar{h}$. The algorithm works by searching image *B* for a best match with the feature in image *A*. The best match is defined as the feature in image *B*

that differs the least with the feature in image *A*. An exhaustive search of image *B* for the feature is impractical and would fail to recognize important constraint of locality that is likely to exist. Lucas and Kanade identify two aspects of the searching algorithm:

1. The method used to search for the minimal difference.
2. The algorithm to calculate the value of the difference.

Lucas and Kanade point out that each of these aspects are loosely coupled so implementation can be realized through any combination of searching and differencing algorithms.

Lucas and Kanade's approach uses the spatial intensity gradient of the image to find the value of $\bar{h}$. The process is iterative and resembles a method similar to the Newton-Raphson method where accuracy increases with each iteration. If the algorithm converges, it converges in $O(M^2 \log N)$ time where N^2 is the size of the image and M^2 the size of the region of possible values of $\bar{h}$.

Lucas and Kanade first describe their solution in the one-dimensional case. Let image *A* be represented by function $F(x)$ and image *B* be represented as $G(x) = F(x+h)$. Lucas and Kanade's solution depends on a linear approximation of $F(x)$ in the neighborhood of x. For small h,

$$G(x) = F(x+h) \approx F(x) + hF'(x), \tag{1}$$

$$h \approx \frac{G(x) - F(x)}{F'(x)} \tag{2}$$

In other words by knowing the rate of change of intensity around *x* in $F(x)$ and the difference in intensity between $F(x)$ and $G(x)$, the offset h can be determined. This approach assumes linearity and will only work for small distances where there are no

local minima in the error function. Lucas and Kanade suggest that by smoothing the image, minima produced by noise in the image can be eliminated.

Equation 2 is correct if $G(x) = F(x+h)$. To find the value of *h* where this is true, the possible values of *h* need to be explored and a best match determined. To identify this best match the following error function is defined:

$$E = \sum_x [F(x+h) - G(x)]^2 \tag{3}$$

The value of *h* can be determined by minimizing (3).

$$\begin{aligned} 0 &= \frac{\partial E}{\partial h} \\ &\approx \frac{\partial}{\partial h} \sum_x [F(x) + hF'(x) - G(x)]^2 \\ &= \sum_x 2F'(x)[F(x) + hF'(x) - G(x)] \end{aligned} \tag{4}$$

$$h \approx \frac{\sum_x F'(x)[G(x) - F(x)]}{\sum_x F'(x)^2} \tag{5}$$

These approximations rely on the linearity of $F(x)$ around *x*. To reduce the effects of non-linearity, Lucas and Kanade propose weighting the error function more strongly where there is linearity and less strongly where there is not linearity. In other words where $F''(x)$ is high, the contribution of that term to the sum should be less. The following equation approximates $F''(x)$

$$F''(x) \approx \frac{G'(x) - F'(x)}{h} \tag{6}$$

Recognizing that this weight will be used in an average, the constant factor $\frac{1}{h}$ can be dropped and the weighting function can be

$$w(x) = \frac{1}{|G'(x) - F'(x)|} \tag{7}$$

Including this weighting function (7), the iterative form of (5) becomes

$$h_0 = 0,$$
$$h_{k+1} = h_k + \frac{\sum_x w(x)F'(x + h_k)[G(x) - F(x + h_k)]}{\sum_x w(x)F'(x + h_k)^2} \tag{8}$$

Equation (8) describes the iterative process where each new value of h adds on to the previous value. The weighting function serves to increase the accuracy of the approximation by filtering out the cases where the linearity assumption is invalid. This in turn will speed up the convergence. Each iteration is calculated until the error function value is below a threshold.

Lucas and Kanade describe how the one-dimensional case can be generalized into an n-dimensional case. Similar to the one-dimensional case, the objective is to minimize the error function:

$$E = \sum_{\bar{x} \in R} \left[F(\bar{x} + \bar{h}) - G(\bar{x})\right]^2 , \tag{9}$$

where $\bar{x}$ and $\bar{h}$ are n-dimensional row vectors. The one-dimensional linear approximation in equation 1 becomes

$$G(\bar{x}) = F(\bar{x} + \bar{h}) \approx F(\bar{x}) + \bar{h} \frac{\partial}{\partial x} F(\bar{x}), \tag{10}$$

where $\frac{\partial}{\partial \bar{x}}$ is the gradient operator with respect to $\bar{x}$. Using this multi-dimensional approximation Lucas and Kanade minimize *E*.

$$
\begin{aligned}
0 &= \frac{\partial}{\partial \bar{h}} E \\
&\approx \frac{\partial}{\partial \bar{h}} \sum_{\bar{x}} \left[F(\bar{x}) + \bar{h} \frac{\partial F}{\partial \bar{x}} - G(\bar{x}) \right]^2 \\
&= \sum_{\bar{x}} 2 \frac{\partial F}{\partial \bar{x}} \left[G(\bar{x}) + \bar{h} \frac{\partial F}{\partial \bar{x}} - G(\bar{x}) \right]
\end{aligned}
\tag{11}
$$

Solving for $\bar{h}$ produces

$$
\bar{h} = \left[\sum_{\bar{x}} \left(\frac{\partial F}{\partial \bar{x}} \right)^T \left[G(\bar{x}) - F(\bar{x}) \right] \right] \left[\sum_{\bar{x}} \left(\frac{\partial F}{\partial \bar{x}} \right)^T \left(\frac{\partial F}{\partial \bar{x}} \right) \right]^{-1}, \tag{12}
$$

The Lucas Kanade method described above works for a translation of a feature. Recognizing this, they generalize their algorithm even further by accounting for an arbitrary linear transformation such as rotation, shear and scaling. This is achieved by inserting a linear transformation matrix *A* into the equations. Equation (1) becomes

$$G(\bar{x}) = F(\bar{x}A + \bar{h}),$$

and the error functions 3 and 9 become

$$E = \sum_{\bar{x}} \left[F(\bar{x}A + \bar{h}) - G(\bar{x}) \right]^2 \tag{13}$$

resulting in a system of linear equations to be solved simultaneously.

Because of the linearity assumption, tracking large displacements is difficult. Smoothing the image can remove the high frequency components that make the linearity assumption more valid and allow for a larger range of convergence. Smoothing the image however, removes information from the image. Another method for tracking large displacements is the pyramidal method. This method described by Bouguet [7] makes use of the pyramidal approach to refine the search for h. Details of how it works are explained in the next section. The pyramidal approach makes use of a pyramid of images

each containing the information from the source image represented in a graduated degree of resolution from coarse to fine.

The Pyramidal Implementation of the Lucas Kanade Correspondence Algorithm

The Open Source Computer Vision Library (OpenCV) sponsored by Intel Corporation is a library written in the C programming language that contains a Lucas-Kanade feature-tracking algorithm. The OpenCV implementation makes use of the pyramid method suggested by Lucas and Kanade in their original paper.

The Residual Function

The OpenCV mathematical formalization differs slightly from the Lucas Kanade formalization. Described by Bouguet [7] the formulation is as follows: Let $A(x, y)$ and $B(x, y)$ represent the images between which the feature correspondence should be determined. OpenCV defines the residual function, analogous to the error function (9), as

$$\varepsilon(\bar{d}) = \varepsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} \left(A(x, y) - B(x + d_x, y + d_y)\right)^2 \qquad (14)$$

This equation defines a neighborhood of size $(2w_x + 1) \times (2w_y + 1)$. While the Lucas-Kanade algorithm defines a *region of interest* over which the error function should be summed, the OpenCV implementation is more specific and defines a small integration window defined in terms of w_x and w_y.

Functional and Performance Requirements

The pyramidal algorithm is designed to meet two important requirements for a practical feature tracker:

1. The algorithm should be accurate. The object of a tracking algorithm is to find the displacement of a feature in two different images. An inaccurate algorithm would defeat the purpose of the algorithm in the first place.

2. The algorithm should be robust. It should be insensitive to variables that are likely to change in real world situations. Variables such as variation in lighting, the speed of image motion and patches of the image moving at different velocities.

In addition to these requirements, in practice, the algorithm should meet a performance requirement:

3. The algorithm should be computationally inexpensive. The purpose of tracking is to identify the motion of features from frame to frame so the algorithm generally will run at a frequency equal to the frame rate. Most vision systems perform a series of processing functions to meet specific goals and functions that are run at a frequency equal to the frame rate of the source video need to use a little of the system resources as possible.

In the basic Lucas Kanade algorithm there is a tradeoff between the accuracy and robustness requirements. In order to have accuracy, a small integration window insures that the details in the image are not smoothed out. Preventing the loss of detail is especially important for boundaries in the image that demarcate occluding regions. The regions are potentially moving at different velocities. For the MAV application this is clearly an important requirement due to the velocity of the camera and the potential difference in velocity with objects to be tracked. On the other hand, to have a robust algorithm, by definition, it must work under many different conditions. Conditions where there are large displacements between images are common are common for the MAV platform and warrant a larger integration window. This apparent conflict defines a zero-sum tradeoff between accuracy and robustness. The solution to meeting each requirement without a likely counterproductive compromise is to define a mechanism that decouples one requirement from the other. The method that succeeds in doing this is the pyramidal approach.

The Pyramid Representation

The pyramid representation of image $A(x,y)$ is a collection of images recursively derived from $A(x,y)$. The images are organized into pyramid levels $L = 1 \ldots L_m$ where the original image $A(x,y)$ is L_0. Each image in the pyramid, increasing in level, is a down sampling of the previous level. For example L_0 is down sampled to produce L_1 and L_1 down sampled to produce L_2 up to L_m. An image of size 360x240 and $L_m = 3$ produces a pyramid of three images with dimensions 180x120, 90x60 and 45x30 pixels at levels L_1, L_2 and L_3 respectively.

Pyramidal Feature Tracking

The goal of feature tracking is to identify the displacement of a feature from one image to another. In the pyramidal approach, this displacement vector is computed for each level of the pyramid [7]. Computing the displacement vector $\bar{d}^L = \begin{bmatrix} d_x^L & d_y^L \end{bmatrix}^T$ is a matter of minimizing the following residual function.

$$\varepsilon^L(\bar{d}^L) = \varepsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L-w_x}^{u_x^L+w_x} \sum_{y=u_y^L-w_y}^{u_y^L+w_y} \left(A^L(x,y) - B^L\left(x+g_x^L+d_x^L, y+g_y^L+d_y^L\right)\right)^2 . \quad (15)$$

Note that this residual function is similar to equation (14) but differs by the term $\bar{g}^L = \begin{bmatrix} g_x^L & g_y^L \end{bmatrix}^T$. This term represents the initial guess used to seed the iterative function. The calculation starts at the highest level of the pyramid with $g^{L_m} = \begin{bmatrix} 0 & 0 \end{bmatrix}$. Using this guess, the displacement vector $\bar{d}^{Lm-1}$ is found by minimizing equation (15). This $\bar{d}$ is then used to find the next $\bar{g}$ using this expression:

$$\bar{g}^{L-1} = 2\left(\bar{g}^L + \bar{d}^L\right) \quad (16)$$

The final displacement found by minimizing the residual function (15) for each level in the pyramid is

$$\bar{d} = \sum_{L=0}^{L_m} 2^L \bar{d}^L \tag{17}$$

The advantage of the pyramid implementation is that a small integration window can be used to meet the accuracy requirement while the pyramidal approach provides a robust method to track large displacements. The size of the maximum detectable displacement is dependent on the number of levels in the pyramid. The gain over the maximum detectible displacement in the underlying Lucas Kanade algorithm is:

$$d_{\max-Gain} = \left(2^{L_m+1} - 1\right) \tag{18}$$

For a three-level pyramid, this produces a gain of 15 times the largest displacement detectable by the underlying Lucas Kanade step.

Parameters of the Pyramidal Feature-tracking Algorithm

Applying the pyramidal algorithm using the OpenCV C-language library is a matter of choosing the best values for the following parameters for the particular application.

- ***I**size* –* This quantity represents the source image size. Specifically the size of level $L = L_0$ in the pyramid representation. Larger images have greater detail and therefore can produce more accurate results. Larger images also require more pyramid levels to track feature displacement. These extra pyramid levels can contribute to more CPU time. Each pyramid level represents a standard Lucas Kanade calculation.

- ***Q**F*.* – This quantity represents which features are selected for tracking. In terms of tracking quality, the upper $(1 - Q_F) \times 100\%$ features are selected for tracking. The method for determining tracking quality is defined by Bouguet, Shi and Tomasi [7, 8].

- ***N**F* – This quantity represents the number of features to track. If the number of features in an image meeting the constraint defined by Q_F is greater than N_F, only the best N_F features are selected for tracking [8].

- ***W*** – This quantity is equivalent to the values w_x and w_y in equation 14. This quantity controls the size of the integration window and therefore determines the accuracy of the tracking algorithm.

- $\boldsymbol{L_m}$ – This quantity represents the number of pyramid levels in the image. As described above, this value determines the maximum detectible displacement of a feature. It also determines how many times the Lucas Kanade algorithm is performed for each feature.

CHAPTER 4
SYSTEM INTEGRATION

System Overview

The following describes an object-oriented system framework in which vision processing components are easily interconnected to form a vision-processing pipeline. Typical vision-processing pipelines involve a number of transformations, filters and samplers connected in a meaningful way to perform a task.

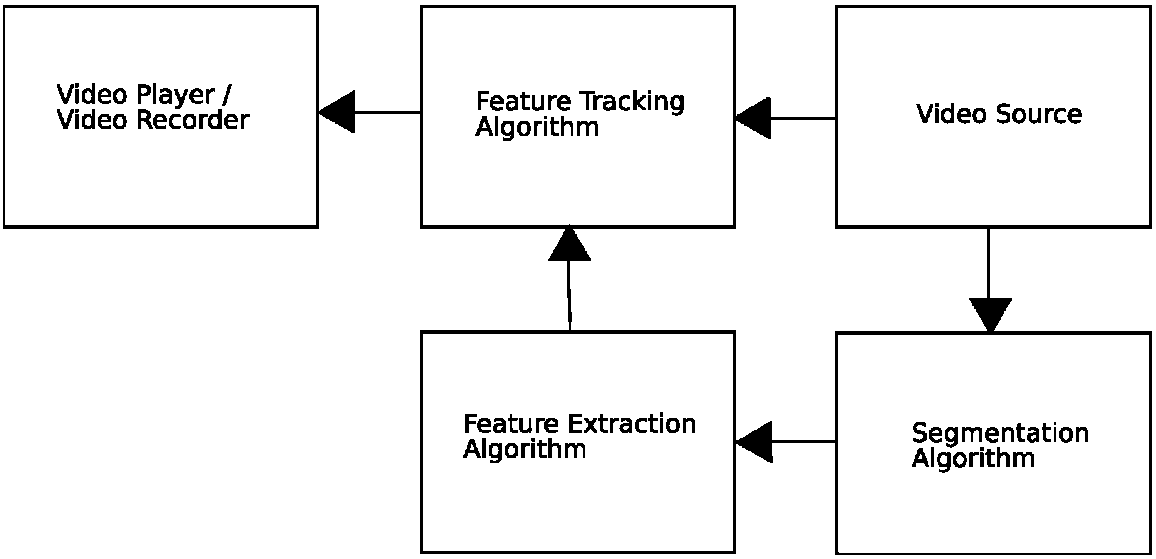


Figure 4-1. The System Overview

The system described by this thesis is formed from the following components:

4. Video Source (a camera or file) – The OpenCV library provides functionality to capture video from a file or a live camera.

5. Feature extraction component – The feature extraction component is described in Chapter 3 as part of the feature tracking system.

6. Image Segmentation Component – The segmentation component is described in Chapter 2.

7. Optical Flow Component – The optical flow component is described in Chapter 3.

8. Video player / video recorder – The OpenCV library provides functionality to write a video stream to a file or to display it on the computer monitor.

The Mediator Design Pattern

By its nature, video processing is a demanding application for which data management is essential to a high performance application. Furthermore, the design process and experimentation processes require that any vision system be made of reusable, decoupled, extensible and manageable components. To meet these requirements, the system described in this thesis utilized the mediator design pattern [9].

The mediator design pattern defines a mediator class that serves as a means to decouple a number of colleague classes. The mediator encapsulates the interaction of the colleague classes to the extent that they no longer interact directly. All interaction happens through the mediator.

The primary role of the mediator in this implementation is to distribute video frames from the output of one component to the inputs of other components. Each component inherits a colleague class which implements an interface that allows the mediator to interact with the individual component. When a component is initialized, it can subscribe to the output of another component. Any frames produced by the source component will automatically be communicated to the subscribing component.

The following describes the typical message exchange between a colleague and the mediator. When the colleague changes its internal state in a way that other colleagues should know about, it communicates that change in state to the mediator class. In this particular implementation, the colleague class notifies the mediator whenever the colleague has produced a frame of video. The mediator then grabs the frames from the

output of the source colleague class and distributes the frames to the input of colleagues subscribing to the output of the source class.

This mediator architecture is designed for expansion. For example, the mediator class could implement a frame cache for an asynchronous system. Another possible improvement is the addition of a configuration file for runtime configuration of the vision system. The system would simply parse the configuration file and based on the contents it could connect the components in the correct order all without a line of code or a compiler.

System Timing and the Observer Design Pattern

A central clock controls the timing of the entire system. The clock is implemented using the observer design pattern. The purpose of this design pattern is to allow a number of objects to observe another object, the subject. Whenever the state of the subject changes, all other subscribing objects, the observers, are notified. In this implementation, the subject is an alarm clock. Any class inheriting the observer class CWatch can subscribe to the clock. As a parameter of the subscribe method call, the observing class passes a time period in milliseconds. The clock will then notify the observer periodically according to the time period.

In this implementation, the clock is useful when capturing data from a live camera. The clock also makes possible multiple frequencies within the system. For example a feature-tracking algorithm can run at a rate of 30Hz while an image segmentation algorithm can run at a different rate such as1Hz.

System Interaction

Figure 4-1 illustrates how the components of the system are organized. The feature-tracking algorithm runs in parallel with the serial combination of the

segmentation algorithm and the feature-extracting algorithm. This organization is necessary to meet the performance requirements of the system. Segmenting an image at the full 30Hz frame rate is both computationally expensive and unnecessary. Instead, the segmentation can happen at a much lower frequency while the feature tracking algorithm runs at a full 30Hz. The mediator pattern described earlier makes implementing a parallel architecture practical and easy. Furthermore the system timing functionality provided by the clock allows for multiple frequencies within the system.

The current implementation is not multi-threaded so this configuration does not currently make full use of the benefits provided by the architecture. For example, the segmentation algorithm may run for one second. In this second, the processor never passes control to the clock object and therefore the more frequent feature tracking updates are never made. In a multi-threaded environment however, this blocking would not occur. The low frequency segmentation algorithm could run concurrently with the higher frequency feature-tracking algorithm for the minor cost of thread management. The analysis in Chapter 5, System Performance, assumes that the system runs in a multi-threaded environment.

CHAPTER 5
SYSTEM PERFORMANCE ANALYSIS

System Performance Requirements

A MAV such as the one developed by the AVCAAF group has a limited performance envelope. As a fixed wing aircraft, it cannot hover nor slow down in order to complete a calculation before making a decision. This characteristic translates into strict performance requirements for the underlying control systems. While the object tracking system described by this paper is not a critical control system, it does run concurrently with the other vision systems and therefore indirectly affects overall MAV performance. While an object tracking system may not provide control critical services, it would likely provide mission critical services for missions such as surveillance, search and rescue or even function as part of a landmark-based navigation system. Another possible role critical to the survival of the MAV is collision avoidance. An object tracking system should meet the following performance requirements.

9. The system should be computationally efficient – It should not require a significant share of the available processing power. Other more critical systems such as flight stability will run concurrently with the tracking system and should not be adversely affected by an inefficient tracking algorithm.

10. The system should run at a rate useful for the particular mission – An update rate of 1 update every 5 seconds may be sufficient for a navigation by landmark mission, but totally unsuitable for navigating a forest of skyscrapers.

Computational Complexity of Each Algorithm

The dominant processes in this system are the feature tracking algorithm and the image segmentation algorithm. This section describes the coupling of these components and how the parameters of the components affect the performance of the system overall

Chapters 2 and 3 describe the complexity of each component. The complexity of the segmentation algorithm is $O(I_{size} \log I_{size})$ while the complexity of the tracking algorithm is $O(L_m N_F w^2 \log I_{size})$. While the tracking algorithm has the w^2 term, experimentation shows that the segmentation algorithm is the most computationally costly. The reason for the disparity in actual CPU time is that the $L_m N_F w^2$ is typically much smaller than the size of the image I_{size}.

Analysis and Discussion

Description of the Test Video Sequence

The experiment was conducted on a video sequence that represents a typical MAV flight. In this sequence the MAV is flying at an approximate altitude of 50 ft and an average speed of 25mph. The conditions are a sky with scattered cumulus clouds and unlimited visibility (no haze). The MAV is flying over a green field with regular patches of brown. Tethered to a fence in the field are two red, helium-filled 8ft diameter balloons. The balloons are partially covered on the bottom with a black tarp. The presence of the tarp produces a black wedge in the circular image of each balloon.

The video sequence captures the relative motion of two balloons. Balloon A is on the left and balloon B is on the right. The first frame in the video sequence contains the first frame in which balloon A entered the picture. The last frame in the video sequence corresponds to the last frame in which balloon B is present.

In the initial frames of the video sequence, both balloons A and B are present. Balloon A is on the left and balloon B is on the right. As the MAV flies toward the balloons, both balloons translate from right to left until only balloon B is within the image. The MAV then turns left toward balloon B and flies directly toward it. After leaving, balloon A never reenters the frame.

To make the turns the MAV must roll and therefore each turn is marked by a rotation of the entire picture. Also, the MAV must make slight altitude changes that are reflected as vertical motion in the picture.

The video sequence is 108 frames long and was captured at 30 frames per second. Each frame in the sequence has a resolution of 720 by 480 pixels and is interlaced.

Preprocessing

The video sequence was preprocessed before running the experiments described in this chapter. First the video was de-interlaced and then it was down sampled to 50% of the original size.

De-interlacing was achieved by duplicating one of the fields for each frame in the interlaced sequence. This reduced the vertical resolution by 50%, but this method prevented the introduction of ghost artifacts in the video.

The video was down sampled by applying a radius-1 Gaussian blur and then a point-sampled resize by half. This method avoids artifacts caused by aliasing, but at the cost of a blurrier output. Bilinear or bicubic filters may be a better choice to down sample the video and maintain quality without producing artifacts but the result of the Gaussian blur was sufficient.

Method

The Segmentation algorithm and tracking algorithms are the primary subjects of the experiment. Each algorithm has a set of parameters as described earlier in their respective chapters. The objective of this experiment is to search the parameter space of these algorithms to find an optimal configuration. The parameter space is huge and therefore an exhaustive search of the parameter space is not feasible. Furthermore, because of the nature of the problem, quantitative experimental results cannot be used to determine the success of each experiment. Instead, a human must evaluate each result set. The human observer must classify the experiment as successful or unsuccessful.

To search the parameter space, the parameters were adjusted with increasing computational efficiency until the algorithm no longer met its requirements. Each of the algorithms was tested individually and compared against its requirements.

Neither algorithm depends on the other to meet its requirements. Each algorithm however will run on the same hardware and is therefore constrained by the availability of CPU time. This analysis first describes how each algorithm behaves independent of the other. The analysis then describes the performance of a system containing these two algorithms, specifically the frequency at which each algorithm can run in real-time given a specific hardware configuration.

The computation performance of each experiment was measured using a code profiler. The profiler measured the total CPU time used by each algorithm.

Environment

The experiments described in this chapter were performed on a Pentium III running at 800MHz with 512 MB RAM.

The Segmentation Algorithm

Each experiment to measure the performance of the segmentation algorithm was performed by running the algorithm on all 108 frames in the video sequence. To verify that the parameter *k* does not affect the computational performance, table 5-1 shows the results for *k* = 100, 1000, 5000 and 10000. The different values of *k* did not change the time of computation, but did change the quality of the output. As expected, lower values of *k* produced smaller components and therefore more components while larger values of *k* produced the opposite. By qualitative observation, *k* = 5000 produced the best segmentation of the test video. This value of *k* was used to perform the remaining experiments.

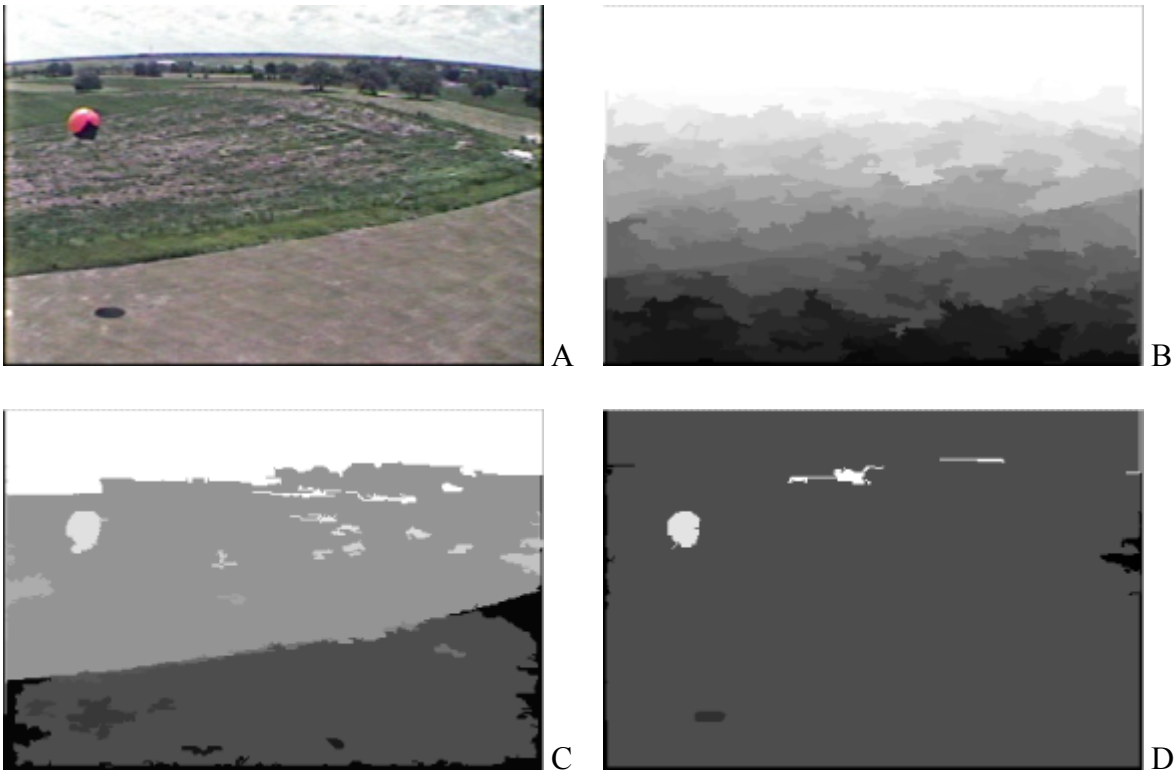


Figure 5-1. Graph-based Segmentation Output. A) Original image, B) *k* = 100, C) *k* = 5,000, D) *k* = 10,000.

Table 5-1. CPU Time for the Graph-Based Segmentation Algorithm

Experiment	k	$C_{min\text{-}size}$ [pixels]	I_{size} [pixels]	Total CPU Time [sec]	Average CPU Time per Frame[sec]
1	100	50	360x240	452.7	4.2
2	1000	50	360x240	406.1	3.8
3	5000	50	360x240	443.0	4.1
4	10000	50	360x240	428.0	4.0
5	5000	50	270x180 (75%)	224.2	2.1
6	5000	50	180x120 (50%)	107.4	1.0
7	5000	50	90x60 (25%)	27.0	0.25

The computational complexity of the segmentation algorithm dictates that the algorithm's time of execution is dependent on the size of the image. Table 5-1 shows the CPU time for the algorithm running on an image scaled by 1.0, 0.75, 0.5 and 0.25. The total CPU time values accurately reflect the algorithm complexity $O(I_{size} \log I_{size})$.

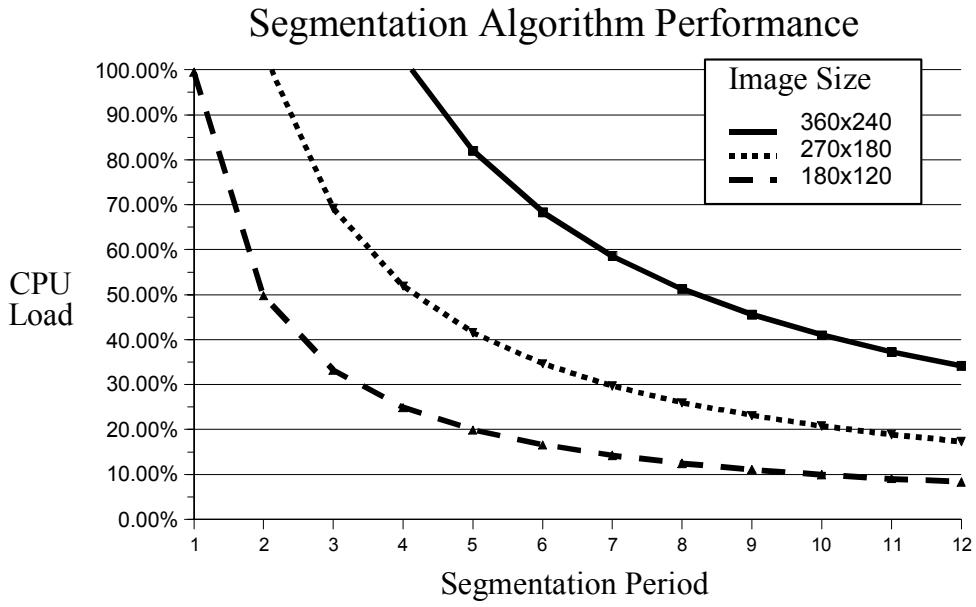


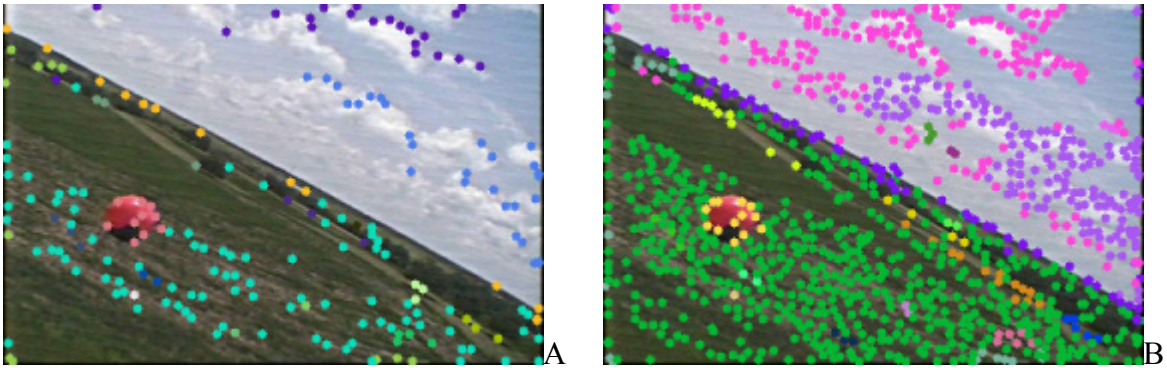
Figure 5-2. Segmentation Algorithm Performance.

The algorithm meets its requirements for scale values 1.0, 0.75 and 0.5 but not for 0.25. There is a particular set of frames (frames 68 - 90) where the algorithm fails at

every scale. In these frames the algorithm confuses a brown patch of dirt and grass in a field with the red balloon in the foreground.

Pyramidal Lucas-Kanade Feature Tracker

The dominant parameters in the feature-tracking algorithm are L_m, w, and N_F. Table 5-2 shows results for combinations of L_m and w. The algorithm requires a set of features to track as input. The OpenCV library contains a function cvGoodFeaturesToTrack that finds the best features suitable for the tracking algorithm [3, 4]. This function requires values for the parameters Q_F and N_F. These parameters were described in chapter 3.



A B

Figure 5-3. Results from the Pyramidal Lucas-Kanade Feature Tracker. A) N_F = 200, B) N_F = 1,000

To improve performance the 30 FPS video was sampled at the lower frequencies of 15, 10 and 5 FPS. Running at lower frequencies requires that the algorithm work for larger displacements of the features between frames. This scenario is precisely what the pyramidal L-K tracker was designed to fulfill. The algorithm maintained a good track for the 15 and 10Hz runs. The algorithm started losing points during tracking for the 5Hz run even with the maximum number of Pyramid levels L_m = 4.

Table 5-2. CPU Time for the Pyramidal Implementation of the Lucas Kanade Feature-tracking Algorithm.

Experiment	*Frames per Second*	L_m	w	N_F	Average CPU Time per Frame [μsec]	CPU Load
8	30	1	2	100	6,677.45	20.03%
9	30	2	2	100	8,495.98	25.49%
10	30	3	2	100	7,287.76	21.86%
11	30	1	3	100	6,909.44	20.73%
12	30	2	3	100	7,950.88	23.85%
13	30	3	3	100	8,758.49	26.28%
14	30	3	3	50	5,617.95	16.85%
15	30	3	3	200	15,172.56	45.52%
16	30	3	3	1000	73,582.15	220.75%
17	15	3	3	200	13,276.56	19.91%
18	10	3	3	200	14,025.36	14.03%
19	5	4	3	200	13,326.06	6.66%

In experiment 16, the feature extractor quality parameter had to be set to 0.005 to get enough points to meet the $N_F = 1000$.

Coupling of Algorithms

The segmentation algorithm requires significantly more computation than the feature tracker. This is easy to understand by recalling the computational complexity of the algorithms. The segmentation algorithm was highly dependent on the size of the image: I_{size}. While the feature tracking algorithm was highly dependent on the number of feature points to be tracked: N_F. This difference is intuitive: the segmentation algorithm must run on every pixel in the image while the feature tracking algorithm must only run on a small set of pixels. Also the tracking algorithm exhibited robust performance at low frequencies: 15 and 10Hz. As with most algorithms there is a tradeoff between accuracy and performance. Figure 5-4 charts the performance of the coupled algorithms against the period of the segmentation algorithm.

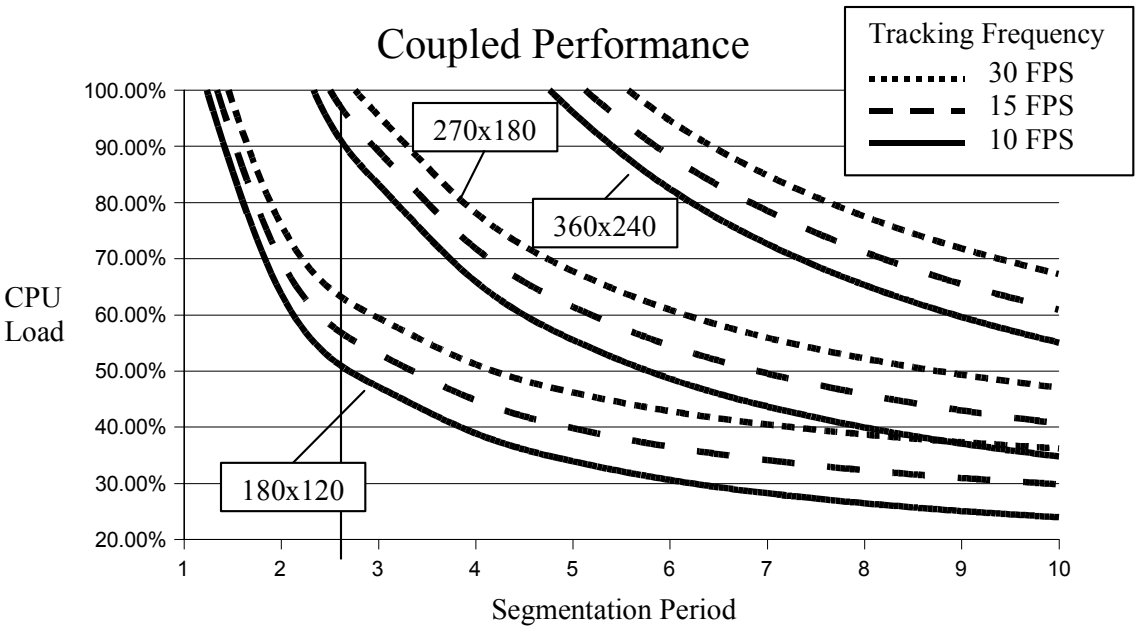


Figure 5-4. The Performance of the Coupled Segmentation and Tracking Algorithms. Results plotted for images with sizes 360x240, 270x180 and 180x120 at tracking frequencies of 30, 15 and 10 FPS.

The AVCAAF MAV flies at a speed of 36.7 ft/sec (25 MPH). Assuming that the MAV is flying in a straight line in a static world, in order to detect an object before it is within 100ft, the MAV must refresh its knowledge of objects once every 2.73 seconds (the time to cover 100ft at 25MPH). Figure 5-4 shows that this can be achieved using the 50% scaled image (180x120) and a tracking frequency between 10 and 30 without loading the CPU any more than 65%. Tracking at 10 FPS puts a 50% load on the processor while tracking at 30 FPS loads the processor approximately 64%. Using the 75% scaled image, the system would exceed the available CPU power at a 30 FPS tracking frequency and use 100% of the processor for tracking frequencies of 10 and 15 FPS. Using 100% of the processor for object tracking is impractical because there are likely other systems more critical to MAV flight requiring CPU resources

CHAPTER 6
CONCLUSION AND FUTURE WORK

The results from chapter 5 show that the Pyramidal Implementation of the Lucas Kanade feature tracker is robust and computationally efficient algorithm. It is capable of meeting functional and performance requirements over a range of configurations. The graph-based segmentation algorithm, while capable of meeting the functional and performance requirements marginally, did not perform in a robust or computationally efficient manner.

Future Work for the Pyramidal Implementation of the Lucas Kanade Feature Tracker

The Lucas Kanade algorithm was designed an image registration algorithm for the purposes of stereo vision, but it has many applications beyond stereo vision. This paper shows how the algorithm can be used to track moving objects from a moving camera. Structure from motion and image mosaic registration are other applications that could be useful in MAV missions. The Lucas Kanade algorithm is a unifying framework [10].

They pyramidal implementation allows for efficient computation of optical flow for a specific set of points in the image. An arbitrary set of points or even the entire image can be calculated. Using the architecture described in Chapter 4, the optical flow processing component can be shared by other video components in the system. This reduces redundant calculations.

Future Work for the Segmentation Algorithm

A possible optimization for the segmentation algorithm is to optimize for distance in the image (the further the distance the less detailed the image has). The segmentation algorithm functions at a fixed level of detail. This fixed level of detail is optimized for a specific distance in the image, but not for all distances in the image. The AVCAAF MAV uses a vision system for estimating the location of the horizon in the image[3]. One improvement would be to use the horizon estimate as a way to identify the distance to a pixel in 3D space. The segmentation algorithm could be adjusted to discriminate at finer detail for pixels further in the image.

LIST OF REFERENCES

[1] S. Todorovic, M.C. Nechyba, "Detection of Artificial Structures in Natural-Scene Images Using Dynamic Trees," *Proc. 17th Int'l. Conf. Pattern Rec.*, Cambridge, UK, Intl. Assoc. Pattern Rec., 2004, pp. 35-39.

[2] S. Todorovic, M.C. Nechyba, "Intelligent Missions for MAVs: Visual Contexts for Control, Tracking and Recognition," *Proc. 2004 IEEE Intl. Conf. on Robotics and Automation*, New Orleans, LA, Apr. 2004, pp. 1640-1645.

[3] S. Todorovic, M.C. Nechyba, P.G. Ifju, "Sky/Ground Modeling for Autonomous MAV Flight," *Proc. IEEE Int'l. Conf. Robotics and Automation*, Taipei, Taiwan, vol. 1, 2003, pp. 1422-1427.

[4] S. Todorovic, M.C. Nechyba ,"Towards Intelligent Mission Profiles of Micro Air Vehicles: Multiscale Viterbi Classification," *Lecture Notes in Computer Science, Computer Vision – ECCV 2004: 8th European Conf. on Computer Vision*, Prague, Czech Republic, vol. 3022, May 2004, pp. 178-189.

[5] P.F. Felzenszwalb and D.P. Huttenlocher, "Efficient Graph-Based Image Segmentation," *Int'l J. Computer Vision*, vol. 59, no 2, September 2004, pp. 167-181.

[6] B.D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *Proc. Image Understanding Workshop*, Washington, D.C., 1981, pp. 121-130.

[7] J. Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the Algorithm," *OpenCV Documentation,* Intel Corporation, Microprocessor Research Labs, Santa Clara, CA, 2000.

[8] J. Shi and C. Tomasi, "Good Features to Track," *Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition*, Seattle, WA, 1994, pp. 593-600.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, MA, 1995.

[10] S. Baker and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework," *Int'l J. Computer Vision*, vol. 56, no. 3, February 2004, pp. 221-255.

BIOGRAPHICAL SKETCH

Ted Belser (Von) was born in Gainesville, Florida, in 1978. Von participated in the International Baccalaureate Program at Eastside High School in Gainesville. He continued his education at the University of Florida where he earned his Bachelor of Science degrees in electrical engineering and computer engineering. While participating in a software development startup company he attended graduate school at the University of Florida and earned a Master of Science degree in electrical engineering.