2009-02-12

A Unified Approach to GPU-Accelerated Aerial Video Enhancement Techniques

Stephen Thayn Cluff
Brigham Young University - Provo

A UNIFIED APPROACH TO GPU-ACCELERATED AERIAL VIDEO ENHANCEMENT TECHNIQUES

by

Stephen T. Cluff

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

April 2009

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Stephen T. Cluff

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Bryan S. Morse, Chair
Date	Michael A. Goodrich
Date	Scott N. Woodfield

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Stephen T. Cluff in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Bryan S. Morse Chair, Graduate Committee
Accepted for the Department	Kent E. Seamons Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

A UNIFIED APPROACH TO GPU-ACCELERATED AERIAL VIDEO ENHANCEMENT TECHNIQUES

Stephen T. Cluff
Department of Computer Science
Master of Science

Video from aerial surveillance can provide a rich source of data for analysts. From the time-critical perspective of wilderness search and rescue operations, information extracted from aerial videos can mean the difference between a successful search and an unsuccessful search. When using low-cost, payload-limited mini-UAVs, as opposed to more expensive platforms, several challenges arise, including jittery video, narrow fields of view, low resolution, and limited time on screen for key features. These challenges make it difficult for analysts to extract key information in a timely manner.

Traditional approaches may address some of these issues, but no existing system effectively addresses all of them in a unified and efficient manner. Building upon a hierarchical dense image correspondence technique, we create a unifying framework for reducing jitter, enhancing resolution, and expanding the field of view while lengthening the time that features remain on screen. It also provides for easy extraction of moving objects in the scene. Our method incorporates locally adaptive warps which allows for robust image alignment even in the presence of parallax and without the aid of internal or external camera parameters. We accelerate the image registration process using commodity Graphics Processing Units (GPUs) to accomplish all of these tasks in near real-time with no external telemetry data.

ACKNOWLEDGMENTS

There are many people who were very instrumental in the creation of this work. I cannot express enough how grateful I am to my advisor, Dr. Bryan Morse, for his endless patience and his dedication to his students. He was the teacher that introduced me to image and signal processing and made it accessible and very interesting to me. He supported my research efforts and provided encouragement in his office, in his lab, and even across time zones. His guidance and many revisions to my work made it much better that it would have otherwise been. Thank you for your kindness and generosity.

Thank you Dr. Michael Goodrich and Dr. Scott Woodfield for serving on my committee and carefully reviewing my thesis.

Mark Duchaineau and Jonathan Cohen at Lawrence Livermore National Laboratory were wonderful collaborators and a pleasure to work with. I met Mark at a point when I was ready to abandon my work. Not only did his amazing work inspire me to continue, but his guidance and support made it possible for me to do so. I consider him to be a second advisor. Jonathan is a true GPGPU wizard and a tremendous writer who helped polish my rough writing.

Nathan Rasmussen and Morgan Quigley spent many hours building, maintaining, and flying mini-UAVs. As a result I had access to excellent aerial video data.

To my beautiful wife, Janelle, I am deeply indebted. Your unwavering support and your selfless sacrifice sustained me through every challenge. I dedicate this work to you.

Contents

Chapter 1

Introduction

Time is of the essence when conducting Wilderness Search and Rescue (WiSAR) operations. With each passing hour the search area radius increases by approximately three kilometers, and the chances of finding the missing person decreases significantly [12]. Any tool or technique that speeds up the search, therefore, can only increase the chances of success.

Miniature unmanned air vehicles (mini-UAVs) have been proposed as a new technology that may be able to provide additional support to rescue workers. A small video camera attached to the mini-UAV allows a worker to quickly scan an area for signs of a missing person. The advantages of using a mini-UAV during the WiSAR operation are

1. they are small and easy to transport to remote locations,
2. they require no more than two people to operate,
3. they are relatively inexpensive to buy and operate,
4. they can cover a large area of rugged terrain quickly, and
5. they can scan the search area without contaminating it.

Using mini-UAVs to assist WiSAR operations is one example of the broader problem of using mini-UAVs for video surveillance in general. There are many other application areas including law enforcement, border patrol, traffic surveillance, and military reconnaissance.

1.1 Challenges Using mini-UAVs for Video Surveillance

Using a mini-UAV as a platform for video surveillance may provide many advantages, but there are still some limitations that need to be overcome for the system to be useful. While their small size makes them very portable, mini-UAVs have a limited payload capacity. This weight limitation not only restricts how much equipment it can carry, but it also indirectly restricts the kinds and number of sensors, processors, and radio transmitters that can be powered by the onboard batteries. Any device that requires power will necessitate additional batteries or will reduce the flight time of the mini-UAV.

Another limitation of the mini-UAV platform is the resolution available from the video stream. High-resolution cameras either weigh too much or consume too much power to be carried by the mini-UAV. Even as these limitations are overcome by new cameras that weigh less and consume less power, the data still needs to be transmitted to the ground station where it can be viewed by an operator. The bandwidth of the radio transmitter now becomes a limiting factor in how much data can be transmitted to the ground station.

In order to get a high enough resolution to identify interesting features on the ground, a high-magnification lens must be used on the camera. This produces a few undesirable effects. First, magnifying the image also magnifies the jittery motion of the mini-UAV as it is buffeted by air turbulence. This emphasizes the constant, high-frequency motion of the camera and makes the video difficult to watch. This limitation can be mitigated by calculating the global frame-to-frame transformation between neighboring frames of the video and using the transformation to stabilize the movement of the video, as in [24].

A second drawback from using a high-magnification lens is that it restricts the field of view of the camera and creates the effect of looking at the ground through a "soda-straw" [4]. It also causes features to move across the viewable area more quickly and limits the time that they are detectable on screen. This combination of limited field of view and fast-moving features can cause a loss of situational awareness and can make it difficult for a human watching the video to relate objects in the current field of view with objects

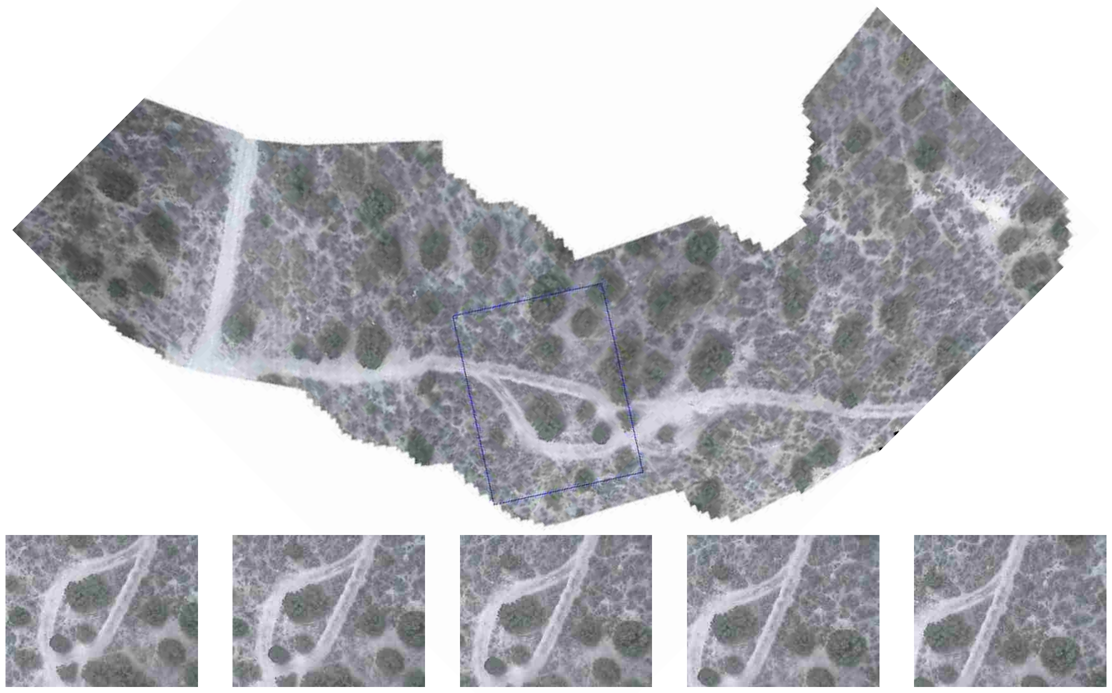


Figure 1.1: A 301-frame mosaic created using the methods described in this work, with the anchor frame outlined in blue. Beneath are five frames surrounding the anchor, acquired 1/30 of a second apart. At the velocity shown by the five frames, the entire visible scene would pass from view in approximately one second. Mosaicking dramatically extends the visible lifetime of the images.

that were observed previously. Current systems overcome these challenges by relying on an abundance of expensive hardware that is easily transported by a full size airplane [20]. The limited payload capacity of mini-UAVs, together with the desire to keep costs reasonable enough to allow local government agencies to purchase the technology, eliminates the possibility of carrying onboard stabilization mechanisms and limits the computing power available onboard. Prior work, however, has shown that this limitation can be overcome through processing the UAV video once it is received at the ground station and presenting it in a mosaic view (Figure 1.1). This can reduce the "soda-straw" effect while providing greater opportunity for detection and identification of objects of interest [24]. Again, the key technology for creating pleasing video mosaics is an accurate registration of the video frames that comprise the mosaic.

A third undesirable effect of using a high-magnification lens is that the resulting narrow field of view means it takes longer to cover the same area. We want to cover the search area as quickly as possible, so we need a way to reduce the search time while providing the necessary resolution to identify features on the ground. This problem is further complicated by the payload and bandwidth limitations that result from using the mini-UAV platform. One method for reducing the time required to cover the search area with the required resolution is to cover a larger portion of the search area on each pass of the UAV, either by flying at a higher altitude or by using a lower-magnification lens. While this would normally reduce the resolution of the video, we can partially recover this loss of resolution by employing a super-resolution technique in which information from overlapping video frames is combined to create an image that has a higher spatial resolution than any of the input images. Once again, this enhancement to the original video relies on having an extremely accurate registration of the images used for super-resolution.

While its application in WiSAR operations has yet to be implemented or evaluated, the ability to identify and track moving objects in surveillance video is a highly desirable feature to assist in the analysis of the video. This is typically done by registering two frames of the video and then doing an image difference to determine what has changed between the two frames. This registration is required to compensate for the camera motion so that the only changes that remain after doing the image difference are due to objects that have moved in the scene. As with the other video enhancement techniques, this registration needs to be extremely accurate. Otherwise, when the image difference is performed any registration misalignments will show up as false positives, indicating a moving object where there was none.

1.2 Global Registration Versus Locally Adaptive Warps

We note that each enhancement to the video, be it stabilization, mosaicking, super-resolution, or identifying moving objects in a scene (mover-detection), requires accurate image registration that can be calculated in a timely manner. One challenge that can plague global registration techniques is motion parallax, or the apparent motion of nearby objects

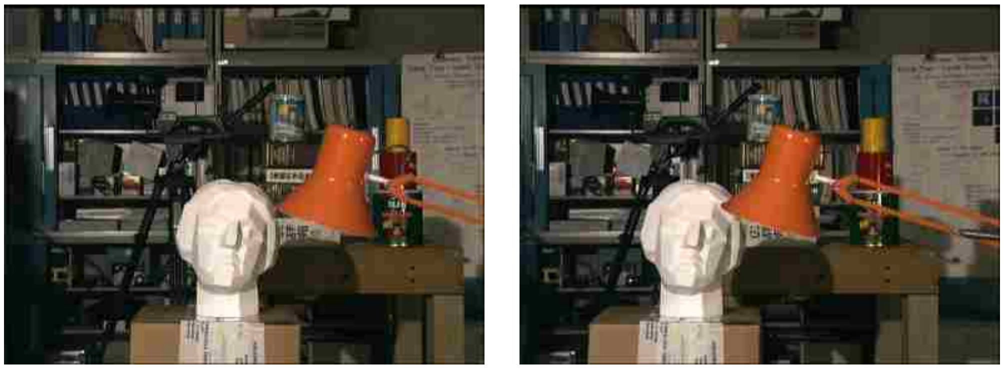


Figure 1.2: Parallax motion. As the camera viewpoint changes objects that are closer to the camera, such as the lamp and the statue, move across the image faster than more distant objects. Images from [27].

relative to more distant objects when viewed from two distinct viewpoints. In aerial video this parallax motion is caused by variation in terrain height or by objects that are tall enough to be significantly closer to the camera than the rest of the background (Figure 1.2).

When the camera motion and scene geometry are known a priori the parallax motion can be compensated for directly and a global registration method is sufficient to produce satisfactory results. However, when the depth variation in the scene is unknown and global registration is used, parallax motion causes blurring in the super-resolution results and false positives in the mover-detection results.

In order to avoid the problems introduced by parallax motion when using global registration we need a technique that allows the transformation to deform locally to account for parallax motion. While techniques such as optical flow are capable of calculating these locally adaptive warps, these algorithms are computationally very expensive. Aerial video surveillance typically generates a large volume of data that needs to be processed in near real-time in order to maintain its relevance. Traditional methods either cannot produce the locally adaptive warps that we require, or they are too computationally intensive to be used with commodity hardware to produce near real-time results.

1.3 Contributions

In order to overcome the limitations of global transformations, we build upon an image correspondence technique, *hierarchical dense correspondence* (HDC), that robustly tracks the general global frame-to-frame motion while providing locally deformable warps that accommodate variations from the global transformation caused by parallax or lens distortion. We show that HDC handles parallax and lens distortion more robustly than global transformations through comparison of super-resolution and mover-detection results using global and local techniques. We also show how HDC provides a unifying framework to simultaneously stabilize aerial video footage, create a video mosaic, enhance the resolution of the video, and extract dynamic events in the video. We provide an example video interface that demonstrates each of these each of these video enhancements running on a modern desktop with a high-end graphics card.

HDC robustly aligns images without the aid of internal or external camera parameters. Eliminating the dependency on camera parameters allows the video interface to provide its enhancements solely through analysis of the video stream, enabling its use with a greater range of video where these parameters are unavailable.

The HDC algorithm has the added benefit of being easily ported to run on commodity Graphics Processing Units (GPUs). The hundreds of gigaflops of processing power and large number of floating point processors provided by modern GPUs allow the image registration to be accelerated to interactive speeds. This acceleration is key to creating a responsive video interface that provides the desired video enhancements in a timely way. We compare the performance of HDC running on the CPU to the performance of HDC when accelerated on the GPU. In addition, because both global transformations and locally adaptive warps can be obtained from HDC, it provides an efficient way to achieve video stabilization, mosaics, super-resolution, and mover-detection.

1.3.1 Individual Contributions

This work is the result of a collaboration between Stephen Cluff and Bryan Morse at Brigham Young University and Mark Duchaineau and Jonathan Cohen at Lawrence Livermore National Laboratory. All four researchers contributed to the jointly authored paper that appears in revised form as Chapter 2 in this thesis, with Cluff being the primary author of the paper. A summary of the differences between the paper as it appears in this thesis and the jointly authored paper appears at the beginning of Chaper 2. The HDC algorithm was developed by Duchaineau and ported to the GPU by Cohen.

Cluff's main contribution was in developing the example interface that demonstrates how video stabilization, mosaics, and super-resolution can all be achieved using HDC as the underlying algorithm. This included several non-trivial implementation details. He used a least squares fit to reduce the HDC mapping to an affine transformation (Section 2.6.1), allowing for efficient compositing of individual video frames and for extrapolation of the video mosaic well beyond the point where the first frame in the mosaic no longer overlaps with the last frame. He introduced the technique of continuously recomputing the transformations used to display older frames in the mosaic history so that the current frame remains undistorted while the accumulated error due to compositing is transferred to the oldest frames in the mosaic (Section 3.1.1). He also introduced the idea of showing the frames that come both before and after the current frame in the same mosaic while processing the video in offine or buffered mode. This allows the reach of the mosaic to be extended while minimizing distortion and without necessitating a costly bundle adjustment (Section 3.1.2). He improved upon the video interface presented in [24] by allowing the user to pan, zoom, and rotate the mosaic, as well as pause the playback and rewind to a previous spot in the mosaic. He also showed that it is possible to stabilize the video using the same affine transformations that were used to construct the mosaic (Section 2.6.3).

In addition to the mosaic improvements, Cluff also introduced real-time super-resolution display using the GPU. He implemented two blending algorithms, nearest neighbor and weighted average, on the GPU to allow interactive display of super-resolution results once the image alignment was computed using HDC. He leveraged the information extracted during the creation of the mosaic by using the relative frame positions of non-adjacent frames to seed the HDC algorithm. This eliminates the need to use a separate bootstrapping technique to align temporally and spatially distant frames.

1.4 Thesis Outline

Chapter 2 contains a paper that was submitted in revised form to the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), although the references section has been moved to appear at the end of this thesis. Sections 2.1 and 2.2 introduce the research, including the challenges that are unique to visualizing mini-UAV video. In Sections 2.3 and 2.4 we review related work and present an overview of our video interface, showing how each of the video enhancements are a direct result of the frame-to-frame alignment that is provided through HDC. The HDC algorithm is detailed in Section 2.5. The process of creating a local mosaic is discussed in Section 2.6. The mosaic provides a starting point for further HDC refinement, which enables super-resolution and mover-detection, described in Sections 2.7 and 2.8, respectively. We draw conclusions and discuss future work in Section 2.10.

Chapter 3 of this thesis contains additional contributions and details that were not included in the CVPR submission due to space constraints. Chapter 4 provides some details necessary for the reader to recreate the work described in this thesis. We conclude with a discussion of contributions, limitations, and future work in Chapter 5.

Chapter 2

Paper Submitted to CVPR

This chapter contains the research paper that was submitted in revised form to the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). The paper contained in this chapter was written to emphasize the contributions of the author of this thesis and focuses on the application of the HDC algorithm for use in video visualization. The paper that was submitted to CVPR was revised to emphasize the advantages of the HDC algorithm over other image registration techniques and to emphasize HDC as a unifying framework for doing mosaicking, super-resolution, and mover-detection.

2.1 Abstract

Video from aerial surveillance can provide a rich source of data for analysts. From the time-critical perspective of wilderness search and rescue operations, information extracted from aerial videos can mean the difference between a successful search and an unsuccessful search. When using low cost, payload-limited mini-UAVs, as opposed to more expensive platforms, several challenges arise, including jittery video, narrow fields of view, low resolution, and limited time on screen for key features. These challenges make it difficult for analysts to extract key information in a timely manner.

We present a video interface that reduces jitter, enhances resolution, and expands the field of view while lengthening the time that features remain on screen. We create a uni-

fying framework for accomplishing all of these tasks in real-time on commodity hardware with no external telemetry data using a novel image correspondence technique.

2.2 Introduction

Over the last decade, video surveillance from Unmanned Aerial Vehicles (UAVs) has become increasingly feasible. UAV-based systems come in a range of sizes and prices. The more expensive systems, costing hundreds of thousands to millions of dollars, include multiple high-resolution cameras, GPS and other sensors, on-board data storage, and state-of-the art transmission to ground. However, the high cost of such systems prohibits their widespread adoption for important applications such as Wilderness Search and Rescue. For such applications miniature UAVs (mini-UAVs), typically in the ten to fifty thousand dollar price range, may be more realistically and widely deployed.

Unfortunately, videos taken from mini-UAVs suffer from a number of problems. Due to their small size, these mini-UAVs are easily buffeted by the wind, making the video jittery and hard to watch. Also, a zoom lens is required to provide enough detail of the ground while maintaining sufficient altitude to avoid crashing the mini-UAV. This creates a "soda-straw" effect, similar to that described in [4], where the features pass very quickly across a narrow field of view and spatial context is quickly lost. This effect makes it difficult for an observer to detect and identify objects and landmarks, let alone to build a mental model of the enduring scene. Finally, the resolution of the video is restricted by both the limited payload capacity of the craft as well as the available bandwidth for transmission to ground. Our system had a payload capacity of up to one kilogram.

Video mosaics have the potential to address these problems. Mosaics are constructed by aligning and compositing multiple frames of a video sequence into a common domain of spatio-temporal locality (see Figure 2.1). It has been demonstrated in [24] that such mosaics can help alleviate the "soda-straw" effect while providing greater opportunity

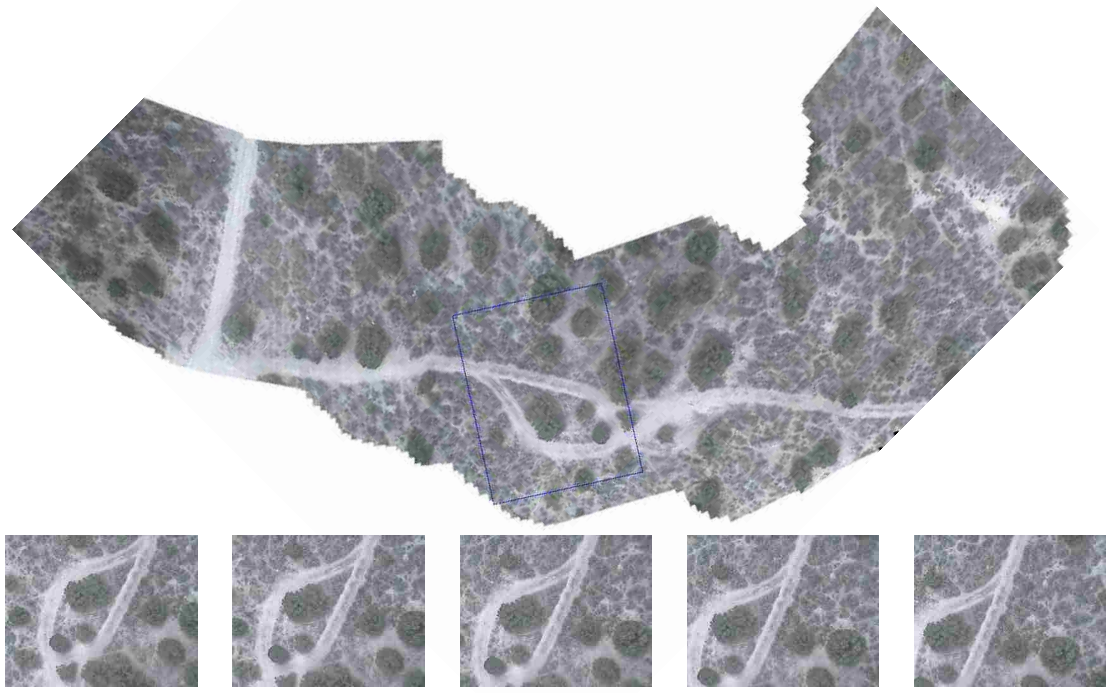


Figure 2.1: A 301-frame mosaic created using the methods described in this work, with the anchor frame outlined in blue. Beneath are five frames surrounding the anchor, acquired 1/30 of a second apart. At the velocity shown by the five frames, the entire visible scene would pass from view in approximately one second. Mosaicking dramatically extends the visible lifetime of the images.

for detection and identification of objects of interest. Visualizing the video using mosaics can extend the lifetime of scene points on the screen by more than an order of magnitude.

In this paper, we improve the construction and display of video mosaics. We apply a *hierarchical dense image correspondence* (HDC) algorithm, combined with strategic affine fitting, to produce accurate alignment of video frames. This approach is robust, even in the presence of motion parallax that defeats purely global alignment methods. Furthermore, it does not require the precise camera position or orientation information that may be available on more expensive platforms, but is typically not as common on mini-UAVs. In this dense correspondence setting, we can stabilize the mosaic by removing unwanted motions. In addition, with proper acquisition focus (over-sharp with respect to CCD resolution), the dense correspondence enables resolution enhancement and image noise reduction of the mosaic imagery. The availability of such *super-resolution* output can produce greater clarity for a given altitude, and could be used to allow higher flight and greater coverage of the search area on the ground. The HDC and mosaic display algorithm are designed for efficient execution on graphics processing units (GPUs), enabling real-time mosaicking and super-resolution using consumer hardware.

We demonstrate our mosaicking and HDC-based algorithms on real video data acquired in the field, including both mini-UAV video footage and small sequences acquired by a human with a Single Lens Reflex (SLR) camera in burst mode. For a variety of such data, our approach provides a more consistent visualization of the video data than simply playing back the frames.

2.3 Related Work

The creation of panoramas or mosaics from collections of images, including video, has been a well-studied area for more than 15 years [23, 25, 30, 29, 28]. (For an excellent survey of this broad area we recommend [31].) Mosaics have been used extensively as a means for summarizing aerial video sequences in ways that provide broader spatial context than individual frames [17, 21, 20], including providing a spatial index back to the source video [16]. They may also be used as a component in a larger strategy for visualizing video content [9, 5].

Once spatially aligned as with a mosaic, multiple video frames can also used to provide super-resolution, or greater resolution than a single source frame alone provides. Although there are a variety of approaches for aligning and warping the source images, modeling their respective point spread functions, and blending them [26, 19, 18, 13, 3, 8, 14, 6, 3, 10, 1], a key component in all methods is the accurate alignment of image content. Since most mosaicking approaches use a global transformation between the images, super-resolution from mosaics can often be limited by local deviations from the global transformation, typically the result of parallax or intrinsic motion in the scene. The HDC algorithm presented here, because it allows arbitrary per-pixel motions different from the general global transformation, can accurately align frames on a per-pixel basis and super-resolve the images even in the presence of minor parallax.

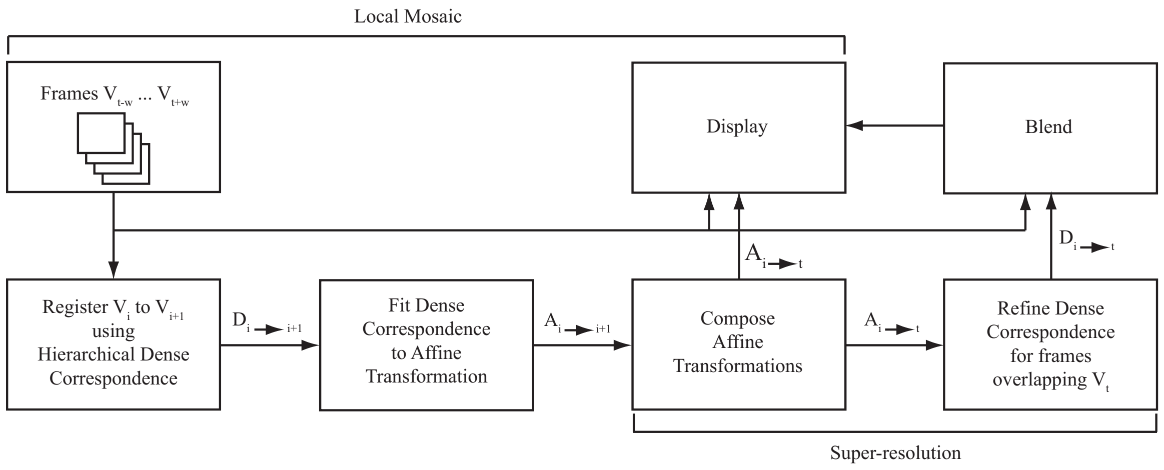
The alignment of frames through mosaics also facilitates the identification of intrinsic motion in the scene by separating out the frame-to-frame differences due to camera motion [2]. While the HDC algorithm allows minor deviations in motion due to parallax, larger motions can be detected and isolated to identify movement.

2.4 System Overview

The system presented in this paper produces a time-varying mosaic, M, from a video, V. Given a temporal window of $2W+1$ frames, the mosaic M_t is produced by combining the frames $\{V_{t-W},\ldots,V_t,\ldots,V_{t+W}\}$ into a single image, with V_t serving as the *anchor frame*. Playing back and interactively manipulating the mosaic sequence $M_{1\ldots N}$ provides a much better understanding of the imaged environment than playing back the original video sequence $V_{1\ldots N}$.

Our system employs two types of $2D$ transformations: hierarchical dense correspondences and affine transformations derived from these (see Figure 2.2). The dense correspondences produce pixel-level mappings between two images, often at sub-pixel accuracy. Without any direct computation of $3D$ depth or camera parameters, they can account for effects such as change of perspective and motion parallax. Affine transformations, on the other hand, provide a single, global mapping between two images that is compact to store and simple to compose to describe the transformation along a sequence of images. A useful tool in our approach is to compute the affine transform that is the best least-squares fit to a given dense correspondence mapping.

Figure 2.3 illustrates the transformations that need to be computed to create a mosaic and enable super-resolution and mover-detection. HDC is used to compute the dense correspondence $D_{i\mapsto i+1}$ between frames V_i and V_{i+1}. This, in turn, is fit to affine $A_{i\mapsto i+1}$. Each time our temporal window advances by a frame, we perform a single HDC and a



Local Mosaic
Frames $V_{t-w} \ldots V_{t+w}$
Display
Blend
Register V_i to V_{i+1} using Hierarchical Dense Correspondence
$D_{i \rightarrow i+1}$
Fit Dense Correspondence to Affine Transformation
$A_{i \rightarrow i+1}$
Compose Affine Transformations
$A_{i \rightarrow t}$
Refine Dense Correspondence for frames overlapping V_t
$D_{i \rightarrow t}$
Super-resolution

Figure 2.2: Overview of video visualization system. To create a temporally local mosaic each frame V_i is mapped to its successor V_{i+1} using HDC to produce $D_{i \mapsto i+1}$. This is then fit to an affine transformation $A_{i \mapsto i+1}$ (Section 2.6.1) and composed with $A_{t \mapsto i}$ to create an affine transformation that maps frame V_{i+1} to the anchor frame V_t (Section 2.6.2). Each frame is then displayed using its corresponding affine transformation to create the mosaic. To achieve super-resolution each dense correspondence $D_{i \mapsto t}$ is seeded with $A_{i \mapsto t}$ and then refined with HDC in order to map each pixel that overlaps V_t. Once registered to sub-pixel accuracy, the source images are blended using either a nearest neighbor or weighted average technique.

single affine fit to incorporate the new frame into the sequence. Composing $A_{t \mapsto i}$ with $A_{i \mapsto i+1}$ produces $A_{t \mapsto i+1}$, which will be used for displaying V_{i+1} as part of M_{t+1}. To display M_t, we apply $A_{i \mapsto t}$ to the corresponding frame i in the temporal window. These frames may be composited with one of several approaches. They may be simply overlaid using the painter's algorithm, the nearest sample may be selected for each output pixel, or a weighted combination may be taken for each output pixel. As we advance to M_{t+1}, we provide continuity by retaining the orientation of the new anchor frame, V_{t+1}, as it was displayed in M_t. We also translate the new anchor frame to the location of the old anchor frame to assist the user in keeping the entire mosaic on-screen.

Performing resolution enhancement requires somewhat more computation. In particular, it requires a dense correspondence $D_{i \mapsto t}$ for each frame contributing to M_t. Although it may seem conceptually possible to compose dense correspondences in a manner similar to the affine transforms, there are two noteworthy obstacles. First, we wish to avoid

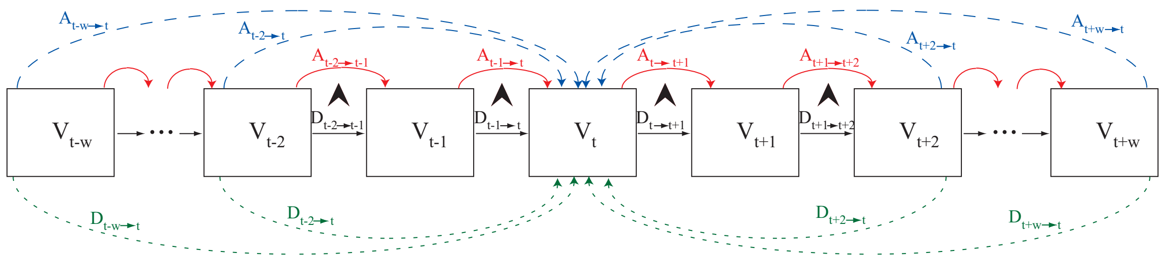


Figure 2.3: Summary of transformations. A dense correspondence is first calculated between each pair of neighboring frames (solid black arrows). This is then fit to an affine transformation (solid red arrows) and then composed together to create a direct transformation from each frame to the anchor frame (dashed blue arrows). These composed affine transformations are then used to seed the dense correspondence (solid green arrows) used in super-resolution.

any procedure that repeatedly resamples the images. Second, the correspondences are not easily composed without clipping to the boundaries of each successive image as the composition proceeds. Instead, we retain the previously described pipeline, but use each affine estimate $A_{i\mapsto t}$ to seed the HDC algorithm for the computation of $D_{i\mapsto t}$. Having a good seed for the HDC computation can be important as source and target are farther apart in the sequence or in the spatial domain. At display time, the HDC mappings are applied to the pixels of each image, and they are blended into a high-resolution output image.

2.5 Hierarchical Dense Image Correspondence

A fundamental building block of our mosaic system is an efficient and robust algorithm for computing dense (per-pixel) correspondences between a source and a destination image. As described in Section 2.4, this enables the mapping of a set of images to the domain of a selected anchor image, and provides the ability to perform super-resolution image enhancement and background/mover segmentation.

As with other dense correspondence algorithms, such as optical flow and stereo matching, the HDC algorithm has a matching component that produces a motion field, and a regularization component that straightens the warp. One key difference from existing algorithms is that the matching and regularization components are decoupled and occur

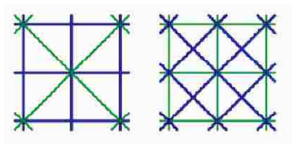


Figure 2.4: Parent (green) and child (blue) levels of a diamond hierarchy. Left: level i. Right: level $i+1$

during two distinct passes. The reason this decoupling is important is that it makes the algorithm amiable to implementation on the GPU where the required calculations are easily distributed to the processing cores to be calculated in parallel, providing a significant acceleration over a CPU-only implementation.

2.5.1 Hierarchical Algorithm

The input to the correspondence algorithm is a source and target image. We currently perform correspondence on greyscale imagery and assume that intensity levels across the sensor field have had proper calibration applied. The output of the algorithm is a 2D coordinate for every source pixel indicating its forward correspondence into the target domain. In our current formulation, this warping function is presumed to be a bijection (i.e. the mesh of warped samples has no folds or gaps). This is of course not true in general, but we can get quite far under this assumption, especially for aerial imagery.

We begin by constructing a diamond hierarchy for the source and target images (see Figure 2.4). Each successive level is low-pass filtered and contains half as many samples as the previous one, similar to Hwa et al. [15]. Notice that in our application, we are applying the hierarchy for the purpose of efficient computation and not just for visual display. As compared to a standard image pyramid, the diamond hierarchy improves the prediction quality as we advance from level to level, and reduces potential grid artifacts in the results.

After hierarchy construction, we apply an iterative, two-phase algorithm at successively finer levels of the hierarchy (see Figure 2.5). Starting at the coarsest level, we initialize the transformation from the source to target pixel locations to either an identity

```
HierarchicalDenseCorrespond($V_{src}, V_{dst}, A_{src \mapsto dst}$)
{
  for $\ell \leftarrow \ell_{coarse}$ to $\ell_{fine}$ {
    if ($\ell = \ell_{coarse}$) {
      if ($A_{src \mapsto dst}$ is non null) $D_\ell \leftarrow$ sample($\ell, A_{src \mapsto dst}$)
      else $D_\ell \leftarrow$ sample($\ell, I$)   // $I =$ identity mapping
    }else $D_\ell \leftarrow$ split($D_{\ell-1}$)   // refine next coarser mapping

    for $j \leftarrow 1$ to $j_{max}(\ell)$ {

      // perform matching motion per pixel
      Move each source pixel's destination position in $D_\ell$ towards closest matching
      isoline in destination image $V_{\ell,dst}$. Clamp motion to a fraction of a pixel width,
      and avoid motion for destination positions with small gradients or missing/bad
      pixels.

      // straighten warp mesh
      Perform least-squares affine fit $A_p$ for the $5 \times 5$ neighborhood centered at each
      source pixel $p$ in $D_\ell$. For each source pixel $p'$ store the average of the affine
      positions computed from the $5 \times 5$ neighborhood of local affine fits $A_p$ centered
      around $p'$.
    }
  }
  return $D_\ell$
}
```

Figure 2.5: The Hierarchical Dense Correspondence processing starts at a coarse resolution with either an identity mapping or an affine mapping estimate from the application. It subsequently performs several iterations per resolution level, alternating per-pixel motions to match intensity and mesh straightening, working to finer and finer levels by splitting the coarser level as a prediction of the next finer level. Typically around ten iterations are required for convergence at the finer, most expensive levels, and coarser levels can have a hundred or more very inexpensive iterations for robustness.

transform or to an appropriate affine transform, if one is available. Then we perform some number of iterations, each of which consists of a *matching phase* followed by a *straightening phase*. The resulting transformation for level i of the hierarchy is used as a starting prediction at level $i+1$ using mesh refinement (see Figure 2.6).

The matching phase is a gradient descent algorithm. For each pixel of the source image, we compute a local gradient at its current position in the target image. We use the

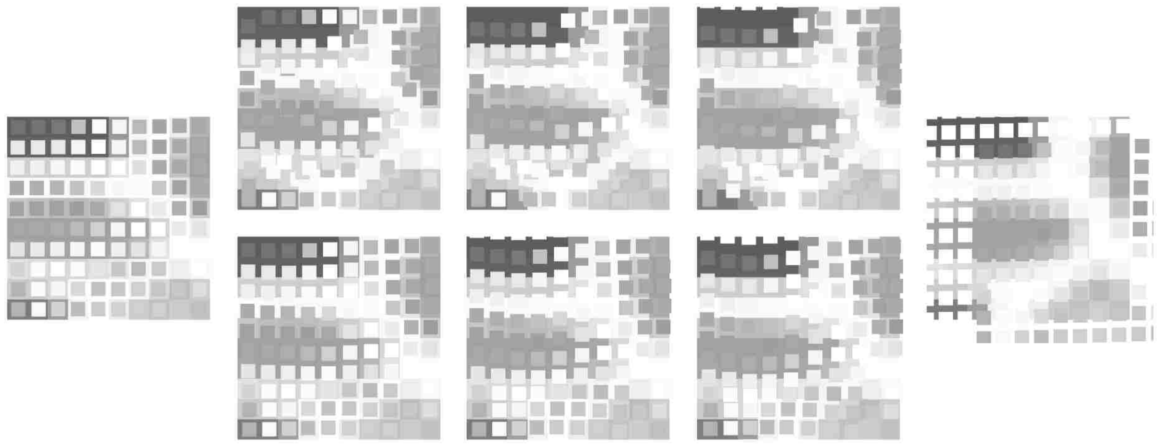


Figure 2.6: HDC stages. Left: source image (smaller squares) super-imposed on destination image. Center: two phases of the HDC algorithm. The top row demonstrates how the pixels move independently during the matching phase, while the bottom row shows the pixel locations after the straightening phase. Right: after several iterations of alternating the matching and straightening phases, the images are aligned. The resulting locations are then propagated down to the next finer level, and the process repeats.

gradient to make a linear prediction of the direction and distance to move the source pixel in the target image to match its intensity. To promote robustness, the size of the step actually taken is clamped to a fraction of a target pixel size. As the gradient magnitude becomes small, the gradient direction becomes more noise than signal, so we disqualify such pixels, as well as those that go outside the bounds of the target image, from motion during the matching phase.

Whereas each source pixel moves independently in the matching phase, the straightening phase uses information about the current locations of source pixel neighborhoods to locally regularize the warp. For each source pixel, a local neighborhood of locations is used to compute an affine transformation from the source to target image (empirical observations show that a 5×5 neighborhood is sufficiently large). The new target location of each source pixel then becomes a weighted average of its target locations as predicted by all the local affine transformations to which it contributes.

2.5.2 GPU Implementation

The architecture of the commodity Graphics Processing Unit (GPU) is ideally suited to accelerate the dense correspondence algorithm. Modern GPUs provide hundreds of gigaflops of processing power, driven by a large number of floating point processing cores (e.g. 128 cores on the NVIDIA 8800 GTX). Furthermore, GPUs are designed to provide highly optimized memory access for programs employing 2D local access patterns. The correspondence algorithm is not significantly hindered by the limitations of the standard GPU programming model, such as reduced efficiency for incoherently branching code and lack of support for scattered write operations or inter-thread communication (though this is actually improving on the latest GPUs).

Both the initial hierarchy construction and the iterative correspondence algorithms map well to the fragment processing units on the GPU. The hierarchy construction requires one “rendering pass” (computational kernel execution) for each level of the hierarchy.

The iterative algorithm is broken into passes for each phase of an iteration. The matching phase, which requires a single pass, maps a fragment thread to each source pixel. A thread reads a source pixel intensity and the current correspondence location in the target image. A gradient is computed from the intensity values at the current target location and this is used to compute the new $2D$ target location for the pixel, which is output to a $2D$ texture via a frame buffer object.

The straightening phase requires two passes. The first pass maps a fragment thread to each source pixel. For neighboring pixels (using a 5×5 neighborhood), the current warp locations are read and contribute to a 6-parameter affine transformation, which is output from the first pass. The second pass also maps a fragment thread to each source pixel. The affine transformations computed in the first pass are read for each member of the source pixel's 5×5 neighborhood. Each of these affine transformations is applied to the thread's source pixel, and a weighted combination of the results is used to compute the pixel's location after the straightening phase, which is output from the fragment program.

We have benchmarked the two algorithm phases on an NVIDIA Geforce 8800 GTX GPU. The matching phase and straightening phases achieve rates of 145 million and 68 million pixel-iterations per second, respectively. This performance is over 230 times that of our unoptimized CPU implementation running on a single processor of an Intel dual-Xeon 3.0 GHz.

2.6 Local Mosaics

Creating a temporally local mosaic of hundreds of video frames in real-time requires a fast and robust algorithm. While a perspective transformation more accurately describes the relationship between successive frames in the video, its non-linear parameters prevent us from calculating the perspective transformation analytically. An iterative solution is required to estimate the non-linear parameters. Since, following the pattern in [24], our goal is not to create a globally correct mosaic but rather to create a temporally local mosaic in real-time, we simplify the transformation between frames to an affine transformation. This allows us to analytically fit the dense correspondence to an affine transformation and compose chains of frame-to-frame affine transformations so we can display several hundred frames of the video in the same coordinate system.

2.6.1 Affine Transformation Estimation

The first step in creating a local mosaic is to create a transformation between each pair of neighboring frames in V. The kinematics of the mini-UAV require that the current frame-to-frame transformation is similar to the previous one. Thus, we initialize the dense correspondence $D_{i \mapsto i+1}$ with the previous affine transformation $A_{i-1 \mapsto i}$. (If $A_{i-1 \mapsto i}$ has not yet been calculated, we initialize $D_{i \mapsto i+1}$ with the identity mapping.) We then run the HDC algorithm to get the dense correspondence $D_{i \mapsto i+1}$ between V_i and V_{i+1}.

In order to create a mosaic that can extend beyond the limits of our anchor frame, we need to combine a chain of correspondences that relates each frame to its neighbors.

Although it is possible to compose the HDC mappings directly, this is computationally very expensive and does not extrapolate well beyond the edges of the anchor frame. To overcome these limitations we reduce each HDC to an affine transformation using a least-squares fit technique. This is accomplished by multiplying the HDC array by the pseudo-inverse matrix to obtain the six parameters of the affine transformation. The x and y components, A^x and A^y of the affine transformation $A_{i\mapsto i+1}$ are calculated independently, as

$$A^x_{i\mapsto i+1} = D^x_{i\mapsto i+1} M (M^T M)^{-1} \tag{2.1}$$

$$A^y_{i\mapsto i+1} = D^y_{i\mapsto i+1} M (M^T M)^{-1} \tag{2.2}$$

where $D^x_{i\mapsto i+1}$ and $D^x_{i\mapsto i+1}$ are the x and y components of $D_{i\mapsto i+1}$, respectively, and M is the N by 3 matrix of pixel locations (x_i, y_i)

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_0 & y_1 & 1 \\ & \vdots & \\ x_0 & y_{N-1} & 1 \\ x_1 & y_0 & 1 \\ & \vdots & \\ x_{N-1} & y_{N-1} & 1 \end{bmatrix} \tag{2.3}$$

giving us an affine transformation $A_{i\mapsto i+1}$ that maps each frame to its successor.

2.6.2 Composing Affine Transformations

To correctly display all the frames composing the local mosaic, we need to combine the frame to frame transformations $A_{i\mapsto i+1}$ to create a transformation $A_{i\mapsto t}$ that maps each frame V_i directly to the anchor frame V_t. We can do this by chaining together, or composing, the frame-to-frame affine transformations, in the following manner:

$$A_{i \mapsto t} = \prod_{k=i}^{t-1} A_{k \mapsto k+1} \tag{2.4}$$

Our goal in creating this temporally local mosaic is to provide additional context for the currently viewed frame while allowing features in the video to persist longer on the screen. When transformations are composed without doing a bundle adjustment, small errors in the transformations will accumulate and distort frames that are temporally distant from the anchor frame. While small distortions are not necessarily inconsistent with our goals, we would like to minimize distortions as much as possible, while maintaining the real-time display of the mosaic. Thus, we select the middle frame of the video sequence V to be the anchor frame, allowing both V_{t-W} and V_{t+W} to be as close to the anchor frame as possible. Since the affine transformations calculated above are only calculated in one direction, we need to invert the affine transformations $A_{i \mapsto t}$ where $i > t$ before doing the composition. We can calculate $A_{i \mapsto t}$ from the previous composition, $A_{i+1 \mapsto t}$, as

$$A_{i \mapsto t} = \begin{cases} A_{i+1 \mapsto t} A_{i \mapsto i+1} & \text{if } i < t \\ A_{i-1 \mapsto t} \hat{A}_{i-1 \mapsto i} & \text{if } i > t \\ I & \text{otherwise} \end{cases} \tag{2.5}$$

where $\hat{A}_{i-1 \mapsto i}$ is the inverse of $A_{i-1 \mapsto i}$ and I is the identity transformation.

Each time t advances, we only need to compute one new frame-to-frame affine transformation in order to display the complete mosaic M_t. The frame-to-frame affine transformations are composed according to Equation 2.5 and then displayed as described in the next section.

2.6.3 Mosaic Display

With each frame V_t now aligned to a common coordinate system through the affine transformation $A_{i \mapsto t}$ they next need to be composited into a single image to create the mosaic M_t.

There are many ways to accomplish this blending (see [31]). We have experimented with three methods of displaying the overlapping frames of the mosaic, namely the painter's algorithm, nearest neighbor selection, and using a weighted combination of the overlapping source images. Our experience has indicated that the painter's algorithm produces satisfactory results and is the fastest of the three methods.

While mosaics can aid in the building of a mental model of the scene, there is a possibility of creating artifacts during the mosaic process, especially in frames far from the anchor frame. In order to ensure that each frame appears undistorted at some point during the playback of the mosaic we keep the anchor frame undistorted at all times. We can, however, filter out unwanted motions by allowing the anchor frame to translate. Experience has shown that it can also be helpful to unwind the rotations, caused by the turning of the plane, by allowing the anchor frame (and hence the mosaic) to counter-rotate. Thus, we can maintain a consistent orientation. Users are also allowed to zoom and pan the mosaic as desired. When the user zooms in far enough so that the anchor frame is occupying more screen real estate than its original dimensions, we then transition to displaying the mosaic in super-resolution.

2.7 Super-Resolution

A key element of Wilderness Search and Rescue is covering the search area as quickly and as thoroughly as possible. Increasing the area that the mini-UAV covers on each pass while still maintaining the resolution needed to identify ground features will shorten the time to search a given area. Super-resolution techniques can enable the mini-UAV to fly at higher altitudes while providing the same resolution as if it was flying at a lower altitude.

We can achieve this increased resolution by combining the information from overlapping video frames. The movement of the mini-UAV causes each frame to be captured from a slightly different viewpoint. As a result, pixels in each source image are integrating over a finite area of the scene that is slightly offset from the other images.

To create a super-resolution image it is necessary to calculate a very accurate alignment for each of the source images that overlaps the anchor frame. This allows us to take advantage of the additional information gained from repeatedly sampling the scene. Significant parallax in the scene due to the movement of the mini-UAV and the depth variation in the scene can cause alignment problems for global registration techniques. The local warps that the HDC approach allows, however, can accommodate parallax motion and give us the needed accuracy in alignment.

To generate a resolution-enhanced image we begin with a collection of overlapping images. One of the images is chosen as the anchor frame and the other frames are warped into the anchor frame's coordinate system. The affine transformations that we calculated during the mosaic construction ($A_{i \mapsto t}$) are used to initialize the HDC warps ($D_{i \mapsto t}$), allowing us to robustly warp frames that only slightly overlap the anchor frame. We then refine the registration using the HDC algorithm to align the images to sub-pixel accuracy.

Once all of the dense correspondences are calculated we combine the source images to create a super-resolution image. Each pixel of each source image is displayed on screen at their floating-point coordinates (Figure 2.7). The pixels are displayed as point sprites with a finite area. Each screen pixel then selects for its color the source pixel it is closest to (nearest neighbor) or a blending of the source pixels that are close by (weighted average). This blending process can be accelerated using a Graphics Processing Unit (GPU) to keep the display of the super-resolution image at interactive speeds.

One indicator of the accuracy of the dense correspondence computation is the ability to create super-resolution images by warping multiple images to a single frame of reference and compositing, as shown in Figures 2.8 and 2.9.

2.8 Mover-Detection

Segmentation of moving objects is performed in two phases: detection and completion. In the detection phase, the goal is to have at least a single pixel of positive detection per

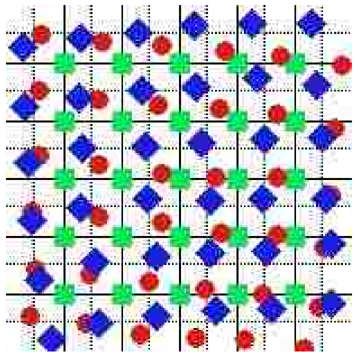


Figure 2.7: Merging multiple image sources into a super-resolution image. The shapes (green squares, red circles, or blue diamonds) represent pixels from separate but overlapping image sources. The solid lines represent the resolution of the input images. The dotted lines represent additional sampling points in the higher resolution image. The pixel color at each intersection is chosen using nearest neighbor or weighted average blending.

moving object, not including parallax motion of buildings, trees, or other tall structures. In the completion phase, pixels for the entire object are determined. Extra "guard pixels" surrounding the object are also labeled as part of the mover to obtain a conservative segmentation. Both phases of this algorithm could be accomplished with well-known but expensive search and correlation strategies (see [7, 22]). For real-time processing on small sensor platforms, fast, local computations are desired.

For detection, two approaches were tested. The first approach uses two buffered frames to perform detection. Pixels are declared likely movers in frame f if (1) their value $v_f(x,y)$ is different from all the values within one pixel in frame $f+1$, (2) this frame-to-frame difference has a high gradient magnitude, (3) the gradient magnitude at frame f is high, and (4) the gradient directions in the two cases above are parallel to one another (either the same or exact opposite directions). This is formulated as

$$\begin{aligned} \vec{g} &= \nabla v_f(x,y) \\ \vec{g}' &= \vec{g}/||\vec{g}|| \end{aligned}$$

$$d_f(x,y) = \min_{dx,dy\in[-1,1]} |v_f(x,y) - v_{f+1}(x+dx,y+dy)|$$

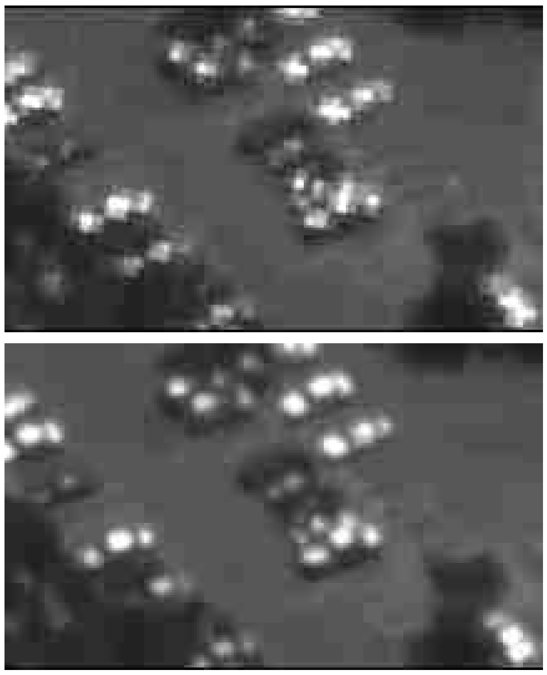


Figure 2.8: 61-frame super-resolution example. Top: one image. Bottom: 61 images warped and composited into a high-resolution image (from the same video sequence as Figure 2.10).

$$\begin{aligned}\vec{h} &= \nabla d_f(x,y) \\ \vec{h}' &= \vec{h}/||\vec{h}||\end{aligned}$$

$$q = (\vec{h}' \cdot \vec{g}')(||\vec{g}|| - g_{m\,\mathrm{min}})||\vec{h}||$$

This consistently finds some pixels per mover but is somewhat sensitive to noise and does not completely eliminate parallax-induced detections of tall building edges.

The second detection approach uses five frames. The median value is obtained for a pixel over these frames. The detection is then the difference between the current frame's value and the least different value from a small neighborhood of the next frame. A small amount of "soft erosion" is performed on the results. This was found to be very reliable at detecting some pixels per mover and was not as sensitive to noise or to building edge parallax.

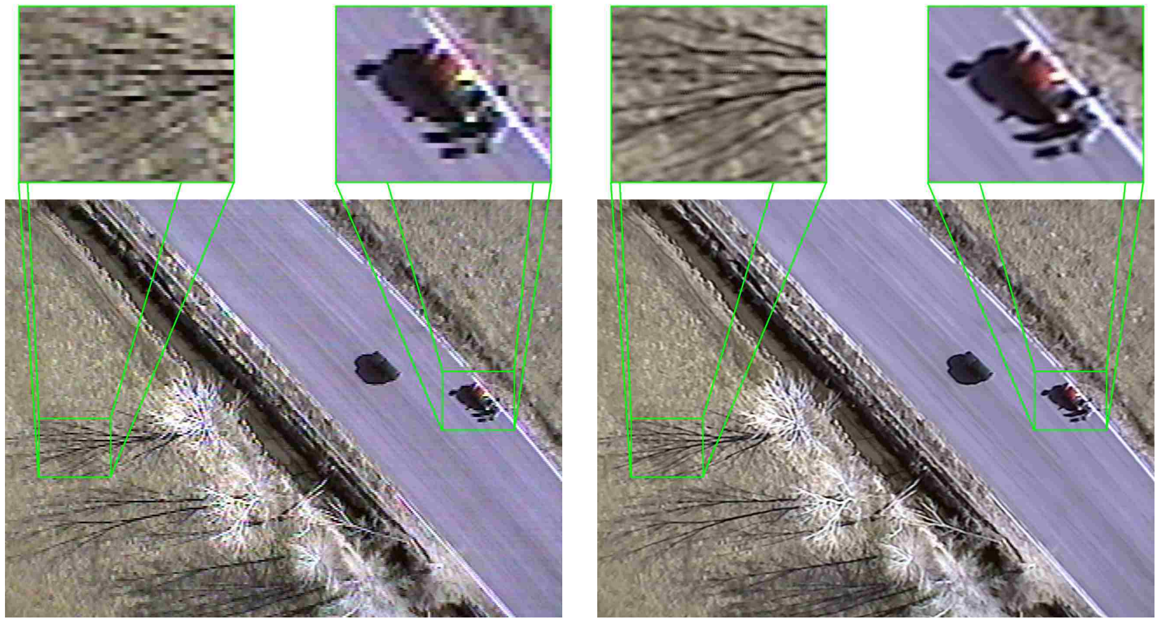


Figure 2.9: 15-frame super-resolution example. Left: example of one of the 15 input frames. Right: super-resolution output. Notice the additional detail in the shadows of the trees and around the red car, as shown in the magnified insets. Also notice how the color artifacts caused by the Bayer filter array in the camera sensor are removed in the super-resolution version, especially around areas that transition from light to dark.

After detecting a set of pixels from each mover, a conservative region of pixels is determined around each complete mover. Using the results of the five-frame detection, there are good starting points to seed a search for all mover pixels. We start with the core pixel of each connected component, defined as the last pixel to be deleted for the component by repeated erosion operations that are restricted to not break components into two pieces. From this starting point, we flood fill to all pixels that are near pixels with frame-to-frame differences with magnitudes above a specified threshold.

Figure 2.10 shows the differences in mover-detection accuracy using a dense correspondence registration versus an affine warp. The dense correspondence matches the background scene much better and eliminates the vast majority of false alarms caused by parallax motion.

2.9 HDC Improvements Over Global Registration

The HDC algorithm provides several advantages over global image registration techniques. HDC can account for parallax motion to provide a more accurate registration. This is important for both mover-detection and super-resolution. Figure 2.10 shows a significant number of false positives when performing an image difference using a global registration technique. This makes it extremely difficult to determine which differences are due to movement of objects within the scene (moving cars in this case) and which differences are due to the parallax motion of tall objects in the scene (tall buildings and elevated tracks). Contrast this with the HDC result where the true movers are much easier to detect because the parallax motion has been eliminated.

When combining multiple, overlapping images to produce a super-resolution result, parallax motion can cause blurring in the final result. This blurring is evident in the super-resolution image produced using affine transformations in Figure 2.11. The super-resolution image created using HDC produces a sharp image, even in the presence of parallax motion. This result is possible because the HDC algorithm has the flexibility to locally warp to movement that is not present in other areas of the images.

2.10 Conclusion

In this paper we have presented an improved approach to video visualization using mosaics. Our approach, based on a judicious combination of affine transformations and hierarchical dense correspondences, performs robustly even on mini-UAV video with no extrasensory information and in jittery flight conditions. The same framework permits not only basic mosaics but fast computation of super-resolution imagery on the GPU, as well as mover-detection. Non-traditional means of visualizing video data, including the mosaics presented here, provide promise of conveying more information to human observers with reduced dependency on temporal viewing.

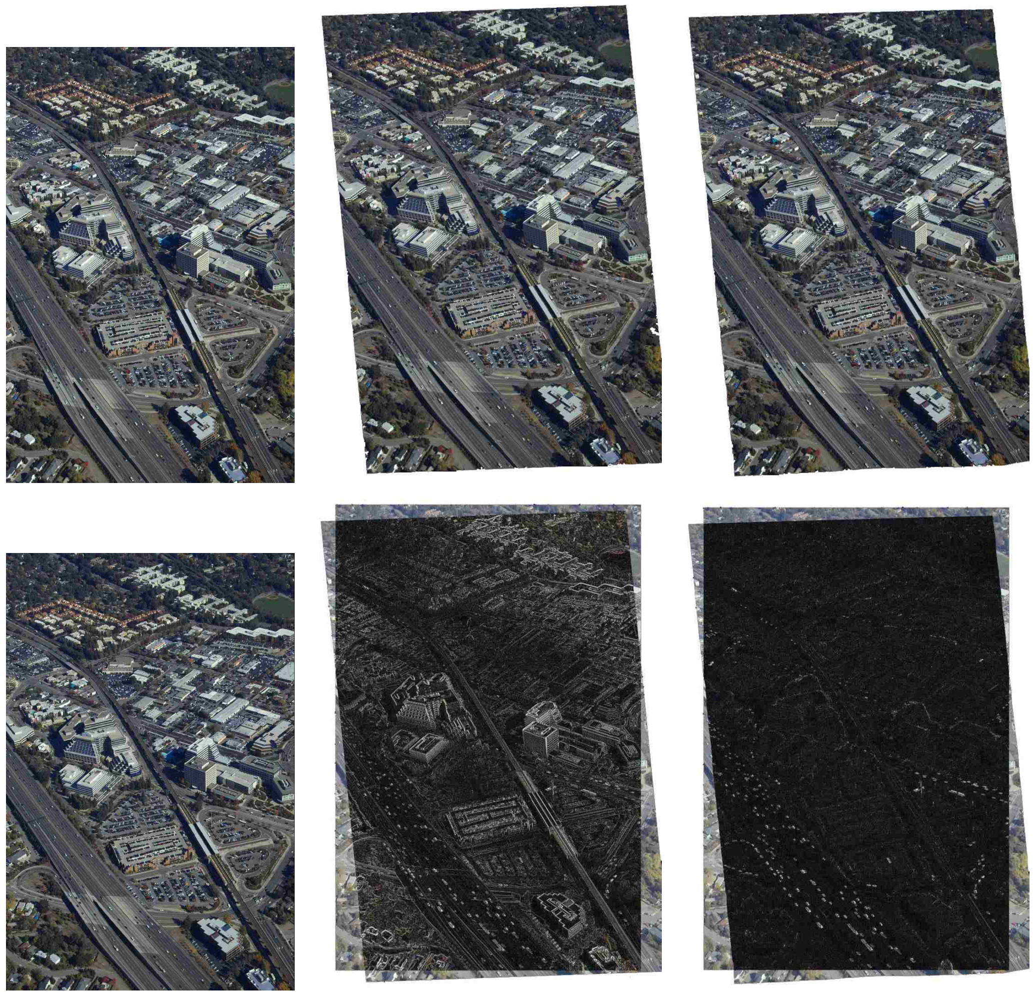


Figure 2.10: Affine warp versus dense correspondence for mover-detection. Left: source and target images. Middle: affine warp with difference image. Right: dense correspondence warp with difference image (significant differences are movers).

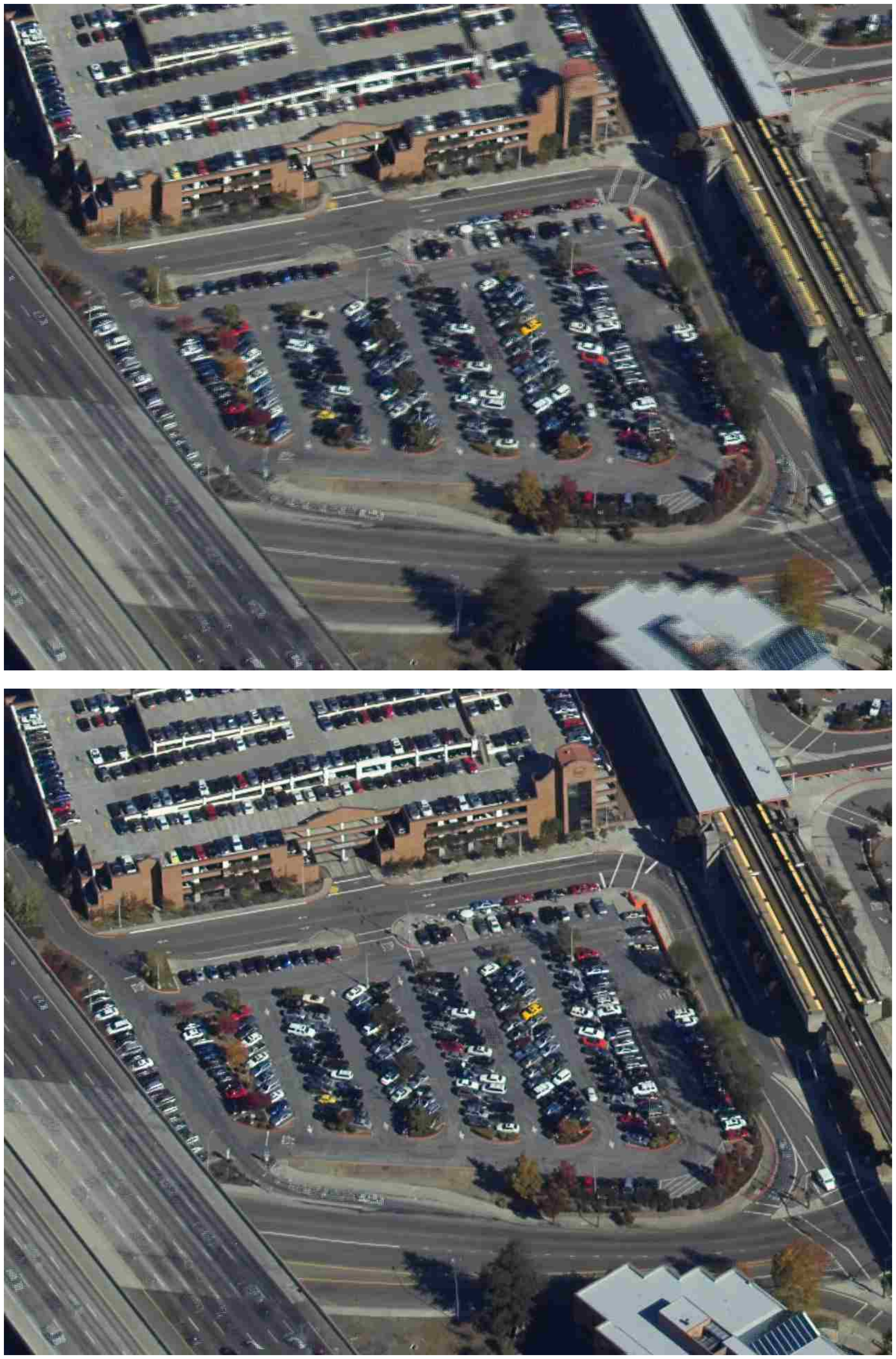


Figure 2.11: Affine warp versus dense correspondence for super-resolution. Top: The tall building in the lower right corner is blurry when using an affine transformation to register the input images. Bottom: the building remains sharp when HDC is used for registration.

Chapter 3

Additional Contributions

This chapter contains details about the individual contributions outlined in Section 1.3.1 that were not included in the paper submitted to CVPR due to space limitations.

3.1 Extending Local Mosaics

There are two innovations that allow us to increase the number of frames that we are able to display in our local mosaics while maintaining minimal distortion. One technique is to continuously re-warp the frames in the history to the coordinate system of the current frame. The other technique extends the mosaic to include both a history of frames as well as a buffered future history when running in offline mode. Both techniques rely on transferring away from the current frame the distortion that builds up when doing purely frame-to-frame alignment with no bundle adjustments.

3.1.1 Continuously Warping History to Current Frame

The approach taken in [11] while constructing the video mosaic is to paste each frame onto a static canvas that each subsequent frame is then aligned to. This techniques works well for short sequences when using a transformation that only allows for translations and rotations. While transformations that allow for greater degrees of freedom, such as affine transformations, can create a more seamless mosaic, they also allow for a greater degree of distortion. While the frame to frame distortion is usually very small, the accumulated

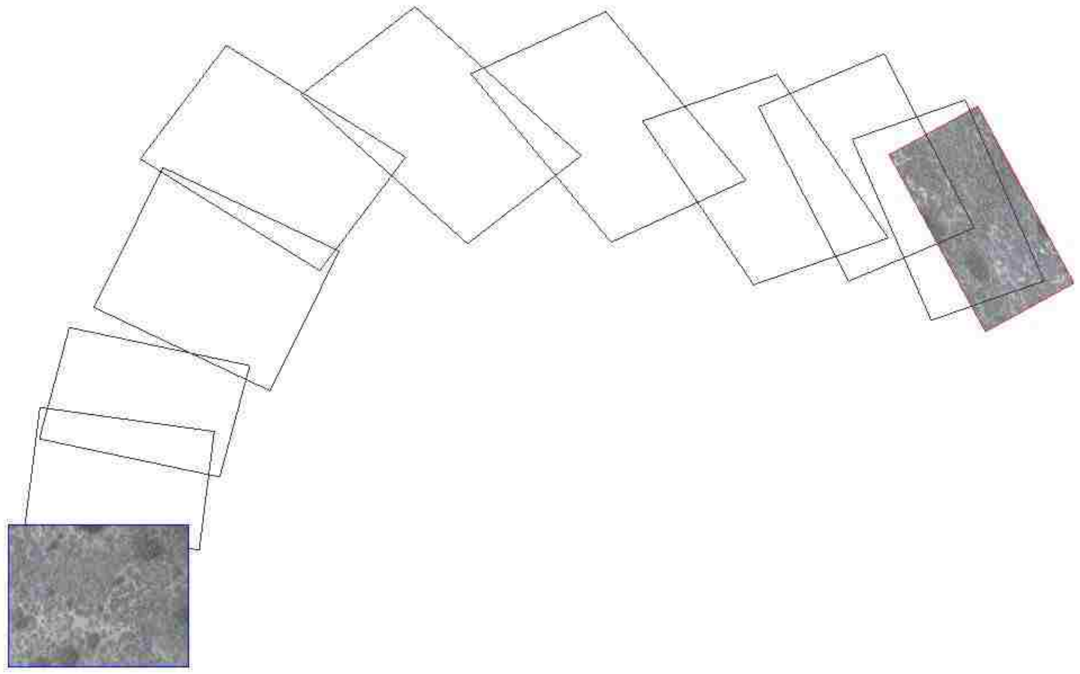


Figure 3.1: Mosaic distortion due to accumulated errors. The first frame pasted onto a static canvas, outlined in blue, is undistorted. As each subsequent frame is aligned to the frames that were previously pasted onto the canvas, small errors in the alignment accumulate and the distortion becomes noticeable. In this example the black rectangles represent the position of every 25th frame, and the 250th frame, outlined in red, is noticeably stretched in one direction due to accumulated errors. If we continued and aligned additional frames to this static canvas then the mosaic would continue to become more and more distorted.

effect over long sequences of several hundred frames can grow to be quite large. If we were to continuously paste frames onto a static canvas using affine transformations, the accumulated distortion would become very noticeable after several hundred frames. Figure 3.1 shows the distortion that is evident after 250 frame to frame affine transformations. The frame outlined in blue was the first frame to be drawn onto the canvas. The black rectangles outline where every subsequent 25th frame would be pasted. The current frame is outlined in red and is the 250th frame in the sequence. By this point the frame is noticeably stretched in one direction. If future frames are aligned to this static canvas, they inherit the accumulated distortion of all the frames already drawn on the canvas.

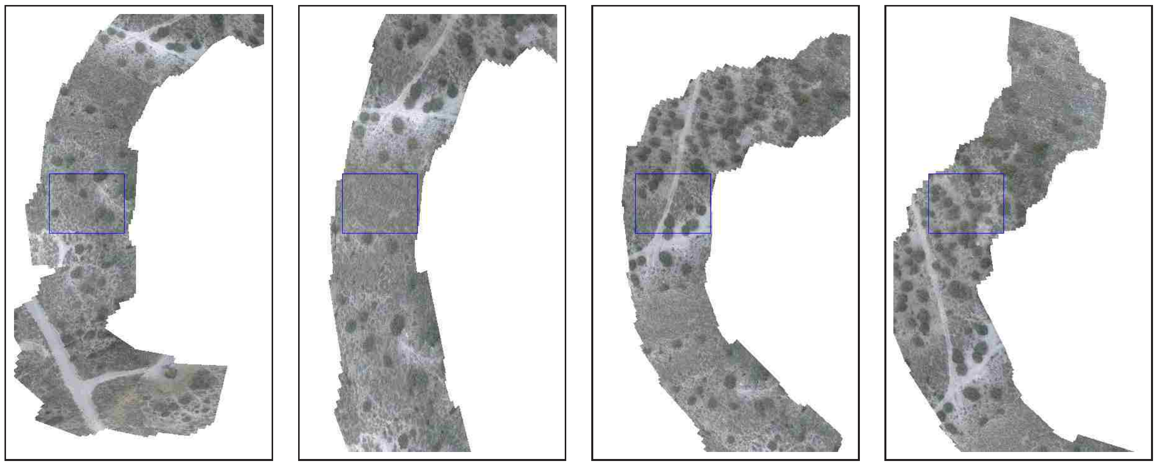


Figure 3.2: Continuously warping the mosaic to the anchor frame. Each image shows a snapshot of the mosaic after 30 frames have elapsed. The anchor frame, outlined in blue, remains stationary as the mosaic "plays through" that fixed location.

In order to mitigate this distortion we continuously warp the history to the current frame. This allows us to draw the current frame on the canvas without any distortion. Frames that are temporally close to the current frame will have minimal distortion, while the accumulated error gets pushed out to the frames that are temporally distant from the current frame. This technique also allows the current frame to remain undistorted throughout the entire length of the video. In Figure 3.1, if we consider the frame outlined in red as the first frame that we received and the frame outlined in blue as the current frame, then we can paste the current frame undistorted onto the canvas and warp the 250 previously received frames to this anchor frame. Then when the next frame is received it becomes the new anchor frame and the frames in the history are warped to align to it. This re-warping to a new anchor frame is accomplished efficiently as described in Section 2.6.2.

While it is difficult to represent using still images, Figure 3.2 gives a general feel for what the mosaic looks like as it plays and is warped to align to each succeeding anchor frame outlined in blue.

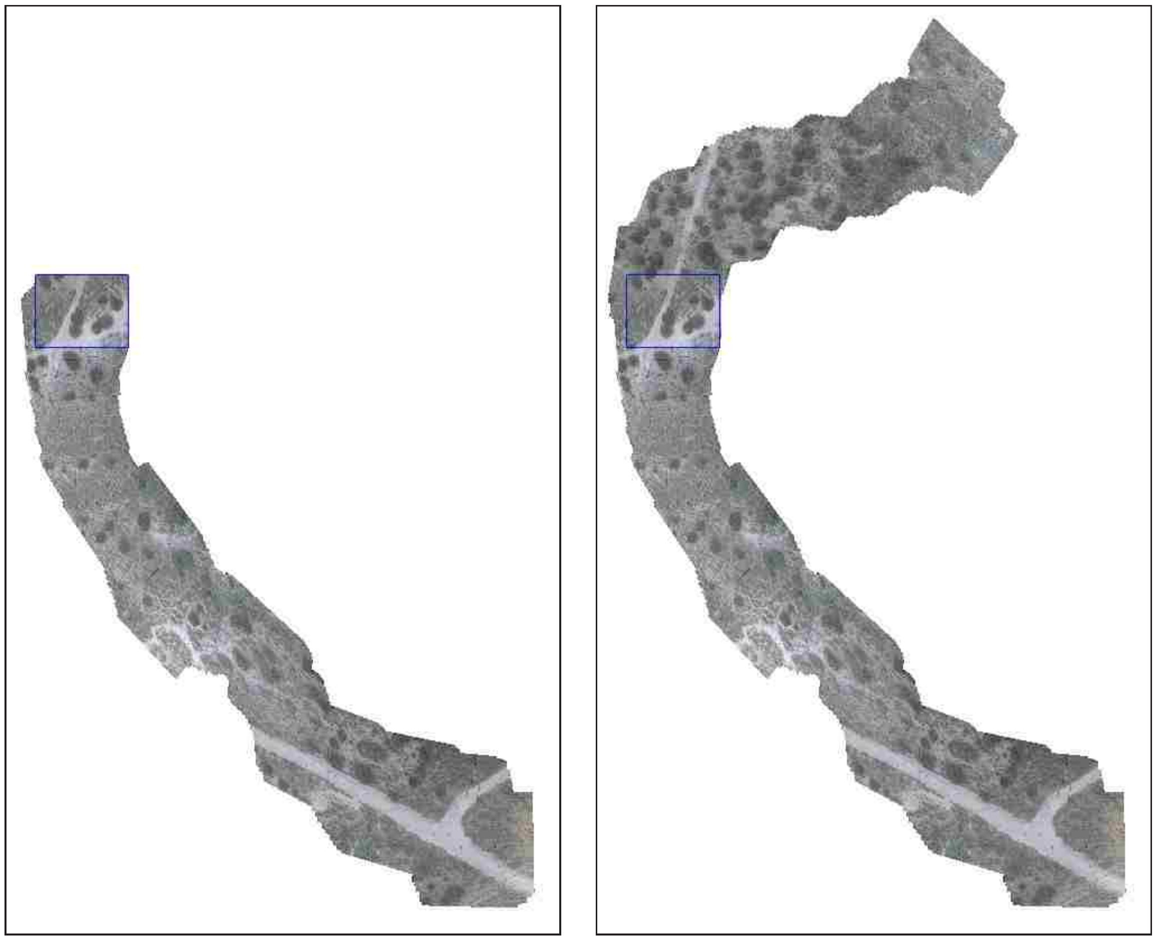


Figure 3.3: Extending the mosaic in offline mode. Left: the mosaic only includes the 250 frames that preceded the anchor frame, outlined in blue. Right: the mosaic includes the 250 frames preceding and 250 frames following the anchor frame.

3.1.2 Displaying History and Future in Offline Mode

We can further extend the mosaic while maintaining minimal distortion when we are in offline mode and have access to the frames both preceding and following the current anchor frame. With access to these frames we can build up the mosaic on both sides of the anchor frame to create a mosaic that is even longer than if we only used the frames that preceded the anchor frame. Figure 3.3 shows an example of a mosaic using only the frames that precede the anchor frame and what the same mosaic looks like if we include the frames that follow the anchor frame.

3.2 Using Mosaic Results to Initialize HDC

In order to get the best possible super-resolution results for a given feature in our aerial video we need to use every input frame that overlaps that feature. This means that we need to align input frames that may be temporally distant from each other to a common frame of reference. While HDC can robustly align images that have a high degree of overlap, when the overlap is not sufficient enough it may produce an incorrect alignment, as in Figure 3.4. In order to ensure that HDC produces the correct alignment even in these cases, we take advantage of the alignment information that was produced during the building of the mosaic. The affine transformations calculated to bring the frames in the mosaic into the coordinate system of the anchor frame (see Section 2.6.2) can be used to give an initial alignment for the HDC algorithm. This alignment is then refined to produce the dense correspondence needed to produce the super-resolution results.

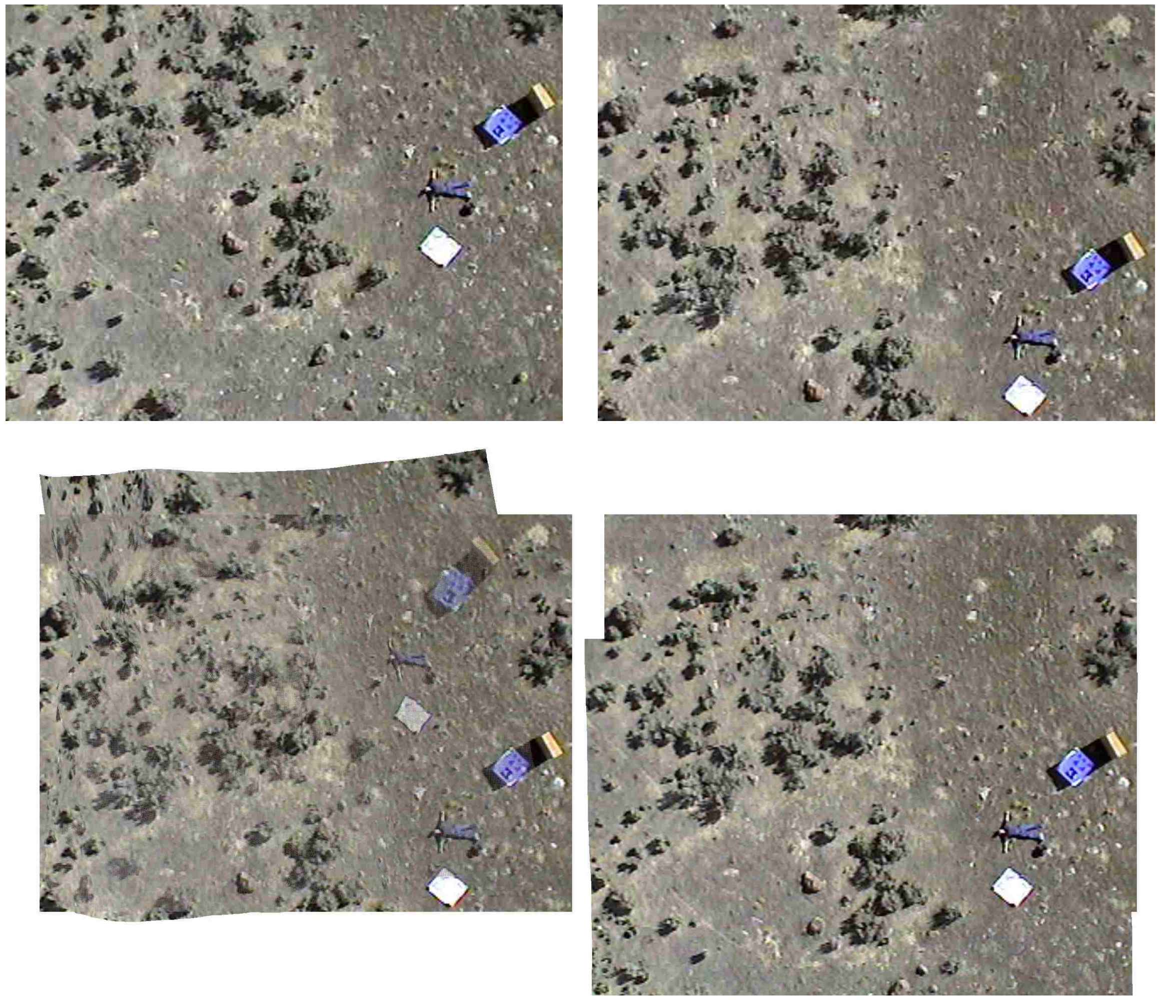


Figure 3.4: Using Mosaic Results to Initialize HDC. Top: two input frames that are temporally distant. Bottom left: HDC alignment when initialized with the identity transformation. The algorithm incorrectly locks on to a local minimum. Bottom right: HDC alignment when initialized with the alignment calculated during the creation of the mosaic. The algorithm correctly aligns the images.

Chapter 4

Additional Methods

This chapter contains additional details about our methods that were not included in the paper submitted to CVPR due to space limitations or intentionally omitted because they are basic concepts generally understood throughout the computer graphics and vision communities. They are included here for completeness.

4.1 Software Libraries

The video interface makes use of utilities and library functions made available through the LibGen library, available for download at http://www.cognigraph.com. The videos were pre-processed using MPlayer to extract individual video frames from movie sequences. The MPlayer application is available for download at http://www.mplayerhq.hu.

4.2 Video Pre-processing

In order to simplify the video interface and focus our efforts on the visualization of the video, we pre-processed the videos and stored the results on disk for use during video playback. The first step in our pre-processing was to extract each video frame and store it as an image file. Each image file was given a unique file name that was identified by the frame number that the particular image was extracted from. This allowed us to play each frame of the video in sequence without having to decode each frame each time we ran the software. The frames were extracted offline using the MPlayer software application. The benefit from doing this was to avoid incorporating software to handle the decoding of the video in our project. One drawback from doing this was the performance hit that came in the form of increased I/O from disk. Since we stored each frame in uncompressed format, we were required to read a greater number of bytes from disk than if we had extracted the frame direct from the video.

The next step during pre-processing was to de-interlace the video frames, if necessary. The odd scan lines were extracted to one image file, while the even scan lines were extracted to a separate image file. The missing scan lines were filled in using a simple center-weighted average of the three pixels above and three pixels below the pixel to be filled in.

In order to compensate for changes in the camera gain and bias during the recording of the video, the final step of our pre-processing was to perform an equalization operation over the entire video frame. This operation examines a neighborhood around each pixel, ordering the pixels in the neighborhood based on their intensity values. The value of the pixel is then reassigned to be the percentile of its rank among its neighbors. The `equalize` utility in LibGen was used to perform this step, with a neighborhood size of 64.

4.3 GPU Accelerated Super-Resolution

In Section 2.7 we mentioned that overlapping images could be combined to generate a super-resolution image. Figure 2.7 illustrates how individual pixels from overlapping images are positioned using their respective dense correspondence positions. To create a super-resolution image we harness the power of modern graphical processing units (GPUs) to display each of the source images on the display, using *fragment programs* to blend the source images where they overlap.

The blending can be accomplished through a variety of techniques. We have experimented with using *nearest-neighbor selection* and *normalized weighted averaging*. Both techniques begin by drawing each pixel of each source image as a point sprite with a finite size that is proportional to the magnification of the displayed image, i.e. the pixels of a source image that has been magnified two times are drawn as point sprites that are two screen pixels wide on each side. With each point sprite drawn on the screen at its corresponding dense correspondence position, we can combine them using a simple averaging technique where each source image provides an equal contribution to the final pixel color, we can select only the source image whose dense correspondence position is closest to the screen pixel position (nearest-neighbor selection), or we can combine them by giving the closest source pixels the greatest weight in the final color of the screen pixel (normalized weighed averaging).

The fragment program in Figure 4.1 implements the nearest-neighbor selection. It first calculates the distance between the source pixel (given by the dense correspondence position), and the fragment, or screen pixel, that is covered by that source pixel. The distance is then stored in the depth buffer. Before each fragment is drawn to the screen its depth is compared against the depth that is stored in the depth buffer. If its depth is less than the existing depth, then the fragment color is written to the output buffer, and the fragment depth is written to the depth buffer. The result of applying this algorithm to each sprite point is a Voronoi diagram where each partition corresponds to the source image that provided the color for the super-resolution image.

Normalized weighted average blending requires a two-pass algorithm. On the first pass each fragment calculates the distance between itself and the center of the source image pixel. The weight for the fragment is inversely proportional to this distance. The color of the fragment is then multiplied by the weight and accumulated off screen in a frame buffer object (FBO). The weights are accumulated in the alpha channel of the FBO. The fragment program code for this first pass is illustrated in Figure 4.2.

The second pass of normalized weighted average blending performs the normalization. In this pass the values in the red, green, and blue channels are divided by the value

```
!!ARBfp1.0

# set zbuff to be equal to distance squared between fragment
# and source color center

# input texture coordinate
ATTRIB tc=fragment.texcoord[0];
ATTRIB color=fragment.color;

# output color value
OUTPUT outColor=result.color;
OUTPUT outDepth=result.depth;

# temporary values
TEMP distx,disty,distsq;
TEMP newcolor;

# get distance between fragment and source color center
SUB distx,{0.5,0.5,0.5,0.5},tc.x;
SUB disty,{0.5,0.5,0.5,0.5},tc.y;
MUL distx,distx,distx;
MUL disty,disty,disty;
ADD distsq,distx,disty;

# set the alpha channel to 1.0
ADD_SAT newcolor,{0.0,0.0,0.0,1.0},color;

# output the depth value
MOV outDepth,distsq;
MOV outColor,newcolor;

END
```

Figure 4.1: Fragment program code for nearest-neighbor selection.

```
!!ARBfp1.0
# set alpha to the weight to be used when blending

# input texture coordinate
ATTRIB tc=fragment.texcoord[0];
ATTRIB inColor=fragment.color;

# output color value
OUTPUT outColor=result.color;
OUTPUT outDepth=result.depth;

# temporary values
TEMP dist,distsq,newWeight,finalColor,half;

# get distance between fragment and source color center
MOV half, {0.5,0.5,0.5,0.5};
SUB dist.xy, half, tc;
MUL dist.xy, dist, dist;
ADD distsq, dist.x, dist.y;

# convert the distance to a weight using:
# [weight=(1/(16*dist+1)]
POW dist, distsq.x, half.x;
MUL newWeight.x, {16.0}, dist;
ADD newWeight.x, {1.0}, newWeight;
RCP newWeight, newWeight.x;

# pre-multiply the color by the weight so we can
# accumulate using glBlendFunc(GL_ONE, GL_ONE)
MUL finalColor.rgb, inColor, newWeight.x;
# store the weight in the alpha channel
MOV finalColor.a, newWeight.x;

# output the new color
MOV outColor, finalColor;
# set the depth to a constant so we get blending instead of
# nearest neighbor coloring
MOV outDepth, {0.0,0.0,0.0,0.0};

END
```

Figure 4.2: Fragment program code for first pass of normalized weighted average blending.

```
!!ARBfp1.0

# divide the color components by the alpha channel (where
# the sum of the weights are stored)

# input texture coord from draw calls
ATTRIB tc=fragment.texcoord[0];

# output color value
OUTPUT outColor=result.color;

# temporary values
TEMP tv,weightSum,finalColor,rcpWeight;

# Get the texture value
TEX tv,tc,texture[0],2D;

# Grab the sum of the weights stored in the alpha channel
MOV weightSum,tv.w;

# Check for weightSum == 0 in preparation for doing a divide
CMP rcpWeight,-weightSum,weightSum,{1,1,1,1};
# invert the sum of weights in preparation for
# multiplication since we don't have a divide.
RCP rcpWeight,rcpWeight.w;

# output the color, divided by the weights
MUL finalColor.rgb,tv,rcpWeight;
# Set the alpha value of the final color. This should be set
# to one if weightSum is non-zero, otherwise it should be
# set to zero
CMP finalColor.a,-weightSum,{1,1,1,1},{0,0,0,0};

MOV outColor,finalColor;

END
```

Figure 4.3: Fragment program code for first pass of normalized weighted average blending.

in the alpha channel. This is done using the fragment program shown in Figure 4.3. The result of this second pass is a high-resolution texture that is then displayed on screen.

The nearest neighbor blending method produces images that maintain sharp edges at the expense of introducing artifacts. The weighted average blending method produces smoother images but is an order of magnitude slower than nearest neighbor blending (see Figure 4.4).

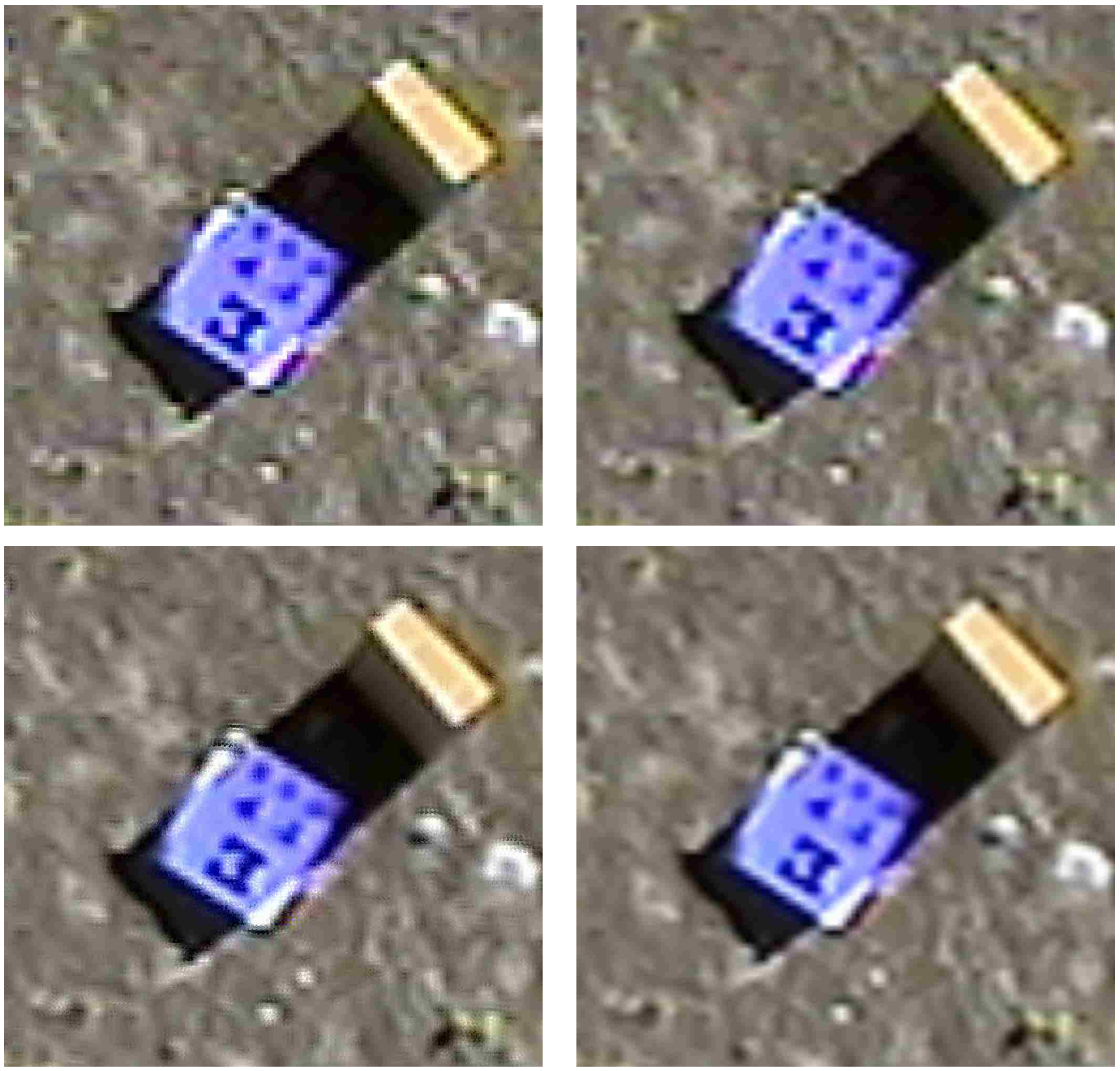


Figure 4.4: Comparison of blending techniques. Top left: portion of one of the input images used to produce the super-resolution result, magnified to show individual pixels. Top right: bilinear interpolation of the same image. Bottom left: super-resolution result using multiple input images and nearest neighbor blending. Bottom right: super-resolution result using multiple input images and weighted average blending.

Chapter 5

Conclusions and Future Work

We have discussed the challenges that arise when using video from a mini-UAV, including jittery video, narrow fields of view, low resolution, and limited time on screen for key features. We have shown how existing techniques provide no efficient means for addressing all of these issues simultaneously. Building upon *hierarchical dense correspondence* (HDC), we create a unifying framework for removing jitter, creating a video mosaic, enhancing the video resolution, and extracting dynamic events in the video. HDC is designed to be accelerated on commodity GPUs allowing near real-time enhancement of the video. The hierarchical nature of the algorithm allows HDC to robustly align video images without the aid of external camera parameters, while the locally adaptive warps eliminates the need for internal camera calibration parameters. This allows for its use on a wider range of aerial videos where these parameters are unavailable. The dense local correspondence built into HDC provides a robust method for compensating for parallax motion that cannot be matched by purely global registration methods.

While the current implementation of the video interface gives pleasing results, there are a few areas that need improvement. Occasionally noise in the video, usually in the form of transmission errors, will cause the HDC algorithm to produce an alignment that is obviously incorrect. The resulting frame-to-frame transformation will be propagated through the mosaic when composed with other frame-to-frame transformations, as described in Section 2.6.2. When these errors occur it should be possible to detect the incorrect transformations and either correct them or prevent them from being included as part of the mosaic, as in [24]. As most of the errors occur during short (less than five frames) video sequences, it may be possible to drop the noisy frames altogether and calculate a frame-to-frame transformation that skips over the unusable frames. It is our opinion that it will be easier to detect erroneous transformations than it will be to analyze the content of each frame to determine if it is usable or not.

There are other efficiencies that could be implemented but have not been included in this work. In the current system when super-resolution is enabled, the HDC algorithm is applied across the entire frame at full resolution for each frame that overlaps the super-resolution area. This sometimes involves computing the dense correspondence for areas of the image that are not even being displayed on screen. It should be possible to cull the inputs to the HDC algorithm to only include the parts of the input frames that will end up being shown on screen. It may even be possible to restrict the algorithm to only run in a small area surrounding the cursor, allowing the super-resolution feature to be even more interactive than it currently is.

Bibliography

[1] S. Baker and T. Kanade. Super resolution optical flow. Technical Report CMU-RI-TR-99-36, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, October 1999.

[2] K. S. Bhat, M. Saptharishi, and P. K. Khosla. Motion detection and segmentation using image mosaics. In *IEEE International Conference on Multimedia and Expo*, volume 3, pages 1577–1580, 2000.

[3] T. E. Boult and G. Wolberg. Local image reconstruction and subpixel restoration algorithms. *CVGIP: Graphical Models and Image Processing*, 55(1):63–77, 1993.

[4] J. Casper and R. R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 33:367–385, 2003.

[5] M. Chen, R. Botchen, R. Hashim, D. Weiskopf, T. Ertl, and I. Thornton. Visual signatures in video visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1093–1100, 2006.

[6] M.-C. Chiang and T. E. Boult. Efficient image warping and super-resolution. In *Proceedings 3rd IEEE Workshop on Applications of Computer Vision*, pages 56–61, December 1996.

[7] R. Collins, A. Lipton, and T. Kanade. A system for video surveillance and monitoring. In *American Nuclear Society 8th Internal Topical Meeting on Robotics and Remote Systems*, 1999.

[8] N. Damera-Venkata and N. L. Chang. Realizing super-resolution with superimposed projection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[9] G. Daniel and M. Chen. Video visualization. In *VIS '03: Proceedings of the 14th IEEE Visualization*, pages 409–416, Washington, DC, USA, 2003.

[10] R. Fransens, C. Strecha, and L. V. Gool. Optical flow based super-resolution: A probabilistic approach. *Computer Vision and Image Understanding*, 106(1):106–115, 2007.

[11] D. Gerhardt. Feature-based mini unmanned air vehicle video euclidean stabilization with local mosaics. Master's thesis, Brigham Young University, 2007.

[12] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, J. Adams, C. Humphrey, and M. Quigley. Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, 25(1):89–110, January 2008.

[13] C. Hein. Real-time implementation of super-resolution imaging algorithm.

[14] T. Huang and R. Tsai. Multi-frame image restoration and registration. In *Advances in Computer Vision and Image Processing*, volume 1, pages 317—339. 1984.

[15] L. M. Hwa, M. A. Duchaineau, and K. I. Joy. Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):355–368, 2005.

[16] M. Irani and P. Anandan. Video indexing based on mosaic representations. *Proceedings of the IEEE*, 86(5):905–921, 1998.

[17] M. Irani, P. Anandan, and S. Hsu. Mosaic based representations of video sequences and their applications. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, pages 605–611, 1995.

[18] M. Irani and S. Peleg. Improving resolution by image registration. *Graphical Models and Image Processing*, 53:231–239, 1991.

[19] D. Keren, S. Peleg, and R. Brada. Image sequence enhancement using sub-pixel displacements. In *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 742–746, June 1988.

[20] R. Kumar, H. Sawhney, S. Samarasekera, S. Hsu, H. Tao, Y. Guo, K. Hanna, A. Pose, R. Wildes, D. Hirvonen, M. Hansen, and P. Burt. Aerial video surveillance and exploitation. *Proceedings of the IEEE: Special Issue on Third Generation Surveillance Systems*, 89(10):1518–1539, October 2001.

[21] R. Kumar, H. S. Sawhney, J. C. Asmuth, A. Pope, and S. Hsu. Registration of video to geo-referenced imagery. *Proceedings of IEEE CVPR*, pages 54–62, August 1998.

[22] A. J. Lipton, H. Fujiyoshi, and R. S. Patil. Moving target classification and tracking from real-time video. *IEEE Workshop on Applications of Computer Vision*, 0:8–14, 1998.

[23] S. Mann and R. W. Picard. Virtual bellows: constructing high quality stills from video. In *IEEE International Conference on Image Processing*, volume 1, pages 363–367, 1994.

[24] B. S. Morse, D. Gerhardt, C. Engh, M. A. Goodrich, N. Rasmussen, D. Thornton, and D. Eggett. Application and evaluation of spatiotemporal enhancement of live aerial video using temporally local mosaics. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2008.

[25] S. Peleg and J. Herman. Panoramic mosaics by manifold projection. In *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 338–343, 1997.

[26] S. Peleg, D. Keren, and L. Schweitzer. Improving image resolution using subpixel motion. *Pattern Recognition Letters*, 5(3):223–226, 1987.

[27] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.

[28] D. Steedly, C. Pal, and R. Szeliski. Efficiently registering video into panoramic mosaics. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1300–1307, Washington, DC, USA, 2005.

[29] Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and texture-mapped models. In *Computer Graphics (SIGGRAPH'97 Proceedings)*, pages 251–258, August 1997.

[30] R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, March 1996.

[31] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–109, 2006.