2010

Efficient object tracking in WAAS data streams

Trevor Clarke

Efficient Object Tracking in WAAS Data Streams

by

Trevor R.H. Clarke

Thesis

Presented to the Faculty of the Graduate School of
Rochester Institute of Technology
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

Committee:
Dr. Roxanne Canosa
Dr. Roger Gaborski
Reynold Bailey

Rochester Institute of Technology
Department of Computer Science
B. Thomas Golisano College of Computing and Information Science

June 2010

Dedicated to Robert Clarke
for his never ending support
in all my endeavors

Acknowledgements

I would like to thank Dr. Roxanne Canosa for her excellent supervision and assistance. She successfully steered me through the many obstacles and setbacks that come with pursuit of an advanced degree.

I would also like the thank Dr. Bischof and the entire CS department support team. No research endeavor can possibly succeed without a top notch group of support professionals.

This thesis would not be possible without high quality data. The Air Force Research Labs, Persistent Surveillance Systems, the RIT College of Imaging Science, and Ball Aerospace and Technologies Corporation all provided data for use in this research.

Last but not least, thanks to my family and friends for motivating me to complete my work. I never would have made it without the support of Jennifer, Bob, Susan, Raymond, and many more.

Trevor R.H. Clarke

Abstract

Wide area airborne surveillance (WAAS) systems are a new class of remote sensing imagers which have many military and civilian applications. These systems are characterized by long loiter times (extended imaging time over fixed target areas) and large footprint target areas. These characteristics complicate moving object detection and tracking due to the large image size and high number of moving objects. This thesis evaluates existing object detection and tracking algorithms with WAAS data and provides enhancements to the processing chain which decrease processing time and increase tracking accuracy. Decreases in processing time are needed to perform real-time or near real-time tracking either on the WAAS sensor platform or in ground station processing centers. Increased tracking accuracy benefits real-time users and forensic (off-line) users. The original contribution of this thesis increases tracking efficiency and accuracy by breaking a WAAS scene into hierarchical areas of interest (AOIs) and through the use of hyperspectral cueing

Contents

List of Figures

List of Tables

1 Overview

Wide area airborne surveillance (WAAS) systems are a new class of remote sensing imagers which have many military and civilian applications. These systems are characterized by long loiter times (extended imaging time over fixed target areas) and large footprint target areas. These systems may be airborne or orbital and may contain sensors from any imaging modality (synthetic aperture radar (SAR), hyperspectral (HSI), thermal (TIR), and electro-optical (EO) systems have all been developed). The data streams generally have high spatial resolution and low temporal resolution. A system from Persistent Surveillance Systems, for example, generates 96 megapixel EO frames at about 1 frame per second. There are two major difficulties created by WAAS systems: a non-stationary camera makes tracking of moving objects difficult and tracking the many objects in a scene is computationally expensive. The main contribution of this thesis will be to adapt existing tracking algorithms to WAAS data streams. This is augmented by a target nomination algorithm used to reduce the size of the tracking space.

WAAS systems often circle a fixed target area in order to increase loiter time. This generates additional image motion which complicates object tracking. Existing tracking algorithms are designed for use in fixed camera situations or traditional unmanned aerial vehicle (UAV) full motion video (FMV). Much work has been done developing and adapting tracking algorithms for use with standard UAV video. Wenshuai Yu[1] has developed a framework for moving target detection and tracking for use with near real-time UAV video. Jiangjian Xiao[2] has shown two methods for tracking vehicles and people in UAV video. Isaac Cohen[3] has developed an algorithm which uses a directed acyclic graph of detected objects to register video and track objects even when the object tracks are partially obscured.

These systems fail to address the second major difficulty with WAAS streams. Large target footprints expose many targets (hundreds or more), many of which are partially obscured for portions of the video sequence. Identifying and accurately tracking this many objects is a computationally expensive operation. Human operators rarely need to track all of these objects simultaneously and concentrate on an area of interest for near real-time exploitation. Target nomination using supplemental HSI or other data will be explored as a way to shrink the tracking space. These techniques will be demonstrated on data from the Air Force Research Labs Angel Fire project, as well as data from aircraft mounted HSI sensors. The original contribution of this thesis is an increase in tracking efficiency and accuracy by breaking a WAAS scene into hierarchical areas of interest (AOIs) and through the use of hyperspectral cueing.

2 Thesis Objective

The objectives of this thesis are to determine the effectiveness of existing tracking algorithms on WAAS data, increase the efficiency of existing algorithms on WAAS data, and increase the accuracy of object tracks in WAAS data.

There is a large body of prior research dealing with object tracking in airborne video data. Most of this research deals with low spatial resolution, full frame rate EO video. A goal of this thesis is to gauge the applicability of these techniques to WAAS data, which has high spatial resolution and low frame rate. Existing algorithms will be modified to work on the lower frame rate WAAS data and their effectiveness judged against human derived object tracks.

The lower spatial resolution video data typically used for object tracking constrains the number of objects available for tracking. The high spatial resolution of WAAS data coupled with common uses of this data, such as large event security monitoring, lead to a large number of objects which need to be tracked. This can complicate tracking as there is a greater number of similar objects following similar tracks. Determining which object belongs to which track and maintaining the proper track can be difficult. This problem is also addressed by this thesis.

3 Background

Object detection and tracking are difficult with WAAS data. Large spatial resolution increases the number of objects which need to be tracked and increases the computational requirements needed for detection and tracking. The low frame rate also complicates tracking as object motion predictions have greater variability from frame to frame.

Most object detection and tracking systems can be structured in a processing pipeline with a number of common steps. Not all steps are present in all systems but this pipeline provides a useful abstraction for comparison and evaluation of different techniques. The pipeline in Figure 1 will be used for discussion.

Many WAAS systems use multiple cameras to capture the image data. These images need to be mosaiced into a single frame. This is typically performed by the sensor's internal processing pipeline using a camera model specific to that particular imaging system. Generic mosaicing algorithms exist but this thesis will assume that the sensor specific mosaicing algorithms are sufficient and will concentrate on other portions of the tracking pipeline.

WAAS platforms are in motion relative to the ground so the apparent motion of objects in a video sequence consists of the actual object's motion combined with the motion of the WAAS platform. Some algorithms require the platform to be stable so algorithms are needed to stabilize a video stream relative to the ground. This stabilization may be accomplished by the platform's mosaicing algorithm but this stabilization is often coarse and may need refinement.

Object detection is a key step in the tracking pipeline. Detection algorithms may require multiple video frames or a single frame for detection. There may be different methods of detection for objects which are part of an existing track. A number of detection algorithms will be discussed in detail later in this section.

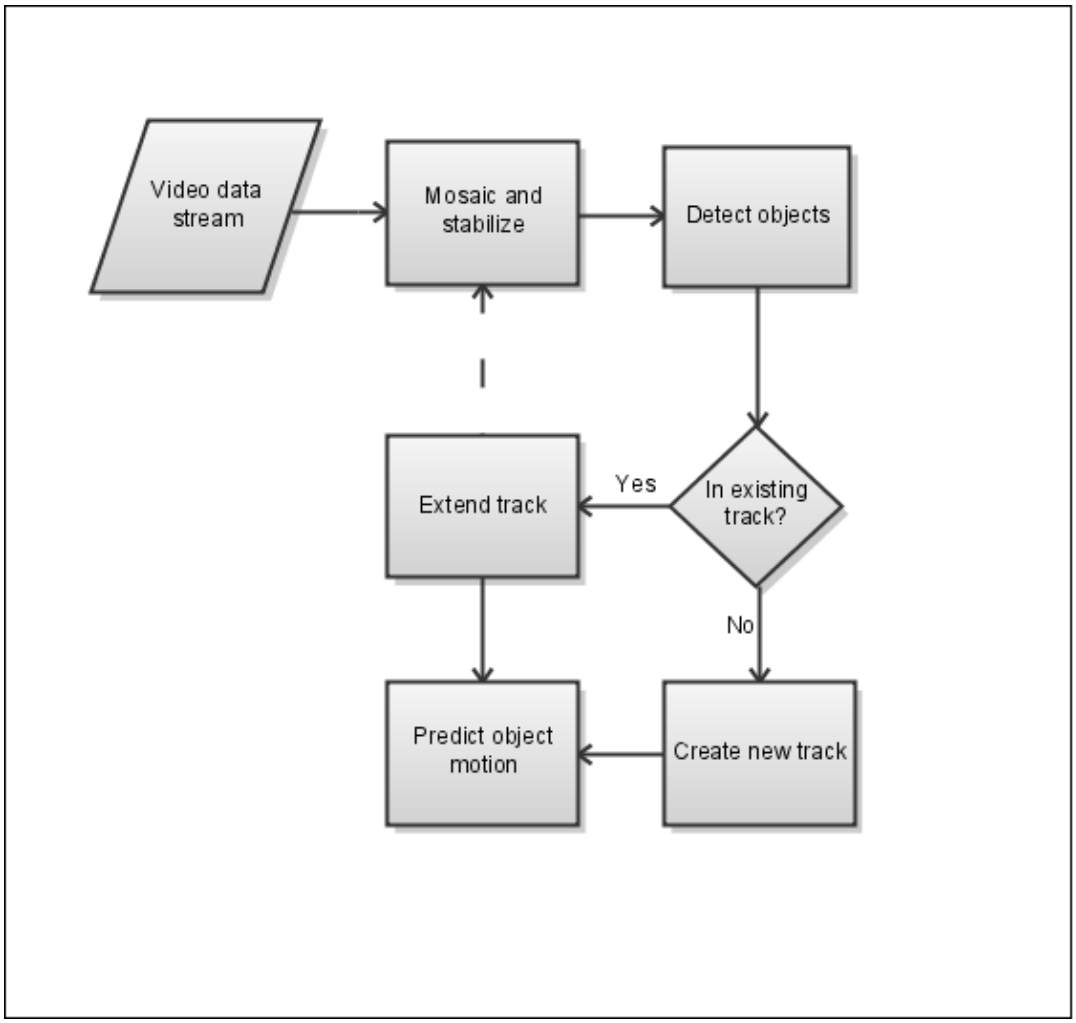

Figure 1: Tracking pipeline

The primary decision point in the tracking pipeline determines if a detected object is in an existing object track or if it represents a new object. This decision may be implicit in the object detection step if a separate algorithm is used to detect existing objects. Alternately, it may be an explicit step in the pipeline. There are a couple of classes of tracking algorithms in general use. Predictive tracking algorithms use existing information about a track and an object to estimate where that object will reside in the next frame. This prediction is used to search for objects within a threshold of the predicted location. The other class is matching algorithms. These algorithms attempt to associate newly detected objects with existing tracks using a variety of metrics. If a match is found, the track is extended; otherwise the object is considered new. Some algorithms may implement a hybrid approach using data from both methods.

The object prediction step is used in predictive tracking algorithms to predict the next probable location for an object. Matching algorithms use this stage to generate model parameters which will be used to match objects in the next frame.

A number of algorithms will be presented and discussed below. Basic information on the algorithm will be presented along with information on how the algorithm has been used in existing tracking applications. Some algorithms may encompass multiple stages in the tracking pipeline or just a single stage.

3.1 RX Filter

A common anomaly detection algorithm, often used as a baseline anomaly detector for comparison with other algorithms, was developed by Reed and Yu [4]. This algorithm is commonly called the RX detector (RXD), RX filter, or simply RX. The classic expression of RXD is shown in Equation 1 and is referred to as $K-RXD$.

$$\delta^{K-RXD}(r) = (r-\mu)^T K_{L\times L}^{-1}(r-\mu) \tag{1}$$

The K refers to the sample covariance matrix, r is the L length vector of spectral intensities at a particular spatial location and μ is the sample mean. A modification shown in Equation 2 uses the sample correlation matrix (R) and is sometimes used in optimized implementation since the sample mean is not needed. It is referred to as $R-RXD$ to distinguish it from the covariance version, K-RXD.

$$\delta^{R-RXD}(r) = r^T R_{L\times L}^{-1} r \tag{2}$$

Both forms of RXD are related to the Mahalanobis distance which measures the similarity between an unknown sample set and a known one. Further explanation of the properties and application of RXD can be found in [5] and a number of other introductory texts on hyperspectral data exploitation.

3.2 Nested Spatial Window Target Detector (NSWTD)

NSWTD is an anomaly detection algorithm for hyperspectral data discussed in [6]. It can be used for object detection in a tracking pipeline. A group of three concentric spatial windows is used to detect objects which are statistically distinct from their background. A spatially large window contains a medium sized window and a small window. The two smaller windows represent potential target sizes and the large window will be used to calculate the background statistics. NSWTD uses a metric known as orthogonal projection divergence (OPD) shown in Equation 3.

$$\begin{aligned} OPD(s_i, s_j) &= \sqrt{s_i^T P_{s_j}{}^{\perp} s_i^T + s_j^T P_{s_i}{}^{\perp} s_j^T)} \\ P_{s_j}{}^{\perp} &= I_{L\times L} - s_k (s_k^T s_k)^{-1} s_k^T \quad \forall\, k = i, j \end{aligned} \tag{3}$$

Where I is an identity matrix, s^T is the signal to noise ratio maximization operator from orthogonal subspace projection[7] and s_i, s_j are vectors representing pixel spectra.

The OPD equation must be calculated twice, once between the inner and middle windows as in Equation 4 and once between the middle and outer windows as in Equation 5.

$$\delta_1^{2W-NSW}(r) = OPD(m_{in}(r), m_{d,1}(r)) \tag{4}$$

$$\delta_2^{2W-NSW}(r) = OPD(m_{mid}(r), m_{d,2}(r)) \tag{5}$$

Where $m_{d,1}$ is the mean of the outer window minus the inner window and $m_{d,2}$ is the mean of the outer window minus the middle window. Finally, the three window NSW is calculated by Equation 6.

$$\delta^{3W-NSW}(r) = max_{i=1,2}(\delta_i^{2W-NSW}(r)) \tag{6}$$

This gives a metric for each pixel which can be thresholded to determine if the pixel is anomalous. According to [8], the NSWTD algorithm is faster than other, common anomaly detection algorithms such as RX while providing good detection results for targets of varying sizes.

3.3 Spectral Angle Mapper (SAM)

SAM is a common hyperspectral classification algorithm[9]. It can be used for object detection in a tracking pipeline or it may be used as an input to a matching-based object tracking algorithm. Spectra are compared pair-wise by treating the spectral vectors as points in n-dimensional space, normalizing to unit vectors, and calculating an angle between the two vectors. Since the vectors are normalized, this method is invariant to illumination level. SAM is a way to quickly determine similarity between two spectra and is a useful filter when the spectral angles are thresholded against a fairly high target angle. Other spectral comparison algorithms provide more accurate similarity results but SAM is extremely fast and is a good first cut for similarity filtering of pixels.

3.4 Optical Flow Estimation

Optical flow is the apparent motion of objects, surfaces, and edges between frames. This motion is typically caused by motion of the camera and motion of individual objects in the scene. A number of methods are available for estimating optical flow but they generally fall into two categories.

Dense estimation techniques map all pixels in the first frame to corresponding pixels in the second frame. Dense techniques are quite accurate and provide motion vectors for all the data in a frame but are generally very computationally expensive. They are typically used on very small scenes or subsets.

Sparse estimation techniques map a subset of pixels from one frame to another. This subset may be an actual subset of pixels chosen by some external criteria or it may amalgamate pixels into blocks and calculate the motion of the blocks. While less accurate, sparse techniques are generally effective for calculating camera motion since a majority or pixels will have the same flow vectors.

The Lucas-Kanade (LK) [10] method was originally proposed as a dense estimation method but has since been adapted to become a sparse estimation method. LK can be used as a sparse estimator since it only relies on spatially local information when estimating a pixel's motion. The motion of a subset of pixels can be calculated and these can be used as an estimate for the motions of the other pixels.

The local information is derived from a small window around the pixel in question. This has a serious drawback in that large motions which fall outside of this window can not be calculated. A pyramidal version of LK can be used to overcome this deficiency. LK is first run on a low detail version of the image, in essence a series of image blocks. This accounts for larger motions since a window of a fixed size encompasses a larger percentage of the frame at a lower detail level. This is repeated on one or more additional detail levels until the full detail frame is used.

LK make a few assumptions about the frames.

1. Brightness consistency - A pixel or object from one frame to another has fairly consistent brightness levels.
2. Temporal persistence - A pixel or object makes only small movements. Small is not an exact qualifier and relates to the overall size of the frame.
3. Spatial coherence - Neighboring pixels belong to the same surface, project to nearby points on the image plane, and have similar motion.

These assumptions can be represented by equations 7 where I is the intensity of pixel $x(t)$ at frame t.

$$\begin{aligned} f(x,t) \equiv I(x(t),t) = I(x(t+dt),t+dt) \\ \frac{\partial f(x)}{\partial t} = 0 \\ \frac{\partial I}{\partial x}|_t(\frac{\partial x}{\partial t}) + \frac{\partial I}{\partial t}|_{x(t)} = 0 \\ v = -\frac{I_t}{I_x} \end{aligned} \tag{7}$$

Starting with the brightness invariance and temporal persistence and applying the chain rule for partial differentiation we end up with the second to last equation. We substitute I_x, v, and I_t for each term to indicate the spatial brightness derivative, the velocity, and the temporal brightness derivative. This yields the final form which is the basis for LK. The form indicated is the one dimensional case but can be easily extended to two dimensions.

The change in space for a given time period is measured by finding strong corners in the edge image of each frame. We search for corresponding corners in a local window and calculate an initial estimate. Since brightness will be somewhat variable, the corners will not correspond exactly

so we use Newton's method to refine the estimate using edge information in the spatial window. A more in depth derivation of LK is available in [11] including a discussion of the strengths and weaknesses of the technique.

3.5 RANSAC

Random sample consensus (RANSAC) [12] is a method used to fit a model to observed data. RANSAC is an iterative method which is robust against outliers.

The algorithm works by randomly selecting a number of points from the observed data and fitting a model to those points. Each point in the data which is not in this set is tested against the model estimate. If the calculated error is less than a threshold it is added to a set of consensus points. The algorithm is halted when the size of this consensus set exceeds a threshold or a maximum number of iterations has occurred. The model with the largest consensus set is selected.

While robust, RANSAC has no upper bound on execution time. The maximum iteration count is used to force an upper bound but exiting the algorithm this way results in a sub-optimal model. It is possible to calculate the probability that RANSAC will generate a model with a maximum error value and a maximum number of iterations. This information can be used to adjust these two parameters in order to control the execution time or the margin of error.

3.6 Hyperspectral Cueing

Cueing as it relates to video tracking uses the video data or other, external data to mark candidate objects or areas of interest. The supplemental data reduces the search space for the primary tracking algorithm or supplements the primary tracking algorithm to increase the object detection accuracy. Hyperspectral cueing uses external hyperspectral imagery to locate areas of interest and to increase tracking accuracy. Anomaly detectors such as K-RXD locate areas of interest based on spectral disimilarity. This areas of interest are intersected with the pixels flagged during motion estimation. The other use of hyperspectral imagery is to supplement tracking of identified objects. Objects in two frames of data are compared using a signature comparison algorithm such as SAM. Comparing the spectral signatures of two objects provides additional information when determining if these are the same object and thus constitute part of an object track.

3.7 Wang Tracking Algorithm

The tracking algorithm used by Wang, et al. [13] is a hybrid object tracking algorithm which uses matching-based and prediction-based techniques. Four values are examined to decide if an object is a candidate for addition to a track: object trajectory, object size, grayscale distribution, and object texture. Wang's tracking algorithm requires a stabilized video stream. Stabilization is accomplished by calculating an edge image for each frame and using a few coincident frames to

estimate the bilinear projective model for the camera motion. [14][15] Object detection in Wang's method utilizes a wavelet transform and the bilinear predictive camera model to remove camera motion. The resultant differences between various levels of the wavelet pyramid identify objects. Wang's end goal is a motion prediction system which is used to compress video streams. Other portions of the compression utilize the wavelet transform, so this is a process with zero additional cost. The wavelet transform can be computationally expensive and will not be discussed in further detail. It was used by Wang since the wavelet transformed needed to be calculated for a non-tracking algorithm thus the data was already available for used for object detection. The object tracking portion of Wang's pipeline is of primary interest for this thesis.

Object position is defined in Equation 8 as the centroid of a detected object.

$$\begin{aligned} c_x &= (\sum_{(i,j)\in O} p_{i,j} \cdot i)/(\sum_{(i,j)\in O} p_{i,j}) \\ c_y &= (\sum_{(i,j)\in O} p_{i,j} \cdot j)/(\sum_{(i,j)\in O} p_{i,j}) \end{aligned} \tag{8}$$

Where O is the set of coordinates of an object area and $p_{i,j}$ is the value of the edge image at position (i,j). Object motion over a few frames is assumed to be a straight line with constant acceleration. Three frames are used to calculate speed v and acceleration a in Equation 9. These values predict a location for the current frame which is compared to the candidate object.

$$S = vt + \frac{1}{2}at^2 \tag{9}$$

With an adequate framerate, object size does not change significantly from frame to frame. A dispersion value as calculated by Equation 10 is a metric for measuring object size. This equation determines how far from the object centroid a nearly uniform gray level extends. (c_x, c_y) represents to object centroid from Equation 8.

$$disp = (\sum_{(i,j)\in O} \sqrt{(i-c_x)^2 + (j-c_y)^2} \cdot p_{i,j})/(\sum_{(i,j)\in O} p_{i,j}) \tag{10}$$

Grayscale distribution (the width of the grayscale histogram) of an object is fairly constant for objects of interest assuming the lighting conditions are constant from frame to frame. Grayscale distribution is calculated as (gr_m, gr_h, gr_l) the mean of groups of pixels in the entire range, upper 10% of the range, and lower 10% of the range respectively.

The object texture measures grayscale variation across an object. Wang estimates texture by calculating the mean of the 10% of pixels with the largest values in the edge image. The method

of calculating the edge image is not discussed, but a reasonable assumption is that a common edge detection method such as the Laplacian method or Sobel method can be used.

The four metrics discussed above are calculated for candidate objects and existing tracks (as the object in the track for the previous frame). The difference between these sets of values is thresholded and metrics outside the threshold are ignored. At least three of the four metrics must be within the threshold for consideration. If no objects fit this criteria for a certain track, it is terminated. If there is an unambiguous correspondence between a single object and a single track, the track is extended. If multiple objects are candidates for a track, or multiple tracks are candidates for an object, a weighted cost is calculated for each potential object and track pairing using Equation 11. The w variables represent the weights and the f and t superscripted variables represent the metrics for the current frame and the track. Wang was not clear how the weights were determined, but it is implied that values were chosen which performed well with a set of sample data.

$$\begin{aligned} dif = w_{tr}(|c_x^f - c_x^t| + |c_y^f - c_y^t|) \\ + w_{disp}(|disp^f - disp^t|) \\ + w_{gr}(|gr_l^f - gr_l^t| + |gr_m^f - gr_m^t| + |gr_h^f - gr_h^t|) \\ + w_{tx}(tx^f - tx^t) \end{aligned} \tag{11}$$

Wang's algorithm is not expected to work well with WAAS data as stated due to the assumption of a reasonably high frame rate in multiple steps in the algorithm.

3.8 Cohen Tracking Algorithm

Cohen and Medioni [3] present an alternate method for object detection and tracking as well as elimination of camera movement using optical flow. This system performs object detection and matching-based object tracking.

Instead of directly modeling the 3D parameters of the camera motion, Cohen estimates the induced optical flow of coincident frames. A small set of feature points (x_i, y_i) in the image are tracked from a reference image I_0 to a target image I_1. These two images are registered by computing the transformation $T_{I_1 I_0}$ which warps I_0 to I_1. The parameters are estimated using an iterative minimization of the least square criterion as in Equation 12.

$$E = \sum_i (I_0(x_i, y_i) - I_1(T(x_i, y_i)))^2 \tag{12}$$

An affine model is used to approximate T and is calculated for at least three different detail

levels in an image pyramid. This transformation is incorporated into the optical flow calculation according to Equation 13.

$$\nabla I_i \nabla^T I_i w = -\nabla I_i \frac{dI_i}{dt} \tag{13}$$

Where $w = (u, v)^T$ is the optical flow. This equation can be used to extrapolate the normal to the optical flow in Equation 14.

$$w_\perp = -\frac{(I_{i+1}(T_{i+1,j}) - I_i(T_{i,j}))}{||\nabla T_{i,j} \nabla I_i(T_{ij})||} \cdot \frac{\nabla T_{ij} \nabla I_i(T_{ij})}{||\nabla T_{ij} \nabla I_i(T_{ij})||} \tag{14}$$

$w_\perp$ can be used to detect residual motion after compensation for optical flow. Large values of $w_\perp$ occur near regions of motion. $w_\perp$ is thresholded and a 4-connectivity scheme is used to aggregate points into regions of residual motion.

Cohen represents potential tracks using a graph. Nodes in the graph represent blobs detected using the normal to the optical flow. Edges in the graph represent relationships between blobs in coincident frames. Edges are created by measuring grayscale similarity between blobs in a neighborhood. The size of the neighborhood is calculated based on the amplitude of the blob's motion. The eigenvalue of the potential transformation from a blob in I_0 to a blob in I_1 are attached to the associated edge. A set of attributes are also associated with each node. Cohen associates the following attributes: frame number, centroid, principal directions, mean, variance, velocity, parent ID and similarity, children IDs and similarities, and length.

Single objects may be detected as multiple blobs when using the optical flow normal. This situation can be detected in the graph when multiple blobs which are spatially close and have nearly the same direction and velocity. When this occurs, the nodes on the graph are merged to a single node and the edges updated accordingly. Detection accuracy of broken objects is increased by maintaining a dynamic template for each object and matching it to potential broken objects. The blob centroids and orientations are aligned over the last five detected frames and a median filter is applied. Cohen calls this process a *median shape template*. This filter can also detect objects which stop and then resume motion. Each blob which has no connected components in subsequent frames will be propagated some number of frames into the future. The template for this node is matched against nodes in these future frames and a positive match indicates an object which has resumed motion.

Object trajectories can be extracted from the graph by finding an optimum path through the graph. Each edge receives a cost calculated in Equation 15 where C_{ij} is the gray level and shape correlation between regions i and j. d_{ij} is the distance between centroids.

$$c_{ij} = \frac{C_{ij}}{1 + d_{ij}^2} \tag{15}$$

An optimal graph path is then calculated from a node with no successors to a node with no parent to determine an object trajectory.

Cohen's algorithm is not expected to work well with WAAS data as stated. The need to build a sizeable graph using many frames before tracks can be extrapolated lends itself to offline data processing. Using low frame rate WAAS data may create a significant delay between data collection and track determination, independent of processing resource requirements.

3.9 Camera motion compensation

The Columbus Large Image Format (CLIF) and CLIF-II data are used as a basis for motion compensation and tracking. Data from a single camera in the six camera system is used. Since a hierarchical subdivision of the image is used to target specific areas of the image to maximize the throughput, using a spatial subset is equivalent to using the entire data set and setting focus to the spatial subset. Scaling up processing to multiple machines would allow simultaneous processing of these additional areas.

4 Approach

This thesis meets the objectives of increasing tracking efficiency and accuracy by breaking a WAAS scene into hierarchical areas of interest (AOIs) and through the use of hyperspectral cueing. The size of these AOIs and the number of AOIs which are actively processed will provide a measure of control over the accuracy/efficiency ratio.

4.1 WAAS Data Characteristics

The primary datasets from the Air Force Research Labs are the Columbus Large Image Format (CLIF[16] and CLIF-II[17]) datasets. Each are approximately 12000x5000 pixels spatially and were collected with a matrix of commercial digital cameras. The panchromatic frames are 8-bit uncompressed. One dataset contains 50 frames and the other contains 56 frames. The data were collected at approximately 2 fps.

The CLIF dataset has a number of camera registration issues resulting in overlapping areas within the image. The data was acquired with a high off NADIR angle (side looking collection) and contains light industrial and farmland areas. The registration makes accurate tracking difficult as it generates a significant number of false detections and duplicate tracks. It is interesting due to

the unique location, the other data used is primarily urban, so this dataset is used for comparison of execution speed but not for accuracy of object and track detection. A sample frame is show in Figure 2.

Figure 2: Sample frame from the CLIF dataset

Within the CLIF-II dataset, the areas captured by cameras 0 and 1 are used. A single camera is a convenient region to use as it represents a reasonably sized subset of the data for the workstation used for analysis, approximately 4000x2600 pixels.

The camera 0 area is a residential neighborhood with two multi-lane roads meeting at an intersection. A number of vehicles are in motion and there are a small number of pedestrians in motion. Lighting is fairly constant and there are no large buildings showing significant parallax.

The camera 1 area contains the Ohio State University football stadium and the surrounding area. A couple of tall buildings show parallax effects and there are some noticeable reflection, sun glint, and shadow effects in and around the stadium. These effects add noise to the image which may be mistaken for moving objects. A small number of vehicles are in motion but this portion of the scene is mostly static.

Hyperspectral data from a number of sensors including NASA's AVIRIS, RIT's MISI, the US Army's COMPASS, the USGS' HYDICE, and the US Air Force's HyCas. Specific details of these

scenes are shown in Table 2.

No current collection system provides synchronized WAAS video and hyperspectral data required for integrated performance analysis of the complete cueing system. Processing throughputs of the individual compone

4.2 Frame Segmentation

An integral part of the optimizations of the various component algorithms is a hierarchical, spatial breakdown of the WAAS data. A WAAS frame was segmented into smaller sub-images arranged in a quadtree. The extent of the segmentation will vary based on the step in the tracking pipeline.

Breaking an image into smaller areas allows selective processing of a subset of the areas based on expected processing requirements. When the average processing requirements of an AOI exceed a threshold, the AOI can be further divided into four smaller AOIs and processing can be limited to one or more of these.

This independent processing of a subset of the image allows for easy exploitation of parallel computing capabilities. A number of AOIs can be processed independently on multiple computation units. Processing of a greater number of concurrent AOIs could be accomplished by increasing the number of available processing units. This aspect will not be fully explored by this thesis.

The frame segmentation is accomplished using a balanced quadtree whose leaf nodes contain sub-frames of a reasonable (user defined) spatial dimension. When objects are being actively tracked, further decomposition in areas of high object density can reduce processing load. Figure 3 shows a sample quadtree decomposition of a scene.

4.3 Motion Compensation

Motion compensation of a sub-frame is accomplished by calculating the optical flow vectors between points in neighboring frames. A pyramidal implementation of the Lucas-Kanade (LK) optical flow calculation method [10] is employed. The LK method is applied to various levels in an image detail pyramid of each sub-frame. This captures large motions using the LK method which employs a small change detection window.

Sparse tracking is used to speed up calculations. Strong corners are located in an edge image and these points are tracked between sub-frames. Once the optical flow vectors between these points are calculated, RANSAC is used to approximate the motion of the sub-frame while ignoring errors in the optical flow calculations.

The 3-D motion of the aircraft mapped to the 2-D image plane requires a perspective transform to accurately capture the sub-frame transformation but calculating this transform can be slow.

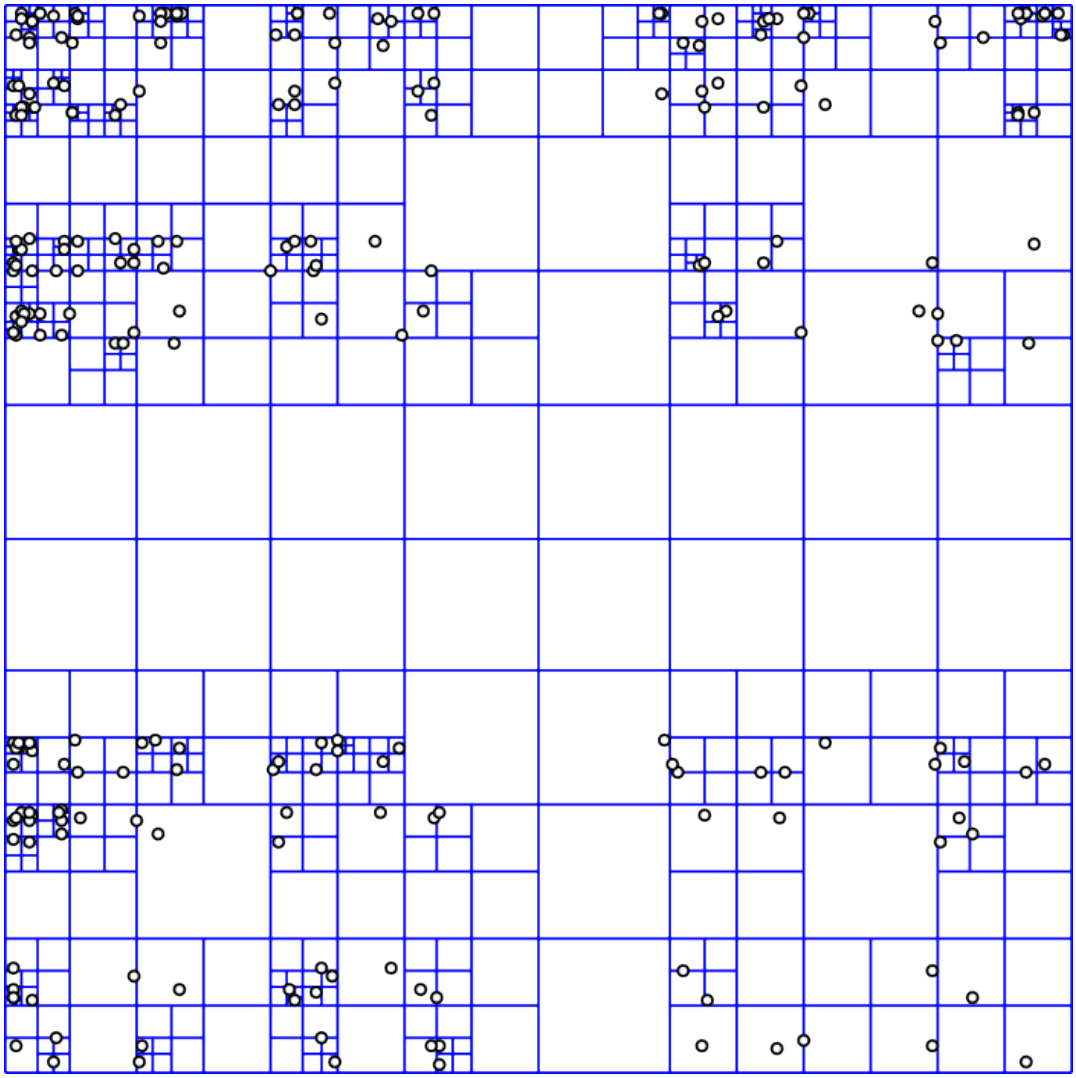

Figure 3: An unbalanced quadtree decomposition has a higher depth in areas of high object density.

An assumption is made that the altitude of the aircraft does not change significantly between neighboring frames which allows a for a fairly accurate estimation of the sub-frame motion using an affine transform. This assumption does not always hold particularly in urban environments with many tall buildings. This fails to compensate for some of the parallax effect seen when looking at these buildings. Parallax is the apparent motion of an object (usually rotation about a point) due to differences in observation location. The CLIF-II dataset provides an example of this as seen in Figure 4. The apparent motion of buildings may register as object tracks but these false positives are obvious to an analyst interpreting the data. Tracking extra objects may slow down execution of the algorithm due to the additional data processing. Compensation for parallax may be addressed in future work. Once the affine transform has been estimated, one of the sub-frames can be transformed in order to remove the camera motion.

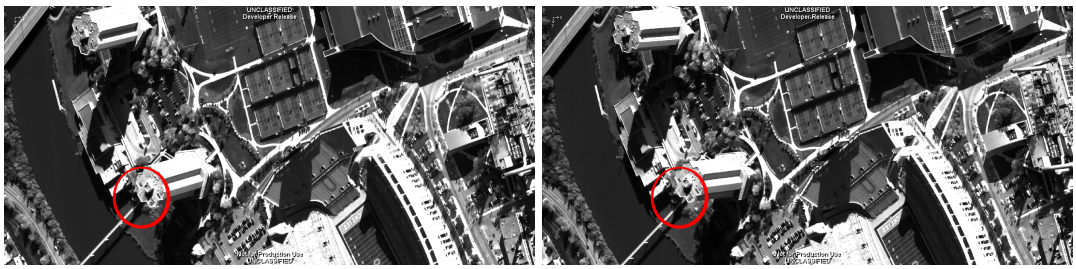

Figure 4: Apparent motion due to parallax in frames 1 and 10 of CLIF-II dataset. The motion is particularly noticeable with the building in the bottom center of the images.

4.4 Object Detection

Two object detection methods are available depending on availability of hyperspectral data. When hyperspectral data is available for a sub-frame, K-RXD is used to detect anomalous pixels in the sub-frame. K-RXD was chosen instead of NSWTD as it is less sensitive to noise than NSWTD. [18] provides a study of the accuracy and performance characteristics of a number of hyperspectral anomaly detection algorithms including RXD and NSWTD. NSWTD is potentially faster to execute and has a higher correct detection rate for high signal to noise data and is worth further exploration for future work.

When hyperspectral data is not available, frame differencing is used to remove the background. The two sub-frames are thresholded to minimize brightness differences between the two frames. Additional noise is removed by eroding the resulting threshold images. This removes single pixel noise often called salt and pepper noise or speckle. The frames are subtracted from each other to remove the background.

In either case, the resulting foreground or anomaly images are run through a morphological opening to fill in small gaps internal to the object blobs. These blobs are labeled using connected component labeling and each is treated as an object candidate.

4.5 Object Tracking

Object tracking was accomplished using a variation of Wang's tracking algorithm.

Each object is added to a directed graph as a vertex. Objects from the base frame are connected to the objects in the current frame and a number of properties are calculated on the vertices and edges.

The vertices contain a dispersion value, an object blob centroid, and a hyperspectral spectra. The edges contain the absolute difference of the dispersion values of the vertices and a velocity calculated from the object centroids. In addition, each edge compares the velocity of the previous

track edge to the current edge. Both the absolute velocity magnitude difference is calculated and the velocity vector angular difference. Finally, the spectral angle between the two objects is calculated for each edge.

Edges with a velocity magnitude exceeding a threshold are removed. This is based on the assumption that objects being tracked (usually vehicles and people) have an upper limit on their speed. A cost is calculated on the remaining edges as a weighted sum of the edge properties. F

Tracking is performed within a quadtree AOI. As a track approaches the edge of an AOI, the tracking algorithm is applied to the parent AOI in the quadtree. This larger area processing is performed for a small number of frames until the location of the object in an AOI lower in the tree can be determined. The object track information is passed to the new AOI for processing.

4.6 Testing and Verification

The following tests indicates the level of success for the thesis.

1. The efficiency of the hierarchical partitioning and hyperspectral cueing.

2. The accuracy of the tracking algorithm without hyperspectral object detection.

3. The accuracy of the hyperspectral object detection and track property.

Performance of the partitioning is determined by executing the tracking algorithm without any partitioning and comparing this to execution of the algorithm with various quadtree partition depths. Throughput in terms of megapixels per second processed is the primary metric.

Comparison of the hyperspectral cueing performance is measured is megapixels per second processed. This is compared to the collection rate of the hyperspectral sensor and to the processing throughput of the optical flow differencing method.

Accuracy of the tracking algorithm is compared to human derived truth data. Two WAAS analysts manually locate and mark objects and tracks in available data sets using techniques employed by US Air Force intelligence services. These sets of truth data are compared and outliers discarded. This technique may under-represent the number of objects and tracks in a data set but should minimize the number of false positives. The accuracy in terms of number of correctly identified tracks, false positives, false negatives, and incomplete or interrupted tracks are the primary metrics.

Table 1: Properties of hyperspectral test data

Sensor	Rows (along track)	Columns (cross track)	Bands	Data size (mpel)
Cueing System	512	256	256	32
Hydice[19]	181	97	168	2.95
COMPASS[20]	550	256	146	20.56
MISI[21]	2000	3689	16	118.05
AVIRIS[22]	512	614	197	61.93
HyCAS[23]	2664	2495	12	79.76

4.7 Hyperspectral Cueing

No current collection system provides synchronized WAAS video and hyperspectral data required for a complete hyperspectral cueing system as described in the background section. A hypothetical HSI collection system is used as the basis for hyperspectral cueing in the tracking system. The properties of this system are consistent with current HSI imaging system technologies. Performance of the system is characterized and conclusions are drawn about the utility of the system. Future work may analyze integrated WAAS and hyperspectral data collections.

The system is a whiskbroom collector which contains a 256x256 pixel CMOS imaging array which collects spectral information along one axis and spatial information along the other. Light enters the system and is reflected off a rotating mirror, passes through a prism and is directed to the detector array. The mirror slowly rotates which collects data across the second spatial dimension. The speed of the mirror and extent of the rotation for which data is captured define the collection rate and width of the collected swath. The size of the imaging array defines the height and each swath and the number of spectral bands collected, 256 for each. Bands are in the visible and near-IR spectral region; 400nm to 1000nm. A swath with a width of 512 pixels can be collected in approximately 10 seconds including time to reset the mirror to the start position. The collection platform can be moved such that a swath can be collected anywhere in the ground area covered by the panchromatic WAAS imager. The maximum time to move the imager across the entire WAAS field of view is less than or equal to 10 seconds. This means that the worst case collection rate including repositioning of the imager is 20 seconds per swath. This yields a collection rate in megapixels between 1.6$mpel/s$ and 3.3$mpel/s$.

A number of existing hyperspectral sensor images have been used for experimental testing. These sensors share some properties with the hypothetical system so they can be used to extrapolate performance characteristics of the hypothetical system. All datasets contain data in the visible and near-IR spectral regions providing complete or nearly complete overlap with the proposed system. The datasets used have the following characteristics.

5 Results and Discussion

5.1 Hyperspectral Anomaly Detection

The K-RXD implementation was tested on 5 hyperspectral data cubes and information on performance was calculated. The resulting anomalous regions were reviews by 2 hyperspectral data analysts to ensure the results are reasonable and correct. Validation is purely subjective and the experience of the analysts is used to lend credibility to the opinions. The analysts visually inspected the spectra of a sample of flagged pixels to surrounding pixels. Inspection took approximately 10 minutes per data set per analyst. In all cases, the analysts felt the indicated anomalous regions are reasonable and correct. A visual comparison of the results with results from a similar anomaly detection algorithm in the ENVI tool indicates nearly identical anomalous pixels. The exact details of the ENVI detector are not known as that information is not made available in the documentation for the tool. ENVI is a frequently used tool for hyperspectral analysis so it is reasonable to assume that it generates valid results.

Figure 5: MISI K-RXD Results (in yellow)

The five data sets used have the following properties. Band counts do not include "bad bands" which contain invalid data. This invalid data is usually due to atmospheric absorption spectral regions which yield very low signal to noise values. In most cases data does not represent a single swath either because the sensor is not a whiskbroom sensor or multiple whisks had already been combined into a single image. All tests were run on a Windows XP 64-bit workstation with 4GB of RAM and an Intel Core2 E8500 dual-core CPU running at 3.17GHz.

Table 3 shows timing information for the K-RXD implementation. The values indicate the average of five executions of the algorithm for each data set. Absolute times are shown for the entire algorithm execution minus the overhead introduced by unrelated processes such as graphical display of the results and for the execution of the covariance matrix calculations. The data is also presented as a throughput in terms of megapixels per second. The covariance matrix calculations are performed by a separate plug-in which prevented certain optimizations such as calculation of the covariance matrix and sample mean in the same loop. Therefore, the execution time of

Table 2: Properties of datasets used for K-RXD timing

Sensor	Rows (along track)	Columns (cross track)	Bands	Data size (mpel)
Hydice	181	97	168	2.95
COMPASS	550	256	146	20.56
MISI	2000	3689	16	118.05
AVIRIS	512	614	197	61.93
HyCAS	2664	2495	12	79.76

Table 3: Execution times and throughput for K-RXD

Sensor	Execution Time (ms)	Throughput ($mpel/s$)	Optimized Throughput ($mpel/s$)
Hydice	3132	0.9417	1.310
COMPASS	20402	1.008	1.441
MISI	77315	1.527	2.737
AVIRIS	66685	0.9287	1.261
HyCAS	52016	1.533	2.762

the loop which calculate the sample mean has been removed and the throughput recalculated to estimate the throughput of an implementation with this optimization.

This yields an average throughput across all the data set of 1.188 $mpel/s$ and a standard deviation of 0.3140. Applying the stated optimization, the average throughput is 1.902 $mpel/s$ and a standard deviation of 0.7763. This is between 57% and 75% of the estimated collection rate of the hypothetical hyperspectral sensor. It is clear that further optimization would be necessary to maintain the hardware collection throughput. A faster processor or more highly optimized implementation seems within reach especially if the anomaly detection is performed on the collection hardware using a highly optimized DSP implementation. The correlation matrix variant R-RXD may also provide a more efficient implementation. Finally, the number of spectral bands in the data strongly effects the calculation throughput. The number of bands increases the size of the matrices and vectors in the RXD calculations which the spatial resolution effects the number of times the RXD calculations are made. The higher order complexity of the RXD calculation (due to three matrix multiplications) accounts for this effect. Minimizing the number of spectral bands needed for adequate anomaly detection can also increase the throughput of the algorithm.

5.2 Motion Compensation

Table 4 shows timing information for the optical flow motion compensation implementation. The values indicate the average of five executions of the algorithm for each data set. Absolute times are shown for the entire algorithm execution for a single frame. Times include overhead introduced by informational display processes. The data is also presented as a throughput in terms of megapixels per second.

This yields an average throughput across all the data set of 6.6614 $mpel/s$ and a standard

Table 4: Execution time and throughput for motion compensation

Sensor	Size (mpel)	Execution Time (ms)	Throughput ($mpel/s$)
CLIF	61.28	6690.00	9.1645
CLIF-II camera 0 (vehicle traffic)	10.23	1769.64	5.8119
CLIF-II camera 1 (noise from sun glint)	10.23	2045.13	5.0079

deviation of 2.2047. These values suggest that processing has a noticeable per frame constant overhead. Much of this can likely be attributed to informational data visualization which would not be needed in a production implementation. Even though the current implementation does not meet the expected 1fps data rate of the sensor, it is reasonable to a assume additional optimizations and a faster CPU can achieve the desired processing rate at the single camera resolution. This sensor has six cameras which can be processed on multiple compute cores.

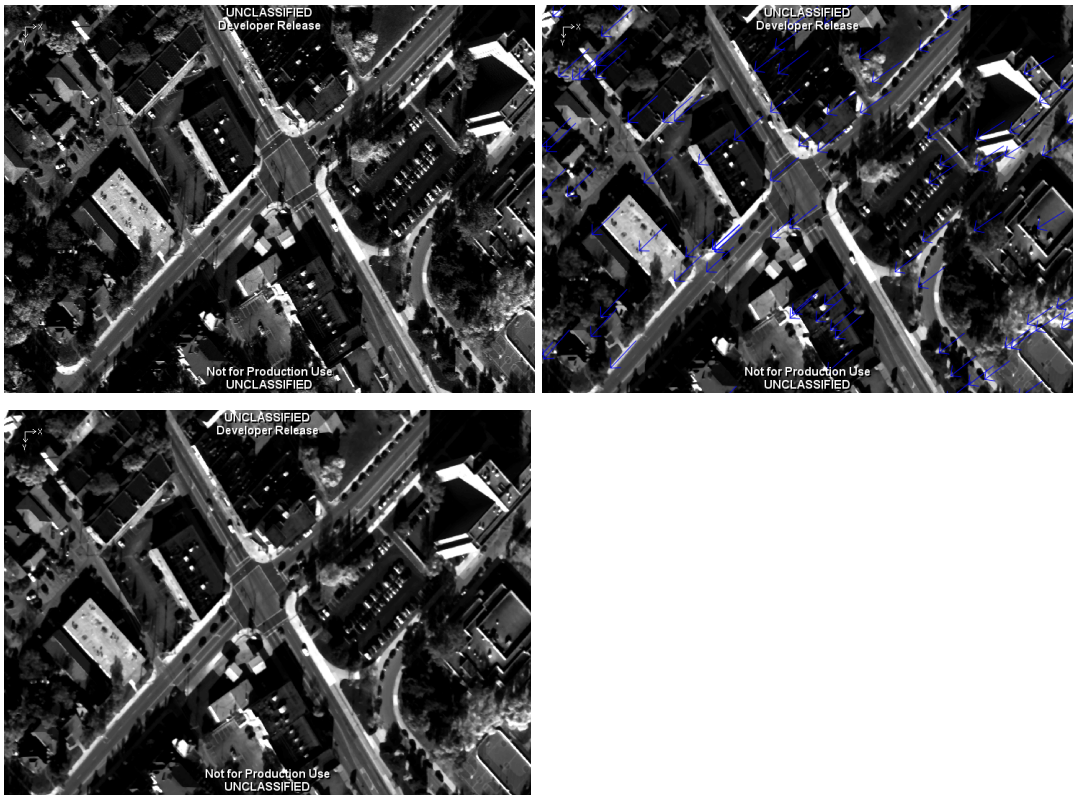

Figure 6: CLIF-II Motion Correction

from top left CLIF-II frame 0, frame 1 with optical flow vectors, frame 0 corrected for sensor motion

5.3 Object Detection

Figure 7 shows two frames from the CLIF-II dataset. Optical flow camera motion correction has been applied to the frames. Objects detected from residual motion are shown and color coded to indicate distinct objects. All moving objects have been at least partially identified although in

Table 5: Execution time and throughput for residual motion object detection

Sensor	Size (mpel)	Execution Time (ms)	Throughput ($mpel/s$)
CLIF-II full scene	61.40	3285.6	19.187
CLIF-II camera 0	10.23	476.68	21.476
CLIF-II camera 1	10.23	483.28	21.228

some cases only part of the object have been flagged. In another case, two cars passing each other are flagged as the same object in frame 1.

Timing information for object detection for two sub-sets of the CLIF-II dataset are shown below. Camera 0 contains an intersection with a number of moving vehicles. Camera 1 has fewer moving vehicles and a significant amount of noise due to shadows and sun glint. The results are averaged over 5 frame transitions. The standard deviation for the camera 0 data is 0.6146 and for camera 1 is 0.4872. Full scene data is also included so that some information on scaling can be inferred. The standard deviation for the full scene data is 1.629.

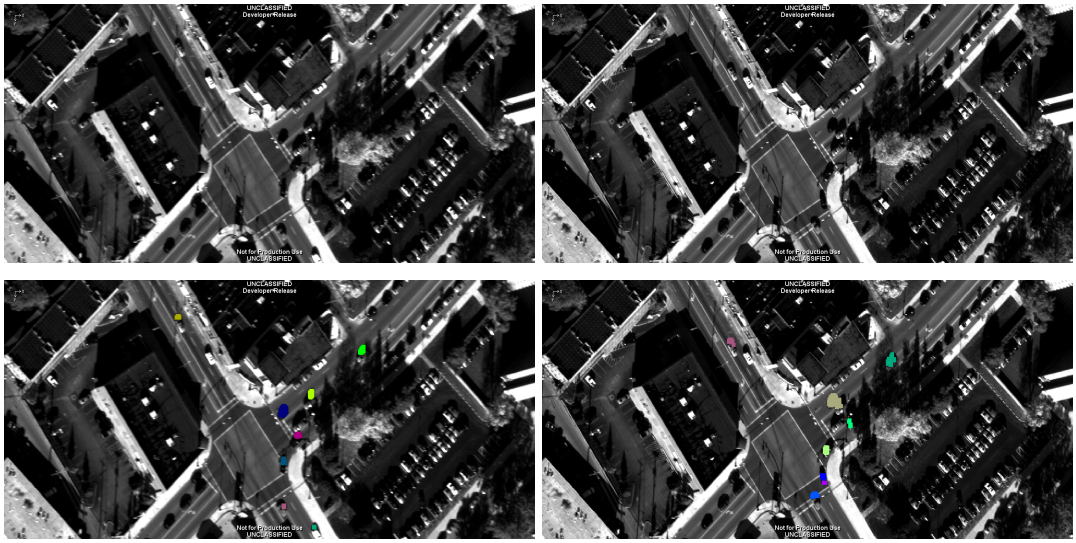

Figure 7: CLIF-II Object Detection

left to right from top CLIF-II frame 0, frame 1, frame 0 with detected object, frame 1 with detected objects

5.4 Object Tracking

A WAAS analyst was asked to locate moving objects and assign tracks over 5 frames of data. A mission objective is always provided when WAAS analysis is performed. This allows the analyst to limit evaluation to relevant data. The situation presented was a need to divert a VIP motorcade through a residential neighborhood. The analyst was asked to provide a situational awareness report within a short suspense of 10 minutes. Figure 8 shows identified tracks in the CLIF-II

camera 0 scene. The blue tracks represent objects located in the raw data stream and the green tracks represents objects located after camera motion was removed. The importance of camera motion compensation can be seen by the number of additional tracks found after motion compensation. Figure 9 shows a similar image comparing the analyst tracks to objects located using the automated method. Detail of the intersection is show in Figure 10 as this area of the data has a significant amount of activity. These images show that the analyst was able to locate 7 additional tracks, some of them indicating very little motion. These slower objects are people which the algorithm has difficulty detecting. This is likely due to the various blurring operations and morphological erosions which remove small detected object candidates. There are also a number of tracks which the algorithm located but it required an additional frame of data. Looking more closely at these locations indicates that the objects are obscured by shadows in all cases. This results in dark areas a low signal to noise ratio which are likely removed during the threshold portion of the object detection algorithm.

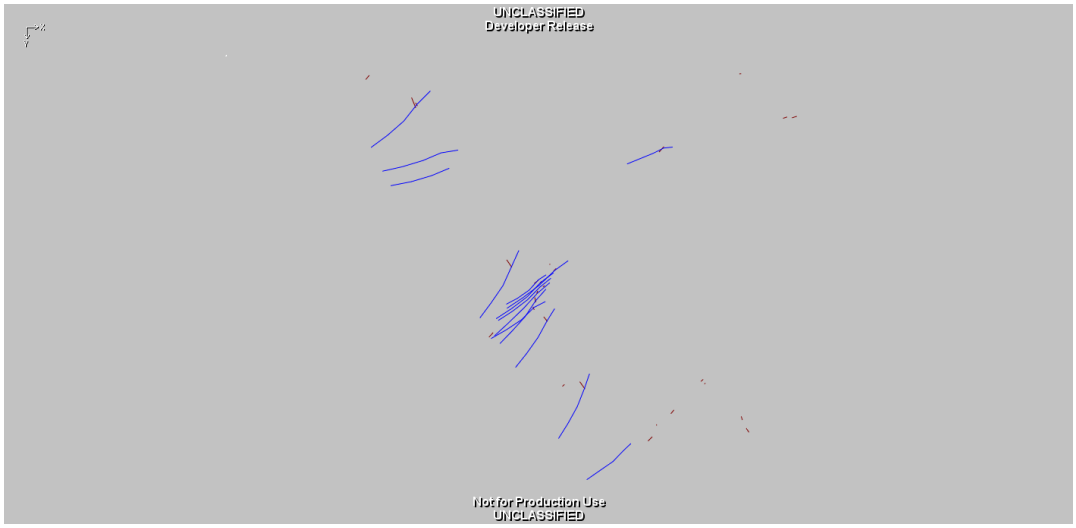

Figure 8: Tracks found in raw data (blue) and in data with camera motion removed (orange)

Another problem can be seen in Figure 10 near the center of the image on the upper right road. Two cars are passing each other but the algorithm detects a single, multi-lobed object instead of two distinct objects. Hyperspectral data for this scene is not available but if it were, the different spectra of the two vehicles, due to different paint colors, would be used to distinguish the two vehicles. Figure 11 compares the spectra of two vehicles with different paint colors.

Tracking objects which leave a sub-frame can be non-optimal in some circumstances as shown in Figure 12. An object moving outside of a sub-frame, represented by locations 1, 2, and 3, will reach the edge of sub-frame A at location 2. Tracking this object to location 3 requires processing of a larger sub-frame. The next larger sub-frame is B but the edge of A is also the edge of B. Therefore the entire frame must be processed in order to locate position 3 in sub-frame C. This

Figure 9: Tracks found in data with camera motion removed (orange lines) and objects found using the automated method. The two colored blobs indicate object locations in different frames.

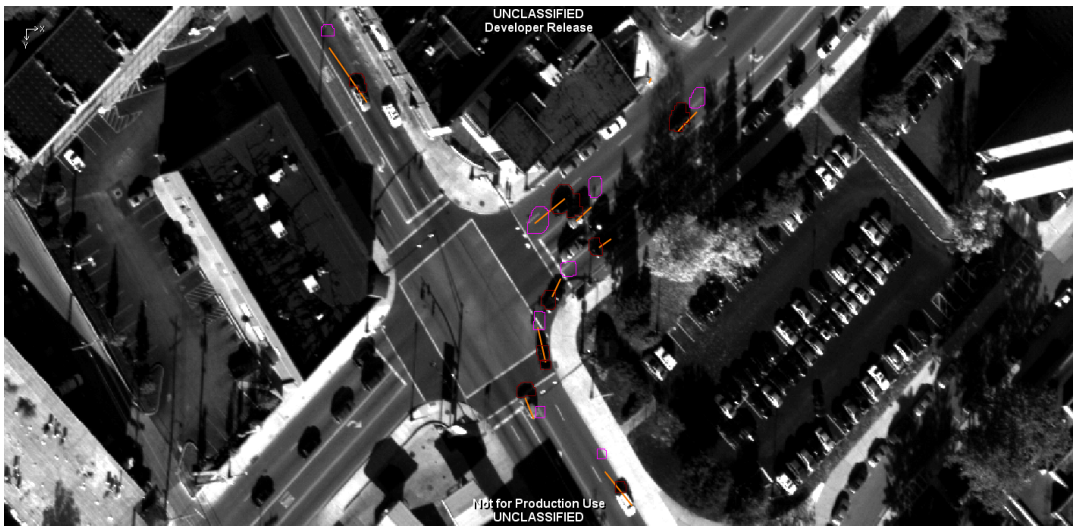

Figure 10: Intersection detail with analyst tracks and automated object detection.

could be resolved by allowing arbitrary sub-frames but this increases the number of possible sub-frames. Another way to address this problem is to locate the adjacent sub-frame instead of using the next larger sub-frame in the quadtree.

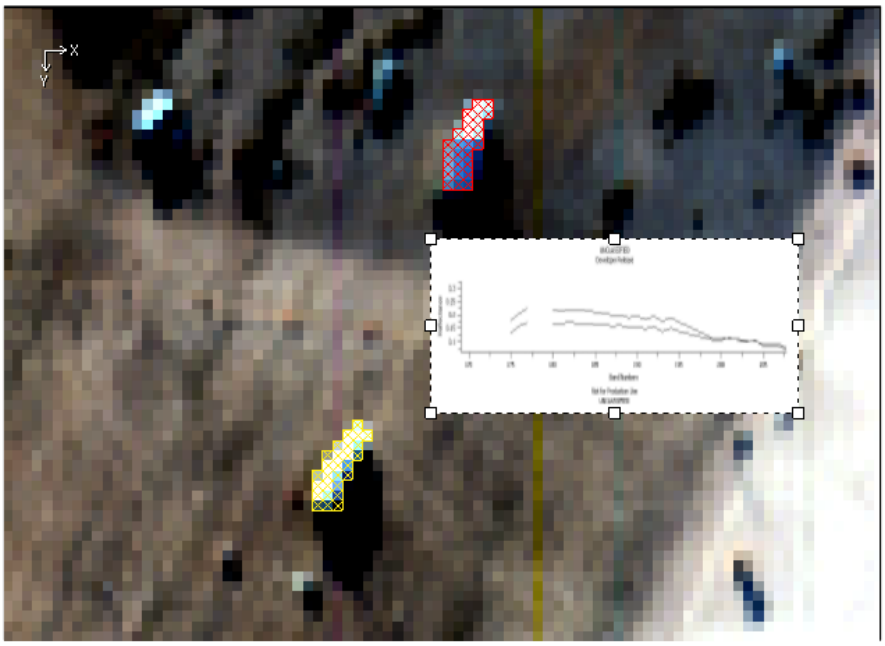

Figure 11: Spectral comparison of two car paint colors.

6 Conclusions

The algorithms presented in this thesis are effective at compensating for camera motion in WAAS data, locating objects of interest in hyperspectral and panchromatic WAAS data, and identifying tracks between interconnected objects. The execution speed and throughput for these algorithms is not sufficient to match the framerate of the WAAS data at full spatial resolution. Processing a portion of the image using a quadtree decomposition on the frames maintains the framerate of the data. The size of the processing area can be adjusted to match the processing capability of the workstation.

The K-RXD hyperspectral anomaly detection algorithm is effective at locating spectrally distinct areas in images with varying signal to noise. However, other anomaly detection algorithms may be more efficient in terms of processing requirements. Further exploration of other anomaly detection algorithms, especially NSWTD may decrease the processing requirements for hyperspectral cueing without serious effect on the quality of the results.

The lack of availability of coordinated hyperspectral and WAAS data collections prevents an integrated test of the proposed tracking system. While this thesis provides arguments that the integrated system will prove efficient and effective in production situations, coordinated data collections of both types of imagery are required to continue research on this topic. The increased interest in WAAS by commercial and government entities suggests that these data collects may

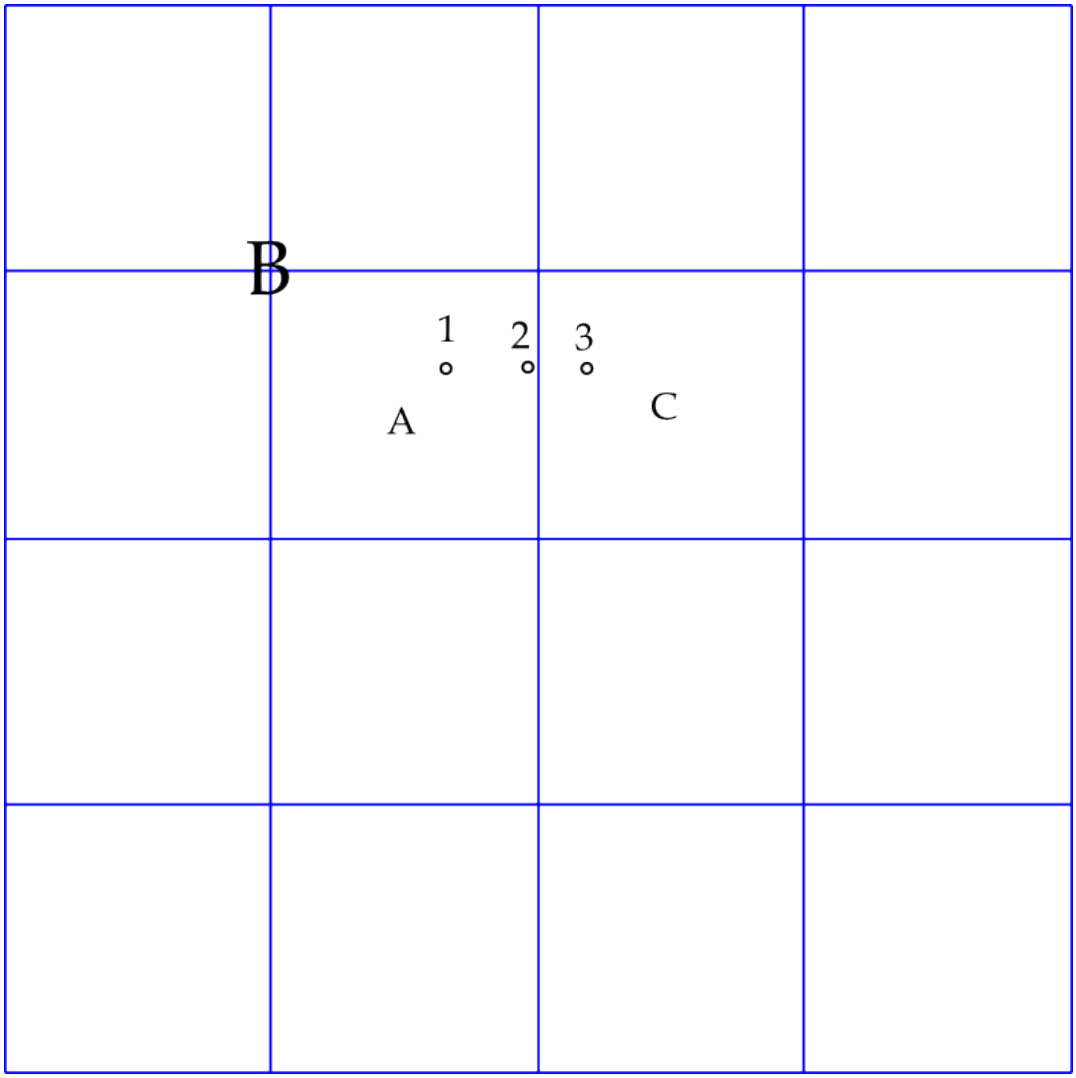

Figure 12: An object at locations 1, 2, and 3 in subsequent frames requires processing of the entire image.

occur in the coming years. Testing the integrated system on operational data is of prime interest for future research.

References

[1] Wenshuai Yu, Xuchu Yu, Penqiang Zhang, and Jun Zhou. A New Framework of Moving Target Detection and Tracking for UAV Video Application. *The International Archives of the Photogrammetry, Remote Sensing, and Spatial Information Sciences*, 37:609–613, 2008. [cited at p. 1]

[2] J. Xiao, C. Yang, F. Han, H. Cheng, et al. Vehicle and Person Tracking in UAV Videos. *Classification of Events, Activities and Relationships, Baltimore, 2007*, 2007. [cited at p. 1]

[3] I. Cohen and G. Medioni. Detecting and tracking moving objects for video surveillance. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 319–325. Citeseer, 1999. [cited at p. 1, 9]

[4] IS Reed and X. Yu. Adaptive multiple-band CFAR detection of an optical pattern withunknown spectral distribution. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(10):1760–1770, 1990. [cited at p. 4]

[5] C.I. Chang. *Hyperspectral data exploitation: theory and applications*. Wiley-Blackwell, 2007. [cited at p. 4]

[6] W. Liu and C.I. Chang. A nested spatial window-based approach to target detection for hyperspectral imagery. In *2004 IEEE International Geoscience and Remote Sensing Symposium, 2004. IGARSS'04. Proceedings*, volume 1, 2004. [cited at p. 4]

[7] Joseph C. Harsanyi, Chein i Chang, and Senior Member. Hyperspectral image classification and dimensionality reduction: an orthogonal subspace projection approach. *IEEE Transactions on Geoscience and Remote Sensing*, 32:779–785, 1994. [cited at p. 4]

[8] SR Soofbaf, H. Fahimnejad, M.J.V. Zoej, and B. Mojaradi. ANOMALY DETECTION ALGORITHMS FOR HYPERSPECTRAL IMAGERY. [cited at p. 5]

[9] R.H. Yuhas, A.F.H. Goetz, and J.W. Boardman. Discrimination among semi-arid landscape endmembers using the spectral angle mapper (SAM) algorithm. In *Summaries of the Third annual JPL airborne geoscience workshop*, volume 1, pages 92–14. Pasadena, CA: JPL Publication, 1992. [cited at p. 5]

[10] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. pages 674–679, 1981. [cited at p. 6, 13]

[11] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning opencv, 1st edition*. O'Reilly Media, Inc., 2008. [cited at p. 7]

[12] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. [cited at p. 7]

[13] Y. Wang, J.F. Doherty, and R.E. Van Dyck. Moving object tracking in video. In *Proceedings of the 29th Applied Imagery Pattern Recognition Workshop*, page 95. IEEE Computer Society, 2000. [cited at p. 7]

[14] S. Mann and R.W. Picard. Video orbits of the projective group: A simple approach to featureless estimation of parameters. *IEEE Transactions on Image Processing*, 6(9):1281–1295, 1997. [cited at p. 8]

[15] S. Lertrattanapanich and NK Bose. Latest results on high-resolution reconstruction from video sequences. *IEIC Technical Report (Institute of Electronics, Information and Communication Engineers)*, 99(505):59–65, 1999. [cited at p. 8]

[16] Columbus large image format (clif) 2006 dataset. Website. https://www.sdms.afrl.af.mil/datasets/clif2006. [cited at p. 11]

[17] Columbus large image format (clif) 2007 dataset. Website. https://www.sdms.afrl.af.mil/datasets/clif2007. [cited at p. 11]

[18] Soofbaf, S.R., ValadanZoej, M.J, and Ashoori, H. Efficient Detection of Anomalies in Hyperspectral Images. *The International Archives of the Photogrammetry, Remote Sensing, and Spatial Information Sciences*, 37:303–308, 2008. [cited at p. 15]

[19] R. Basedow, P. Silverglate, W. Rappoport, R. Rockwell, D. Rosenberg, K. Shu, R. Whittlesey, and E. Zalewski. The HYDICE instrument design and its application to planetary instruments. In J. F. Appleby, editor, *Advanced Technologies for Planetary Instruments*, pages 1–+, 1993. [cited at p. 17]

[20] J. Zadnik, D. Guerin, R. Moss, A. Orbeta, R. Dixon, C. G. Simi, S. Dunbar, and A. Hill. Calibration procedures and measurements for the COMPASS hyperspectral imager. In S. S. Shen & P. E. Lewis, editor, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5425 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 182–188, August 2004. [cited at p. 17]

[21] J.R. Schott. Modular imaging spectrometer instrument (misi). In *ACSM/ASPRS Annual Convention and Exposition*, February 1993. [cited at p. 17]

[22] G. Vane, R.O. Green, T.G. Chrien, H.T. Enmark, E.G. Hansen, and W.M. Porter. The airborne visible/infrared imaging spectrometer (AVIRIS). *Remote Sensing of Environment*, 44(2-3):127–143, 1993. [cited at p. 17]

[23] NASIC. Hyperspectral collection and analysis system (hycas). Technical report, National Air Intelligence Center, November 2001. [cited at p. 17]

A Source Code Listing

The following pages contain source code listings for the test application. This source code builds a plug-in for the Opticks analysis workbench which can be obtained from http://www.opticks.org

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Ball Aerospace & Technologies Corporation
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #include "AoiElement.h"
11 #include "AppVerify.h"
12 #include "BitMaskIterator.h"
13 #include "DataAccessorImpl.h"
14 #include "DataRequest.h"
15 #include "DesktopServices.h"
16 #include "ObjectResource.h"
17 #include "PlugInArgList.h"
18 #include "PlugInManagerServices.h"
19 #include "PlugInRegistration.h"
20 #include "PlugInResource.h"
21 #include "ProgressTracker.h"
22 #include "RasterDataDescriptor.h"
23 #include "RasterElement.h"
24 #include "RasterUtilities.h"
25 #include "SpatialDataView.h"
26 #include "Rx.h"
27 #include "ThresholdLayer.h"
28 #include <gsl/gsl_blas.h>
29 #include <gsl/gsl_matrix.h>
30
31 REGISTER_PLUGIN_BASIC(Tracking, Rx);
32
33 Rx::Rx()
34 {
35    setName("Rx");
36    setDescriptorId("{55f85fb3-1a65-4686-9fdf-2c759383fbcb}");
37    setSubtype("Anomaly Detection");
38    setMenuLocation("[Tracking]/RX");
39 }
40
41 Rx::~Rx()
42 {
43 }
44
45 bool Rx::getInputSpecification(PlugInArgList*& pArgList)
46 {
47    VERIFY(pArgList = Service<PlugInManagerServices>()->getPlugInArgList());
48    VERIFY(pArgList->addArg<Progress>(ProgressArg(), NULL));
49    VERIFY(pArgList->addArg<RasterElement>(DataElementArg()));
50    VERIFY(pArgList->addArg<SpatialDataView>(ViewArg()));
51    VERIFY(pArgList->addArg<AoiElement>("AOI", NULL));
52    return true;
53 }
54
55 bool Rx::getOutputSpecification(PlugInArgList*& pArgList)
56 {
57    VERIFY(pArgList = Service<PlugInManagerServices>()->getPlugInArgList());
58    VERIFY(pArgList->addArg<RasterElement>("Results"));
59    return true;
60 }
61
62 bool Rx::execute(PlugInArgList *pInArgList, PlugInArgList *pOutArgList)
63 {
64    VERIFY(pInArgList);
65    ProgressTracker progress(pInArgList->getPlugInArgValue<Progress>(ProgressArg()),
       "Executing RX.", "COAN", "{f5a21b68-013b-4d32-9923-b266e5311752}");
```

```
66
67    RasterElement* pElement = pInArgList->getPlugInArgValue<RasterElement>
       (DataElementArg());
68    SpatialDataView* pView = pInArgList->getPlugInArgValue<SpatialDataView>(ViewArg());
69    AoiElement* pAoi = pInArgList->getPlugInArgValue<AoiElement>("AOI");
70
71    if (pElement == NULL)
72    {
73       progress.report("No element specified.", 0, ERRORS, true);
74       return false;
75    }
76    RasterElement* pCov = NULL;
77    { // scope
78       bool success = true;
79       ExecutableResource covar("Covariance", std::string(), progress.getCurrentProgress
       (), isBatch());
80       success &= covar->getInArgList().setPlugInArgValue(DataElementArg(), pElement);
81       if (isBatch())
82       {
83          success &= covar->getInArgList().setPlugInArgValue("AOI", pAoi);
84       }
85       success &= covar->execute();
86       pCov = static_cast<RasterElement*>(
87          Service<ModelServices>()->getElement("Inverse Covariance Matrix",
        TypeConverter::toString<RasterElement>(), pElement));
88       success &= pCov != NULL;
89       if (!success)
90       {
91          progress.report("Unable to calculate covariance.", 0, ERRORS, true);
92          return false;
93       }
94    }
95    const RasterDataDescriptor* pDesc = static_cast<const RasterDataDescriptor*>
       (pElement->getDataDescriptor());
96    const BitMask* pBitmask = (pAoi == NULL) ? NULL : pAoi->getSelectedPoints();
97    BitMaskIterator iter(pBitmask, pElement);
98    FactoryResource<DataRequest> pReq;
99    pReq->setInterleaveFormat(BIP);
100   pReq->setRows(pDesc->getActiveRow(iter.getBoundingBoxStartRow()), pDesc->
       getActiveRow(iter.getBoundingBoxEndRow()));
101   pReq->setColumns(pDesc->getActiveColumn(iter.getBoundingBoxStartColumn()), pDesc->
       getActiveColumn(iter.getBoundingBoxEndColumn()));
102   DataAccessor acc(pElement->getDataAccessor(pReq.release()));
103
104   ModelResource<RasterElement> pResult(static_cast<RasterElement*>(
105      Service<ModelServices>()->getElement("RX Results", TypeConverter::toString
       <RasterElement>(), pElement)));
106   if (pResult.get() != NULL && !isBatch())
107   {
108      Service<DesktopServices>()->showSuppressibleMsgDlg("RX Results Exists",
109         "The results data element already exists and will be replaced.",
110         MESSAGE_WARNING, "Rx/ReplaceResults");
111      Service<ModelServices>()->destroyElement(pResult.release());
112   }
113   pResult = ModelResource<RasterElement>(
114      RasterUtilities::createRasterElement("RX Results", iter.getNumSelectedRows(),
       iter.getNumSelectedColumns(), FLT8BYTES, true, pElement));
115   if (pResult.get() == NULL)
116   {
117      progress.report("Unable to create results.", 0, ERRORS, true);
118      return false;
119   }
120   FactoryResource<DataRequest> pResReq;
121   pResReq->setWritable(true);
122   DataAccessor resacc(pResult->getDataAccessor(pResReq.release()));
123   if (!acc.isValid() || !resacc.isValid())
```

```
124    {
125       progress.report("Unable to access data.", 0, ERRORS, true);
126       return false;
127    }
128    { // scope temp matrices
129       int bands = pDesc->getBandCount();
130       EncodingType encoding = pDesc->getDataType();
131       gsl_matrix covMat = {bands, bands, bands, reinterpret_cast<double*>(pCov->
       getRawData()), NULL, 0};
132       gsl_matrix* pPixelMat = gsl_matrix_alloc(bands, 1);
133       gsl_matrix* pTemp = gsl_matrix_alloc(1, bands);
134       gsl_matrix* pMuMat = gsl_matrix_calloc(bands, 1);
135
136       for (int row = iter.getBoundingBoxStartRow(); iter != iter.end(); ++iter)
137       {
138          LocationType loc;
139          iter.getPixelLocation(loc);
140          if (loc.mY > iter.getBoundingBoxEndRow()) // work around a bug
141          {
142             break;
143          }
144          if (loc.mY > row)
145          {
146             row = static_cast<int>(loc.mY);
147             progress.report("Calculating means", row * 50 / iter.getNumSelectedRows(),
       NORMAL);
148          }
149          acc->toPixel(static_cast<int>(loc.mY), static_cast<int>(loc.mX));
150          VERIFY(acc.isValid());
151          for (int band = 0; band < bands; ++band)
152          {
153             double val = Service<ModelServices>()->getDataValue(encoding, acc->
       getColumn(), band);
154             gsl_matrix_set(pPixelMat, band, 0, val);
155          }
156          gsl_matrix_add(pMuMat, pPixelMat);
157       }
158       gsl_matrix_scale(pMuMat, 1.0 / iter.getCount());
159       iter.firstPixel();
160
161       for (int row = iter.getBoundingBoxStartRow(); iter != iter.end(); ++iter)
162       {
163          LocationType loc;
164          iter.getPixelLocation(loc);
165          if (loc.mY > iter.getBoundingBoxEndRow()) // work around a bug
166          {
167             break;
168          }
169          if (loc.mY > row)
170          {
171             row = static_cast<int>(loc.mY);
172             progress.report("Calculating RX", row * 49 / iter.getNumSelectedRows() + 50
       , NORMAL);
173          }
174          acc->toPixel(static_cast<int>(loc.mY), static_cast<int>(loc.mX));
175          resacc->toPixel(static_cast<int>(loc.mY) - iter.getBoundingBoxStartRow(),
       static_cast<int>(loc.mX) - iter.getBoundingBoxStartColumn());
176          VERIFY(acc.isValid() && resacc.isValid());
177
178          for (int band = 0; band < bands; ++band)
179          {
180             double val = Service<ModelServices>()->getDataValue(encoding, acc->
       getColumn(), band);
181             gsl_matrix_set(pPixelMat, band, 0, val);
182          }
183          gsl_matrix_sub(pPixelMat, pMuMat);
```

```
184         gsl_matrix resMat = {1, 1, 1, reinterpret_cast<double*>(resacc->getColumn()), ↙
     NULL, 0};
185         gsl_matrix_set_zero(pTemp);
186         gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, pPixelMat, &covMat, 0.0, pTemp);
187         gsl_blas_dgemm(CblasNoTrans, CblasNoTrans, 1.0, pTemp, pPixelMat, 0.0, &      ↙
     resMat);
188       }
189       gsl_matrix_free(pTemp);
190       gsl_matrix_free(pPixelMat);
191    }
192    if (!isBatch())
193    {
194       ThresholdLayer* pLayer = static_cast<ThresholdLayer*>(pView->createLayer         ↙
     (THRESHOLD, pResult.get()));
195       pLayer->setXOffset(iter.getBoundingBoxStartColumn());
196       pLayer->setYOffset(iter.getBoundingBoxStartRow());
197       pLayer->setPassArea(UPPER);
198       pLayer->setRegionUnits(STD_DEV);
199       pLayer->setFirstThreshold(pLayer->convertThreshold(STD_DEV, 2.0, RAW_VALUE));
200    }
201    if (pOutArgList != NULL)
202    {
203       pOutArgList->setPlugInArgValue<RasterElement>("Results", pResult.get());
204    }
205    pResult.release();
206
207    progress.report("Complete", 100, NORMAL);
208    progress.upALevel();
209    return true;
210 }
```

```
1  /*
2   * The information in this file is
3   * Copyright(c) 2010 Trevor R.H. Clarke
4   * and is subject to the terms and conditions of the
5   * GNU Lesser General Public License Version 2.1
6   * The license text is available from
7   * http://www.gnu.org/licenses/lgpl.html
8   */
9
10 #ifndef RX_H__
11 #define RX_H__
12
13 #include "AlgorithmShell.h"
14
15 class Rx : public AlgorithmShell
16 {
17 public:
18    Rx();
19    virtual ~Rx();
20
21    virtual bool getInputSpecification(PlugInArgList*& pArgList);
22    virtual bool getOutputSpecification(PlugInArgList*& pArgList);
23    virtual bool execute(PlugInArgList* pInArgList, PlugInArgList* pOutArgList);
24 };
25
26 #endif
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #include "AnnotationElement.h"
11 #include "AoiElement.h"
12 #include "AoiLayer.h"
13 #include "ApiUtilities.h"
14 #include "ColorMap.h"
15 #include "ColorType.h"
16 #include "DataAccessor.h"
17 #include "DataAccessorImpl.h"
18 #include "DataRequest.h"
19 #include "DesktopServices.h"
20 #include "GraphicGroup.h"
21 #include "GraphicObject.h"
22 #include "LayerList.h"
23 #include "MessageLogResource.h"
24 #include "PlugInRegistration.h"
25 #include "PseudocolorLayer.h"
26 #include "RasterDataDescriptor.h"
27 #include "RasterData.h"
28 #include "RasterElement.h"
29 #include "RasterLayer.h"
30 #include "SpatialDataView.h"
31 #include "Statistics.h"
32 #include "StringUtilities.h"
33 #include "ThresholdLayer.h"
34 #include "TrackingManager.h"
35 #include "TrackingUtils.h"
36 #include <opencv/cv.h>
37 #include <BlobResult.h>
38 #include <map>
39 #include <set>
40 #include <time.h>
41 #include <vector>
42 #include <QtCore/QtDebug>
43 #include <opencv/highgui.h>
44
45 REGISTER_PLUGIN_BASIC(Tracking, TrackingManager);
46
47 #define SQR(x) ((x) * (x))
48
49 // Define this to generate an annotation layer showing the optical flow vectors
50 // This will slow down processing quite a bit.
51 //#define SHOW_FLOW_VECTORS
52
53 // Define this to calculate connected components on the object results
54 #define CONNECTED
55
56 // Define the threshold for locating objects
57 #define OBJECT_THRESHOLD 15
58
59 // Maximum number of corners to use when calculating optical flow.
60 // Higher numbers may result in more accurate calculations but may also slow down ↙
       calculations.
61 // There's a point where increasing this number does nothing as there are only so many ↙
       strong corners in a frame.
62 #define MAX_CORNERS 500
63
64 // Maximum change in location allowed before a track is considered...lower values ↙
```

```
    shrink the possible search space for matching objects
#define MAX_SPEED 150

namespace
{
static const int CodeDeltas[8][2] =
{ {1, 0}, {1, -1}, {0, -1}, {-1, -1}, {-1, 0}, {-1, 1}, {0, 1}, {1, 1} };
}

const char* TrackingManager::spPlugInName("TrackingManager");

TrackingManager::TrackingManager() :
      mCalcBaseObjects(true),
      mPaused(false),
      mpDesc(NULL),
      mpElement(NULL),
      mBaseFrameNum(-1),
      mBaseAcc(NULL, NULL),
      mpBaseCorners(new CvPoint2D32f[MAX_CORNERS]),
      mCurrentFrameNum(-1),
      mpGroup(NULL),
      mpTracks(NULL),
      mCornerCount(MAX_CORNERS),
      mpRes(NULL),
      mpRes2(NULL)
{
   mpAnimation.addSignal(SIGNAL_NAME(Animation, FrameChanged), Slot(this, &
    TrackingManager::processFrame));
   mpLayer.addSignal(SIGNAL_NAME(Subject, Deleted), Slot(this, &TrackingManager::
    clearData));
   setName(spPlugInName);
   setDescriptorId("{c5f096e1-1584-4d7c-aa6f-29c4e422aad1}");
   setType("Manager");
   setSubtype("Video");
   setAbortSupported(false);
   allowMultipleInstances(false);
   executeOnStartup(true);
   destroyAfterExecute(false);
   setWizardSupported(false);
   setHandle(ModuleManager::instance()->getService());
}

TrackingManager::~TrackingManager()
{
   mpBaseFrame.reset(NULL);
}

bool TrackingManager::getInputSpecification(PlugInArgList*& pArgList)
{
   pArgList = NULL;
   return true;
}

bool TrackingManager::getOutputSpecification(PlugInArgList*& pArgList)
{
   pArgList = NULL;
   return true;
}

bool TrackingManager::execute(PlugInArgList*, PlugInArgList*)
{
   return true;
}

void TrackingManager::setTrackedLayer(RasterLayer* pLayer)
{
```

```
128    mpLayer.reset(pLayer);
129    mpAnimation.reset((pLayer == NULL) ? NULL : pLayer->getAnimation());
130    mpBasePyramid.reset(NULL);
131    mpBaseCorners.reset(NULL);
132    mpBaseFrame.reset(NULL);
133    mBaseAcc = DataAccessor(NULL, NULL);
134    initializeDataset();
135 }
136
137 void TrackingManager::setPauseState(bool state)
138 {
139    mPaused = state;
140 }
141
142 void TrackingManager::setFocus(LocationType loc, int maxSize)
143 {
144    if (mpDesc == NULL)
145    {
146       return;
147    }
148    LocationType dloc;
149    mpLayer->translateScreenToData(loc.mX, loc.mY, dloc.mX, dloc.mY);
150    Opticks::PixelLocation minBb(0,0);
151    Opticks::PixelLocation maxBb(mpDesc->getColumnCount(), mpDesc->getRowCount());
152    uint8_t level = TrackingUtils::calculateNeededLevels(maxSize, maxBb, minBb);
153    TrackingUtils::subcubeid_t id = TrackingUtils::calculateSubcubeId(dloc, level, maxBb
       , minBb);
154    TrackingUtils::calculateSubcubeBounds(id, level, maxBb, minBb);
155    mpFocus->clearPoints();
156    GraphicObject* pRect = mpFocus->getGroup()->addObject(RECTANGLE_OBJECT);
157    pRect->setBoundingBox(LocationType(minBb.mX, minBb.mY), LocationType(maxBb.mX, maxBb
       .mY));
158    pRect->setFillState(false);
159    mMinBb = minBb;
160    mMaxBb = maxBb;
161    mMaxBb.mX--;
162    mMaxBb.mY--;
163    initializeFrame0();
164 }
165
166 void TrackingManager::processFrame(Subject& subject, const std::string& signal, const
       boost::any& val)
167 {
168    if (mPaused)
169    {
170       return;
171    }
172    VERIFYNRV(mpLayer.get());
173    try
174    {
175       unsigned int curFrame = mpAnimation->getCurrentFrame()->mFrameNumber;
176       mCurrentFrameNum = curFrame;
177       int width = (*mpBaseFrame).width;
178       int height = (*mpBaseFrame).height;
179       bool fullScene = (width == mpDesc->getColumnCount() && height == mpDesc->
        getRowCount());
180
181       FactoryResource<DataRequest> req;
182       if (!fullScene)
183       {
184          req->setRows(mpDesc->getActiveRow(mMinBb.mY), mpDesc->getActiveRow(mMaxBb.mY),
        height);
185          req->setColumns(mpDesc->getActiveColumn(mMinBb.mX), mpDesc->getActiveColumn
       (mMaxBb.mX), width);
186       }
187       req->setBands(mpDesc->getActiveBand(curFrame), mpDesc->getActiveBand(curFrame),
```

```
      1);
188     req->setInterleaveFormat(BSQ);
189     DataAccessor curAcc = mpElement->getDataAccessor(req.release());
190
191     if (!curAcc.isValid())
192     {
193        return;
194     }
195     IplImageResource pCurFrame(NULL);
196     if (fullScene)
197     {
198        pCurFrame = IplImageResource(width, height, 8, 1, reinterpret_cast<char*>
    (curAcc->getColumn()));
199     }
200     else
201     {
202        pCurFrame = IplImageResource(width, height, 8, 1);
203        for (int row = 0; row < height; ++row)
204        {
205           if (row > 0)
206           {
207              curAcc->nextRow();
208              VERIFYNRV(curAcc.isValid());
209           }
210           memcpy((*pCurFrame).imageData + (row * width), curAcc->getRow(), (*
    pCurFrame).width);
211        }
212     }
213     CvSize pyr_sz = cvSize((*mpBaseFrame).width + 8, (*mpBaseFrame).height / 3);
214     IplImageResource pCurPyramid(pyr_sz.width, pyr_sz.height, IPL_DEPTH_32F, 1);
215     std::auto_ptr<CvPoint2D32f> pCurCorners(new CvPoint2D32f[MAX_CORNERS]);
216     cvCalcOpticalFlowPyrLK(mpBaseFrame, pCurFrame, mpBasePyramid, pCurPyramid,
    mpBaseCorners.get(), pCurCorners.get(), mCornerCount,
217        cvSize(10,10), 5, mpFeaturesFound, mpFeatureErrors, cvTermCriteria
    (CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.3), 0);
218
219     // display flow vectors in an annotation layer
220     std::vector<std::pair<CvPoint2D32f, CvPoint2D32f> > corr;
221
222     if (mpGroup != NULL)
223     {
224        mpGroup->removeAllObjects(true);
225     }
226     for (int i = 0; i < mCornerCount; ++i)
227     {
228        if (mpFeaturesFound[i] == 0 || mpFeatureErrors[i] > 550)
229        {
230           continue;
231        }
232        if (mpGroup != NULL)
233        {
234           GraphicObject* pObj = mpGroup->addObject(ARROW_OBJECT);
235           pObj->setBoundingBox(LocationType(mpBaseCorners.get()[i].x, mpBaseCorners.
    get()[i].y),
236              LocationType(pCurCorners.get()[i].x, pCurCorners.get()[i].y));
237        }
238        corr.push_back(std::make_pair(mpBaseCorners.get()[i], pCurCorners.get()[i]));
239     }
240     srand((unsigned int)time(NULL));
241     CvMat* pMapMatrix = TrackingUtils::ransac_affine(corr, 15, 5.0f, 5);
242     if (pMapMatrix != NULL)
243     {
244        IplImageResource pXform(width, height, 8, 1);
245        IplImageResource pTemp(width, height, 8, 1);
246        IplImageResource pRes(width, height, 8, 1, reinterpret_cast<char*>(mpRes->
    getRawData()));
```

```
247          IplImageResource pRes2(width, height, 8, 1, reinterpret_cast<char*>(mpRes2->
     getRawData()));
248          cvCopy(pCurFrame, pXform); // initialize to the current frame so any "offsets"
      have a consistent background
249          cvWarpAffine(mpBaseFrame, pXform, pMapMatrix, CV_INTER_LINEAR); // warp the
     base frame to the current camera position
250          cvCopy(pXform, mpBaseFrame); // copy the transformed base frame back to the
     raster element
251          cvSmooth(pXform, pRes, CV_MEDIAN, 5, 5); // smooth the base frame to remove
     high frequency noise
252          cvSmooth(pCurFrame, pTemp, CV_MEDIAN, 5, 5); // smooth the current frame to
     remove high frequency noise
253
254          // threhold the results
255          cvThreshold(pCurFrame, pTemp, OBJECT_THRESHOLD, 255, CV_THRESH_BINARY); //
     threshold the current frame
256          cvThreshold(pXform, pRes, OBJECT_THRESHOLD, 255, CV_THRESH_BINARY); //
     threshold the base frame
257          // erode to remove some noise
258          cvErode(pTemp, pTemp); // erode the current frame
259          cvErode(pRes,pRes); // erode the base frame
260          // difference the frames
261          cvSub(pTemp, pRes, pRes2); // subtract the base from the current and store in
     res2
262          cvSub(pRes, pTemp, pRes);  // subtract the current from the base and store in
     res
263          // remove final small differences with an open
264          cvErode(pRes,pRes, NULL, 3); // open the current frame
265          cvDilate(pRes,pRes,NULL,3);
266
267          cvErode(pRes2,pRes2, NULL, 3); // open the base frame
268          cvDilate(pRes2,pRes2,NULL,3);
269
270          std::vector<TrackVertex> curObjs;
271          { // scope blobs
272             CBlobResult blobs(pRes, NULL, 0);
273 #ifdef CONNECTED
274             for (int bidx = 0; bidx < blobs.GetNumBlobs(); ++bidx)
275             {
276                CBlob blob(blobs.GetBlob(bidx));
277                blob.FillBlob(pRes, CV_RGB(bidx+1,bidx+1,bidx+1));
278             }
279 #endif
280             curObjs = updateTrackObjects(blobs, pCurFrame, true);
281          } // scope blobs
282          { // scope blobs
283             CBlobResult blobs(pRes2, NULL, 0);
284 #ifdef CONNECTED
285             for (int bidx = 0; bidx < blobs.GetNumBlobs(); ++bidx)
286             {
287                CBlob blob(blobs.GetBlob(bidx));
288                blob.FillBlob(pRes2, CV_RGB(bidx+1,bidx+1,bidx+1));
289             }
290 #endif
291             if (mCalcBaseObjects)
292             {
293                mBaseObjects = updateTrackObjects(blobs, mpBaseFrame, false);
294             }
295             else
296             {
297                // apply affine transform to the coords
298                for (size_t idx = 0; idx < mBaseObjects.size(); ++idx)
299                {
300                   mTracks[mBaseObjects[idx]].mCentroidB.mX =
301                      (mTracks[mBaseObjects[idx]].mCentroidA.mX * cvGetReal2D(pMapMatrix
     , 0, 0)) +
```

```
302                      (mTracks[mBaseObjects[idx]].mCentroidA.mY * cvGetReal2D(pMapMatrix↙
     , 0, 1)) +
303                      cvGetReal2D(pMapMatrix, 0, 2);
304                   mTracks[mBaseObjects[idx]].mCentroidB.mY =
305                      (mTracks[mBaseObjects[idx]].mCentroidA.mX * cvGetReal2D(pMapMatrix↙
     , 1, 0)) +
306                      (mTracks[mBaseObjects[idx]].mCentroidA.mY * cvGetReal2D(pMapMatrix↙
     , 1, 1)) +
307                      cvGetReal2D(pMapMatrix, 1, 2);
308                }
309             }
310          } // scope blobs
311          matchTracks(curObjs);
312
313          if (!fullScene)
314          {
315             mBaseAcc->toPixel(mMinBb.mY, mMinBb.mX);
316             VERIFYNRV(mBaseAcc.isValid());
317             for (int row = 0; row < height; ++row)
318             {
319                if (row > 0)
320                {
321                   mBaseAcc->nextRow();
322                   VERIFYNRV(mBaseAcc.isValid());
323                }
324                memcpy(mBaseAcc->getRow(), (*mpBaseFrame).imageData + (row * width), (* ↙
     mpBaseFrame).width);
325             }
326          }
327          mpElement->updateData();
328          if (mpRes != NULL)
329          {
330             mpRes->updateData();
331          }
332          if (mpRes2 != NULL)
333          {
334             mpRes2->updateData();
335          }
336
337          cvReleaseMat(&pMapMatrix);
338          mBaseObjects = curObjs;
339       }
340
341       // prep for next frame
342       mpBasePyramid = pCurPyramid;
343       mpBaseCorners.reset(pCurCorners.release());
344       mpBaseFrame = pCurFrame;
345       mBaseAcc = curAcc;
346       mBaseFrameNum = mCurrentFrameNum;
347
348       mCornerCount = MAX_CORNERS;
349       cvGoodFeaturesToTrack(mpBaseFrame, mpEigImage, mpTmpImage, mpBaseCorners.get(), &↙
     mCornerCount, 0.01, 5.0);
350       cvFindCornerSubPix(mpBaseFrame, mpBaseCorners.get(), mCornerCount, cvSize(10, 10)↙
     , cvSize(-1, -1), cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.03));
351
352       mCalcBaseObjects = false;
353    }
354    catch (cv::Exception& err)
355    {
356       Service<DesktopServices>()->showMessageBox("OpenCV Error", err.err + "\n" + err. ↙
     file + ":" + StringUtilities::toDisplayString(err.line) + "\n" + err.func);
357    }
358 }
359
360 void TrackingManager::clearData(Subject& subject, const std::string& signal, const     ↙
```

```
    boost::any& val)
{
   setTrackedLayer(NULL);
}

void TrackingManager::initializeDataset()
{
   if (mpLayer.get() == NULL)
   {
      mpElement = NULL;
      mpDesc = NULL;
      return;
   }
   VERIFYNRV(mpElement = static_cast<RasterElement*>(mpLayer->getDataElement()));
   VERIFYNRV(mpDesc = static_cast<RasterDataDescriptor*>(mpElement->getDataDescriptor
    ()));
   if (mpDesc->getBytesPerElement() != 1)
   {
      // only 8-bit supported right now
      mpElement = NULL;
      mpDesc = NULL;
      return;
   }

   mpGroup = NULL;
   mpTracks = NULL;
#ifdef SHOW_FLOW_VECTORS
   {
   // Create elements and views to show the optical flow vectors
   ModelResource<AnnotationElement> pAnno(static_cast<AnnotationElement*>(
      Service<ModelServices>()->getElement("Flow Vectors", TypeConverter::toString
     <AnnotationElement>(), mpElement)));
   bool created = pAnno.get() == NULL;
   if (created)
   {
      pAnno = ModelResource<AnnotationElement>("Flow Vectors", mpElement);
      static_cast<SpatialDataView*>(mpLayer->getView())->createLayer(ANNOTATION, pAnno.
     get());
   }
   mpGroup = pAnno->getGroup();
   pAnno.release();
   }
#endif
   {
   // Create elements and views to show the optical flow vectors
   ModelResource<AnnotationElement> pTrackAnno(static_cast<AnnotationElement*>(
      Service<ModelServices>()->getElement("Object Tracks", TypeConverter::toString
     <AnnotationElement>(), mpElement)));
   bool created = pTrackAnno.get() == NULL;
   if (created)
   {
      pTrackAnno = ModelResource<AnnotationElement>("Object Tracks", mpElement);
      static_cast<SpatialDataView*>(mpLayer->getView())->createLayer(ANNOTATION,
     pTrackAnno.get());
   }
   mpTracks = pTrackAnno->getGroup();
   pTrackAnno.release();
   }

   // Create an AOI to define a sub-area for processing.
   mpFocus = static_cast<AoiElement*>(
      Service<ModelServices>()->getElement("Focus", TypeConverter::toString<AoiElement>
     (), mpElement));
   if (mpFocus == NULL)
   {
      mpFocus = static_cast<AoiElement*>(
```

```
420         Service<ModelServices>()->createElement("Focus", TypeConverter::toString
     <AoiElement>(), mpElement));
421       AoiLayer* pLayer = static_cast<AoiLayer*>(
422          static_cast<SpatialDataView*>(mpLayer->getView())->createLayer(AOI_LAYER,
     mpFocus));
423       pLayer->setSymbol(BOX);
424       pLayer->setColor(ColorType(0, 0, 128));
425    }
426    mMinBb = Opticks::PixelLocation(0, 0);
427    mMaxBb = Opticks::PixelLocation(mpDesc->getColumnCount()-1, mpDesc->getRowCount()-1)
     ;
428
429    initializeFrame0();
430 }
431
432 void TrackingManager::initializeFrame0()
433 {
434    if (mpLayer.get() == NULL)
435    {
436       mpElement = NULL;
437       mpDesc = NULL;
438       return;
439    }
440    try
441    {
442       unsigned int baseFrame = mpLayer->getDisplayedBand(GRAY).getActiveNumber();
443       mBaseFrameNum = baseFrame;
444       mCurrentFrameNum = -1;
445
446       int width = mMaxBb.mX - mMinBb.mX + 1;
447       int height = mMaxBb.mY - mMinBb.mY + 1;
448       bool fullScene = (width == mpDesc->getColumnCount() && height == mpDesc->
     getRowCount());
449
450       FactoryResource<DataRequest> baseRequest;
451       if (!fullScene)
452       {
453          baseRequest->setRows(mpDesc->getActiveRow(mMinBb.mY), mpDesc->getActiveRow
     (mMaxBb.mY), height);
454          baseRequest->setColumns(mpDesc->getActiveColumn(mMinBb.mX), mpDesc->
     getActiveColumn(mMaxBb.mX), width);
455       }
456       baseRequest->setBands(mpDesc->getActiveBand(baseFrame), mpDesc->getActiveBand
     (baseFrame), 1);
457       baseRequest->setInterleaveFormat(BSQ);
458       mBaseAcc = mpElement->getDataAccessor(baseRequest.release());
459
460       if (!mBaseAcc.isValid())
461       {
462          return;
463       }
464       // Wrap the accessors with OpenCV data structures and create some temporary
     arrays.
465       if (fullScene)
466       {
467          mpBaseFrame = IplImageResource(width, height, 8, 1, reinterpret_cast<char*>
     (mBaseAcc->getColumn()));
468       }
469       else
470       {
471          mpBaseFrame = IplImageResource(width, height, 8, 1);
472          for (int row = 0; row < height; ++row)
473          {
474             if (row > 0)
475             {
476                mBaseAcc->nextRow();
```

```
477               VERIFYNRV(mBaseAcc.isValid());
478            }
479            memcpy((*mpBaseFrame).imageData + (row * width), mBaseAcc->getRow(), (*
    mpBaseFrame).width);
480         }
481      }
482
483      mpEigImage = IplImageResource(width, height, IPL_DEPTH_32F, 1);
484      mpTmpImage = IplImageResource(width, height, IPL_DEPTH_32F, 1);
485      mCornerCount = MAX_CORNERS;
486      mpBaseCorners.reset(new CvPoint2D32f[MAX_CORNERS]);
487      cvGoodFeaturesToTrack(mpBaseFrame, mpEigImage, mpTmpImage, mpBaseCorners.get(), &
    mCornerCount, 0.01, 5.0);
488      cvFindCornerSubPix(mpBaseFrame, mpBaseCorners.get(), mCornerCount, cvSize(10, 10)
    , cvSize(-1, -1), cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.03));
489
490      CvSize pyr_sz = cvSize((*mpBaseFrame).width + 8, (*mpBaseFrame).height / 3);
491      mpBasePyramid = IplImageResource(pyr_sz.width, pyr_sz.height, IPL_DEPTH_32F, 1);
492
493      // Create elements and views to show identified objects
494      mpRes = NULL;
495      mpRes2 = NULL;
496
497      RasterElementArgs args={height, width, 1, 0, 1, 1, mpElement, 0, NULL}; // BSQ,
    uchar (ushort)
498      DataElement* pTmp = Service<ModelServices>()->getElement("Current Objects",
    TypeConverter::toString<RasterElement>(), mpElement);
499      if (pTmp != NULL)
500      {
501         Service<ModelServices>()->destroyElement(pTmp);
502      }
503      mpRes = static_cast<RasterElement*>(createRasterElement("Current Objects", args))
    ;
504 #pragma message(__FILE__ "(" STRING(__LINE__) ") : warning : Work around for OPTICKS-
    932 (tclarke)")
505      std::vector<int> badValues(1, 0);
506      mpRes->getStatistics()->setBadValues(badValues);
507 #ifdef CONNECTED
508      RasterLayer* pPseudo = static_cast<RasterLayer*>(
509         static_cast<SpatialDataView*>(mpLayer->getView())->createLayer(RASTER, mpRes))
    ;
510      pPseudo->setXOffset(mMinBb.mX);
511      pPseudo->setYOffset(mMinBb.mY);
512      pTmp = Service<ModelServices>()->getElement("Base Objects", TypeConverter::
    toString<RasterElement>(), mpElement);
513      if (pTmp != NULL)
514      {
515         Service<ModelServices>()->destroyElement(pTmp);
516      }
517      mpRes2 = static_cast<RasterElement*>(createRasterElement("Base Objects", args));
518      mpRes2->getStatistics()->setBadValues(badValues);
519      RasterLayer* pPseudo2 = static_cast<RasterLayer*>(
520         static_cast<SpatialDataView*>(mpLayer->getView())->createLayer(RASTER,
    mpRes2));
521      pPseudo2->setXOffset(mMinBb.mX);
522      pPseudo2->setYOffset(mMinBb.mY);
523      pPseudo->setStretchUnits(GRAYSCALE_MODE, RAW_VALUE);
524      pPseudo->setStretchValues(GRAY, 0, 47);
525      pPseudo2->setStretchUnits(GRAYSCALE_MODE, RAW_VALUE);
526      pPseudo2->setStretchValues(GRAY, 0, 47);
527      ColorMap cmap("C:/Opticks/COAN/Tracking/Release/SupportFiles/ColorTables/
    pseudocolor.clu");
528      pPseudo->setColorMap(cmap.getName(), cmap.getTable());
529      pPseudo2->setColorMap(cmap.getName(), cmap.getTable());
530 #endif
531
```

```
532       mCalcBaseObjects = true;
533    }
534    catch (const cv::Exception& exc)
535    {
536       MessageResource msg("OpenCV error occurred.", "tracking", "{b16ae5e9-ef7e-474f-
       9c45-8a061aa3cda2}");
537       msg->addProperty("Code", exc.code);
538       msg->addProperty("Error", exc.err);
539       msg->addProperty("Function", exc.func);
540       msg->addProperty("File", exc.file);
541       msg->addProperty("Line", exc.line);
542       msg->finalize(Message::Failure);
543    }
544 }
545
546 std::vector<TrackingManager::TrackVertex> TrackingManager::updateTrackObjects
       (CBlobResult& blobs, IplImage* pFrame, bool current)
547 {
548    // Get contour points for each blob
549    std::vector<TrackVertex> objects;
550    for (int blobi = 0; blobi < blobs.GetNumBlobs(); ++blobi)
551    {
552       std::map<int, std::set<int> > pts;
553       CBlob blob = blobs.GetBlob(blobi);
554       CBlobContour* pCon = blob.GetExternalContour();
555       CvTreeNodeIterator iter;
556       cvInitTreeNodeIterator(&iter, pCon->GetContourPoints(), 0);
557       CvSeq* pContour;
558       while((pContour = reinterpret_cast<CvSeq*>(cvNextTreeNode(&iter))) != 0 )
559       {
560          CvSeqReader reader;
561          int count = pContour->total;
562          int elem_type = CV_MAT_TYPE(pContour->flags);
563          cvStartReadSeq(pContour, &reader, 0);
564          if (CV_IS_SEQ_CHAIN_CONTOUR(pContour))
565          {
566             cv::Point pt = ((CvChain*)pContour)->origin;
567             char prev_code = reader.ptr ? reader.ptr[0] : '\0';
568
569             for (int i = 0; i < count; i++)
570             {
571                char code;
572                CV_READ_SEQ_ELEM(code, reader);
573
574                if (code != prev_code)
575                {
576                   prev_code = code;
577                   pts[pt.y].insert(pt.x);
578                }
579
580                pt.x += CodeDeltas[(int)code][0];
581                pt.y += CodeDeltas[(int)code][1];
582             }
583          }
584          else if (CV_IS_SEQ_POLYLINE(pContour) && elem_type == CV_32SC2)
585          {
586             cv::Point pt1, pt2;
587             int shift = 0;
588
589             count -= !CV_IS_SEQ_CLOSED(pContour);
590             CV_READ_SEQ_ELEM(pt1, reader);
591             pts[pt1.y].insert(pt1.x);
592
593             for(int i = 0; i < count; i++)
594             {
595                CV_READ_SEQ_ELEM(pt2, reader);
```

```
596                pts[pt2.y].insert(pt2.x);
597                pt1 = pt2;
598             }
599          }
600       }
601       // Fill the contour and calculate some properties
602       Opticks::PixelLocation centroid(0,0);
603       std::vector<cv::Point> filledPoints;
604       for (std::map<int, std::set<int> >::iterator ptiter = pts.begin(); ptiter != pts.↙
      end(); ++ptiter)
605       {
606          int startCol = -1;
607          for (std::set<int>::iterator coliter = ptiter->second.begin(); coliter !=     ↙
      ptiter->second.end(); ++coliter)
608          {
609             if (startCol == -1)
610             {
611                startCol = *coliter;
612             }
613             else
614             {
615                for (int col = startCol; col <= *coliter; ++col)
616                {
617                   filledPoints.push_back(cv::Point(col, ptiter->first));
618                   centroid.mX += col;
619                   centroid.mY += ptiter->first;
620                }
621                startCol = -1;
622             }
623          }
624       }
625       centroid.mX /= filledPoints.size();
626       centroid.mY /= filledPoints.size();
627       double tmpNum = 0.0, tmpDen = 0.0;
628       for (std::vector<cv::Point>::iterator ptiter = filledPoints.begin(); ptiter !=   ↙
      filledPoints.end(); ++ptiter)
629       {
630          double val = cvGetReal2D(pFrame, ptiter->y, ptiter->x);
631          tmpNum += sqrt((double)SQR(ptiter->x - centroid.mX) +
632                         SQR(ptiter->y - centroid.mY)) * val;
633          tmpDen += val;
634       }
635       TrackVertex obj;
636       if (current || mCalcBaseObjects)
637       {
638          obj = boost::add_vertex(mTracks);
639          mTracks[obj].mFrameNum = current ? mCurrentFrameNum : mBaseFrameNum;
640          mTracks[obj].mDispersion = static_cast<float>(tmpNum / tmpDen);
641       }
642       else
643       {
644          // locate the correct item.
645       }
646       if (current)
647       {
648          mTracks[obj].mCentroidA = centroid;
649       }
650       else
651       {
652          mTracks[obj].mCentroidB = centroid;
653       }
654       objects.push_back(obj);
655    }
656    return objects;
657 }
658
```

```
659 void TrackingManager::matchTracks(const std::vector<TrackVertex>& curObjs)
660 {
661    if (mpTracks != NULL)
662    {
663       mpTracks->removeAllObjects(true);
664    }
665    for (std::vector<TrackVertex>::const_iterator base = mBaseObjects.begin(); base !=
       mBaseObjects.end(); ++base)
666    {
667       std::pair<TrackTraits::in_edge_iterator, TrackTraits::in_edge_iterator> edges =
       boost::in_edges(*base, mTracks);
668       bool hasInVel = false;
669       Opticks::Location<int, 2> inVel(0, 0);
670       double inVelAng = 0.0;
671       if (edges.first != edges.second)
672       {
673          inVel = mTracks[*(edges.first)].mVelocity;
674          inVelAng = atan((double)inVel.mY / inVel.mX);
675          hasInVel = true;
676       }
677       // add connections which meet certain minimum criteria
678       TrackEdge minE;
679       float minCost = 999999999999999;
680       for (std::vector<TrackVertex>::const_iterator cur = curObjs.begin(); cur !=
       curObjs.end(); ++cur)
681       {
682          Opticks::Location<int, 2> vel(mTracks[*cur].mCentroidA.mX - mTracks[*base].
       mCentroidB.mX,
683                                        mTracks[*cur].mCentroidA.mY - mTracks[*base].
       mCentroidB.mY);
684          double velDiff = 0.0;
685          if (hasInVel)
686          {
687             velDiff = fabs(inVelAng - atan((double)vel.mY / vel.mX));
688          }
689          if (vel.length() < MAX_SPEED)
690          {
691             TrackEdge e = boost::add_edge(*base, *cur, mTracks).first;
692             mTracks[e].mVelocity = vel;
693             mTracks[e].mVelDiff = velDiff;
694             mTracks[e].mSpeedDiff = fabs(vel.length() - inVel.length());
695             mTracks[e].mDiffDispersion = fabs(mTracks[*base].mDispersion - mTracks[*
       cur].mDispersion);
696             mTracks[e].mCost = mTracks[e].mDiffDispersion * 0.2
697                              + mTracks[e].mVelDiff * 0.5
698                              + mTracks[e].mSpeedDiff * 0.3;
699             if (mTracks[e].mCost < minCost)
700             {
701                minCost = mTracks[e].mCost;
702                minE = e;
703             }
704          }
705       }
706       TrackTraits::edge_iterator ei, eitmp, ei_end;
707       for (boost::tie(ei, ei_end) = boost::edges(mTracks); ei != ei_end;)
708       {
709          eitmp = ei;
710          ++eitmp;
711          if (*ei != minE)
712          {
713             boost::remove_edge(*ei, mTracks);
714          }
715          else
716          {
717             LocationType start(mTracks[boost::source(*ei, mTracks)].mCentroidB.mX,
       mTracks[boost::source(*ei, mTracks)].mCentroidB.mY);
```

```
718             LocationType stop(mTracks[boost::target(*ei, mTracks)].mCentroidA.mX,
    mTracks[boost::target(*ei, mTracks)].mCentroidA.mY);
719             mpTracks->addObject(LINE_OBJECT)->setBoundingBox(start, stop);
720          }
721          ei = eitmp;
722       }
723    }
724    // diff in velocity
725    // ang = atan(A.y/A.x) - atan(B.y/B.x); if (ang > 180) ang = 360 - ang
726    // mag = fabs(A.distance() - B.distance())
727 }
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #ifndef TRACKINGMANAGER_H__
11 #define TRACKINGMANAGER_H__
12
13 #include "Animation.h"
14 #include "AttachmentPtr.h"
15 #include "ConfigurationSettings.h"
16 #include "DataAccessor.h"
17 #include "ExecutableShell.h"
18 #include "RasterData.h"
19 #include "RasterLayer.h"
20 #include "TrackingUtils.h"
21 #include <boost/any.hpp>
22 #include <boost/graph/graph_traits.hpp>
23 #include <boost/graph/adjacency_list.hpp>
24
25 class AoiElement;
26 class CBlobResult;
27 class GraphicGroup;
28 class RasterDataDescriptor;
29 class RasterElement;
30
31 class TrackingManager : public ExecutableShell
32 {
33 public:
34    SETTING(InitialSubcubeSize, TrackingManager, unsigned int, 0);
35
36    static const char* spPlugInName;
37
38    TrackingManager();
39    virtual ~TrackingManager();
40
41    virtual bool getInputSpecification(PlugInArgList*& pArgList);
42    virtual bool getOutputSpecification(PlugInArgList*& pArgList);
43    virtual bool execute(PlugInArgList* pInArgList, PlugInArgList* pOutArgList);
44
45    void setTrackedLayer(RasterLayer* pLayer);
46    void setPauseState(bool state);
47    void setFocus(LocationType loc, int maxSize);
48
49    struct TrackVertexProps
50    {
51       TrackVertexProps() : mFrameNum(-1), mCentroidA(0,0), mCentroidB(0,0), mTexture(0.↙
       0), mDispersion(0.0) {}
52
53       int mFrameNum;                      // frame number where this objects was found
54       Opticks::PixelLocation mCentroidA; // position when this is the "current" frame  ↙
      (pre-transform)
55       Opticks::PixelLocation mCentroidB; // position when this is the "base" frame     ↙
      (post-transform)
56       float mTexture;                     // grayscale texture
57       float mDispersion;                  // grayscale dispersion
58       // optional HSI sig goes here
59    };
60    struct TrackEdgeProps
61    {
62       TrackEdgeProps() : mVelocity(0,0), mVelDiff(0.0), mSpeedDiff(0.0),              ↙
       mDiffDispersion(0.0) {}
```

```
63
64        Opticks::Location<int, 2> mVelocity; // previous frame's centroid B to this frame↙
       's centroid A
65        float mVelDiff;                      // angular difference between this velocity ↙
       vector and the previous location's
66        float mSpeedDiff;                    // magnitude difference between this       ↙
       velocity vector and the previous location's
67        float mDiffDispersion;               // difference in dispersion values
68        float mCost;                         // total "cost" of this track
69     };
70     typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::bidirectionalS,      ↙
        TrackVertexProps, TrackEdgeProps> TrackGraph;
71     typedef boost::graph_traits<TrackGraph> TrackTraits;
72     typedef boost::graph_traits<TrackGraph>::vertex_descriptor TrackVertex;
73     typedef boost::graph_traits<TrackGraph>::edge_descriptor TrackEdge;
74
75  protected:
76     void processFrame(Subject& subject, const std::string& signal, const boost::any&    ↙
        val);
77     void clearData(Subject& subject, const std::string& signal, const boost::any& val);
78     std::vector<TrackVertex> updateTrackObjects(CBlobResult& blobs, IplImage* pFrame,   ↙
        bool current);
79     void matchTracks(const std::vector<TrackVertex>& curObjs);
80
81  private:
82     void initializeDataset();
83     void initializeFrame0();
84
85     TrackGraph mTracks;
86
87     bool mCalcBaseObjects; // should the base objects be calculated or results from the ↙
        previous iteration used?
88     std::vector<TrackVertex> mBaseObjects;
89     bool mPaused;
90     AttachmentPtr<RasterLayer> mpLayer; // tracked layer
91     AttachmentPtr<Animation> mpAnimation; // tracked animation
92     const RasterDataDescriptor* mpDesc; // tracked element descriptor
93     RasterElement* mpElement; // tracked element
94
95     // base frame information, passed to the next iteration
96     int mBaseFrameNum;
97     DataAccessor mBaseAcc;
98     IplImageResource mpBaseFrame;
99     IplImageResource mpEigImage;
100    IplImageResource mpTmpImage;
101    std::auto_ptr<CvPoint2D32f> mpBaseCorners;
102    IplImageResource mpBasePyramid;
103
104    // pass feature state to the next iteration
105    char mpFeaturesFound[500];
106    float mpFeatureErrors[500];
107
108    int mCurrentFrameNum;
109    GraphicGroup* mpGroup; // draw flow vectors
110    GraphicGroup* mpTracks; // draw tracks
111
112    int mCornerCount;
113
114    RasterElement* mpRes; // base frame object blobs
115    RasterElement* mpRes2; // current frame object blobs
116
117    // sub-frame AOI and bounding box information
118    AoiElement* mpFocus;
119    Opticks::PixelLocation mMinBb;
120    Opticks::PixelLocation mMaxBb;
121 };
```

```
122
123 #endif
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #include "AppConfig.h"
11 #include "LocationType.h"
12 #include "RasterData.h"
13 #include "TrackingUtils.h"
14 #include <stdlib.h>
15 #include <opencv/cv.h>
16
17 #if defined(WIN_API)
18 #include <boost/cstdint.hpp>
19 using boost::uint8_t;
20 #endif
21
22 namespace TrackingUtils
23 {
24 subcubeid_t calculateSubcubeId(LocationType location, uint8_t levels,
25                                Opticks::PixelLocation maxBb,
26                                Opticks::PixelLocation minBb)
27 {
28    subcubeid_t id(levels);
29    for (uint8_t level = 0; level < levels; ++level)
30    {
31       Opticks::PixelLocation center(static_cast<int>((maxBb.mX - minBb.mX + 1) / 2.0 + ↙
     minBb.mX),
32                                     static_cast<int>((maxBb.mY - minBb.mY + 1) / 2.0 + ↙
     minBb.mY));
33       if (location.mX >= center.mX && location.mY <= center.mY) // quadrant 0
34       {
35          minBb.mX = center.mX;
36          maxBb.mY = center.mY;
37       }
38       else if (location.mX < center.mX && location.mY <= center.mY) // quadrant 1
39       {
40          id |= 1 << (level * 2 + 4);
41          maxBb = center;
42       }
43       else if (location.mX < center.mX && location.mY > center.mY) // quadrant 2
44       {
45          id |= 2 << (level * 2 + 4);
46          maxBb.mX = center.mX;
47          minBb.mY = center.mY;
48       }
49       else // quadrant 3
50       {
51          id |= 3 << (level * 2 + 4);
52          minBb = center;
53       }
54    }
55    return id;
56 }
57
58 bool calculateSubcubeBounds(subcubeid_t subcubeId, uint8_t& levels, Opticks::          ↙
     PixelLocation& maxBb, Opticks::PixelLocation& minBb)
59 {
60    levels = subcubeId & 0x0f;
61    subcubeId >>= 4;
62    for (uint8_t level = 0; level < levels; ++level)
63    {
```

```
64       Opticks::PixelLocation center(static_cast<int>((maxBb.mX - minBb.mX + 1) / 2.0 + minBb.mX),
65                                     static_cast<int>((maxBb.mY - minBb.mY + 1) / 2.0 + minBb.mY));
66       switch (subcubeId & 0x03) // quadrant number
67       {
68       case 0:
69          minBb.mX = center.mX;
70          maxBb.mY = center.mY;
71          break;
72       case 1:
73          maxBb = center;
74          break;
75       case 2:
76          maxBb.mX = center.mX;
77          minBb.mY = center.mY;
78          break;
79       case 3:
80          minBb = center;
81          break;
82       default:
83          break; // unreachable
84       }
85       subcubeId >>= 2;
86    }
87    return subcubeId == 0;
88 }
89
90 uint8_t calculateNeededLevels(uint32_t maxSubcubeSize,
91                               Opticks::PixelLocation maxBb,
92                               Opticks::PixelLocation minBb)
93 {
94    maxBb = (maxBb - minBb) + 1;
95    for (uint8_t level = 0; level <= 12; ++level) // maximum number of levels which can be represented
96    {
97       uint32_t curSize = maxBb.mX * maxBb.mY;
98       if (curSize <= maxSubcubeSize)
99       {
100         return level;
101      }
102      maxBb.mX /= 2;
103      maxBb.mY /= 2;
104   }
105   return 0xff; // Invalid return
106 }
107
108 CvMat* ransac_affine(std::vector<std::pair<CvPoint2D32f, CvPoint2D32f> >& corr, int maxIter, float thresh, unsigned int numNeeded)
109 {
110   if (corr.empty())
111   {
112      return NULL;
113   }
114   thresh = thresh * thresh; // calculating sqrt for each test is expensive, so we square the check value to change from RMS to MSE
115   unsigned int bestCount = 0;
116   CvMat* pBest = NULL;
117   CvMat* pTest = cvCreateMat(2, 3, CV_32F);
118   CvMat* pVer = cvCreateMat(3, 1, CV_32F);
119   CvMat* pVerRes = cvCreateMat(2, 1, CV_32F);
120   pVer->data.fl[2] = 1.0;
121   for (int it = 0; it < maxIter; ++it)
122   {
123      // build mss
124      int pMss[3];
```

```
125       pMss[0] = rand() % corr.size();
126       do
127       {
128          pMss[1] = rand() % corr.size();
129       }
130       while (pMss[1] == pMss[0]);
131       do
132       {
133          pMss[2] = rand() % corr.size();
134       }
135       while (pMss[2] == pMss[0] || pMss[2] == pMss[1]);
136       CvPoint2D32f pSrc[3] = {corr[pMss[0]].first, corr[pMss[1]].first, corr[pMss[2]]. ↙
      first};
137       CvPoint2D32f pDst[3] = {corr[pMss[0]].second, corr[pMss[1]].second, corr[pMss[2]]↙
      .second};
138
139       cvGetAffineTransform(pSrc, pDst, pTest);
140
141       // test the mss
142       unsigned int fitCount = 0;
143       for (unsigned int j = 0; j < corr.size(); ++j)
144       {
145          if (j == pMss[0] || j == pMss[1] || j == pMss[2])
146          {
147             continue;
148          }
149          pVer->data.fl[0] = corr[j].first.x;
150          pVer->data.fl[1] = corr[j].first.y;
151          cvGEMM(pTest, pVer, 1.0, NULL, 0.0, pVerRes);
152          float tmpX = pVerRes->data.fl[0] - corr[j].second.x;
153          float tmpY = pVerRes->data.fl[1] - corr[j].second.y;
154          float mse = (tmpX * tmpX + tmpY * tmpY) / 2.0f;
155          if (mse < thresh)
156          {
157             // fits the affine transform
158             fitCount++;
159          }
160       }
161       if (fitCount >= numNeeded && fitCount > bestCount)
162       {
163          if (pBest == NULL)
164          {
165             pBest = pTest;
166             pTest = cvCreateMat(2, 3, CV_32F);
167          }
168          else
169          {
170             std::swap(pBest, pTest);
171          }
172          bestCount = fitCount;
173       }
174    }
175    cvReleaseMat(&pTest);
176    cvReleaseMat(&pVer);
177    cvReleaseMat(&pVerRes);
178    return pBest;
179 }
180 }
181
182 IplImageResource::IplImageResource() : mpImage(NULL), mShallow(false)
183 {
184 }
185
186 IplImageResource::IplImageResource(IplImage* pImage) : mpImage(pImage), mShallow(false)
187 {
188 }
```

```
189
190 IplImageResource::IplImageResource(IplImageResource& other)
191 {
192    reset(other.get());
193    if (other.isShallow())
194    {
195       release();
196    }
197    else
198    {
199       take();
200    }
201    other.mpImage = NULL;
202 }
203
204 IplImageResource::IplImageResource(int width, int height, int depth, int channels) :
       mShallow(false)
205 {
206    mpImage = cvCreateImage(cvSize(width, height), depth, channels);
207 }
208
209 IplImageResource::IplImageResource(int width, int height, int depth, int channels, char
       * pData) : mShallow(true)
210 {
211    mpImage = cvCreateImageHeader(cvSize(width, height), depth, channels);
212    mpImage->imageData = pData;
213 }
214
215 IplImageResource::~IplImageResource()
216 {
217    reset(NULL);
218 }
219
220 bool IplImageResource::isShallow()
221 {
222    return mShallow;
223 }
224
225 void IplImageResource::reset(IplImage* pImage)
226 {
227    if (mpImage != NULL)
228    {
229       if (mShallow)
230       {
231          mpImage->imageData = NULL;
232       }
233       cvReleaseImage(&mpImage);
234    }
235    mpImage = pImage;
236    mShallow = false;
237 }
238
239 IplImage* IplImageResource::get()
240 {
241    return mpImage;
242 }
243
244 IplImage* IplImageResource::release()
245 {
246    mShallow = true;
247    return mpImage;
248 }
249
250 IplImage* IplImageResource::take()
251 {
252    mShallow = false;
```

```
253    return mpImage;
254 }
255
256 IplImageResource::operator IplImage*()
257 {
258    return get();
259 }
260
261 IplImage& IplImageResource::operator*()
262 {
263    return *get();
264 }
265
266 IplImageResource& IplImageResource::operator=(IplImageResource& other)
267 {
268    mShallow = other.mShallow;
269    mpImage = other.mpImage;
270    other.mShallow = true;
271    other.mpImage = NULL;
272    return *this;
273 }
274
275 dataptr::dataptr(DataElement* pElement, DataPointerArgs args)
276 {
277    int own(0);
278    mpData = createDataPointer(pElement, &args, &own);
279    mOwns = (own != 0);
280 }
281
282 dataptr::~dataptr()
283 {
284    if (mOwns)
285    {
286       destroyDataPointer(mpData);
287    }
288 }
289
290 dataptr::operator void*()
291 {
292    return mpData;
293 }
294
295 dataptr::operator char*()
296 {
297    return reinterpret_cast<char*>(mpData);
298 }
299
300 void* dataptr::get()
301 {
302    return mpData;
303 }
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #ifndef TRACKINGUTILS_H__
11 #define TRACKINGUTILS_H__
12
13 #include "AppConfig.h"
14 #include "LocationType.h"
15 #include "RasterData.h"
16
17 #include <opencv/cv.h>
18 #if defined(WIN_API)
19 #include <boost/cstdint.hpp>
20 using boost::uint8_t;
21 #endif
22
23 namespace TrackingUtils
24 {
25 typedef uint32_t subcubeid_t;
26
27 /**
28  * Calculate a sub-cube ID for a given point.
29  *
30  * A sub-cube ID uses a quadtree to encode a sub-cube of an image.
31  * It is a 32-bit unsigned value. The least significant 4 bits encode the number of
32  * the remaining bit pairs which are significat. The remaining bits encode the child
33  * ID at each level of a quad tree, least significat to most. The quadrants are 0-3 ↙
       where
34  * 0 is the upper right quadrant (smallest y, largest x) and the remaining follow anti-↙
       clockwise.
35  * The id 0x00000000 indicates the entire image. The id 0x00000902 (0b10010010) ↙
       indicates
36  * there are 2 significant bit pairs. Those 4 bits point to the 1 quadrant (upper-left)↙
        and the
37  * 2 sub-quadrant (lower-left). The id 0xffffffff is an invalid ID and can be used for ↙
       initialization
38  * of variables, default error values, etc.
39  *
40  * @param location
41  *        The point to encode.
42  * @param levels
43  *        The number of levels deep to calculate.
44  * @param maxBb
45  *        The largest x, y values (inclusive) of the bounding box used for calculations↙
       .
46  * @param minBb
47  *        The smallest x, y values (inclusive) of the bounding box used for ↙
       calculations.
48  * @return The sub-cube ID.
49  */
50 subcubeid_t calculateSubcubeId(LocationType location, uint8_t levels, Opticks:: ↙
       PixelLocation maxBb,
51                                Opticks::PixelLocation minBb = Opticks::PixelLocation(0,↙
        0));
52
53 /**
54  * Calculate the extents in the sub-cube for a sub-cube ID.
55  *
56  * @param subcubeId
57  *        The ID.
```

```
 58  * @param levels
 59  *        Output param containing the number of levels deep.
 60  * @param maxBb
 61  *        Input/output parameter.
 62  *        Initially The largest x, y values (inclusive) of the bounding box used for calculations.
 63  *        On return, contains the largest x, y values of the specified sub-cube.
 64  * @param minBb
 65  *        Input/output parameter.
 66  *        Initially The smallest x, y values (inclusive) of the bounding box used for calculations.
 67  *        On return, contains the smallest x, y values of the specified sub-cube.
 68  * @return True on success, false otherwise. If false, the output parameters are undefined. False usually
 69  *         indicates an invalid subcubeId
 70  */
 71 bool calculateSubcubeBounds(subcubeid_t subcubeId, uint8_t& levels, Opticks::PixelLocation& maxBb, Opticks::PixelLocation& minBb);
 72
 73 /**
 74  * Calculate the quadtree level needed to get a sub-cube of at most the specified size.
 75  *
 76  * @param maxSubcubeSize
 77  *        Maximum size of the subcube in pixels^2.
 78  * @param maxBb
 79  *        The largest x, y values (inclusive) of the bounding box used for calculations.
 80  * @param minBb
 81  *        The smallest x, y values (inclusive) of the bounding box used for calculations.
 82  * @return The level needed. If it can't be represented, 0xff will be returned.
 83  */
 84 uint8_t calculateNeededLevels(uint32_t maxSubcubeSize, Opticks::PixelLocation maxBb, Opticks::PixelLocation minBb = Opticks::PixelLocation(0, 0));
 85
 86 /**
 87  * Determines the affine transform between a set of correspondances using RANSAC.
 88  */
 89 CvMat* ransac_affine(std::vector<std::pair<CvPoint2D32f, CvPoint2D32f> >& corr, int maxIter, float thresh, unsigned int numNeeded);
 90 }
 91
 92 /**
 93  * C allocation memory resource.
 94  *
 95  * Functions like an std::auto_ptr but for memory allocated with malloc() and friends.
 96  */
 97 template<typename T>
 98 class ca_ptr
 99 {
100 public:
101    ca_ptr() : mpData(NULL) {}
102
103    ca_ptr(T* pData) : mpData(pData) {}
104
105    ~ca_ptr()
106    {
107       if (mpData != NULL)
108       {
109          free(mpData);
110       }
111    }
112
113    T* get()
114    {
115       return mpData;
```

```
116    }
117
118    T* release()
119    {
120       T* pData = mpData;
121       mpData = NULL;
122       return pData;
123    }
124
125    void reset(T* pData)
126    {
127       if (mpData != NULL)
128       {
129          free(mpData);
130       }
131       mpData = pData;
132    }
133
134    operator T*()
135    {
136       return get();
137    }
138
139    T** operator&()
140    {
141       return &mpData;
142    }
143
144 private:
145    T* mpData;
146 };
147
148 /**
149  * Resource to manage an OpenCV IplImage.
150  *
151  * Can wrap existing memory (shared) or alloc its own memory.
152  */
153 class IplImageResource
154 {
155 public:
156    IplImageResource();
157    IplImageResource(IplImage* pImage);
158    IplImageResource(IplImageResource& other);
159    IplImageResource(int width, int height, int depth, int channels);
160    IplImageResource(int width, int height, int depth, int channels, char* pData);
161    ~IplImageResource();
162    bool isShallow();
163    void reset(IplImage* pImage);
164    IplImage* get();
165    IplImage* release();
166    IplImage* take();
167    operator IplImage*();
168    IplImage& operator*();
169    IplImageResource& operator=(IplImageResource& other);
170
171 private:
172    IplImage* mpImage;
173    bool mShallow;
174 };
175
176 class dataptr
177 {
178 public:
179    dataptr(DataElement* pElement, DataPointerArgs args);
180    ~dataptr();
181    operator void*();
```

```
182    operator char*();
183    void* get();
184
185 private:
186    void* mpData;
187    bool mOwns;
188 };
189
190 #endif
191
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #include "AnimationController.h"
11 #include "AnimationServices.h"
12 #include "AnimationToolBar.h"
13 #include "AttachmentPtr.h"
14 #include "DesktopServices.h"
15 #include "ModelServices.h"
16 #include "LabeledSection.h"
17 #include "LayerList.h"
18 #include "MouseMode.h"
19 #include "OpticalFlowWidget.h"
20 #include "PlugInManagerServices.h"
21 #include "RasterLayer.h"
22 #include "SessionManager.h"
23 #include "Slot.h"
24 #include "SpatialDataView.h"
25 #include "SpatialDataWindow.h"
26 #include "TrackingManager.h"
27 #include <QtGui/QAction>
28 #include <QtGui/QCheckBox>
29 #include <QtGui/QCursor>
30 #include <QtGui/QHBoxLayout>
31 #include <QtGui/QMouseEvent>
32 #include <QtGui/QPushButton>
33 #include <QtGui/QSpinBox>
34 #include <vector>
35
36 OpticalFlowWidget::OpticalFlowWidget(QWidget* pParent) :
37       LabeledSectionGroup(pParent),
38       mpToolbar(SIGNAL_NAME(AnimationToolBar, ControllerChanged), Slot(this, &
       OpticalFlowWidget::updateAnimation)),
39       mpActiveMode(NULL)
40 {
41    mpToolbar.reset(static_cast<AnimationToolBar*>(Service<DesktopServices>()->getWindow
       ("Animation", TOOLBAR)));
42    QWidget* pSelectWidget = new QWidget(this);
43    QHBoxLayout* pSelectLayout = new QHBoxLayout(pSelectWidget);
44
45    mpMaxSize = new QSpinBox(pSelectWidget);
46    mpMaxSize->setRange(16, 8192);
47    mpMaxSize->setSingleStep(16);
48    mpMaxSize->setValue(1024);
49    mpMaxSize->setSuffix(" pel^2");
50    pSelectLayout->addWidget(mpMaxSize);
51    QPushButton* pSelectButton = new QPushButton("Select Area", pSelectWidget);
52    pSelectLayout->addWidget(pSelectButton);
53    VERIFYNR(connect(pSelectButton, SIGNAL(clicked()), this, SLOT(activateSelectMode
       ())));
54
55    QCheckBox* pPause = new QCheckBox("Pause", pSelectWidget);
56    pSelectLayout->addWidget(pPause);
57    VERIFYNR(connect(pPause, SIGNAL(toggled(bool)), this, SLOT(updatePause(bool))));
58
59    pSelectLayout->addStretch(10);
60    LabeledSection* pDataSelectSection = new LabeledSection("Data Set", this);
61    pDataSelectSection->setSectionWidget(pSelectWidget);
62    addSection(pDataSelectSection);
63
```

```
64    addStretch(100);
65
66    mpSelectMode = Service<DesktopServices>()->createMouseMode("OpticalFlowCenterSelect"↙
       , Qt::CrossCursor);
67 }
68
69 OpticalFlowWidget::~OpticalFlowWidget()
70 {
71 }
72
73 void OpticalFlowWidget::updateAnimation(Subject& subject, const std::string& signal,   ↙
       const boost::any& val)
74 {
75    AnimationController* pController = boost::any_cast<AnimationController*>(val);
76    if (pController == NULL || pController->getAnimations().empty())
77    {
78       return;
79    }
80    Animation* pAnimation = pController->getAnimations().front();
81    std::vector<Window*> windows;
82    Service<DesktopServices>()->getWindows(windows);
83    for (std::vector<Window*>::iterator window = windows.begin(); window != windows.end ↙
       (); ++window)
84    {
85       SpatialDataWindow* pWindow = dynamic_cast<SpatialDataWindow*>(*window);
86       SpatialDataView* pView = pWindow == NULL ? NULL : pWindow->getSpatialDataView();
87       if (pView != NULL)
88       {
89          std::vector<Layer*> layers;
90          pView->getLayerList()->getLayers(RASTER, layers);
91          for (std::vector<Layer*>::iterator layer = layers.begin(); layer != layers.end↙
       (); ++layer)
92          {
93             RasterLayer* pLayer = static_cast<RasterLayer*>(*layer);
94             if (pLayer != NULL && pLayer->getAnimation() == pAnimation)
95             {
96                std::vector<PlugIn*> manager =
97                   Service<PlugInManagerServices>()->getPlugInInstances(TrackingManager:↙
       :spPlugInName);
98                if (!manager.empty())
99                {
100                  static_cast<TrackingManager*>(manager.front())->setTrackedLayer      ↙
       (pLayer);
101               }
102               return;
103            }
104         }
105      }
106   }
107 }
108
109 void OpticalFlowWidget::updatePause(bool state)
110 {
111   std::vector<PlugIn*> manager = Service<PlugInManagerServices>()->getPlugInInstances ↙
       (TrackingManager::spPlugInName);
112   if (!manager.empty())
113   {
114      static_cast<TrackingManager*>(manager.front())->setPauseState(state);
115   }
116 }
117
118 void OpticalFlowWidget::activateSelectMode()
119 {
120   View* pView = Service<DesktopServices>()->getCurrentWorkspaceWindowView();
121   if (pView != NULL)
122   {
```

```
123       mpActiveMode = pView->getCurrentMouseMode();
124       if (mpActiveMode == mpSelectMode)
125       {
126          mpActiveMode = NULL;
127       }
128       pView->addMouseMode(mpSelectMode);
129       pView->setMouseMode(mpSelectMode);
130       pView->getWidget()->installEventFilter(this);
131    }
132 }
133
134 bool OpticalFlowWidget::eventFilter(QObject* pObj, QEvent* pEvent)
135 {
136    if (pEvent != NULL && pEvent->type() == QEvent::MouseButtonRelease)
137    {
138       View* pView = Service<DesktopServices>()->getCurrentWorkspaceWindowView();
139       if (pView != NULL)
140       {
141          QPointF loc = static_cast<QMouseEvent*>(pEvent)->posF();
142          loc.setY(pView->getWidget()->height() - loc.y());
143          std::vector<PlugIn*> manager = Service<PlugInManagerServices>()->
      getPlugInInstances(TrackingManager::spPlugInName);
144          if (!manager.empty())
145          {
146             int maxSz = mpMaxSize->value();
147             static_cast<TrackingManager*>(manager.front())->setFocus(LocationType(loc.x
      (), loc.y()), maxSz*maxSz);
148          }
149
150          pView->setMouseMode(mpActiveMode);
151          mpActiveMode = NULL;
152          pView->getWidget()->removeEventFilter(this);
153          return true;
154       }
155    }
156    return false;
157 }
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #ifndef OPTICALFLOWWIDGET_H__
11 #define OPTICALFLOWWIDGET_H__
12
13 #include "AnimationToolBar.h"
14 #include "AttachmentPtr.h"
15 #include "LabeledSectionGroup.h"
16 #include <boost/any.hpp>
17
18 class MouseMode;
19 class QComboBox;
20 class QSpinBox;
21
22 class OpticalFlowWidget : public LabeledSectionGroup
23 {
24    Q_OBJECT
25
26 public:
27    OpticalFlowWidget(QWidget* pParent = NULL);
28    virtual ~OpticalFlowWidget();
29
30 protected slots:
31    void updatePause(bool state);
32    void activateSelectMode();
33
34 protected:
35    bool eventFilter(QObject* pObj, QEvent* pEvent);
36
37 private:
38    void updateAnimation(Subject& subject, const std::string& signal, const boost::any&
       val);
39
40    AttachmentPtr<AnimationToolBar> mpToolbar;
41    MouseMode* mpSelectMode;
42    MouseMode* mpActiveMode;
43    QSpinBox* mpMaxSize;
44 };
45
46 #endif
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #include "AppVerify.h"
11 #include "DesktopServices.h"
12 #include "MenuBar.h"
13 #include "OpticalFlow.h"
14 #include "OpticalFlowWidget.h"
15 #include "PlugInRegistration.h"
16 #include "ToolBar.h"
17
18 REGISTER_PLUGIN_BASIC(Tracking, OpticalFlow);
19
20 OpticalFlow::OpticalFlow()
21 {
22    setName("OpticalFlow");
23    setDescription("Calculate optical flow field.");
24    setDescriptorId("{38c907e0-1a57-11df-8a39-0800200c9a66}");
25    setSubtype("Video");
26 }
27
28 OpticalFlow::~OpticalFlow()
29 {
30 }
31
32 QAction* OpticalFlow::createAction()
33 {
34    // Add a menu command to invoke the window
35    MenuBar* pMenuBar = Service<DesktopServices>()->getMainMenuBar();
36    VERIFYRV(pMenuBar, NULL);
37    QAction* pAction = pMenuBar->addCommand("&View/&Optical Flow");
38
39    ToolBar* pToolBar = static_cast<ToolBar*>(Service<DesktopServices>()->getWindow(
       "Tracking", TOOLBAR));
40    if (pToolBar == NULL)
41    {
42       pToolBar = static_cast<ToolBar*>(Service<DesktopServices>()->createWindow(
       "Tracking", TOOLBAR));
43    }
44    VERIFYRV(pToolBar, NULL);
45    pToolBar->addButton(pAction);
46
47    return pAction;
48 }
49
50 QWidget* OpticalFlow::createWidget()
51 {
52    return new OpticalFlowWidget();
53 }
```

```
1  /*
2   * The information in this file is
3   * Copyright(c) 2010 Trevor R.H. Clarke
4   * and is subject to the terms and conditions of the
5   * GNU Lesser General Public License Version 2.1
6   * The license text is available from
7   * http://www.gnu.org/licenses/lgpl.html
8   */
9
10 #ifndef OPTICALFLOW_H__
11 #define OPTICALFLOW_H__
12
13 #include "DockWindowShell.h"
14
15 class OpticalFlow : public DockWindowShell
16 {
17 public:
18    OpticalFlow();
19    virtual ~OpticalFlow();
20
21 protected:
22    virtual QAction* createAction();
23    virtual QWidget* createWidget();
24
25 };
26
27 #endif
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #include "PlugInRegistration.h"
11
12 REGISTER_MODULE(Tracking);
```

```
 1 /*
 2  * The information in this file is
 3  * Copyright(c) 2010 Trevor R.H. Clarke
 4  * and is subject to the terms and conditions of the
 5  * GNU Lesser General Public License Version 2.1
 6  * The license text is available from
 7  * http://www.gnu.org/licenses/lgpl.html
 8  */
 9
10 #ifndef TRACKINGVERSION_H
11 #define TRACKINGVERSION_H
12
13 #define TRACKING_NAME "Tracking"
14 #define TRACKING_NAME_LONG "Object trackgin"
15 #define TRACKING_COPYRIGHT "Copyright © 2010, Trevor R.H. Clarke"
16 #define TRACKING_VERSION_NUMBER "1.0.0Unofficial"
17 #define TRACKING_IS_PRODUCTION_RELEASE false
18
19 #endif
20
```

```
 1 <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
 2 <ConfigurationSettings xmlns="https://comet.balldayton.com/standards/namespaces/2005/v1/↙
       comet.xsd">
 3
 4   <opticks build_revision="9296" release_date="09 June 2009" version="4.3.3rc1"/>
 5
 6   <group name="settings" version="3">
 7     <attribute name="TrackingManager" type="DynamicObject" version="3">
 8       <attribute name="InitialSubcubeSize" type="unsigned int">
 9           <value>262144</value> <!-- 512x512 -->
10       </attribute>
11     </attribute>
12   </group>
13 </ConfigurationSettings>
14
```