The Pennsylvania State University

The Graduate School

Department of Electrical Engineering

EXTENDING A BIOLOGICAL CELL TRACKING SYSTEM TO TRACK RANGE DATA OF PEDESTRIANS

A Thesis in

Electrical Engineering

by

Ryan James Poore

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2017

The thesis of Ryan James Poore was reviewed and approved* by the following:

William E. Higgins
Distinguished Professor of Electrical Engineering

Richard L. Tutwiler
Senior Scientist, Professor of Acoustics
Applied Research Laboratory Department Head Imaging Systems and Processing
Thesis Advisor

Kultegin Aydin
Professor of Electrical Engineering
Head of the Department of Electrical Engineering

*Signatures are on file in the Graduate School

ABSTRACT

The challenges faced when constructing a system to track biological cells coincides with the challenges when tracking pedestrians: they have complex topological shapes; wide range of behaviors; they move abruptly; they interact with other objects; their shape deforms; they leave the field of view (FOV) and enter the FOV; they split and merge. The similarities suggest that a system for tracking cells potentially works well for pedestrians.

This work presents an automated tracking system that extends a framework, designed for tracking hundreds of biological cells in phase contrast microscopy, to tracking multiple human sized objects with LIDAR range data. It integrates various classic image-processing techniques with an adaptive interacting multiple model (IMM) estimator, a topologically constrained active contour, and spatiotemporal trajectory optimization. The framework processes the data with multiple independent collaborating modules. This module design facilitates a straight forward substitution of algorithms within a module without affecting the rest of the system.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere appreciation and gratitude to my advisor, Dr. Richard Tutwiler, for his expertise and guidance. I thank my committee Chair, Dr. William Higgins, for stepping into that role and taking on the responsibility. I thank SherryDawn Jackson for dealing with all my last-minute atypical request. I thank Donald Natale, Matthew Baran, and John Durkin for collecting the LIDAR data.

Chapter 1

Introduction

1.1 Motivation

The goal of this work is to autonomously track multiple objects in a real-life scene as they interact with their surroundings. The approach presented in this work is to take an existing system designed for tracking hundreds of biological cells, and implement it for tracking pedestrians. The framework and algorithms presented in [1] are adapted for this work because of the module architecture and the challenges the author addressed. The initial detection algorithm is modified due to the different type of data processed: range data of people in an outdoor environment vs. cells in phase contrast microscopy. Cells in phase contrast microscopy normally appear as dark regions surrounded by bright halo artifacts, while the data analyzed in this paper lacks that same distinct contrast. Furthermore, the data collected for this work has a more complex background (see figure 1). However, the rest of the system is replicated without significant modification. The architecture is modular, in that each module processes the data independently and they collaborate to produce the trajectories. This module architecture allows for the seamless adjustment in processing a particular aspect without changing the entire system. For example, changing the method that process the initial segmentation without effecting the components that evolve the contours or manage the lost and occluded objects.

1.2 Object Tracking System

The objective of an object tracker is to generate the trajectory of an object over time by locating its position, and often its complete region, in every frame. The task of determining these trajectories can be separated into two approaches: (1) detecting the objects in each frame and then establishing correspondence between frames or (2) evolve a model that represents the object from frame to frame [12]. The object tracker in this work deploys a combination of both approaches. Shape, region, and edge information is utilized for a coarse segmentation while a geometric active contour model is implemented to evolve the objects across frames.

The system described in this paper divides the tasks associated with creating trajectories of multiple objects into 5 components: (1) Detection Module, (2) Contour Evolution Module, (3) Motion Filter, (4) Track Compiler, (5) Track Linker. The details of each are explained in chapter 3. The detection module uses shape, region, and edge information to detect and provide a rough segment of the objects. The contour evolution module implements a geometric active contour model that incorporates topological constraints to govern under what circumstances an object can split or merge. An IMM filter is utilized to enhance the tracking, and a track linking module is implemented to assist with complete and long term occlusion. The track linker is developed to consider the objects' location in space and time, along with their dynamics, when attempting to link lost objects or occluded objects.

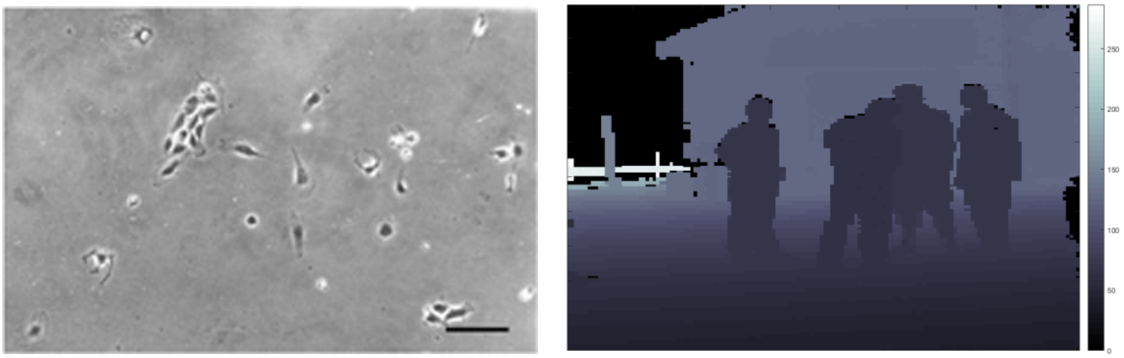


Figure 1-1: (a) Phase-contrast microscopy of putative RCE cells taken from NIH Open-i project. (b) A sample range map from the data collect for this work.

1.3 Thesis Outline

Chapter 2 of this paper provides a brief overview of the sensor equipment along with the specifications of the collected data. Chapter 3 delineates the system implemented for the tracking model. The first section of Chapter 3 discusses the initial segmentation and detection. The second section goes through the object contour and propagator module. It involves a discussion on geometric active contours, contour evolution, topological constraints, and the energy/speed equation. The third section reviews the motion filter along with the interacting multiple models (IMM) and Kalman filter, which are used for state predictions and in the motion component of the energy term. The fourth section discusses the object track compilation and the fifth section reviews the object track linker module. Chapter 4 presents the results. Chapter 5 provides the conclusion.

Chapter 2

3D Flash LIDAR/VNIR

2.1 LIDAR Hardware

The data utilized in this work was collected from a 3D Flash LIDAR Video Camera built by Advanced Scientific Concepts Inc. Figure 1 below displays the LIDAR sensor used to collect the data. The model name is the modular Portable 3D Flash LIDAR Camera integrated with1.57um, air-cooled, diode-pumped laser, and a 9-degree field of view. The camera can image through obscuration (e.g. dust, smoke, fog) and has an effective capture range from 5cm to 1km [11]. The short capture time (typically ½ a microsecond or less) emphasizes the camera's ability to capture 3D images without motion distortion. Table 1 displays the instrumentation along with it specifications.

Advanced Scientific Concepts Inc. Portable 3D Flash LIDAR Camera	9º field of view lens and diffuser LD-2809-PM: 9º field of view; 128 x 128 pixel resolution 85mm EFFL (Field of View 8.6º x 8.6º), F/1.4 (60mm aperture) @ 1.57 um Range ~ 300 meters @ 20% reflectance 2D IR Camera, co-aligned with Portable 3D FLVC (Part number: 2D-C015-IR) 2D visible Camera, co-aligned with Portable 3D FLVC (Part number 2D-C00M-VS) http://www.advancedscientificconcepts.com/products/older-products/portable.html

Table 2-1: Advanced Scientific Concepts Inc. Portable 3D Camera Specifications

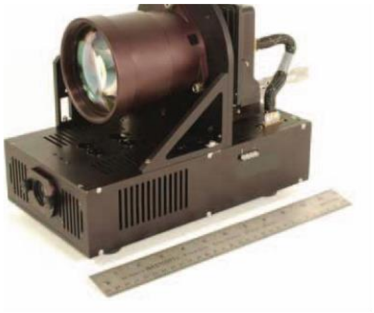


Figure 2-1: The LIDAR/VNIR sensor (Picture taken from http://www.advancedscientificconcepts.com).

2.2 LIDAR/ VNIR Data

The ASC 3D Flash LIDAR Camera generates a range image, and a reflectivity intensity image at 128 x128 pixels. Figure 3 displays an example of the output from the LIDAR (resampled at 600 x 800) for the (a) range data and (b) the intensity data. There is also an optional VNIR component that generates a RGB image at 640x480 pixels. Figure 2 (C) displays the output of the VNIR data (resampled to 600 x 800). The intensity and VNIR sensor output is displayed in Figure 2 for completeness but only the range is processed for this work.

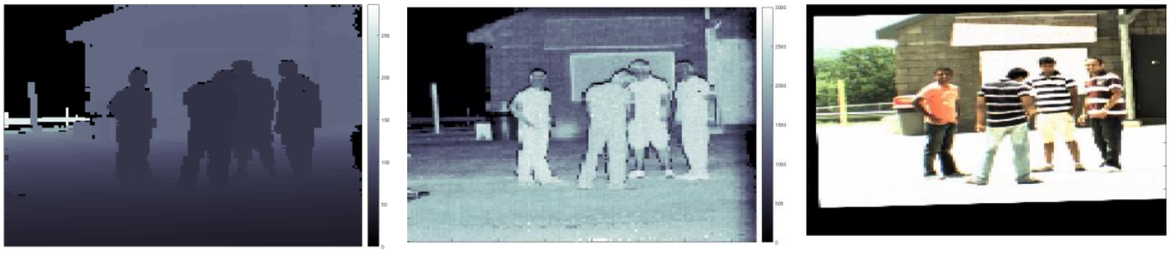


Figure 2-2 : LIDAR sensor output of Rick Tutwiler's graduate students, resampled to 600 x 800 (a - left) Range, 128 x128 pixels (b - right) Intensity, 128 x 128 (c) VNIR sensor output resampled to 600 x 800: RGB, 640x480x3 pixels.

Chapter 3

Object Tracking System

The system processes data sequentially and filters through the different modules as display in figure 3-1. The final output is recorded to a Elasticsearch database and a flat file. The output consists of the trajectories for each object, along with their states and parameters (the details are presented in 3.5).

Data is first processed by the Detection Module utilizing various classic image processing techniques. The output of the module is a map of segmented regions (χ) that represents its region energy. This map of potential object regions is distributed to the Contour Evolution Module, as well as, the Track Compiler module.

The Contour Evolution module generates an initial coarse map of the labeled object regions ($\widehat{\psi}(x, y)$). The labels represent their corresponding track assignments, so the map is essentially the initial attempt at propagating the objects tracked in the last frame to the current frame. The module performs this task by using the active contour model framework, this is further explored in section 3.2. Topological constraints are also incorporated into the framework; the constraints are formulated to allow the contours to split while preventing the contours of different objects from merging. The speed function that is used to drive the evolution of the contour incorporates components associated with the objects' region features, edges, and dynamics. The region component is based on the object region map ($\widehat{\psi}(x, y)$) passed by the Detection Module. The Motion filter provides the past dynamics while the edge calculation is performed given the original input data.

The Track Compiler Module receives the initial coarse map of the labeled object regions ($\widehat{\psi}(x, y)$) from the Contour Evolution Module, along with the initial object region map (χ) from the Detection Module. It then determines how to classify each object track: update an existing track, terminate a track, create a new track, or split a track into child tracks. The result is an updated map of the labeled object regions ($\widehat{\psi}(x, y)$) along with the intermediate track segments (also called tractlets) for the current frame (discussed further in 3.3).

The Track Linker Module attempts to link the tracks that were split in the past by analyzing the objects in the spatiotemporal image volume along with their past dynamics. Any successful merging of tracks is fed back to the track compiler for reference in the next frame.

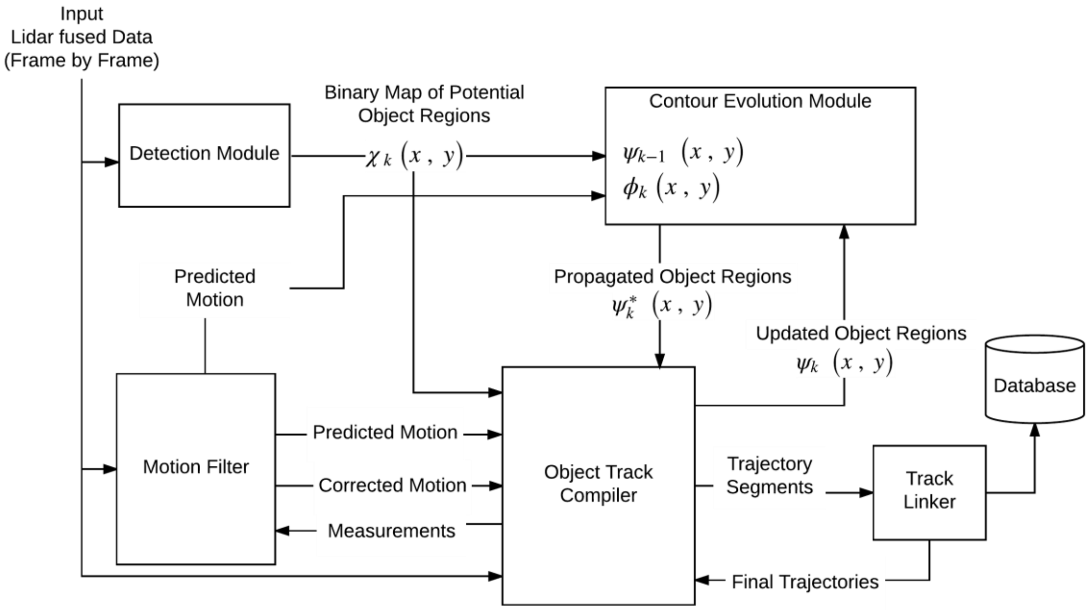


Figure 3-1: System overview

3.1 Object Detection Module Overview

The system starts with the detection and segmentation of object regions from the input data. The image is initially filtered to remove the image plane using a Random Sample Consensus (RANSAC) algorithm, as displayed in figure 3-2. Threshold filtering is implemented to eliminate object regions that possess properties inconsistent with the physical constraints for a given target (such as objects size, intensity, or range). The segmentation is enhanced by utilizing a Bayesian maximum a posteriori probability (MAP) classifier; followed by a coarse shape filter to match human sized and shaped objects, along with a range filter to filter out objects that are "too close" or "too far." The output is a binary map of segmented regions (χ_r).

In parallel to the above processes, the original input is sent through a canny edge detector to produce an edge map. The edge map is further processed to fill in enclosed regions and then filtered based on the mean and standard deviation of neighboring pixels. The output of these two processes are combined with a binary "or" operation to produce the initial segmentation of potential object regions (χ).

3.1.1 RANSAC

The Random Sample Consensus (RANSAC) algorithm was first published by Fischler and Bolles at SRI International in 1981 [14]. In the RANSAC algorithm, an assumption is made that the data contains inliers whose distribution can be explained by a mathematical model, along with noise and outliers whose distribution does not fit the model. The algorithm separates outliers of the observation data from the inliers. To implement the RANSAC algorithm, the following two steps are repeated in an iterative fashion [23]:

(1) Hypothesize: Select a random sample set from the input and use that set to compute the model parameters

(2) Test: test which datum are consistent with the model estimated in the previous step. A data element will be considered as an outlier if it does not fit the fitting model instantiated by the set of estimated model parameters within some error threshold that defines the maximum deviation attributable to the effect of noise.

The set of inliers obtained for the fitting model is called consensus set. The RANSAC algorithm will iteratively repeat the above two steps until the obtained consensus set in certain iteration has enough inliers.

The Detection Module implements the RANSAC method with least squares orthogonal distance minimization to find the ground plane. Orthogonal minimization is selected since it is assumed that there is error in all three dimensions of our irregularly spaced data [13]. Principal component analysis is used to further refine the ground plane estimate. Once the estimation for the ground plane is finished, the pixels associated with the ground plane are removed. The objects are in the outliers of the data along with some noise. The process of removing the plane will result in the loss of some object pixels. This will manifest as holes in the objects. The holes are closed with a simple combination of dilation and erosion. Figure 3-2 displays an example from the collected data set.

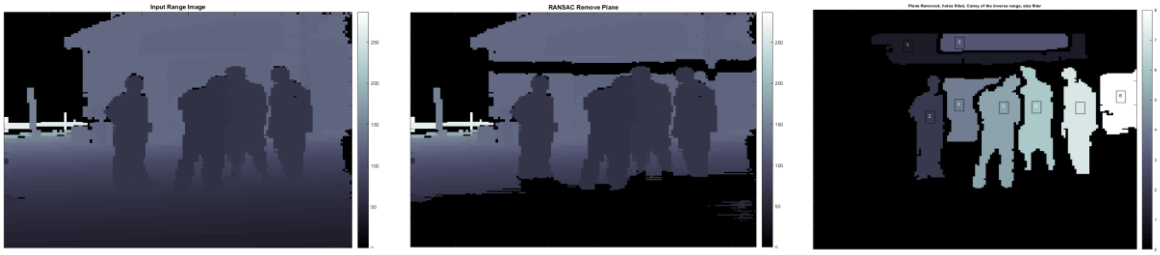

Input Range Image
RANSAC Remove Plane
Plane Removed, holes filled, Canny of the inverse range, size filter
1
2
3
4
5
6
7
8

Figure 3-2: Detection Module processing: (a) The range image of a frame. (b) The Range Image with the ground plane removed (c) The output of the Detection Module, denoted as the propagated region labeling map,($\hat{\psi}(x, y)$).

3.2 Contour Evolution Module

This module is responsible for propagating any object associated with an identification in the last frame, to the current frame. Since the goal is to simultaneously track multiple objects (such as multiple people or groups of people in a scene), a model that allows for significant deformation, while handling separation and merging is desired. To achieve this condition, geometric active contours are utilized.

3.2.1 Geometric Active Contours

Active contours are generally classified as either parameterized active contours (explicit contours) or geometric active contours (implicit contours) based on their representation and implementation. They are often used in image processing for edge detection, shape modeling, segmentation, and motion tracking. Parametric active contours were introduced by Kass et al. in 1988 [24]. They defined active contours as energy-minimizing splines guided by external constraint forces that pull them toward features such as lines and edges. An internal energy term was also defined to impose a smoothness constraint on the moving curve. The primary issue with parameterized contours is their lack of topological handling.

Geometric active contours were independently introduced by Caselles et al. [4] and Malladi et al. [5]. They are based on the theory of curve evolution and the level set method [15]. In this method, a curve evolves using only geometric measures, resulting in a contour evolution that is independent of the curve’s parameterization. This avoids the need to repeatedly reparametrize the curve or to explicitly handle changes in topological [16].

3.2.2 Level Set Framework

The Level Set Method is a computational technique for tracking moving interfaces that depend on an implicit representation of the interface whose equation of motion is numerically approximated using schemes built from those for hyperbolic conservation laws [25]. It was introduced by Osher and Sethian [7] for computing the solution to fluid interface problems.

The classic level set method consists of three main components: an implicit data representation of a hypersurface, a set of PDEs that govern how the surface moves, and the corresponding numerical methods for implementing [17].

Following the derivation presented by Sethian [7]. Given a closed N −1 dimensional hypersurface Γ(t), the goal is to evolve this hypersurface over time as it propagates along its normal direction with speed F.

In two-dimensions (although not limited to only two-dimensions), a closed contour can divide a three-dimensional space into two separate subdomains consisting of the inside regions, the outside regions, and the interface. These are denoted:

$$\begin{aligned} \Omega^{+} &= \text{region outside } \Gamma(t) \\ \partial\Omega &= \Gamma(t) \\ \Omega^{-} &= \text{region outside } \Gamma(t) \end{aligned} \qquad 3\text{-}1$$

Embed the hypersurface (Γ) as the zero level set of a higher dimensional function, $\phi(\bar{x}, t)$.

Let $\phi(\bar{x}, t = 0)$, for $\bar{x} \in \mathbb{R}^{N}$ be defined by

$$\phi(\bar{x}, t = 0) = \pm d \qquad 3\text{-}2$$

Where d is the distance from the position, $\bar{x}$, to Γ(t=0). The positive d is chosen if $\bar{x}$ is outside (Ω^{+}) and the negative d if the point is inside (Ω^{-}).

Therefore, $\phi(\bar{x}, t = 0)\colon \mathbb{R}^{n} \to \mathbb{R}$ such that

$$\Gamma(\mathrm{t}=0)=(\bar{\mathrm{x}}|\phi(\bar{\mathrm{x}},\mathrm{t}=0)=0). \qquad 3\text{-}3$$

Figure 3-3 depicts this relationship.

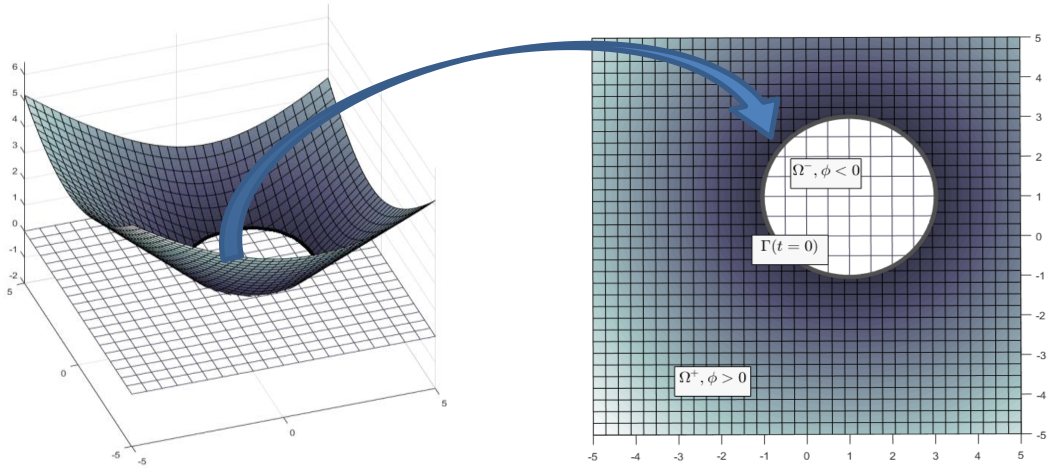


Figure 3-3: Level Set Method: (a) Level Set hypersurface (b) the 2D boundary

Consider a point on the propagating interface, and let x(t) be the path of the point on the propagating interface Γ(t = 0). For each $\bar{\mathrm{x}} \in \partial\Omega$, apply the constraint $\phi(\bar{\mathrm{x}},\mathrm{t}) = 0$:

$$\frac{\mathrm{d}\phi(\bar{\mathrm{x}},\mathrm{t})}{\mathrm{dt}}=0 \qquad 3\text{-}3$$

Apply the chain rule

$$\frac{\mathrm{d}\phi(\bar{\mathrm{x}},\mathrm{t})}{\mathrm{dt}}=\frac{\partial\phi}{\delta\mathrm{t}}+\nabla\phi(\bar{\mathrm{x}},\mathrm{t})\cdot\left(\frac{\mathrm{d}\bar{\mathrm{x}}}{\mathrm{dt}}\right) \qquad 3\text{-}4$$

$\left(\frac{\mathrm{d}\bar{\mathrm{x}}}{\mathrm{dt}}\right)$ in equation 3-4 is the velocity at the point ($\bar{\mathrm{x}}$) on the hypersurface. Set equation 3-4 to zero to achieve a Euler-Lagrange equation (representing the steady state of the contour):

$$\frac{\partial\phi}{\delta\mathrm{t}}+\nabla\phi(\bar{\mathrm{x}},\mathrm{t})\cdot\frac{\mathrm{d}\bar{\mathrm{x}}}{\mathrm{dt}}=0 \qquad 3\text{-}5$$

In equation 3-2, the initial surface (ϕ) was defined as the Euclidean distance from $\bar{x}$ to Γ:

$$\phi(\bar{x}, t = 0) = \pm d \quad \bar{x} \in \mathbb{R}^n \tag{3-6}$$

The distance from a point $\bar{x}$ to a set $\partial\Omega$:

$$d(\bar{x}) = \min_{\bar{x}_C \in \partial\Omega} |\bar{x} - \bar{x}_C| \tag{3-7}$$

The ϕ takes on the distance from the boundary with a sign depending on which region it resides:

$$\phi(\bar{x}) = \begin{cases} -d(\bar{x}) & \bar{x} \in \Omega^- \\ 0 & \bar{x} \in \partial\Omega \\ d(\bar{x}) & \bar{x} \in \Omega^+ \end{cases} \tag{3-8}$$

There are an infinite number of choices for the level set function, but in practice the signed distance function is preferred for its stability in numerical computations [15].

In the level set method, it is assumed the curve moves in the normal direction to itself. If the velocity $\left(\frac{d\bar{x}}{dt}\right)$ is constrained to the normal direction, then the speed, F, which corresponds to the movement of the contour in the normal direction with respect to time, is defined as:

$$F = \left(\frac{d\bar{x}}{dt}\right) \cdot \bar{n} \tag{3-9}$$

Where $\bar{n}$ is the unit normal and can be defined on the contour as

$$\bar{n} = \frac{\nabla\phi(\bar{x}, t)}{|\nabla\phi(\bar{x}, t)|} \tag{3-10}$$

Since $\nabla\phi(\bar{x}, t)$ is normal to the surface, its inner product with a vector tangential to the surface is zero and therefore $\frac{d\bar{x}}{dt}$ from equation 3-9 will be in the normal direction. Furthermore,

$\nabla\phi(\bar{x},t) \cdot \nabla\phi(\bar{x},t) = |\nabla\phi(\bar{x},t)|^2$ and $\frac{|\nabla\phi(\bar{x},t)|^2}{|\nabla\phi(\bar{x},t)|} = |\nabla\phi(\bar{x},t)|$. Combining equations 3-5, 3-9, and 3-10:

$$\frac{\partial\phi}{\delta t} + F\,|\nabla\phi(\bar{x},t)| = 0 \qquad 3\text{-}11$$

As indicated from equation 3-11 above, the important decision lies with the construction of the speed function. That is the component that guides the contour evolution.

3.2.3 Two-Cycle Algorithm

The classic implementation of a level set method requires evaluating partial differential equations using computational expensive numerical methods. To reduce the computational complexity, this work implements a two-cycle algorithm with smoothness regulation adapted from [1,2,17]. In addition to reducing the computation burden, the algorithm is relatively straightforward to implement and it allows for easy integration of topological constraints into the contour evolution. The method implemented by [18] gains additional reduction in computational cost by careful consideration of data types, for example, using integer valued arrays, performing integer only calculations, and only saving the sign of the calculated speed function.

A two-dimensional map is created to store the location of all the objects; the map is called a region labeling map ($\psi_k(x,y)$) and is defined by:

$$\psi_k(x,y) = \begin{Bmatrix} n, & \text{if pixel } (x,y) \text{ is part of the object n} \\ 0, & \text{if pixel } (x,y) \text{ is part of the background} \end{Bmatrix} \qquad 3\text{-}12$$

The two linked-list that describe the inside and outside of an object's boundary are labeled L_{in} and L_{out}, respectively. They are initialized according to the region labeling map ($\psi_k(x,y)$) and are defined as follows:

$$L_{out}(n) = \{\ (x,y)\ |\ \psi(x,y) = n \text{ and } \exists(x',y') \in\ N_4(x,y) \text{ where } \psi(x',y') \neq n\ \} \qquad 3\text{-}13$$

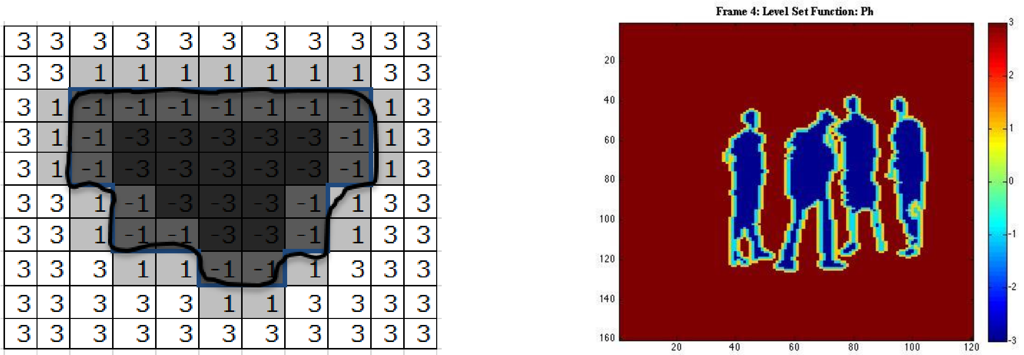
$$L_{in}(n) = \{\ (x,y)\ |\ \psi(x,y) = n \text{ and } \exists(x',y') \in\ N_4(x,y) \text{ where } (x',y')\ \in\ L_{out}(n)\ \} \qquad 3\text{-}14$$

Figure 3-4 shows how they fit around an objects contour.

In this implementation, one level set function is defined to represent all the objects' contours. The level set function ($\phi(x,y)$) is used in conjunction with the region labeling map ($\psi_k(x,y)$) to indicate what region of the function belongs to a given tracked object. It approximates the signed distance function and is defined as:

$$\phi(x,y) = \left\{ \begin{array}{rl} 3 & \text{if } (x,y) \text{ is an exterior pixel} \\ 1 & \text{if } (x,y) \text{ is } \in \text{outside boundary} \\ -1 & \text{if } (x,y) \text{ is } \in \text{inside boundary} \\ -3 & \text{if } (x,y) \text{ is an interoir pixel} \end{array} \right\} \qquad 3\text{-}15$$

The algorithm has two cycles: update cycle and a regularization cycle. The update cycle moves the contour while regularization cycle provides smoothing. The movement of the contour is realized by iteratively switching elements between the two linked-list that represent the adjacent sides of the contour. This can be viewed as an extreme case of the narrow band scheme with a two-pixel resolution [1,2,18]. The algorithm is described in the next four subsections and the pseudo code is presented in table 3-1 and table 3-2.



3	3	3	3	3	3	3	3	3	3	3
3	3	1	1	1	1	1	1	1	3	3
3	1	-1	-1	-1	-1	-1	-1	-1	1	3
3	1	-1	-3	-3	-3	-3	-3	-1	1	3
3	1	-1	-3	-3	-3	-3	-3	-1	1	3
3	3	1	-1	-3	-3	-3	-1	1	3	3
3	3	1	-1	-1	-3	-3	-1	1	3	3
3	3	3	1	1	-1	-1	1	3	3	3
3	3	3	3	3	1	1	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3

Figure 3-4: The level set function, $\phi_k(x,y)$ for (a) an arbitrary shape. (b) for the LIDAR/VNIR image data displayed in Figure 2-1 and Figure 3-1, the red represents the background or exterior pixels (value = 3), the yellow represents the outside boundary (value =1), the light blue represents the inside boundary (value =-1), and the dark blue represents all the object pixels or interior pixels (value =-3).

3.2.3.1 Initialization

The two arrays of linked-lists that represent the inside ($\mathrm{L_{in}(n)}$) and outside ($\mathrm{L_{out}(n)}$) of the contours are initialize based on the Region Labeling Map from the previous frame ($\psi_{k-1}(x,y)$). Equation 3-15 displays this assignment. The level set function ($\phi(x,y)$) is initialized based on the two arrays of linked list as described in equations 3-13 and 3-14. The pixels for the outside linked list are set to zero ($\mathrm{L_{out}}$ is not part of the object). The object identification numbers from last frame are combined with the current frame's temporary identification numbers to create a complete list of objects to process ($\mathrm{N_{k-1}} \cup \{n\}$). Table 3-1 shows the pseudo code, under the initialization section of the code.

3.2.3.2 Update Cycle

The update cycle is responsible for calculating the speed function and evolving the contours based on its sign (only the sign is needed for the determination). The speed values should be positive for all pixels in $\mathrm{L_{in}}$ and negative for all pixels in $\mathrm{L_{out}}$, thus any pixel in $\mathrm{L_{out}}$ with $\mathrm{F} > 0$

gets switched to $\mathrm{L_{in}}$and vice versa if $\mathrm{F} < 0$. The cycle basically scans through the two linked-list; first outward, then inward. During each evolution, redundant elements of $\mathrm{L_{in}}$and $\mathrm{L_{out}}$are removed. Table 3-1 shows the pseudo code for this algorithm, under the section marked Update Cycle.

3.2.3.3 Stopping Condition

To regulate the time and resources allocated to the contour evolution process, a stopping condition is enforced. The contour evolves until the boundary stops progressing or a maximum number of iterations is achieved. The following rules define it:

(a) The Speed, $\hat{\mathrm{F}}(\mathrm{x},\mathrm{y})$, at each neighboring pixels satisfies:

$$\hat{\mathrm{F}}(\mathrm{x},\mathrm{y}) \leq 0 \;\forall\; (\mathrm{x},\mathrm{y}) \in \mathrm{L_{out}} \;\text{ and }\; \hat{\mathrm{F}}(\mathrm{x},\mathrm{y}) \geq 0 \;\forall\; (\mathrm{x},\mathrm{y}) \in \mathrm{L_{in}}$$

(b) A specified maximum number of iterations are reached

If rule (a) is satisfied, this indicates that the two boundaries, $\mathrm{L_{in}}$ and $\mathrm{L_{out}}$, differ on the direction the curve should evolve and thus signifying convergence is attained. If the maximum iterations are reached, then convergence may not be possible: real data is infused with noise and may not always converge.

3.2.3.4 Regularization Cycle

The second cycle provides the smoothness to the contour, In the traditional level set method, the curvature-based terms that are associated with the speed function are often used to regularize the level set function. Since $\phi(\mathrm{x},\mathrm{y})$ is the signed distance function, $|\nabla\phi(\mathrm{x},\mathrm{y})| = 1$, the regularized term can be rewritten as the Laplacian of the level set function: $\Delta\phi(\mathrm{x},\mathrm{y})$. The evolution of a function according to its Laplacian is equivalent to Gaussian filtering [1]. Thus, a Gaussian filtering process is implemented to regularize the level set function.

This regularization has a similar effect as the curvature-dependent terms in the speed function but avoids the computational expense associated with the curvature calculations.

As outlined in table 3-1, (under the regularization cycle) the cycle applies the Gaussian filter to the level-set function $\phi(x, y)$ and then update the linked lists (L_{out} and L_{in}) based on the sign of the result. Similar to the update cycle, the smoothing evolution will generate redundant points that are removed at the end of each evolution step. Table 3-1 shows the pseudo code for this algorithm, under the section marked Regularization Cycle. The discrete Gaussian kernel, G in table 3-1, is approximated with: $G = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$. The distance in table 3-1, is calculated with: $\mathrm{distance}(n^*, n|x, y) = |I_k(x^*, y^*) - I_k(x, y)|$, where $(x^*, y^*) \in N_4(x, y), \psi(x^*, y^*) = n^*, \text{and } \psi(x, y) = n$.

3.2.3.5 Finalization

After the stop condition is attained and the regulation cycle completes, the algorithm performs a likelihood test to determine the final region labeling map assignments for the values at the L_{out} locations. The pseudo code is presented in Table 3-1, finalization section.

```
Region Labeling Map $\widehat{\psi}_{k-1}(x,y), U, V, T_{max}$
Initialize
    $\widehat{\psi}_{k-1}(x,y) \leftarrow \widehat{\psi}_{k-1}(x,y)$
    $N_{k-1} \leftarrow \emptyset$
    foreach LabeledRegion $\Omega_n \subset \widehat{\psi}_{k-1}(x,y)$
        Initialize $\phi(x,y), L_{in}(n), L_{out}(n)$
        foreach $(x,y) \in L_{out}(n)$ { $\widehat{\psi}_{k-1}(x,y) \leftarrow \emptyset$ } end
    end
end
Main Loop: Two Cycles
    for $t = 1 : T_{max}$
        Update Cycle
            for $u = 1 : U_{max}$
                foreach $n \in N_{k-1}$
                    // Calculate the Speed values inside/outside the contour
                    foreach $(x,y) \in L_{out}(n) \cup L_{out}(n)$ { Compute $\hat{F}(x,y)$ } end
                    // Outward evolution
                    foreach $(x,y) \in L_{out}(n)$
                        if $\hat{F}(x,y) > 0$ {Switch$L_{out}$To$L_{in}(x,y)$} end
                    end
                    Update$L_{in}(L_{in}(n))$
                    // Inward evolution
                    foreach $(x,y) \in L_{in}(n)$
                        if $\hat{F}(x,y) < 0$ {Switch$L_{in}$To$L_{out}(x,y)$} end
                    Update$L_{out}(L_{out}(n))$
                    end
                    if StopConditionTrue() { Exit Update Cycle } end
                end
            end
        end // End Update
        Regulation Cycle //Smooth to account for curvature
            for $\upsilon = 1 : V_{max}$
                foreach $n \in N_{k-1}$
                    foreach $(x,y) \in L_{out}(n)$
                        if $(G * \phi)(x,y) < 0$ {Switch$L_{out}$To$L_{in}(x,y)$} end
                    end
                    Update$L_{in}(L_{in}(n))$
                    foreach $(x,y) \in L_{in}(n)$
                        if $(G * \phi)(x,y) > 0$ {Switch$L_{in}$To$L_{out}(x,y)$} end
                    Update$L_{out}(L_{out}(n))$
                    end
                end
            end
        end // End Regulation
        if StopConditionTrue() { Exit Main loop } end
end
Finalize
    foreach $n \in N_{k-1}$
```

```
        foreach $(x, y) \in L_{out}(n)$
            $\widehat{\psi}_{k-1}(x, y) \leftarrow \text{argmin}_{n^* \in N^*} \text{distance}(n^*, n|x, y)$
        end
    end
end
```

Table 3-1: Pseudo code for Two-Cycle algorithm (adapted from k. Li)

```
SwitchL$_{in}$ToL$_{out}$$(x, y)$
    foreach $(x, y) \in L_{in}(n)$
        remove $(x, y)$ from $L_{in}(n)$ and add it to $L_{out}(n)$
        foreach $(x^*, y^*) \in N_4(x, y)$ with $\phi(x^*, y^*) = -3$
            Add $(x^*, y^*)$ to $L_{in}(n)$; $\phi(x^*, y^*) \leftarrow -1$
        end
    end
end

SwitchL$_{out}$ToL$_{in}$$(x, y)$
    foreach $(x, y) \in L_{out}(n)$
        if $T_r(x, y) \neq 1$ {Exit} end
        remove $(x, y)$ from $L_{out}(n)$ and add it to $L_{in}(n)$
        foreach $(x^*, y^*) \in N_4(x, y)$ with $\phi(x^*, y^*) = 3$
            Add $(x^*, y^*)$ to $L_{out}(n)$; $\phi(x^*, y^*) \leftarrow 1$
        end
    end
end

UpdateL$_{in}$$(L_{in}(n))$
    foreach $(x, y) \in L_{in}(n)$
        if $\phi(x^*, y^*) < 0, \forall (x^*, y^*) \in N_4(x, y)$
            remove $(x, y)$ from $L_{in}(n)$; $\phi(x, y) \leftarrow -3$
        end
    end
end

UpdateL$_{out}$$(L_{out}(n))$
    foreach $(x, y) \in L_{out}(n)$
        if $\phi(x^*, y^*) > 0, \forall (x^*, y^*) \in N_4(x, y)$
            remove $(x, y)$ from $L_{out}(n)$; $\phi(x, y) \leftarrow 3$
        end
    end
end
```

Table 3-2: Pseudo code for Two-Cycle algorithm aux functions (adapted from k. Li)

3.2.4 Topological Constraints

Since multiple objects are tracked simultaneously, multiple contours will need to be simultaneously evolved. When multiple contours evolve, there is potential for the contours to incorrectly merge. Similar to the biological cells in [1], the two primary goals here are to: (1) allow the contours to split: (2) allow contours that came from the same original identification to merge while preventing contours from different objects to merge. To protect against this splitting and incorrect merging of contours that do not belong to the same object, topological constraints are employed.

The relaxed topological number, adapted from [1], is used to test if a pixel should evolve based on the aforementioned requirements. It is calculated by:

$$\mathrm{T_r}(\mathrm{x},\mathrm{y}) = \min\left[\alpha(\mathrm{x},\mathrm{y}), \max\left(\mathrm{T_{obj}}(\mathrm{x},\mathrm{y}), \mathrm{T_{bg}}(\mathrm{x},\mathrm{y})\right)\right] \tag{3-16}$$

$\alpha(\mathrm{x},\mathrm{y})$ is the number of overlapping object regions, $\mathrm{T_{obj}}(\mathrm{x},\mathrm{y})$ is the number of 4-connected components in the set $\Omega_\mathrm{N} \cap \mathrm{N_8}(\mathrm{x},\mathrm{y})$ and $\mathrm{T_{bg}}(\mathrm{x},\mathrm{y})$ is the number of 8-connected components in the set $\Omega_\mathrm{O} \cap \mathrm{N_8}(\mathrm{x},\mathrm{y})$. The $\max\left(\mathrm{T_{obj}}(\mathrm{x},\mathrm{y}), \mathrm{T_{bg}}(\mathrm{x},\mathrm{y})\right)$ is from the definition of a “simple” point; it prevents merging of objects. The $\alpha(\mathrm{x},\mathrm{y})$ is the added component that allows objects to merge if they have the same object ID. Figure x shows an example from the data sets collected.

3.2.5 Speed Equation

The speed function ($\hat{\mathrm{F}}$) is responsible for driving the contour towards the correct boundary. In this work, the function utilized incorporates information derived from each object’s region, edge, and motion.

$$\hat{F}(x, y) = \hat{F}_{region}(x, y) + w_{edge}\,\hat{F}_{edge}(x, y) + w_{motion}\,\hat{F}_{motion}(x, y) \qquad 3\text{-}17$$

Where w_{edge} and w_{motion} are the weights associated with the edge component and the motion component, respectively.

3.2.5.1 Region Component

To calculate the region component of the speed function, the initial segmentation (χ) from section 2.1 is utilized. The output from the segmentation module is based on the binary union of a region-based detection and an edge-based detection. It is derived from a region energy term presented by [19]. It represents the joint posterior probability that each pixel in frame k belongs to a certain propagated region, subject to a penalty on the total length of the boundary.

The region component of the speed function is calculated by:

$$\hat{F}_{region}(x, y) = \begin{Bmatrix} 1, & \text{if}\,\chi\ \ (x, y) > 0 \\ 0, & \text{otherwise} \end{Bmatrix} \qquad 3\text{-}18$$

3.2.5.2 Edge Component

The edge component is a measure of the “edgeness” at the boundary; it is inspired by geodesic active contours [20]. This can be interpreted as the length of a curve in a Riemannian Space where the metric is induced by image edges [1]. It is calculated by:

$$\begin{gathered} \hat{F}_{edge}(x, y) = -\big(e(x+1, y) - e(x-1, y)\big)\big(\phi(x+1, y) - \phi(x-1, y)\big) + \\ (e(x, y+1) - e(x, -1))(\phi(x, y+1) - \phi(x, y-1)) \end{gathered} \qquad 3\text{-}19$$

Where e(x,y) is defined as the Euclidean distance transform of the edge map of the intensity. The canny edge detector is used to calculate the edge map of the range. This technique is used because the definition induced fewer local minima as opposed to the gradient-based definition [1].

3.2.5.3 Motion Component

The motion component is included to incorporate past (and predicted future) dynamics of the object into the energy equation. It represents the joint probability that the object regions reside at the locations predicted by their respective motion filters. It is calculated by:

$$\hat{F}(x,y)_{\text{motion}} = \sum_{n \in N_{k-1}} \left(\log N(x,y \mid \hat{z}_{n,k|k-1}, S_{n,k-1}) - \tau \right) R_n(\psi(x,y)) \tag{3-20}$$

$$\text{Where, } R_n\big(\psi(x,y)\big) = \begin{Bmatrix} 1 & \text{if } \psi(x,y) = n \\ 0 & \text{if } \psi(x,y) \neq n \end{Bmatrix} \tag{3-21}$$

$\hat{z}_{n,k|k-1}$ and $S_{n,k-1}$ are the mean and covariance, respectively, of the bivariate normal distribution, $N(\cdot|x,S)$. The mean is the predicted centroid position for object n. It is calculated using the combined predicted state ($\hat{s}_{k|k-1}$) as follows:

$$\hat{z}_{n,k|k-1} = H\,\hat{s}_{k|k-1}. \tag{3-22}$$

$S_{n,k-1}$ can be considered an elliptical approximation of the objects shape by second moment matching. It is calculated by:

$$S_{n,k-1} = \text{cov}\{(x,y) \mid \psi_{k-1}(x,y) = n\} \tag{3-23}$$

3.2.6 Interacting Multiple Models (IMM)

An Interacting multiple models (IMM) motion filter is implemented for the motion-based calculation of the speed function from equation 3-17. The IMM filter operates multiple Kalman filters in parallel, each of which is matched to a unique motion model. A Markov chain governs the model transitions with a probability that is associated with switching from one model to the next. The result is a filter that computes an optimal weighted sum of the Kalman filter outputs. The stages of the IMM process consists of a mixing step, followed by filtering, and then combining. The IMM acts as the high-level architecture, to help fuse the different motion models.

3.2.6.1 State Equations

To model the dynamics of the objects a motion model is utilized. The motion model is defined as:

$$s_k = F^i s_{k-1} + v^i_{k-1}, i \in \{1,2 \ldots M\}, \text{ where M is the number of motion models} \qquad 3\text{-}24$$

Where, s_k, is the state vector of an object in frame k, and contains the position $(x\ y)$, velocity $(\dot{x}\ \dot{y})$, and acceleration $(\ddot{x}\ \ddot{y})$: $s_k = (x\ \dot{x}\ \ddot{x}\ y\ \dot{y}\ \ddot{y})$. F^i is the state transition matrix for model i and v^i_{k-1} is the process noise vector (zero-mean Gaussian process with covariance $\mathbf{Q}^i$) for model i. To relate the states to the measurements, a measurement model is defined as:

$$z_k = Hs_k + w_k \qquad 3\text{-}25$$

Where z_k is the measurement vector that contains the measured centroid position of the object. H is the measurement matrix and w_k is the measurement noise vector (zero-mean Gaussian process with covariance **R**).

Three different motion models are implemented for the state transition matrix: constant speed (F^1), constant acceleration (F^2), and random walk (F^3). By fixing these values instead of estimating them with an EM algorithm, the model weights can be used to indicate a dominate type of motion [1].

$$F^1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad F^2 = \begin{bmatrix} 1 & T_s & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T_s & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$F^3 = \begin{bmatrix} 1 & T_s & \frac{T_s^2}{2} & 0 & 0 & 0 \\ 0 & 1 & T_s & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T_s & \frac{T_s^2}{2} \\ 0 & 0 & 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3-26}$$

The measurement matrix is defined as:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3-27}$$

3.2.6.2 Kalman Filter: Prediction

The predictions are calculated with a Kalman filter (note: other predicting filters can be used here because of the modality of the IMM**).**

The initial combined covariance, γ_{k-1}^{0j}, and state $\hat{s}_{k-1}^{0j}$ are calculated given the weights, ρ_{k-1}^{i}, states, $\hat{s}_{k-1}^{i}$, and covariance, γ_{k-1}^{i}, of the previous frame.

$$\hat{s}_{k-1}^{0j} = \sum_{i} \rho_{k-1}^{i\,|\,j}\ \hat{s}_{k-1}^{i} \qquad 3\text{-}28$$

$$\gamma_{k-1}^{0j} = \sum_{i} \rho_{k-1}^{i\,|\,j}\ \left[\gamma_{k-1}^{i} + \left(\hat{s}_{k-1}^{i} - \hat{s}_{k-1}^{0j}\right)\left(\hat{s}_{k-1}^{i} - \hat{s}_{k-1}^{0j}\right)'\right],$$
$$\text{where } \rho_{k-1}^{i\,|\,j} = \frac{P_{ij}\ \rho_{k-1}^{i}}{\rho_{k|k-1}^{j}} \text{ and } \rho_{k|k-1}^{j} = \sum_{i} P_{ij}\rho_{k-1}^{i} \qquad 3\text{-}29$$

The weights, $\rho_{k|k-1}^{i}$, control the amount a given motion model will contribute to the final prediction (along with its covariance). They are a function of all the transition probabilities from all the other motion models. It utilizes the probability that the motion model will switch from a different model to the one currently under considered. The states predication for each model ($\hat{s}_{k|k-1}^{j}$) and the covariance predication for each model ($\gamma_{k|k-1}^{j}$) are calculated given the combined initial covariance (γ_{k-1}^{0j}) and the combined initial state ($\hat{s}_{k-1}^{0j}$).

$$\hat{s}_{k|k-1}^{j} = F^{j}\hat{s}_{k-1}^{0j} \qquad 3\text{-}30$$

$$\gamma_{k|k-1}^{j} = F^{j}\ \gamma_{k-1}^{0j}\ (F^{j})' + Q^{j} \qquad 3\text{-}31$$

P_{ij}, in equation 3-29, is the Markovian transition probability of switching from motion model i to motion model j of the current frame. The transition probabilities are often combined into a matrix to make the Transition Probability Matrix (TPM). The TPM is defined as:

$$P_{ij} = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix} \qquad 3\text{-}32$$

A fixed matrix is chosen empirically although in the case of [1], online minimum mean-square error estimation of the TPM is implemented with the idea that a fixed TPM would impede discovery of unknown cell behavioral variations.

Figure 7-2 displays the State diagram of the transition probabilities.

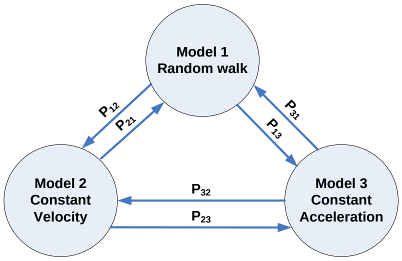


Figure 3-5: Graphic representation of the Transition Probabilities

The combined predicted state ($\hat{s}_{k|k-1}$) and the combined predicted covariance ($\gamma_{k|k-1}$) are calculated by combining the filter outputs from equations 3-30 and 3-31. The combining is performed as follows:

$$\hat{s}_{k|k-1} = \sum_{j} \rho^{j}_{k|k-1}\, \hat{s}^{j}_{k|k-1} \qquad \text{3-33}$$

$$\gamma_{k|k-1} = \sum_{j} \rho^{j}_{k|k-1} \left[\gamma^{j}_{k|k-1} + (\hat{s}^{j}_{k|k-1} - \hat{s}_{k|k-1})(\hat{s}^{j}_{k|k-1} - \hat{s}_{k|k-1})'\right] \qquad \text{3-34}$$

3.2.6.3 Kalman Filter: Update/Correction

Given the predicted states from equation 3-30 ($\hat{s}^{j}_{k|k-1}$), covariance from equation 3-31 ($\gamma^{j}_{k|k-1}$), and measured Centroid (z_k) the Kalman filters are utilized to obtain the updated states (s^{j}_{k}) and covariance (γ^{j}_{k}).

$$s_k^j = s_{k|k-1}^j + K_k^j(Z_k - Hs_{k|k-1}) \qquad 3\text{-}35$$

$$\gamma_k^j = \hat{\gamma}_{k|k-1}^j - K_k^j\left(Z_k - H\hat{s}_{k|k-1}^j\right), \qquad K_k^j = \gamma_{k|k-1}^j H'\left(H\gamma_{k|k-1}^j H' + R\right)^{-1} \qquad 3\text{-}36$$

K is known as the Kalman gain. Note $\hat{Z}_{k|k-1} = H\hat{s}_{k|k-1}$ is the predicted centroid that is used for the motion component of the speed function (equation 3-17). The likelihood that a model j is activated in the current frame (k) is

$$\lambda_k^j = \frac{\exp\left[-\frac{1}{2}(y_k^j)'\left(\mathbf{S}_k^j\right)^{-1}(y_k^j)\right]}{\sqrt{2\pi\det(\mathbf{S}_k^j)}} \qquad 3\text{-}37$$

Where $y_k^j = Z_k - \hat{Z}_{k|k-1}$; this is known as the innovation of the Kalman filter j. Note the $\mathbf{S}_k^j$ in equation 22 is the associated measurement covariance matrix. The combined state and covariance is computed using equations 18 and 19, however $\rho_{k|k-1}^j$ is replaced with $\rho_k^j = \frac{\rho_{k|k-1}^j\lambda_k^j}{\sum_i \rho_{k|k-1}^i\lambda_k^i}$.

The algorithm sets the weight of the motion filter to zero for the first five frames, so that the Kalman filter can build up history. The initial model weights are set equal to indicate that they are all equally likely.

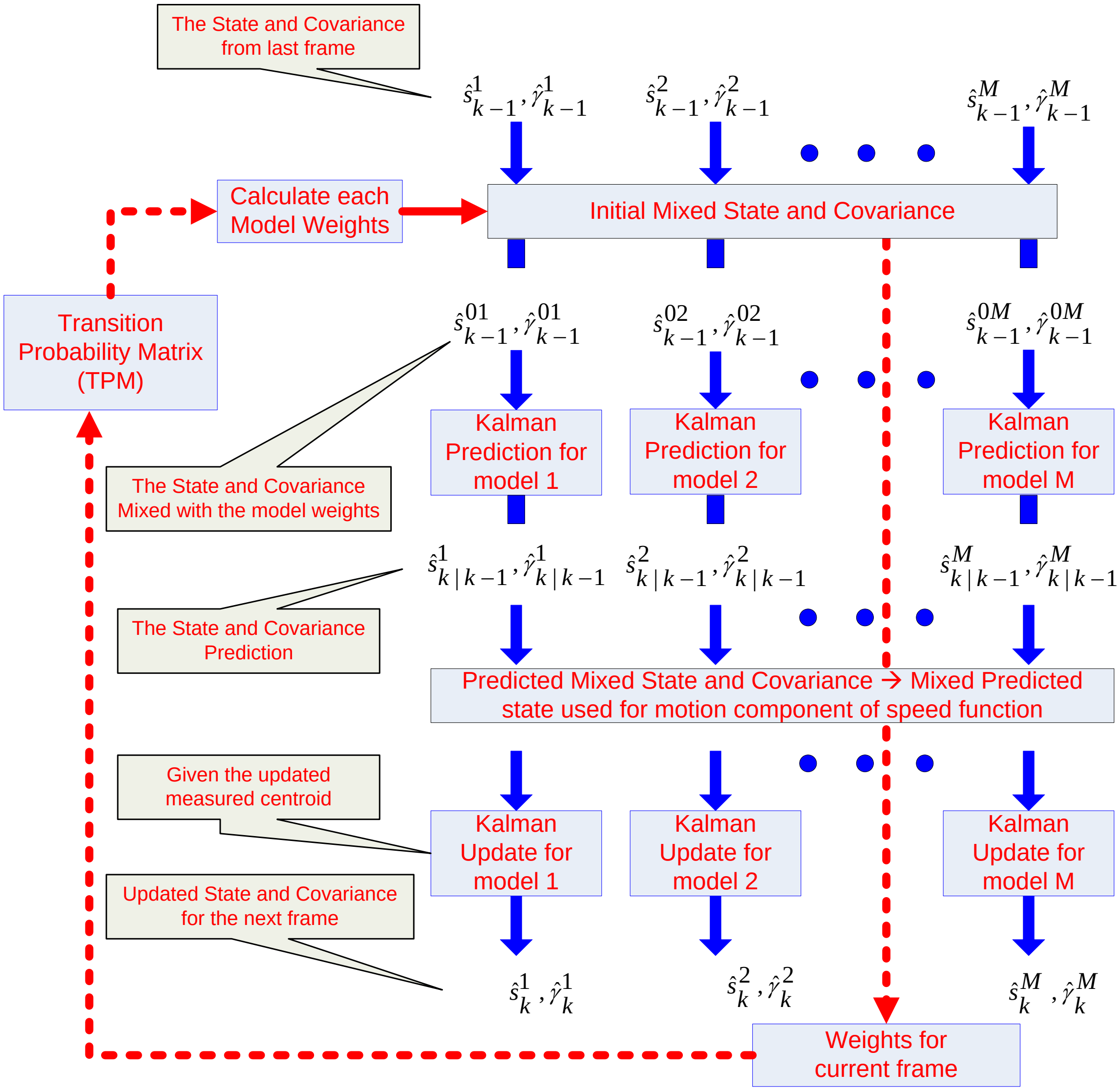

The State and Covariance from last frame
$\hat{s}^1_{k-1}, \hat{\gamma}^1_{k-1}$
$\hat{s}^2_{k-1}, \hat{\gamma}^2_{k-1}$
$\hat{s}^M_{k-1}, \hat{\gamma}^M_{k-1}$
Calculate each Model Weights
Initial Mixed State and Covariance
Transition Probability Matrix (TPM)
$\hat{s}^{01}_{k-1}, \hat{\gamma}^{01}_{k-1}$
$\hat{s}^{02}_{k-1}, \hat{\gamma}^{02}_{k-1}$
$\hat{s}^{0M}_{k-1}, \hat{\gamma}^{0M}_{k-1}$
The State and Covariance Mixed with the model weights
Kalman Prediction for model 1
Kalman Prediction for model 2
Kalman Prediction for model M
$\hat{s}^1_{k|k-1}, \hat{\gamma}^1_{k|k-1}$
$\hat{s}^2_{k|k-1}, \hat{\gamma}^2_{k|k-1}$
$\hat{s}^M_{k|k-1}, \hat{\gamma}^M_{k|k-1}$
The State and Covariance Prediction
Predicted Mixed State and Covariance → Mixed Predicted state used for motion component of speed function
Given the updated measured centroid
Kalman Update for model 1
Kalman Update for model 2
Kalman Update for model M
Updated State and Covariance for the next frame
$\hat{s}^1_k, \hat{\gamma}^1_k$
$\hat{s}^2_k, \hat{\gamma}^2_k$
$\hat{s}^M_k, \hat{\gamma}^M_k$
Weights for current frame

Figure 3-6: The Block diagram of the IMM structure and equations. Note: M equals 3 in the case of this paper: (1) random motion, (2) constant velocity (3) constant acceleration

3.3 Track Compiler

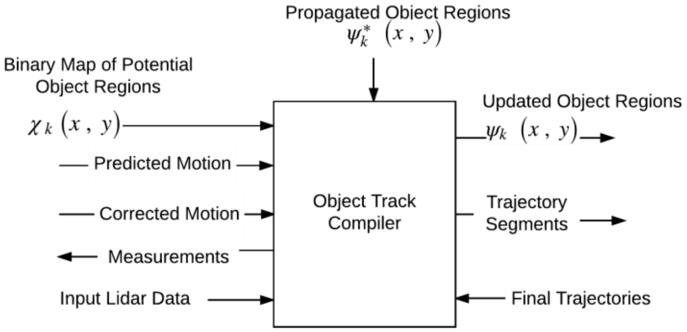


Figure 1-7: Track Compiler diagram

The track compiler module is responsible for creating the track segments for each of the objects and updating the map of labeled regions ($\psi_k(x,y)$). Once the contour evolution is complete, the tracking algorithm confirms or adjusts each of the identifications and matches it up with its track, if one exists. Any object that does not match to a single track is further processed to determine what occurred.

Let $N_k$represent the set of labels for all track segments created up to the current frame, k. A track segment is labeled "active" in frame k if it was successfully tracked in frame k-1, otherwise it is labeled "inactive".

The algorithm starts by comparing the map of labeled regions ($\psi_k(x,y)$) with the map of potential object regions (χ (x,y)). Any region that does not overlap between the two maps is considered either a new object or a lost object. If only one object from the map of labeled regions overlaps ($\psi_k(x,y)$) with a region from the potential object region map (χ (x,y)), then that object is assigned the same track segment. If multiple regions overlap and are separated by a minimum distance (d_{min}), the track compiler considers the following options: (1) the object split into multiple objects, so assigned it a "child" id that is associated to the original parent object: (2)

the object is a previously occluded object, so assign it the occluded object’s id number: (3) it is a new object that has entered the FOV so assign it a new track id number.

The Pseudo code is presented in table 3-3. The Pseudo code for the additional function calls are presented in table 3-4

```
$\Omega_0 \leftarrow \{(x,y)|\hat{\psi}_k(x,y) = 0\}$
foreach connected component $\omega \subset \chi_k$
    if $w \subset \chi_k$ then AddTrack($N_{new}, k, w$)
foreach ActiveTrack $n \in N_{k-1}$
    $\Omega_n \leftarrow \{(x,y)|\hat{\psi}_k(x,y) = n\}$
    if $\Omega_n = \emptyset$ then DeactivateTrack(n)
    else if MultipleObjectsOverlap( $\Omega_n$)then
        if ObjectSplit($\Omega_n$)then
            foreach connected component $\omega \subset \Omega_n$
                CreateChildTrack($n_{child}, n, k, \omega$)
        else
            $\hat{\omega} \leftarrow$ SelectBestMatch($n, k, \Omega_n$)
            UpdateTrack($n, k, \hat{\omega}$)
            foreach connected component $\omega \subset \Omega_n \setminus \hat{\omega}$
                AddTrack($n_{child}, \omega, k$)
            end
        end
    else
        UpdateTrack($n, k, \hat{\omega}$)
    end
end
```

Table 3-3: Pseudo code for Track Compiler (adapted from k. Li)

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//// AddTrack creates a new track segment labeled $n_{new}$; fills region ω with $n_{new}$; and initializes the
//// cell state based on measurements of ω.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 AddTrack($\omega, n_{new}, k$)
    CreateNewTrackSegment($n_{new}$)
    FillRegion($\omega, \psi, n_{new}$)
    CalculateDynamics($\omega$)
Send
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////  updates the track segment n using the features of region ω, including the centroid
////  location, mean intensity, area, and eccentricity. The centroid and the mean intensity are
////  fed to the motion filter to obtain a filtered state of object n in frame k.
////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
UpdateTrack($n, k, \omega$)
    CalculateDynamics($\omega$)
    UpdateTrackSegment(n   )
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////  Add a child track for the object n with a unique label $n_{child}$, and fills the region ω with $n_{child}$.
////  Calculate the object state based on the measured centroid location and mean intensity of ω,
////  and the predicted state of object  n (parent).
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
AddChildTrack($n_{child}, \omega, n, k$)
    CreateNewTrack($n_{new)}$
     CalculateDynamics($\omega$)
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//// Selects component  $\hat{\omega} \in \Omega_n$ that best matches the dynamics of object n.  Use the innovation
//// likelihood and select the one which maximizes it.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
SelectBestMatch($n_{child}, \omega, n, k$)
    CreateNewTrack($n_{new)}$
     CalculateDynamics($\omega$)
    SelectMaxInnovationLikelihood()
end
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////  returns true if region $\Omega_n$ has multiple  connected components and the minimum distance
////  between any two points in different components is greater than a preset threshold $d_{min}$
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
ObjectSplit($\Omega_n$)
    Foreach ConnectedComponent cc in  $\Omega_n$
        For i =  cc  :  NumberOfConnectComponents
```

```
            DoMultipleComponentsOverlapRegion(cc)
        end
    end
end
```

Table 3-4: Pseudo code for Track Compiler auxiliary functions (adapted from k. Li)

3.4 Track Linker

The main goal here is to find tracks that split and came back together or reemerge in accordance to the following physical parameters:

(1) No object is removed unless it leaves the FOV or is considered noise.

(2) No new object is added unless it enters from outside the FOV or it is matched to a previously occluded object.

The track linker attempts to match lost or previously occluded tracks by using the following method:

Let $N_{\mathrm{found}} = \{n_f | f = 1,2 \ldots, F\}$, denote the set of track segments that start after the first frame. $N_{\mathrm{lost}} = \{n_l | l = 1,2 \ldots, L\}$ denote the set of track segments that end before the current frame. For each node pair, (n_l, n_f), an arc <n_l, n_f> is created between the two nodes if the last centroid of $n_l$is within the double cone centered at the first centroid location of n_f.

$$\sqrt{(x_l - x_f)^2 + (y_l - y_f)^2} \leq |K_l - K_f| R + R_0 \qquad 3\text{-}38$$

With $|K_l - K_f| \leq \frac{D}{2}$. The $|k_l - k_f|$ is to account for the number of frames that past with an object track lost. D, R, and R_0 are fixed parameters. $k_l$and $k_f$are the frames when the referenced object was lost and found, respectively. Each arc is assigned a weight (ω_{lf}) and is defined as the maximum innovation likelihood ($\omega_{lf} = \lambda^{\max}_{n_l,k_f}(n_f)$) of track n_l on the measurement of track $n_f$in frame k_f. It is calculated using equation 3-37:

$$\lambda_{n_l,k_f}^{\max}(n_f) = \frac{\exp\left[-\frac{1}{2}(y_k^{\max})'(\mathbf{S}_k^{\max})^{-1}(y_k^{\max})\right]}{\sqrt{2\pi\det(\mathbf{S}_k^{\max})}} \qquad 3\text{-}39$$

A maximum-likelihood matching algorithm is computed between tracks n_l and n_f. The maximum innovation likelihood is also calculated for the one-to-two matching. In this case, the spatial temporal mean of the two found objects (n_{f_1} and n_{f_2} for example) is assigned as the starting point. The problem is solved as an integer programming problem:

$$\max_{x} \mathbf{f}^T \mathbf{x} \text{ subject to } \mathbf{A}^T\mathbf{x} \leq \mathbf{b} \qquad 3\text{-}40$$

Where **A** be a constraint matrix of size H x (L+F), **b** is an (L+F) x 1 vector of ones, and **f** is a likelihood vector of size H x 1. H is the total number of matchings hypotheses, both one-to-one and one-to-two. L and F are the number of elements in N_{lost} and N_{found}. The constraint matrix contains the different hypotheses between the elements of N_{lost} and N_{found}. The likelihood vector holds the corresponding maximum innovation likelihood for the hypotheses in **A**. The constraint matrix and likelihood vector are constructed according to the pseudo code in table 3-6 and equation 3-41 (one-to-one match) ,3-42 (one-to-two match). The solution selects the matching hypotheses with the greatest maximum innovation likelihood that conforms to the constraints imposed by C. Since this method utilizes the spatial temporal cone to filter and the maximum innovation likelihood to match, both the location and the dynamics are incorporated into the linking of lost tracks.

```
$N_{lost} \leftarrow \emptyset$ and $N_{found} \leftarrow \emptyset$
foreach track $n \in N_k$
    if LostTrack(n, k) { Add n to $N_{lost}$ }
    else if FoundTrack(n, k) { Add n to $N_{found}$}
end
MatchTracks($N_{lost}, N_{found}$)
foreach track $n_l \in N_{lost}$
    if IsMatchedToOne($n_l, n_f \in N_{found}$) { LinkTracks( $n_l, n_f$) }
    else if IsMatchedToTwo($n_l; n_{f_1}, n_{f_2} \in N_{found}$) { LinkTracks( $n_l, n_f$) }
end

foreach track $n \in N_k$
    if IsShort(n, k) { DeleteTrack(n) }
end
```

Table 3-5: Pseudo code for Track Linker

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//// MatchTracks
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
MatchTracks($N_{lost}, N_{found}$)
    foreach track $n_l \in N_{lost}$
        foreach track $n_f \in N_{found}$
            if InsideDoubleCone($n_l, n_f, k_{l,} k_f$)
                $\omega_{lf}$ = CalculateMaxInnovationLikelihood($n_l, n_f, k_{l,} k_f$);
                // Store connections to test for one-to-two connections after this loop
                PotentialSelection.Append($n_l, n_f, k_{l,} k_f, \omega_{lf}$)
                ConstraintMatrix.Append($n_l, n_{f,}$);
                LikelihoodVector.Append($\omega_{lf}$);
            end
        end
    end
    foreach OneToTwoMatches($n_l$) $h' \in$ PotentialSelection
        $\omega_{lf_1 f_2}$ = CalculateMaxInnovationLikelihoodWithSpatialTemporalMean ($n_l, n_{f_1}, n_{f_2}, k_{l,} k_f$);
        ConstraintMatrix.Append($n_l, n_{f_1}, n_{f_2}$)
        LikelihoodVector.Append($\omega_{lf_1 f_2}$);
    end

    // Solve for the solution vector x from the integer programming problem
    x = intlinprog(LikelihoodVector, ConstraintMatrix,1)
end
```

Table 3-6 Pseudo code for Track Linker auxiliary functions (adapted from k. Li).

$$A(h,i) = \begin{cases} 1, & \text{if } i = l, i = L + f \\ 0, & \text{otherwise} \end{cases}$$

3-41

$$d(h) = \omega_{lf}$$

$$A(h,i) = \begin{cases} 1, & \text{if } i = l, i = L + f_1, i = L + f_2 \\ 0, & \text{otherwise} \end{cases}$$

3-42

$$d(h) = \omega_{lf_1f_2}$$

3.5 Object Database

After the tracks are finalized, a new entry is sent to the object database. The database is a Elasticsearch database using the open source Elastic Stack [https://www.elastic.co/products]. The data is also stored as csv format flat file post-analysis. The database stores the following information: Track (“Parent”) ID, Child ID, Centroid, Area, Covariance, Predicted Centroid, motion filter statistics, and all the state vector information.

Chapter 4

Results

4.1 Detection, Segmentation, and Tracking Results

The system discussed in chapter 3 was tested with Flash LIDAR data. The Range data is the only component processed, any intensity image displayed is only for the sake of showing the objects and background. It is sometimes difficult to visually discern the objects and background in the range data image, in those situations the results are overlaid on top of the intensity image.

Figure 4-1 shows a few examples of the detection and segmentation of a frame. The contours are layered on top of the intensity image. The top two images show an example where the lower limit range filter was set to high. This is why there is a straight line cutting where the contours stop growing on their legs. The bottom set shows an example where the max iterations of the two-cycle algorithm should be increased since some of the contours did not completely grow. In both cases, the segmentation works well and the contours approximate the objects.

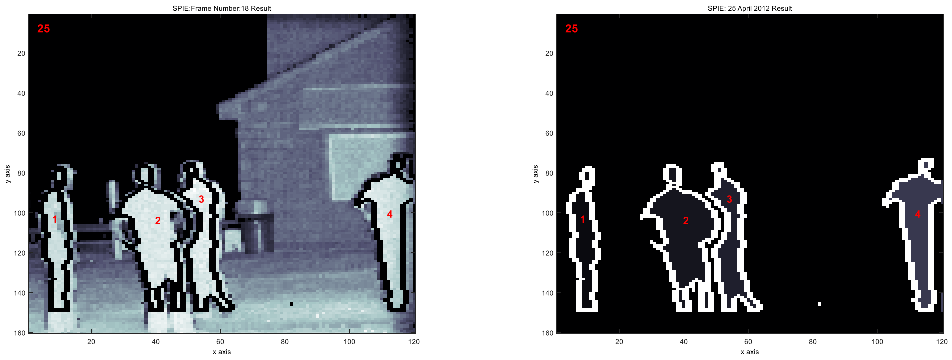


Figure 4-1: Segmentation example, Range Cutoff: (Left) The contours displayed on the intensity image. (Right) The segmentation of the input frame

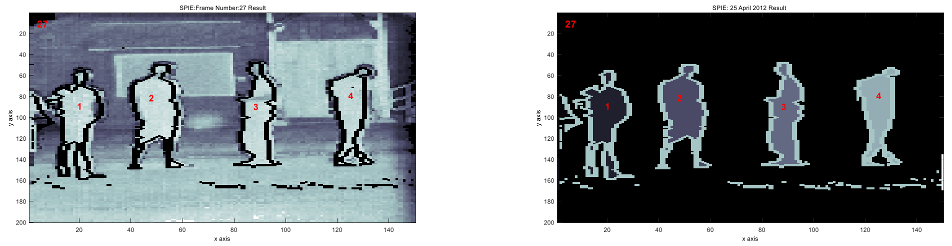


Figure 4-2: Segmentation example, contour cycle limit: (Left) The contours displayed on the intensity image. (Right) The segmentation of the input frame

Figure 4-3 shows another segmentation along with a graphical representation of how the elliptical approximation used in the motion filter fits to a segmented object.

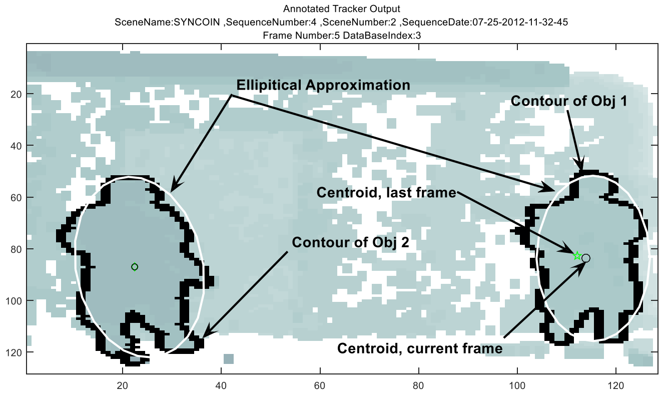

Annotated Tracker Output
SceneName:SYNCOIN ,SequenceNumber:4 ,SceneNumber:2 ,SequenceDate:07-25-2012-11-32-45
Frame Number:5 DataBaseIndex:3
Ellipitical Approximation
Contour of Obj 1
Centroid, last frame
Contour of Obj 2
Centroid, current frame
20
40
60
80
100
120

Figure 4-3: Example of Elliptical approximation of contours used in the motion filter

Figure 4-4 presents a short sequence of frames along with its track assignments. The sequence shows that the topological constraint correctly prevented objects 2 and 3 from merging. It also automatically identified object 5 as it enters the frame. Since object 5 is detected near the FOV boarder, it is assigned it a new ID.

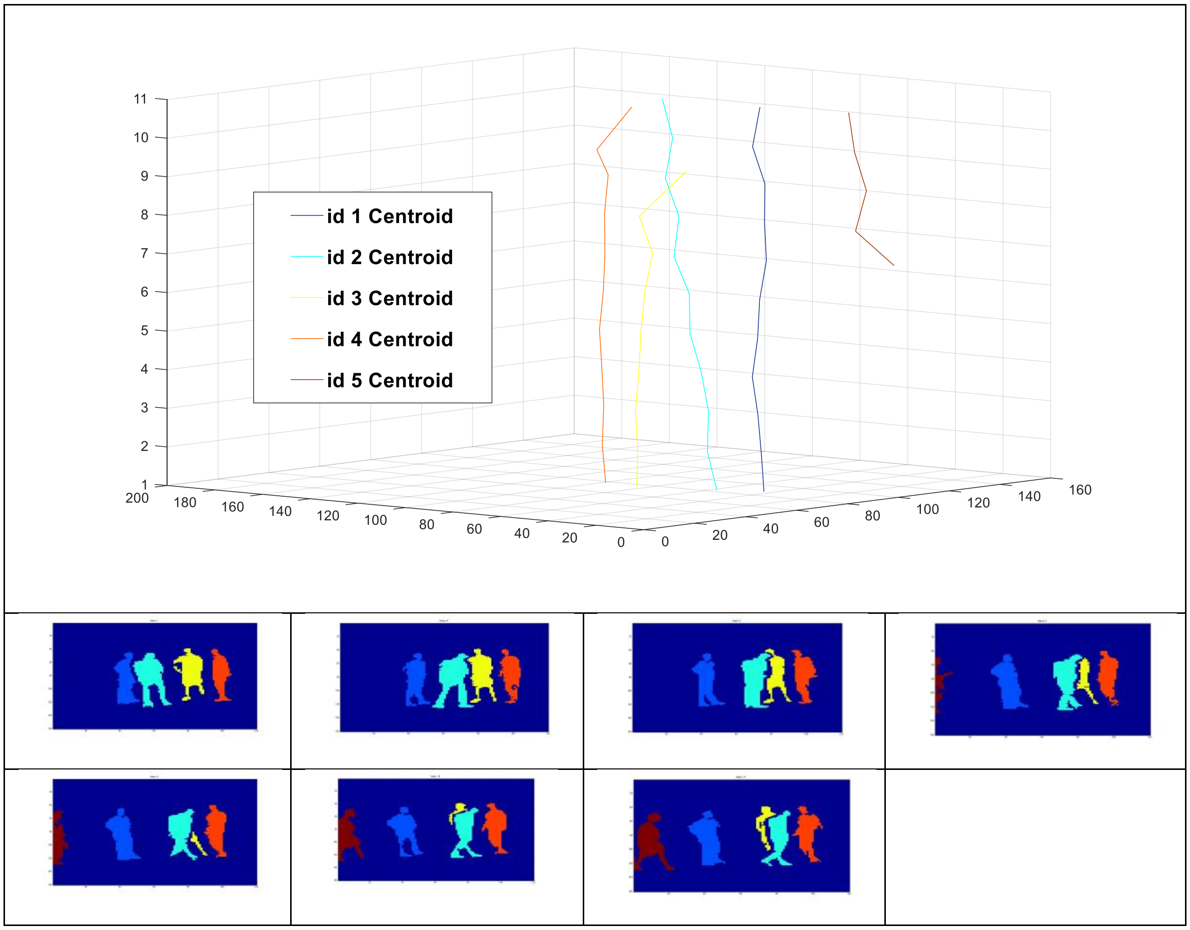


Figure 4-4: 3-D Tracks for multiple objects (vertical axis is the frame number). Object 1 is dark blue, 2 is light blue, 3 is yellow, 4 is orange, r is maroon. (a) Frame 1 (b) Frame 4 (c) Frame 6 (d) Frame 8 (e) Frame 9 (f) Frame 10 (g) Frame 11

Figure 4-5 displays the predicted positions from the IMM motion filter as the objects move. The solid line is the measured position and the dotted is the predicted. The predicted position is able to converge within a few frames unless an abrupt change occurs.

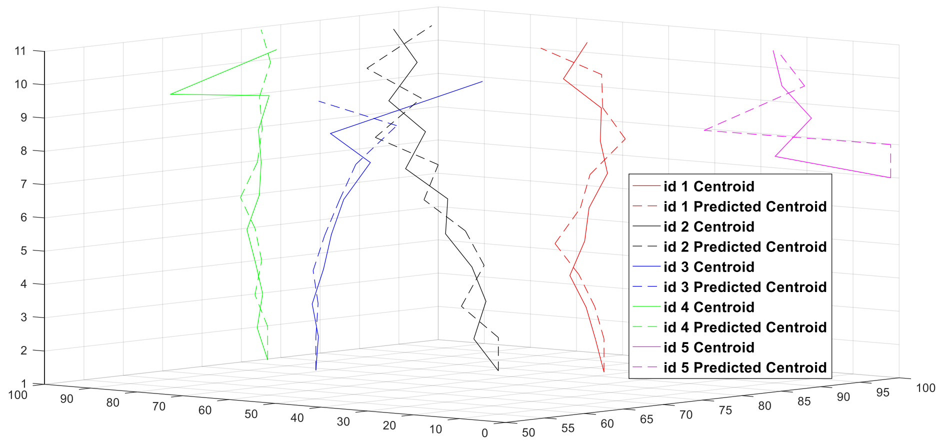
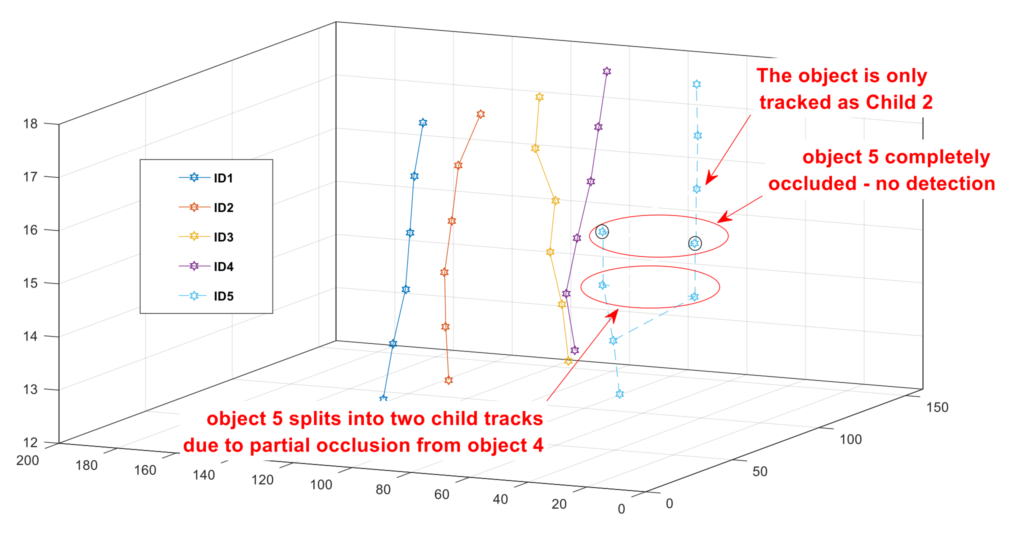


Figure 4-5: The predicted and measured positions from the IMM motion filter. The dashed line is the predicted data and the solid line is the measured.

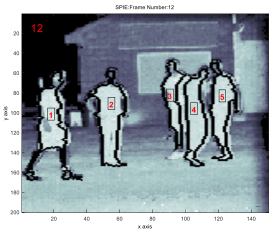
Figure 4-6 presents a short sequence of frames along with its track assignments. This sequence displays two main points. First, object 5 gets partially occluded by object 4 and causes a split. The tracking system correctly identifies the situation and assigns two child IDs to object 5. Second, in the next frame, it is completely occluded and the tracking system flags them as occluded and propagates their position based on the predicted position. In the following frame, child 2 emerges and is tracked.



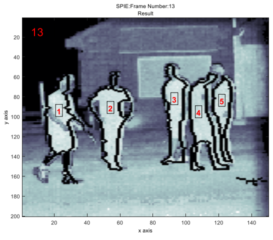
The object is only tracked as Child 2

object 5 completely occluded - no detection

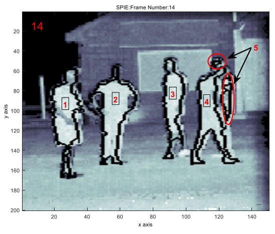
object 5 splits into two child tracks due to partial occlusion from object 4



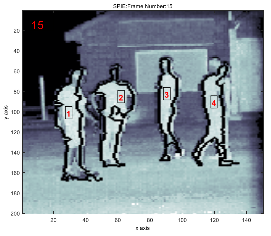
Frame 12



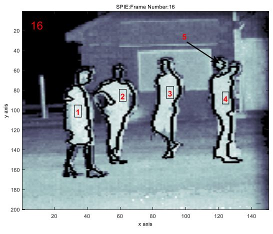
Frame 13



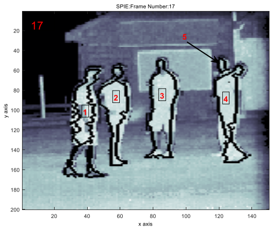
Frame 14



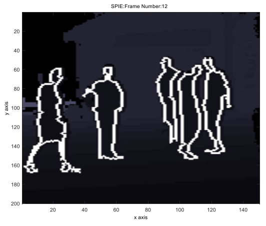
Frame 15



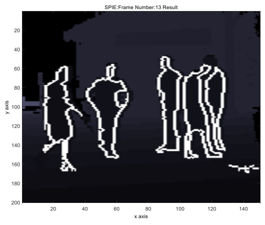
Frame 16



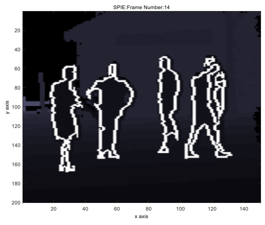
Frame 17



Frame 12



Frame 13



Frame 14

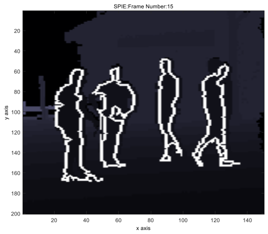
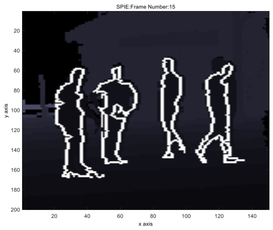
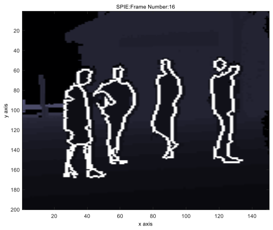
		
Frame 15	Frame 16	Frame 17

Figure 4-6: Object Track and Link (top) A segment of the center positions for the frames listed below the plot (top). A series of the Intensity images with the contours overlaid (middle). A series of the Range images with the contours overlaid (bottom).

The topological constraint that stopped objects with different IDs from merging caused issues in two scenarios: (1) if the object enters a vehicle, (2) if the object initially is segmented into two objects. Consider the scenario where a person enters a car, the car drive somewhere and the person gets out. If the person and the vehicle have separate track ID's, they will not be able to merge when the person enters the car; it will be flagged as occluded. The car then drives off, the person gets out and they are now either assigned a new ID (or potentially get assigned a false association if they exit the vehicle close to other tracked objects). Either way, at that point the connection to the original object track is broken. If the topological constraints allowed the objects to merge, then the track could persist through that situation. The system implemented for tracking split objects could be extended to account for linking merged objects. The second scenario is displayed in Figure 4-7, the person on the right has an item across their waist. This item causes the segmentation to split the initial detection into two objects. A few frames later the person is segmented as a single object. Since the object properties are significantly different than either one of the small objects, the algorithm flagged the two smaller objects as occluded and assigned it a new temporary id. This case can also be avoided if the objects of different ID's are allowed to merge.

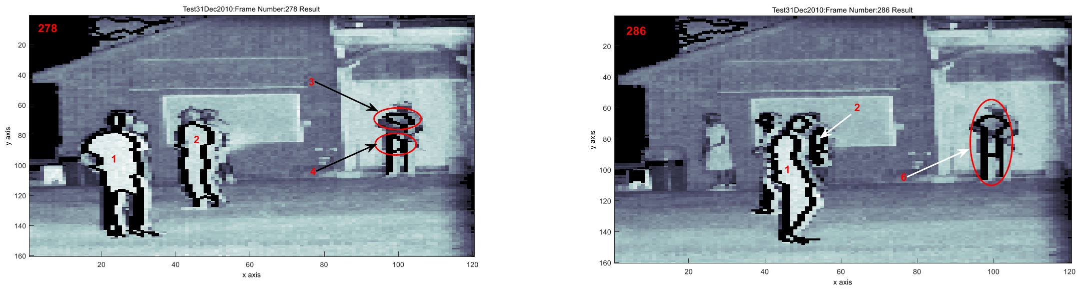


Figure 4-7: Segmentation of object that is initially split and later is detected as a single object.

Chapter 5

Conclusion

The system performed with a varying degree of success. It is able to segment the image to at least a close approximation for the majority of the frames. However, if often picks up a few false detections and sometimes completely misses people in the scene. If the segmentation step works, then the tracker behaves as expected. The IMM adapts quickly to changing motion and assists the contour evolution, as well as, the track linking. There are cases when the tracker assigns the incorrect id to an object but that is normally explained by the detection module providing a false object.

The most significant issue was dealing with large displacements between frames, or equivalently, large relative velocity.   If the frame rate is not high or the objects are moving too fast, it can cause false assignments and dropped tracks. If the motion filter correctly models the dynamics, it helps since it can influence the evolution via the speed function from the predicted dynamics. However, in the data tested for this work, it normally would drop the track and create the object as a new temporary object.

The data processed consisted of scenes where the camera was stationary, as well as, panning. In the case where the camera was panning, objects that were moving in the opposite direction of the camera's rotation had a large positional displacement between frames. The large jump normally caused the tracker to drop the track.

In the future, other detection and segmentation algorithms can be introduced to assist with the initial detection. Other rules can be incorporated to deal with large relative velocities or the camera motion can be determined and included in the calculations. Also, it would be interesting to analyze the IMM filters to look for abnormal behavior indicators or to incorporate all the different data components of the LIDAR output to see what benefit it provides.

Bibliography

[1] Li, K., Miller E.,Chen M., Smith, A. S., Kanade T., Weiss L., Campbell P., “Cell population tracking and lineage construction with spatiotemporal context,”, Medical Image Analysis, 546-566 (2008).

[2] Shi Y., Karl W., "Real-Time Algorithm for the Approximation of Level-Set-Based Curve Evolution," IEEE Transactions on image processing, VOL. 17, NO. 5, 645-656 (May, 2008).

[3] Natale, D., Tutwiler, R., “3DSF: Three-dimensional Spatiotemporal Fusion,” Defense Transformation and Net-Centric Systems 2011, Vol. 8062 80620E-1

[4] V. Caselles, F. Catte, T. Coll, and F. Dibos, “A geometric model for active contours in image processing”, Numer. Math., vol. 66, pp. 1-31, 1993.

[5] R. Malladi, J.A. Sethian, and B.C. Vemuri, “Shape modeling with front propagation: a level set approach”, IEEE Trans. Patt. Anal. Mach. Intell., vol. 17, pp. 158-175, 1995.

[6] Osher, Stanley. "Level set methods." Geometric level set methods in imaging, vision, and graphics. Springer New York, 2003. 3-20

[7] S. Osher and James A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. J. Computational Physics, 79(1):12-49, 1988

[8] Shi, Yonggang, and William Clement Karl. "Real-time tracking using level sets." 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). Vol. 2. IEEE, 2005

[9] Osher S, Sethian J. Fronts propagation with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations. J Comput Phys. 1988;79:12–49

[10] Shi, Yonggang, and William Clem Karl. “A Real-Time Algorithm for the Approximation of Level-Set-Based Curve Evolution.” IEEE transactions on image processing: a publication of the IEEE Signal Processing Society 17.5 (2008): 645–656. PMC. Web. 26 Nov. 2016.

[11] T. Laux, "ASC's 3D Flash LIDAR™ Camera: The Science behind ASC's 3D Depth Imaging Video Camera," SMPTE International Conference on Stereoscopic 3D for Media and Entertainment, New York, NY, USA, 2010, pp. 1-10. doi: 10.5594/M001404

[12] Luo, Wenhan, et al. "Multiple Object Tracking: A Literature Review." arXiv preprint arXiv:1409.7618 (2014).

[13] D. J. Natale, R. L. Tutwiler, M. S. Baran and J. R. Durkin, "Using full motion 3D Flash LIDAR video for target detection, segmentation, and tracking," 2010 IEEE Southwest Symposium on Image Analysis & Interpretation (SSIAI), Austin, TX, 2010, pp. 21-24.

[14] Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24, 6 (June 1981), 381-395.

[15] Han, Xiao, Chenyang Xu, and Jerry L. Prince. "A topology preserving level set method for geometric deformable models." IEEE Transactions on Pattern Analysis and Machine Intelligence 25.6 (2003): 755-768.

[16] Xu, Chenyang, Anthony Yezzi, and Jerry L. Prince. "A summary of geometric level-set analogues for a general class of parametric active contour and surface models." Variational and Level Set Methods in Computer Vision, 2001. Proceedings. IEEE Workshop on. IEEE, 2001.

[17] Tsai, Richard, and Stanley Osher. "Review article: Level set methods and their applications in image science." Communications in Mathematical Sciences 1.4 (2003): 1-20.

[18] Shi, Yonggang, and William Clem Karl. "A real-time algorithm for the approximation of level-set-based curve evolution." IEEE transactions on image processing 17.5 (2008): 645-656.

[19] Zhu, Song Chun, and Alan Yuille. "Region competition: Unifying snakes, region growing, and Bayes/MDL for multiband image segmentation." IEEE transactions on pattern analysis and machine intelligence 18.9 (1996): 884-900.

[20] Caselles, Vicent, Ron Kimmel, and Guillermo Sapiro. "Geodesic active contours." International journal of computer vision 22.1 (1997): 61-79.

[21] Cremers, Daniel, Mikael Rousson, and Rachid Deriche. "A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape." International journal of computer vision 72.2 (2007): 195-215

[22] Goldenberg, Roman, et al. "Fast geodesic active contours." IEEE Transactions on Image Processing 10.10 (2001): 1467-1475

[23] Zuliani, Marco. "RANSAC for Dummies." Vision Research Lab, University of California, Santa Barbara (2009).

[24] Kass, Michael, Andrew Witkin, and Demetri Terzopoulos. "Snakes: Active contour models." International journal of computer vision 1.4 (1988): 321-331.

[25] Sethian, James A. "Evolution, implementation, and application of level set and fast marching methods for advancing fronts." Journal of computational physics 169.2 (2001): 503-555.