UNIVERSITY OF CALIFORNIA SAN DIEGO

Spot, an Algorithm for Low-Resolution, Low-Contrast, Moving Object-Tracking with a Non-Stationary Camera

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Electrical Engineering (Intelligent System, Robotics, and Control)

by

Christopher Lee Crutchfield

Committee in charge:

Professor Curt Schurgers, Chair
Professor Ryan Kastner, Co-Chair
Professor Nikolay Atanasov

2023

The Thesis of Christopher Lee Crutchfield is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

EPIGRAPH

The world is one big data problem.

Andrew McAfee

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude and appreciation to all the individuals and institutions who have contributed to the successful completion of this master's thesis. It was a long and arduous road, but it was mine, and for that, I am grateful.

First and foremost, I am immensely grateful to my thesis advisors, Professor Curt Schurgers and Professor Ryan Kastner, for their guidance, expertise, and unwavering support throughout this research journey. Their invaluable insights, constructive feedback, and continuous encouragement have been instrumental in shaping this thesis and pushing me to strive for excellence. I further would like to express my appreciation for them providing me with further opportunities beyond my master's. I look forward to our future endeavors on other interesting and exciting projects.

I extend my sincere thanks to the faculty members of the Electrical and Computer Engineering department at the University of California, San Diego, for providing a conducive academic environment and for their dedication to imparting knowledge. Their commitment to excellence in teaching and research has inspired me to explore new horizons and pursue intellectual challenges.

I would like to express my gratitude to the research staff, technicians, and administrative personnel at Engineers for Exploration, whose assistance and cooperation were crucial in conducting experiments, accessing necessary resources, and facilitating a smooth research process. Their expertise and willingness to share their knowledge have been invaluable.

I would also like to express my appreciation to my friends and classmates who have provided constant encouragement, stimulating discussions, and a supportive network. Their belief in my abilities and their friendship have made this academic journey more enjoyable and meaningful.

I would like to express my thanks to my wife, who has stood by me throughout this entire journey, encouraging me when I felt that I was unable to succeed. Your help and support have pushed me to my limits and beyond. I would not have had the courage to persevere without your

guidance.

I also would like to thank my parents, who helped me immensely throughout my schooling career. Their continued persistence allowed me to course correct when I had worried I had irreparably damaged my school career.

Finally, thank you to the coauthors of the documents I have published during my tenure at UC San Diego.

Chapter 2 and Chapter 4 contain material originally published in International Conference on Methods and Techniques in Behavioral Research (Measuring Behavior), October 2020. Crutchfield, Christopher L.; Sutton, Jake; Ngo, Anh; Zadorian, Emmanuel; Hourany, Gabrielle; Nelson, Dylan; Wang, Alvin, McHenry-Crutchfield, Fiona; Forster, Deborah; Strum, Shirley C.; Kastner, Ryan; Schurgers, Curt. The thesis author was the primary investigator and author of this paper.

"Sherlock: A Multi-Objective Design Space Exploration Framework" is a work that is tangential to the work detailed within this write-up, and therefore no content is reproduced therein. This document was originally published in ACM Transactions on Design Automation Electronic Systems, Vol. 27, No 4., Article 33, March 2022. Gautier, Quentin; Althoff, Alric; Crutchfield, Christopher L.; and Kastner, Ryan.

"Underwater Depth Calibration Using a Commercial Depth Camera" is a work that is unrelated to the work detailed within this write-up, and therefore no content is reproduced therein. That said, the thesis author would like to acknowledge this work regardless. It was originally published at International Conference on Underwater Networks & Systems, November 2022. Wong, Emily M.; Humphrey, Isabella; Switzer, Scott; Crutchfield, Christopher L.; Hui, Nathan; Schurgers, Curt; and Kastner, Ryan.

To all those who have contributed directly or indirectly to this thesis, please accept my heartfelt appreciation. Your involvement and support have played a significant role in shaping my academic and personal growth, and I am truly grateful for your contributions.

Thank you all.

VITA

2018 Bachelor of Science, University of California, San Diego

2023 Master of Science, University of California, San Diego

PUBLICATIONS

"Underwater Depth Calibration Using a Commercial Depth Camera", International Conference on Underwater Networks & Systems, November 2022

"Sherlock: A Multi-Objective Design Space Exploration Framework", ACM Transactions on Design Automation Electronic Systems, Vol. 27, No. 4, Article 33, March 2022

"Baboons on the Move: Enhancing Understanding of Collective Decision Making through Automated Motion Detection from Aerial Drone Footage", International Conference on Methods and Techniques in Behavioral Research (Measuring Behavior), October 2020

ABSTRACT OF THE THESIS

Spot, an Algorithm for Low-Resolution, Low-Contrast, Moving Object-Tracking with a Non-Stationary Camera

by

Christopher Lee Crutchfield

Master of Science in Electrical Engineering (Intelligent System, Robotics, and Control)

University of California San Diego, 2023

Professor Curt Schurgers, Chair
Professor Ryan Kastner, Co-Chair

The ability to track moving objects in a video stream is helpful for many applications, from pedestrian and vehicle tracking in a city to animal tracking for ecology and conservation. This write-up introduces Spot, an algorithm for moving object tracking in low-resolution, low-contrast videos. This write-up will discuss two motivating examples to guide the development of Spot–satellite-based surveillance of vehicles in cityscapes and animal tracking using drones for ecological purposes.

Spot uses image processing techniques to generate a pipeline to track moving objects frame-to-frame. It then leverages Bayesian Filtering techniques to use the frame-to-frame motion

to track individual identity between consecutive frames.

Each stage of Spot's pipeline–both image processing and the Bayesian Filtering portions of the pipeline–introduces many parameters. To determine which parameters are ideal for a particular dataset, a design space exploration tool, dubbed Sherlock, is used to choose the optimal parameters. As part of this, we evaluate multiple possible objective functions and demonstrate the importance of selecting an appropriate one.

Spot is competitive with other modern, moving object-tracking algorithms on cityscape data, outperforming others in some of the metrics presented. For tracking animals from drone footage, Spot demonstrated an ability to track wildlife at a similar rate to its performance in some cityscape videos.

Chapter 1

Introduction

Object tracking in video footage is a common problem encountered in many applications. In this context, the "object" is anything we want to follow within the video. For example, in surveillance applications, this could be any person moving into the environment we are monitoring. Applications are vast, from tracking comets in the sky to a football in a sports broadcast.

Within this ample application space of object tracking in videos, we are specifically interested in the problem when we need to track multiple small, hard-to-find objects. More accurately, the objects themselves can be large, but in the video, each object is only a few pixels in area. As such, these objects have few distinguishing features, making them hard to keep track of. In addition, often, these small objects are hard to distinguish from the background. This specific subset of tracking problems can occur in a variety of applications. For example, the camera could be on a satellite used to track cars in urban spaces. This is useful for monitoring traffic conditions, aiding in traffic congestion planning, etc. Because of the distance between the satellite and Earth, these cars will appear as tiny moving objects in the video, occupying only a small number of pixels. A similar challenge can occur when using drones for wildlife monitoring, as seen in Crutchfield et al.[6] with baboons. In this case, the distance between the camera and the object is much smaller. Still, the animals being tracked can also be much smaller, resulting in the challenge of tracking tiny objects within a video.

We aim to focus on this class of tracking problems, where we need to track multiple objects that appear small within the video. Specifically, we consider objects that are fewer than 30 by 30 pixels and are moving with respect to their background. In common applications, such as the vehicle and wildlife tracking examples mentioned above, the objects are not engineered to stand out against the background. This is a common feature of real-world tracking problems: the objects we are interested in often do not naturally stand out. As such, we will assume that the contract between the moving objects and their background is small. That is to say, we will focus on targets with an average luminance of less than 0.2. The luminance of an object is calculated by doubling the object's size so that its region is equal parts background and foreground, then calculating the standard deviation for each object.

Furthermore, as also seen in the earlier examples, we are often interested in tracking more than one object in a scene, often many of them. The problem we are considering is thus that of tracking many objects in video footage, where each object is only represented by a few pixels with relatively low contrast compared to the background.

To constrain our problem, we will specifically consider scenarios where the camera is semi-stationary. In the traffic example, this refers to the satellite view being that of a fixed eye in the sky, which is typically the case of the time scales under consideration. In the wildlife monitoring example, this would mean that the drone would be hovering in a position lock. These assumptions are not unreasonable and cover an important set of object-tracking scenarios. However, note that the camera is not fixed in the scenarios and that small position perturbations are likely to occur. As our example scenarios, and common applications in which these tracking problems manifest, rely on aerial camera platforms, we will assume that these perturbations will be present and that we will need to tailor our solution accordingly. In addition, the background against which the objects (cars, animals, etc.) move is not technically fixed, even if the camera is absolutely stationary. For example, clouds may cast moving shadows, wind may cause the movement of trees and shimmer on the water, or other changes may occur. We will refer to the effects of slight camera motion and this background changes as "pseudo-motion" of the

background. Our goal is to achieve object tracking that is robust to this pseudo-motion.

1.1 Problem Specification

The problem research that we will address in this thesis can thus be summarized as: Tracking of multiple moving targets from semi-stationary video footage under the following constraints:

- Targets are moving with respect to the background.
- Targets which are less than 30×30 pixels.
- Targets have an average luminance of less than 0.2.
- There may be pseudo-motion from an unfixed camera, limited such that it affects less than 20% of the total pixels.
- There may be pseudo-motion from objects within the scene.

We will present a general solution to this problem. We will describe our proposed algorithm, dubbed Spot. We will then discuss a methodology to select appropriate algorithm parameter settings and evaluate its performance. Specifically, we will use the two examples of motivation, traffic monitoring, and baboon monitoring as evaluation scenarios from vastly different application domains. However, the proposed approach is generic and not constrained to only these examples.

1.2 Related Work

The primary space we will consider for algorithms that fit the criteria presented in Section 1.1 is vehicle tracking from satellite footage, which is more prevalent than applications in wildlife. As both examples we will examine fit the above criteria, these algorithms should largely be transferable between both use cases.

Yin et al. [22] provide a recent overview of the space, which we can use for comparison. Throughout the remainder of this write-up, we will examine four algorithms as they compete most closely with Spot based on the average F1 scores provided by Yin et al. [22].

The first of these algorithms is Adaptive Gaussian Mixture Model (AGMM) [10]. AGMM extends existing methods for using GMMs for modeling pixels and improving upon the process of background subtraction. It does this by learning variations in the background. This compensates for wind, shimmering water, or other environment-based pseudo-motion. AGMM improves upon previous work with GMMs by using multiple learning rates to better account for scenarios where static objects are placed in a scene. As with the previous work on GMMs, AGMM expects a static camera, the movement from a satellite or drone may produce extra noise during detection. AGMM may also struggle with the low-contrast nature of the videos as laid out in Section 1.1.

Next, we consider ClusterNet [13], which uses a two-stage algorithm to look for motion. The first is ClusterNet, an algorithm that produces regions of interest based on motion clusters. These regions of interest are then fed into FoveaNet, which is a consensus method for determining whether the region of interest to determine if a vehicle or vehicles are present. The output of FoveaNet is the vehicles within the cluster. Likely, FoveaNet's focus on vehicles will not translate well to different spaces, such as our wildlife tracking problem.

Third is D&T presented by Ao et al. [2]. D&T, much like the algorithm we will present in Chapter 2, D&T leverages a Bayesian Filter, specifically, a Kalman Filter. It assigns each vehicle to be tracked, a Kalman filter at the start. Initially, accelerations are assumed to be zero and corrected later. The algorithm makes assumptions about the kind of movement expected as a step of its Kalman Filter. Then hypothesis of the next step is associated with a track. These associations are then used to update the overall Kalman Filter. As this algorithm makes assumptions about movement, particularly in the prediction step of the Kalman Filter, we expect it to perform more poorly on targets that do not match the moving vehicle assumptions.

The final like-algorithm we will consider is Motion Model Baseline (MMB) presented by Yin et al. [22]. MMB uses the "spatial-temporal information" encoded in videos while

"suppressing false alarms in the background" [22]. MMB assumes the background is mainly unchanging but may be noisy. MMB makes some assumptions about the shape to filter out the noise: the target object "has a regular geometrical pattern, texture shape, and motion models," and noise usually does not [22]. This assumption does not necessarily hold for soft, bodied objects that may not have regular geometrical patterns. Once it has the background removed, it can generate regions of interest. These regions may be incomplete. It addresses this by leveraging a model of the background. Finally, it makes some assumptions about the trajectory being "continuous and regular" [22], which is likely invalid for more sporadic animals.

The next chapter will formally introduce our contribution to the space, a generic moving-object detection algorithm called Spot. In later chapters, we will evaluate Spot against the vehicle tracking problem other like-algorithms presented here aim to solve and the baboon tracking problem, which also fits the constraints that Spot was designed to solve as listed in Section 1.1.

Chapter 2

Spot Algorithm

The previous chapter discussed the general moving object tracking problem and its many applications. It further provided two representative example applications we will evaluate Spot against, satellite-based vehicle tracking and drone-based wildlife tracking, specifically baboons. These two examples were chosen because they fit the constraints in Section 1.1. That is to say, they are domains where the subjects of interest are small with respect to the frame, the subjects move with respect to the background, and they are low-contrast. This chapter proposes a solution to the moving-object tracking problem called Spot.

2.1 Assumptions

To begin our construction of Spots, we make the observation, as laid out in our constraints, that objects tracked from the camera footage have a reduced number of pixels available–an average of 30×30 pixels–and a low luminance–less than 0.2. This makes tracking them incredibly challenging. Therefore, we cannot expect methods that only look at a single frame to identify moving objects consistently. Instead, we shall focus on methods that exploit both temporal locality–the relationships between consecutive frames–and spatial locality–the relationship between nearby pixels. To do this, we introduce our first assumption, (1) The background scene being observed does not change drastically from frame to frame. Therefore, the changes between the current and previous frames constitute the foreground–our regions of interest. We can then

use this assumption to look for changes between frames by comparing pixel intensities.

Our following observation in the construction of Spot is that the camera moves with respect to the background introducing pseudo-motion. To correct this, we will aim to "fix" the camera position of previous frames to match the camera position of the current frame. To do so, we must make another assumption (2) the camera must be relatively stable so that when the frames are adjusted, few image artifacts are created. The more a frame moves, there is less of the image to compare against, and therefore the realignment is less accurate. Since the previously stated goal is to compare intensities, if the camera moves significantly, the data to compare against will not be present. Therefore, this assumption will hold.

Finally, we need a way to find the same objects between two frames to perform this alignment. Therefore, our final assumption is (3) the background of the video must be sufficiently featureful. The background must have well-distributed regions of sufficiently different contrast levels to select unique, common features between frames. This assumption will allow us to align the current video frame with previous frames, allowing us to compare them more easily. This assumption will hold almost any real-world dataset looking at a land-based environment.

To summarize, the three assumptions that Spot makes are:

1. The observed background scene does not change drastically from frame to frame. Therefore, the changes between the current and previous frames constitute the foreground, our region of interest.

2. The camera must be relatively stable to create few image artifacts when the frames are adjusted.

3. The background of the video must be sufficiently featureful. The background must have well-distributed regions of sufficiently different contrast levels to select unique, common features between frames.

2.2 Pipeline

In the remaining sections of this chapter, we will present the pseudocode outline of Spot. Algorithm 2 will demonstrate that it is pipelined in nature. The subsequent sections will discuss each of the components of that pipeline in more detail. Spot takes F as an input parameter. F represents the list of valid frames from a singular video. The output of Spot is represented by R, which is the set of moving objects discovered by Spot per frame.

To best approximate this pipeline, we consider each stage as a dimensionality reduction, taking an input of one dimensionality and reducing it to a new dimensionality.

Algorithm 2. Spot Pipeline

Input: F
Output: R

$H \leftarrow \emptyset$ ▷ Queue of historical frames
$R \leftarrow \emptyset$ ▷ Results of the moving object detection
$\Phi \leftarrow \emptyset$ ▷ Particle filters with one matching each detected region
for $F_t \in F$ **do**
  $P_t \leftarrow$ preprocess_frame(F_t, k) ▷ See Algorithm 3
  $A_t \leftarrow$ motion_detection(P_t, H) ▷ See Algorithm 4
  $R_t \leftarrow$ detect_blobs(A_t) ▷ See Algorithm 5
  $T_t \leftarrow$ calculate_transformation_matrix(P_t, P_{t-1}) ▷ See Algorithm 4.1.1
  $R_t, \Phi \leftarrow$ particle_filter(R_t, T_t, Φ) ▷ See Algorithm 6
  $R \leftarrow R \cup \{R_t\}$
end for

At the beginning of the algorithm, we initialize an empty queue to hold our list of historical frames, H. This will be used to be able to help us track motion later. We then create an empty set to hold our final results, R. The final initialization step is to create an empty set of Particle Filters, Φ, which will be used to track our objects over time. Then we loop over each frame from the video, performing each step of our pipeline.

After Spot has performed `preprocess_frame` and `motion_detection`, the following dimension reduction has been performed, where $\mathbb{Z}$ is the set of integers, $\mathbb{B}$ is the set of booleans, m represents the total number of pixels in the current frame, F_t, and c represents the total number

of color channels of the current frame, F_t.

$$\mathbb{Z}^{m \times c} \Rightarrow \mathbb{B}^m \tag{2.1}$$

This represents the reduction of a single frame down to a motion mask. Given this motion mask, in `detect_blobs`, we perform the following action, where r are the number of the blobs detected by blob detection.

$$\mathbb{B}^m \Rightarrow \mathbb{Z}^{r \times 2} \tag{2.2}$$

Finally, combining both of these steps, it is clear that Spot performs the following reduction.

$$\mathbb{Z}^{m \times c} \Rightarrow \mathbb{Z}^{r \times 2} \tag{2.3}$$

The pipeline and its respective dimensionality reductions are the primary contributions of this write-up. Within the next sections, we will discuss each stage in that pipeline and will note any additional contributions as they come up.

2.3 Preprocess the image

Algorithm 3. Preprocess Frame

Input: F_t
Output: P_t

$k_p \leftarrow$ kernel_from_configuration()

$P_t \leftarrow$ convert_to_grayscale(F_t)
$P_t \leftarrow P_t * k_p$

As indicated above, the combination of preprocessing and motion detection must transform the current frame into a motion mask where 0 represents no motion, and a value of 1

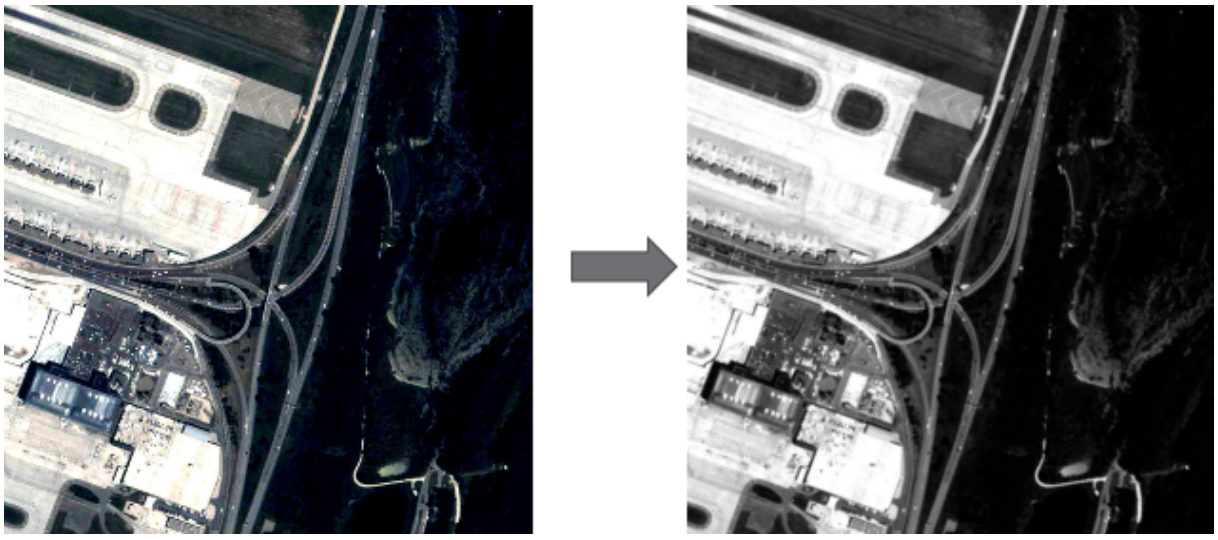


Figure 2.1. The results of Algorithm 3 when applied to one frame of a sample video

represents motion. As our motion detection portion, which will be further discussed in Section 2.4, of the pipeline requires $\mathbb{Z}^m$, and our input frame is $\mathbb{Z}^{m\times c}$, we must combine the results from motion detection across multiple color channels or choose a single color channel to run motion detection against.

There are many ways to approach this, we have experimentally determined that using a simple grayscale color transformation performs well, allowing us to go from our input frame dimension to the required motion-detection input frame dimension. We represent this transformation as $\mathbb{Z}^{m\times c} \Rightarrow \mathbb{Z}^m$

This still leaves some flaws in the image that are still necessary to correct, such as intensity differences caused by sensor imperfections. We blur the grayscaled frame by convolving a kernel with the grayscale frame. The kernel convolved with the frame is one of the parameters that can be controlled for the algorithm.

The details of these steps can be seen in Algorithm 3, where F_t is a single frame, k_p is the blur kernel pulled from configuration, and P_t is the preprocessed frame. The results of this process can be seen in Figure 2.1

2.4 Motion Detection

Algorithm 4. Spot Motion Detection

Input: P_t, H
Output: A_t

$h \leftarrow$ history_count_from_configuration()
$H' \leftarrow \emptyset$
$M \leftarrow \emptyset$
$Q \leftarrow \emptyset$

if count$(H) = h$ **then**
    H.pop()
end if
for $H_i \in H$ **do**
    $H_i', M_i \leftarrow$ register_history_frames(H_i, P_t) ▷ See Algorithm 4.1
    $H_t' \leftarrow H' \cup \{H_i\}$
    $M \leftarrow M \cup \{M_i\}$

    $Q \leftarrow$ quantize_frame(H_i', Q) ▷ See Algorithm 4.2
end for

H'.push(P_t)

$Q \leftarrow$ quantize_frame(P_t, Q)
$W_t \leftarrow$ generate_weights(Q) ▷ See Algorithm 4.3
$D_t \leftarrow$ history_of_dissimilarity(P_t, H') ▷ See Algorithm 4.4
$I_t \leftarrow$ intersect_frames(Q, H') ▷ See Algorithm 4.5.1
$U_t \leftarrow$ union_intersected_frames(I_t) ▷ See Algorithm 4.5.2
$A_t \leftarrow$ subtract_background(A_t) ▷ See Algorithm 4.6
$A_t \leftarrow$ moving_foreground(D_t, A_t, W_t) ▷ See Algorithm 4.7
$A_t \leftarrow$ apply_masks(A_t, M) ▷ See Algorithm 4.8

In this case, since we had the `preprocess_frame` step reduce the dimensionality of our input frame to be compatible with our motion detection model, we can have the motion detection perform directly on the output of the previous step.

We will leverage portions of the algorithm proposed by Ray and Chakraborty [14]. In this proposed algorithm, they implement a queue of historical frames. Using this queue, we can observe a single pixel over multiple frames, allowing us to better build a background

representation than a single frame can. An object moving through a pixel can then be filtered out, unlike a single frame. We can also track the number of times a pixel changes in the previous h frames; if the pixel has changed in all h historical frames, represent pseudo-motion from non-foreground motion as discussed in Chapter 1, such as shimmering water or blowing grass. The number of historical frames used, h is a parameter to the algorithm.

The details of this algorithm can be seen in Algorithm 4, where P_t is a preprocessed frame, H is a list of previous historical frames, and A_t is the moving foreground–the objects we wish to track, as opposed to U_t, which is the representation of the mostly static background.

The final result of this algorithm stage is a motion mask where 0 represents no motion, and 1 represents pixels that moved, the moving foreground. This converts the input frame from $\mathbb{Z}^m$ to a boolean mask in $\mathbb{B}^m$.

2.4.1 Register Historical Frames

Algorithm 4.1. Register Historical Frames

Input: H_i, P_i
Output: H_i', M_i

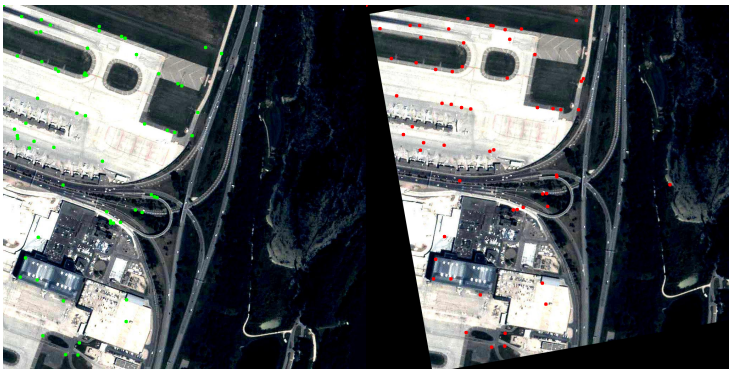
$T_i \leftarrow$ calculate_transformation_matrix(H_i, P_i) ▷ See Algorithm 4.1.1
$H_i', M_i \leftarrow$ transform_frame(H_i, T_i) ▷ See Algorithm 4.1.2

Given that we are comparing many historical frames against the current frame, the camera's motion can become a significant source of error.

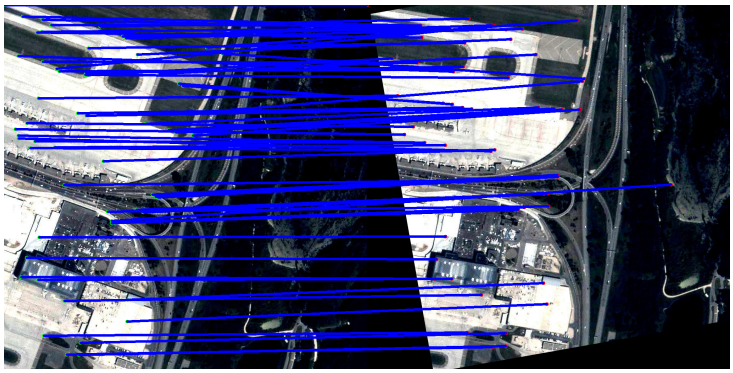
A possible solution to this is to attempt to transform each of the historical frames, H_i to be in the same coordinate space as the current grayscale frame, P_t which results in the warped frame H_i' which is in the same coordinate space as P_t.

2.4.1.1 Calculate Transformation Matrix

We can do this by looking at common features between the current frame and any given historical frame and then compute the necessary transformation matrix. We find the features using a FAST [17] feature detector to get the list of features. These features are pretty well



(a) Features labeled between two frames



(b) Features matched between two frames

Figure 2.2. (a) demonstrates locating the features using the FAST feature detector and SSC. (b) demonstrates the features being matched using the ORB feature matcher.

Algorithm 4.1.1. Calculate Transformation Matrix

Input: H_i, P_i
Output: T_i

$k_{H_i} \leftarrow$ FAST.detect_keypoints(H_i) [17]
$k_{P_i} \leftarrow$ FAST.detect_keypoints(P_i) [17]

$k_{H_i} \leftarrow$ ssc(k_{H_i}) [3]
$k_{P_i} \leftarrow$ ssc(k_{P_i}) [3]

$d_{H_i} \leftarrow$ ORB.generate_descriptions(H_i, k_{H_i}) [18]
$d_{P_i} \leftarrow$ ORB.generate_descriptions(P_i, k_{P_i}) [18]

$N \leftarrow$ match(d_{H_i}, d_{P_i}) ▷ Match features from H_i and P_i
Sort by distance $n_i \in N$

$T_i \leftarrow$ find_homography(k_{H_i}, k_{P_i})

---

distributed over the entire frame, but there are many more features than necessary. As such, we filter the points using Suppression via Square Covering (SSC) [3] to ensure that the points are well distributed throughout the frame. The process of matching features can be visualized in Figure 2.2a.

We now need to match the features between H_i and P_t. The first step is to create feature descriptions with ORB [18]. We then match the features, choosing the n best-matched features. The matched features can be seen in Figure 2.2b.

Given these features, we can use them to fit a transformation matrix that transforms the location of the features of H_i to P_t. In our implementation, we do this matching via Random Sample Consensus (RANSAC) to find the homography. RANSAC does this by randomly choosing two samples, then calculating a transformation matrix between those samples. It then checks to see how many other samples match this generated matrix. This process is repeated several times, choosing the matrix that fits the most samples.

2.4.1.2 Transform Frame

Algorithm 4.1.2. Transform Frame

Input: H_i, T_i
Output: H_i', M_i

$M \leftarrow$ ones_like(H_i) ▷ Matrix of ones same size as H_i

$H_i' \leftarrow$ warp_perspective(H_i, T_i)
$M_i \leftarrow$ warp_perspective(M, T_i)

With this transformation matrix, we leverage Algorithm 4.1.2 to warp H_i generating H_i' which is now in the same coordinate plane as the current frame, P_i. For use later in Algorithm 4.8, we also generate a mask that can be used to mask away the portion of the image lost due to this transformation.

Algorithm 4.2. Quantize Frame

Input: H_i', Q
Output: Q

$s \leftarrow$ quantization_factor_from_config()

$q_i \leftarrow \frac{H_i' \times s}{2^8 - 1}$
$Q \leftarrow Q \cup \{q_i\}$

2.4.2 Quantize Frame

To filter out changes in background lighting and other sensor errors, we introduce a way to control the algorithm's sensitivity. We do this by providing a threshold for change. When considering large amounts of data, this can be performance intensive. A more performant thresholding method is quantizing the frame and the historical frames and then comparing the quantized frames as seen in Algorithm 4.2. This is also performed by Ray and Chakraborty [14].

Quantization here compresses the image, so all pixel values are between 1 and s. This effectively thresholds what pixels are considered to be the same. Originally, Ray and Chakraborty [14] had compressed the values between 1 and 10. Our change here allows for increased sensitivity to contrast changes. Larger values of s result in lower threshold values and, therefore, are less sensitive to changes in color, allowing us to handle lower contrast tasks than the algorithm initially designed for. The parameter s needs to be supplied to the configuration. It expects a history frame, H_i, and the existing set of quantized frames, Q. It outputs a set of quantized frames, including the Q passed in. It required a quantization factor, s to come from the configuration.

2.4.3 Generate Weights

To reduce the impact of shimmering water or blowing grass, we must differentiate between pseudo-motion caused by the background and true motion caused by a moving object. Considering how often a particular pixel changes, we observe that pixels that change consistently likely do not represent a moving object. Instead, they represent something shimmering as an

Algorithm 4.3. Generate Weights

Input: P_t, Q
Output: W_t

$W_t \leftarrow$ zeros_like(P_t) ▷ A zero matrix the same size as P_t

for $Q_i \in Q$ for $i > 0$ **do**
    $W_i \leftarrow (|Q_i - Q_{i-1}| \leq 1)$
    $W_t \leftarrow W_t + W_i$
end for

object that is moving should induce change that moves through pixels, not stays at the same pixels [14].

Algorithm 4.3 is adapted from Ray and Chakraborty [14], which generates a weight value per pixel. The weights represent the number of times per-pixel motion was detected in the historical frames used to generate Q.

The weight matrix is used in combination with Algorithm 4.4 and Algorithm 4.6 to calculate the moving foreground in Algorithm 4.7.

2.4.4 Generate History of Dissimilarity

Algorithm 4.4. Generate History of Dissimilarity

Input: P_t, H'
Output: D_t

$D_t \leftarrow$ zeros_like(P_t) ▷ Matrix of zeros the same size as P_t

for $H'_i \in H$ for $i > 1$ **do**
    $M_i \leftarrow |Q_i - Q_{i-1}| \leq 1$
    $D_i \leftarrow |H'_i - H'_{i-1}|$
    $D_i[M_i] \leftarrow 0$
    $D_t \leftarrow D_t + D_i$
end for

$D_t \leftarrow D_t/n$ ▷ n is the number of frames in H' + 1

Algorithm 4.4 measures a pixel's change within the historical frames, $|H'_i - H'_{i-1}| \ \forall$ $H'_i \in H$. This is the average change of a pixel throughout the historical frames. This will later be

used in Algorithm 4.7 to compare against the moving foreground; foreground values with more change than the average change are likely to be true movement and not pseudo-movement.

2.4.5 Generation of Static Background

Between Algorithm 4.5.1 and Algorithm 4.5.2, the goal is to create an estimate of the static, non-moving background.

2.4.5.1 Intersect Frames

Algorithm 4.5.1. Intersect Frames

Input: Q, H'
Output: I_t

$I_t \leftarrow \emptyset$
$g \leftarrow \emptyset$

for $i, j = i+1 | i \forall i <= n$ **do**
    $g \leftarrow g \cup \{(i, j)\}$
end for

for $i, j \in g$ **do**
    $M_{ij} \leftarrow |Q_i - Q_j| \leq 1$
    $I_{ij} \leftarrow H'_i[M_{ij}]$
    $I_t \leftarrow I_t \cup I_{ij}$
end for

The goal of the first step is to remove pixels that are likely to have changed between the frames due to true motion caused by a moving object or pseudo-motion caused by environmental factors. Each pair of two frames are intersected by comparing each pixel. If the pixels are the same, the value is copied over to the result; if the value is different, then the intensity in the result is 0 [14]. This yields $h-1$ intersected frames for h historical frames, where h is the number of historical frames defined as a parameter to the algorithm discussed earlier. Each of these $h-1$ intersected frames represents the static portions of the background between two consecutive historical frames.

2.4.5.2 Union Intersected Frames

Algorithm 4.5.2. Union Intersected Frames

Input: I_t
Output: U_t

$U_t \leftarrow$ zeros_like(I_0)

for $I_i \in I_t$ **do**
    $U_t[U_t = 0] \leftarrow I_i[U_t = 0]$
end for

Once we intersect each pair of h frames, the $h-1$ intersected frames are combined by a union operation [14]. The goal of the union operation is to fill the gaps created by the intersection operation with the actual background. Starting with an empty matrix U_t with all intensities 0, we can begin by looping over each $I_i \in I_t$. Since the intersections have gaps, the union of these frames should fill them. We should be able to estimate the entire background better for each iteration. Since the background remains static, each I_i should have the same intensity for portions of the background observed within a margin of error as defined by the quantization step. After all, iterations are complete, we have an estimate of the static background. The output is the union, U_t. See Algorithm 4.5.2.

2.4.6 Subtract Background

Algorithm 4.6. Subtract Background

Input: P_i, U, W
Output: A_i

$P'_t \leftarrow$ zero_weights(P_t, W_t) ▷ See Algorithm 4.6.1
$U'_t \leftarrow$ zero_weights(U_t, W_t)

$A_t \leftarrow |P'_t - U'_t|$

This algorithm extracts the foreground from the current frame. It uses the estimated static background, U_t, from Algorithm 4.5.2 and the weights, W_t, representing the number of times a pixel changes in the previous historical frames as generated in Algorithm 4.3. It first zeros out

all portions of the frame that have moved in every historical frame to address pseudo-motion from the environment. It then subtracts the estimated background from the current frame leaving only the foreground pixels. This algorithm is adopted from Ray and Chakraborty [14]. See Algorithm 4.6.

2.4.6.1 Zero Weights

Algorithm 4.6.1. Zero Weights

Input: Z_t, W_t
Output: Z'_t

$h \leftarrow$ history_from_configuration()
$Z'_t \leftarrow Z_t$
$Z'[W_t \geq h-1] = 0$

Algorithm 4.6.1 portion of the method is directly adapted from [14]. It requires h, the number of historical frames pulled from the configuration.

2.4.7 Compute Moving Foreground

Our goal now is to generate a candidate moving foreground mask. We will leverage the weights matrix, W, representing the number of times a pixel has changed in the previous historical frames. We will also leverage the history of dissimilarity, D_t, the average amount of change per pixel frame to frame. The last portion will be the foreground, A_t. We produce our candidate moving foreground mask by combining these three as defined by Ray and Chakraborty [14], as seen in Algorithm 4.7. Figure 2.3 visually represents this combination step.

To combine these three matrices, we first categorize how often a pixel changes based on W. We do not consider pixels that change many times within the historical frames. This is because these represent environmental pseudo-motion. We next categorize how different the foreground is compared to the background. We will only consider foreground values that represent more change than average. This is done to reduce misclassification.

Algorithm 4.7. Compute Moving Foreground

Input: D_t, A_t, W_t

Output: A'_t

$h \leftarrow$ history_from_configuration()

$x \leftarrow \frac{255}{3}$

$W_{t,low} \leftarrow W_t \leq \lfloor \frac{h-1}{3} \rfloor$

$W_{t,medium} \leftarrow \lfloor \frac{h-1}{3} \rfloor < W_t \wedge W_t < h-1$

$W_{t,levels} \leftarrow W_{t,low} + 2W_{t,medium}$

$A_{t,low} \leftarrow A_t \leq \lfloor x \rfloor$

$A_{t,medium} \leftarrow \lfloor x \rfloor < A_t \vee A_t < \lfloor 2x \rfloor$

$A_{t,high} \leftarrow A_t \geq \lfloor 2x \rfloor$

$A_{t,levels} \leftarrow A_{t,low} + 2A_{t,medium} + 3A_{t,high}$

$D_{t,low} \leftarrow D_t \leq \lfloor x \rfloor$

$D_{t,medium} \leftarrow \lfloor x \rfloor < D_t \vee D_t < \lfloor 2x \rfloor$

$D_{t,high} \leftarrow D_t \geq \lfloor 2x \rfloor$

$D_{t,levels} \leftarrow D_{t,low} + 2D_{t,medium} + 3D_{t,high}$

$A'_t \leftarrow W_{t,levels} = 2 \wedge A_{t,levels} \geq D_{t,levels}$

$A'_t \leftarrow A'_t \vee (W_{t,levels} = 1 \wedge A_{t,levels} > D_{t,levels})$

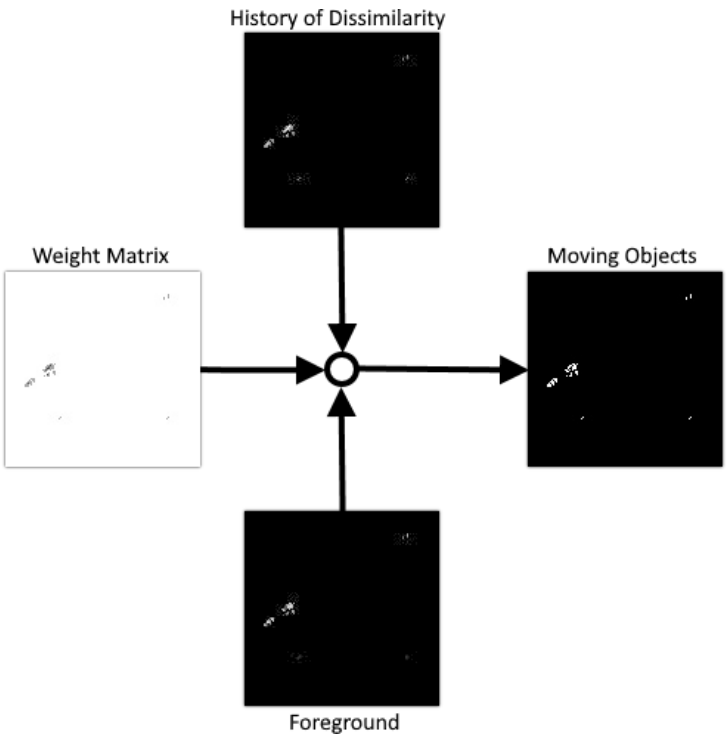


Figure 2.3. The unfiltered moving object mask is the combination of the results from Algorithm 4.3, Algorithm 4.4, and Algorithm 4.6

2.4.8 Apply Masks

Algorithm 4.8. Apply Masks

Input: A_t, M
Output: A'_t

$A'_t \leftarrow A_t$
for $M_i \in M$ **do**
    $A'_t \leftarrow A'_t \times M_i$
end for

We have generated a noisy mask of the moving foreground at this point. This mask has inappropriately marked regions with no data in more than one historical frame as part of the moving foreground. This needs to be corrected.

We can assume that the outer edges do not contain relevant information as the camera moves. Using this assumption, we compute the "dead" regions caused by camera motion, i.e. the areas that are cropped out after correcting the camera motion. These motions are then masked

away in Algorithm 4.8 using masks generated in Algorithm 4.1. This addition for correcting the camera motion is another contribution proposed in this write-up.

2.4.9 Noise Reduction/Connecting the Blobs

Algorithm 4.9. Noise Reduction

Input: A_t
Output: A'_t

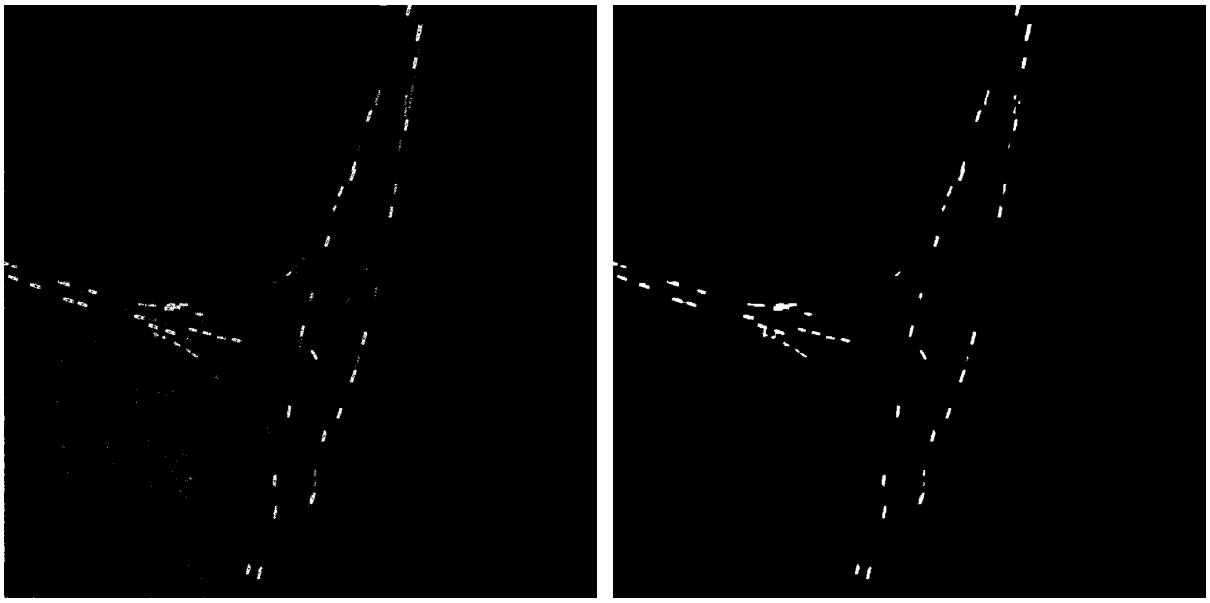
$\varepsilon \leftarrow \varepsilon$_from_config()
$m \leftarrow$ min_samples_from_config()
$k_o \leftarrow$ dbscan_kernel_from_config()

$A'_t \leftarrow \text{dbscan}(A_t, \varepsilon, s)$ [7]
$A'_t \leftarrow A'_t \oplus k_o$ ▷ $\oplus$ refers to the dilate operation
$A'_t \leftarrow A'_t \ominus k_o$ ▷ $\ominus$ refers to the erode operation

To reduce the noise seen in Figure 2.4, we first run DBScan [7] against the results of Algorithm 4.8. This noise differs from the noise discussed in Section 2.4.7. The noise Section 4.7 is primarily focused on is caused by pseudo-motion. Conversely, the noise we focus on here is caused by subtle differences introduced by Section 2.4.1. We can filter out this noise by observing that the pixels of our objects will move as a dense group. DBScan allows Spot to determine which motion-candidate pixels are part of a dense group; the denser the group, the higher the probability that each pixel is true motion.

This process removes most of the paint-splatter noise seen in Figure 2.4a, but may leave these groups potentially unconnected. We have observed this to be the case because a single object, particularly a soft-bodied one, may only have a few pixels connecting two dense groups depending on how it moves. To correct this, we perform a dilation operation with kernel k_o. This connects the groups by expanding each motion pixel until they overlap. Assuming that the objects to be tracked are sufficiently separated, this does not impact accuracy. When this assumption does not hold, it may result in connecting more than one distinct object.

Finally, the now connected regions are larger than the motion regions discovered in Algorithm 4.7. To correct this, it is necessary to perform the opposite operation, erode. Using



(a) Moving Foreground **(b)** Reduced Noise

Figure 2.4. As seen in (a), the moving foreground results have paint-splatter-like noise. (b) applies the DBScan [7] to remove the paint-splatter like noise.

the same kernel, k_o, we reduce the connected regions to the original size. The final result can be seen in Figure 2.4b.

At the end of this stage, we have completed the motion detection task by computing the moving foreground mask. In the next section, we will convert this mask to a list of regions of interest.

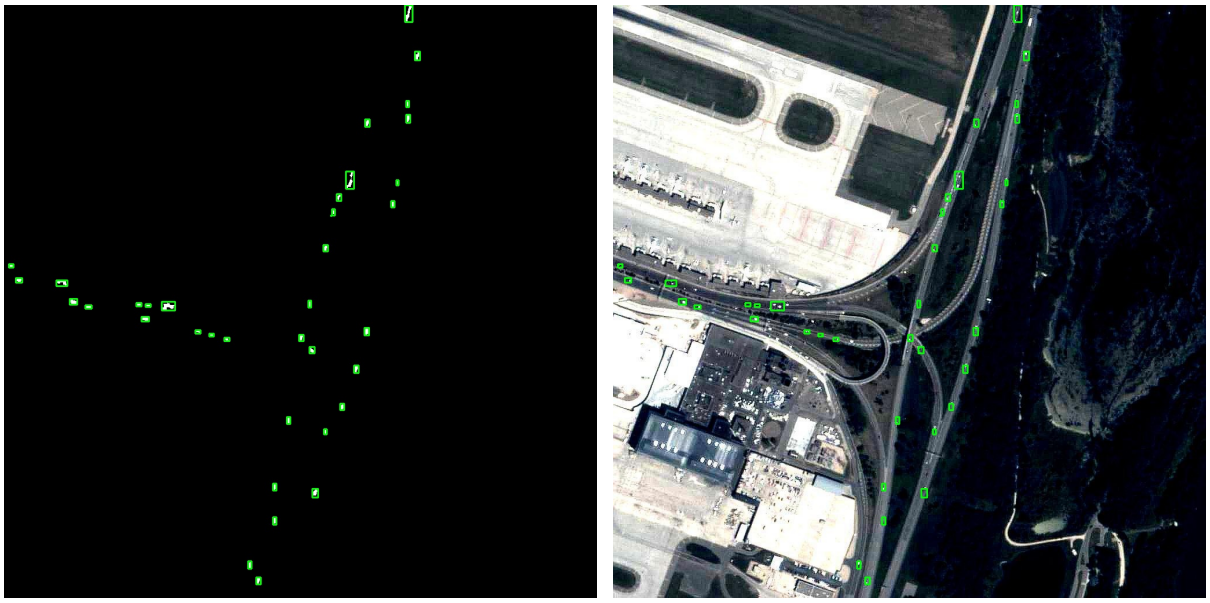
2.5 Region Detection

Algorithm 5. Region Detection

Input: A'_t
Output: R_t

$C_t \leftarrow$ find_contours(A'_t)[19]
$R_t \leftarrow$ bounding_rect(C_t) ▷ calculates up-right bounding rectangle

At the end of Section 2.4, we now have a mask representing the moving foreground. To convert this mask into regions of interest, we must now detect each of the blobs within this mask.



(a) Region Detection on Reduced Noise Frame **(b)** Regions on Original Color Frame

Figure 2.5. Region detection is performed on the reduced noise moving foreground from Algorithm 2.4 (a). These regions can then be used on the original frame to bound the moving objects (b).

To do this Algorithm 5 first finds the contours of the connected regions leveraging an algorithm defined by Suzuki and Abe [19].

To define a bounding box, this can then detect the top-left and bottom-right contour points. These bounding regions represent the objects moving within this frame compared to the previous one. Figure 2.5 visualizes these regions of interest by drawing bounding boxes around them.

2.6 Particle Filter

The previous step produced a list of regions that changed between the current and previous frames. To answer the moving-object tracking problem, it is necessary to match moving regions from one frame to the next; otherwise, there is no relationship between a region in one frame and the same region in another. To do this, we propose using some Bayesian Filter; in this case, we opted for a particle filter to track each region. This would mean assigning each region to track its

Algorithm 6. Particle Filter

Input: R_t, T_t, Φ
Output: R_t, Φ

$\Gamma_t \leftarrow \emptyset$

for $\Phi_i \in \Phi$ **do**
  $\Phi_i \leftarrow$ particle_filter_predict(Φ_i) ▷ See Algorithm 6.1
  $\Phi_i \leftarrow$ particle_filter_transform(Φ_i, T_t) ▷ See Algorithm 6.2
  $\Phi_i \leftarrow$ particle_filter_update(Φ_i) ▷ See Algorithm 6.3
  $\Phi_i \leftarrow$ particle_filter_resample(Φ_i) ▷ See Algorithm 6.4

  for $r_{t,i,j} \in R_t$ **do**
    $\gamma_{t,i,j} \leftarrow$ particle_filter_calculate_probability($\Phi_i, r_{t,i,j}$)
    $\Gamma_t \leftarrow \Gamma_t \cup \{\gamma_{t,i,j}\}$
  end for
end for

$\Gamma_t \leftarrow$ reverse_sort(Γ_t) ▷ Sort the probabilities

$R'_{t,\text{used}} \leftarrow \emptyset$
for $\gamma_{t,i} \in \Gamma_t$ **do**
  if $\gamma_{t,i} > 0$ **then**
    $R'_{t,\text{used}} \leftarrow R'_{t,\text{used}} \cup \{r_{t,i}\}$ ▷ Get the list of used regions
  end if
end for
$R'_{t,\text{unused}} \leftarrow R_t - R_{t,\text{used}}$
$\Phi \leftarrow \Phi \cup \{\Phi_{t,i,\text{unused}} | r_{t,\text{unused},i} \in R'_{t,\text{unused}}\}$ ▷ Create a new particle filter for each unused region

own particle filter. A particle filter can accomplish the task of tracking a region frame-to-frame by creating many ”particles” representing the object's current state–the location and size of the bounding region. Each of these particles is assigned a likelihood. Between observations, we can perturb these particles to produce a probability map of where any object is likely to be. We can then leverage this probability map to decide which observation this frame will likely be the same region.

After all particle filters have been associated with a region, a new filter is created for each newly detected region, the specifics of this can be seen in Algorithm 6.

2.6.1 Predict

The first portion of the particle filter we will discuss is the prediction step. This step is common to all Bayesian filters and allows it to compensate for changes in the observed state based on probabilistic expectations of how those changes occur.

Since Spot is generic, it is impossible to know in what direction a particular object will likely move to the next frame. For particular objects, such as vehicles, assumptions may be made based on historical velocity, but other objects, such as those observed in wildlife tracking, may change direction erratically.

Instead of assuming a particular kind of movement, in Algorithm 6.1, we assume the object will be near its previous location. For each particle, we also assume that the amount it can move is a function of the space it takes up in the frame, which we measure by taking the diagonal of the bounding region. This decision is made to bound the amount of movement we expect. In this case, we assume that a long train may move faster than a smaller animal. We then sample from a normal distribution to decide how much to move the particle. This helps account for some of the randomnesses of the objects being tracked since it does not assume how the object will move, just the amount of motion.

Next, we decide the direction in which the object will travel. This is done assuming any possible direction is equally as likely. As stated before, better assumptions may be made based

Algorithm 6.1. Particle Filter Predict

Input: Φ_i

Output: Φ_i

for $\phi_{i,j} \in \Phi_i$ **do**

$x_{i,j,\text{top\_left}}, y_{i,j,\text{top\_left}} \leftarrow \phi_{i,j,\text{top\_left}}$

$x_{i,j,\text{bottom\_right}}, y_{i,j,\text{bottom\_right}} \leftarrow \phi_{i,j,\text{bottom\_right}}$

$w_{i,j} \leftarrow x_{i,j,\text{bottom\_right}} - x_{i,j,\text{top\_left}}$

$h_{i,j} \leftarrow y_{i,j,\text{bottom\_right}} - y_{i,j,\text{top\_left}}$

$d_{i,j} \leftarrow \sqrt{w_{i,j}^2 + h_{i,j}^2}$

$d'_{i,j} \leftarrow \mathcal{N}(0, 0.01) \times d_{i,j}$ ▷ Sample from a normal distribution with $\mu = 0, \sigma = 0.01$

$r_{i,j} \leftarrow \mathcal{U}(0,1) \times 2\pi$ ▷ Sample from a uniform distribution over the range $[0,1]$

$\Delta_{i,j,x} \leftarrow \text{round}(d'_{i,j} \sin(r_{i,j}))$

$\Delta_{i,j,y} \leftarrow \text{round}(d'_{i,j} \cos(r_{i,j}))$

$\Delta_{i,j,\text{top\_left},x} \leftarrow \text{round}(\mathcal{U}(0,1) - 0.5)$

$\Delta_{i,j,\text{top\_left},y} \leftarrow \text{round}(\mathcal{U}(0,1) - 0.5)$

$\Delta_{i,j,\text{bottom\_right},x} \leftarrow \text{round}(\mathcal{U}(0,1) - 0.5)$

$\Delta_{i,j,\text{bottom\_right},y} \leftarrow \text{round}(\mathcal{U}(0,1) - 0.5)$

if $\Delta_{i,j,\text{top\_left},x} \geq \Delta_{i,j,\text{bottom\_right},x}$ **then**

$\Delta_{i,j,\text{bottom\_right},x} \leftarrow \Delta_{i,j,\text{top\_left},x}$ ▷ Correct the case where the left point passes the right

end if

if $\Delta_{i,j,\text{top\_left},y} \geq \Delta_{i,j,\text{bottom\_right},y}$ **then**

$\Delta_{i,j,\text{bottom\_right},y} \leftarrow \Delta_{i,j,\text{top\_left},y}$ ▷ Correct the case where the top point passes the bottom

end if

$\phi_{i,j,\text{top\_left},x} \leftarrow x_{i,j,\text{top\_left}} + \Delta_{i,j,x} + \Delta_{i,j,\text{top\_left},x}$

$\phi_{i,j,y,\text{top\_left}} \leftarrow y_{i,j,\text{top\_left}} + \Delta_{i,j,y} + \Delta_{ij,\text{top\_left},y}$

$\phi_{i,j,\text{bottom\_right},x} \leftarrow x_{i,j,\text{bottom\_right}} + \Delta_{i,j,x} + \Delta_{i,j,\text{bottom\_right},x}$

$\phi_{i,j,\text{bottom\_right},y} \leftarrow y_{i,j,\text{bottom\_right}} + \Delta_{i,j,y} + \Delta_{i,j,\text{bottom\_right},y}$

$\phi_{i,j,\text{observed}} \leftarrow \text{False}$

end for

on apriori knowledge of what is being tracked, but we do not make those assumptions for Spot.

Finally, we decide how we will transform the shape of the bounding region. This is done to account for the possibility that an object may change shape from frame to frame. Again, since we cannot make assumptions on how this may happen, we assume any shift in change is possible.

2.6.2 Transform

Algorithm 6.2. Particle Filter Transform

Input: Φ_i, T_t
Output: Φ_i

for $\phi_{i,j} \in \Phi_i$ **do**

$x_{i,j,\text{top\_left}}, y_{i,j,\text{top\_left}} \leftarrow \phi_{i,j,\text{top\_left}}$

$x_{i,j,\text{bottom\_right}}, y_{i,j,\text{bottom\_right}} \leftarrow \phi_{i,j,\text{bottom\_right}}$

$$p_{i,j,\text{top\_left}} \leftarrow \begin{bmatrix} x_{i,j,\text{top\_left}} \\ y_{i,j,\text{top\_left}} \\ 1 \end{bmatrix}$$

$$p_{i,j,\text{bottom\_right}} \leftarrow \begin{bmatrix} x_{i,j,\text{bottom\_right}} \\ y_{i,j,\text{bottom\_right}} \\ 1 \end{bmatrix}$$

$p'_{i,j,\text{top\_left}} \leftarrow \text{round}(T_t p_{\text{top\_left}})$

$p'_{i,j,\text{bottom\_right}} \leftarrow \text{round}(T_t p_{\text{bottom\_right}})$

$\phi_{i,j,\text{top\_left},x} \leftarrow p_{i,j,\text{top\_left},x}$

$\phi_{i,j,\text{top\_left},y} \leftarrow p_{i,j,\text{top\_left},y}$

$\phi_{i,j,\text{bottom\_right},x} \leftarrow p_{i,j,\text{bottom\_right},x}$

$\phi_{i,j,\text{bottom\_right},y} \leftarrow p_{i,j,\text{bottom\_right},y}$

end for

Since the camera may move from frame to frame, we must transform particles in the previous frame's coordinate space into the current frame's coordinate space. We do this by applying the transformation matrix from P_{t-1} to P_t calculated by Algorithm 4.1.1 to each of the particle filters and, therefore, each particle within a filter. We then transform the location of the top-left and bottom-right of the bounding box represented by each pixel, resulting in the particles being in the coordinate space from of P_t instead of P_{t-1} by the end of Algorithm 6.2.

2.6.3 Update

Algorithm 6.3. Particle Filter Update

Input: Φ_i
Output: Φ_i

for $\phi_{i,j} \in \Phi_i$ **do**
  $W \leftarrow \emptyset$

  for $R_{i,t} \in R_t$ **do**
    $w_{i,j,t} \leftarrow \text{iou}(\phi_{i,j,\text{region}}, R_{i,t})$ ▷ Calculates intersection over union between two regions
    $W \leftarrow W \cup \{w_{i,j,t}\}$
  end for

  $w_{i,j} \leftarrow \max(W)$

  if $w = 0$ **then**
    continue ▷ This particle does not intersect with a known region
  end if

  $\phi_{i,j,\text{weight}} \leftarrow \phi_{i,j,\text{weight}} w_{i,j}$
end for

This subsection once again details a portion of the classical Bayesian filter. It is responsible for considering an observation and updating the particles as a result. In our case, the observation is a single region of interest determined by Section 2.5.

To determine which region of interest corresponds to this particle filter, the Algorithm 6 leverages a weighted average of all of the particles within the filter. The region of interest most likely to relate to this particle filter is the one used as the observation.

Algorithm 6.3, then iterations over each particle and calculates the intersection over union (IOU) of the particle region with each region found in P_i. The largest IOU is then used as the new weight if it is a nonzero value. If the value is 0, we assume the object has been lost and do not move the particle since we have no observations for this particle.

Algorithm 6.4. Particle Filter Resample

Input: Φ_i
Output: Φ'_i

$p \leftarrow$ number_of_particles_from_config()
$\Phi'_i \leftarrow \emptyset$

$W_i \leftarrow \{w_{i,\text{region}} = 0 | \phi_{i,j,\text{region}} \forall \Phi_i\}$ ▷ For all unique regions in Φ_i

for $\phi_{i,j} \in \Phi_i$ **do**
    $w_{i,\text{region}} \leftarrow w_{i,\text{region}} + \phi_{i,j,\text{region}}$
end for

for $w_{\text{region}} \in W_i$ **do**
    $p_{i,\text{region}} \leftarrow \text{round}(p \frac{w_{i,\text{region}}}{\sum_{\text{region}} w_{i,\text{region}}})$

    if $\text{count}(\Phi'_i) + p_{i,\text{region}} > p$ **then**
        $p_{i,\text{region}} \leftarrow p - \text{count}(\Phi'_i)$
    end if

    if $p_{i,\text{region}} = 0$ **then**
      break
    end if

    $\Phi'_i \leftarrow \Phi'_i \cup \{\phi_{i,j} | j \forall [0, p_{i,\text{region}})\}$
end for

2.6.4 Resample

After an update, some particles have a much larger weight than others since the update step boosts the weight of particles representing the observation. Others still no longer represent objects worth tracking as they did not align with observations. To address this, Algorithm 6.4 relocates all particles.

It does this by calculating the total weight of the particles assigned to a particular region. Afterward, the particles within a particular particle filter are redistributed to better represent the high-weighted regions of interest, resetting the particle filter to a state ready to repeat the prediction and update steps. The total weight is then used to calculate the number of particles that should represent this region in the new set.

2.6.5 Calculate Probability of Region Match

Algorithm 6.5. Particle Filter Calculate Probability

Input: $\Phi_i, r_{t,i,j}$
Output: $\gamma_{t,i,j}$

$p \leftarrow$ number_of_particles_from_config()

$\gamma_{t,i,j} \leftarrow \sum_{\phi_{i,j} \in \Phi_i} \frac{1}{p} \text{iou}(\phi_{i,j,\text{region}}, r_{t,i,j})$

To decide which particle filter likely represents which region, $r_{t,i,j}$, a weighted sum of all of the IOUs is calculated as performed by Algorithm 6.5. A weighted sum allows the algorithm to combine the results of many particles and account for the fact that each particle may point to a unique region in the worst case.

This probability allows us to relate the particle filters and regions of interest over time. With this construction of the particle filter, we have a means of tracking objects provided by the region detection and motion tracking algorithms.

Table 2.1. The parameters of the algorithm

Parameter Name	Parameter Description
k_p	The kernel used in Algorithm 3 to blur the gray-scale input frame
h	The number of historical frames to keep in the queue as used by Algorithm 2, Algorithm 2, Algorithm 4.6.1, and Algorithm 4.7
s	The quantization factor used by Algorithm 4.2 to quantize each pixel of the frame
ε	The threshold distance for DBScan [7] as used in Algorithm 4.9
m	The minimum number of samples DBScan needs to consider a point a core point as used in Algorithm 4.9
k_o	The kernel used for dilating and eroding to reconnect the blobs after performing DBScan as used in Algorithm 4.9

2.7 Algorithm Parameters

With the explanation of the algorithm complete, Table 2.1 now contains a list of all the parameters used by the algorithm and a short description.

2.8 Acknowledgements

Chapter 2 contains material originally published in International Conference on Methods and Techniques in Behavioral Research (Measuring Behavior), October 2020. Crutchfield, Christopher L.; Sutton, Jake; Ngo, Anh; Zadorian, Emmanuel; Hourany, Gabrielle; Nelson, Dylan; Wang, Alvin, McHenry-Crutchfield, Fiona; Forster, Deborah; Strum, Shirley C.; Kastner, Ryan; Schurgers, Curt. The thesis author was the primary investigator and author of this paper.

Chapter 3

Parameter Selection and Metrics

In the previous chapter, we described Spot, a low-contrast, low-resolution moving object tracking algorithm. This chapter will describe how we select the configuration parameters and metrics to evaluate the algorithm and configuration parameters.

3.1 Parameter Optimization

When performing parameter optimization, there are often multiple metrics that we wish to optimize. Consider an abstract optimization problem where we may wish to optimize two variables, x and y. We can formulate this as a design space exploration problem. In such a problem, the inputs are referred to as knobs. These knobs are inputs to an algorithm or configurations of a design that we, as designers, can modify. Each knob has a range of values it is valid for. Figure 3.1 can be used to visualize such a design exploration problem for the two-dimensional case. Higher dimensional cases can be harder to visualize.

Here, the blue triangles represent samples of particular settings of the knobs. Each sample is then measured in the x and y variables, known as the output variables. When reviewing these metrics, we can see that each of these samples has trade-offs with one another. That said, some samples are more optimal than others. We have highlighted these more optimal points in red. They are referred to as Pareto optimal points.

Each Pareto optimal represents a point where variable x cannot be improved without

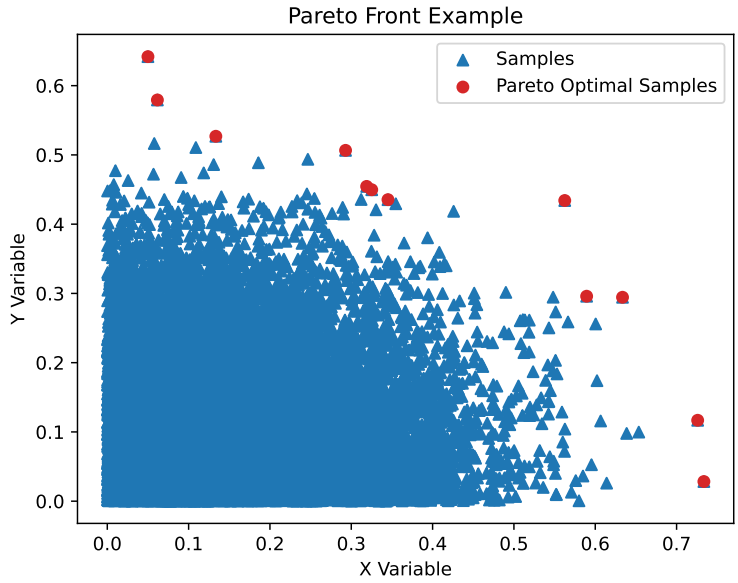


Figure 3.1. Example of a Pareto front on synthetic data.

negatively impacting variable *y* or vice versa. These red Pareto optimal points are referred to as the Pareto front for this design. Since each value not on the Pareto front is objectively worse than those on the Pareto front, the only values we are genuinely interested in are those on the Pareto front. The Pareto front represents a set of optimal points, meaning that no value within that set is more optimal than another value in that set. We can only choose a point from this set when we have a specific application with specific requirements.

In the case of Spot, we will aim to find the Pareto front for Spot parameter sets for related videos. This search for parameter sets will not be transferable between unrelated videos since, for example, wildlife and vehicles are very different categories of objects to be tracked.

To find the Pareto front for Spot, we need to define the design space for a video evaluated by Spot. We will define the input space as the configuration parameters in Table 2.1. The output space will be the accuracy of Spot in solving the moving object detection problem. Next, we will define the metrics to evaluate Spot's performance.

3.2 Base Metrics

To continue our construction of Spot's Pareto front, we must first define the metrics we intend to optimize. We will use standard metrics for moving object detection. We must have videos that already know where the moving objects are to leverage these standard metrics. This list of moving objects and their locations per frame will make up our ground truth.

To produce the metrics, we will begin with three base metrics. (1) True positive (TP) when an object detected is where we expect it to be. (2) False positive (FP) when we detect a moving object, but there is not one. (3) False negative (FN) when there is a moving object, but we fail to detect it. More specifically, to evaluate these, we will consider a true positive when the bounding box provided by the ground truth overlaps the bounding box detected. A false positive is when the bounding box detected does not overlap with any region the ground truth provides. And finally, a false negative is when the bounding boxes provided by the ground truth do not overlap with any detected region. These definitions were chosen to match Yin et al. [22] as interpreted by this author.

3.3 Derived Metrics

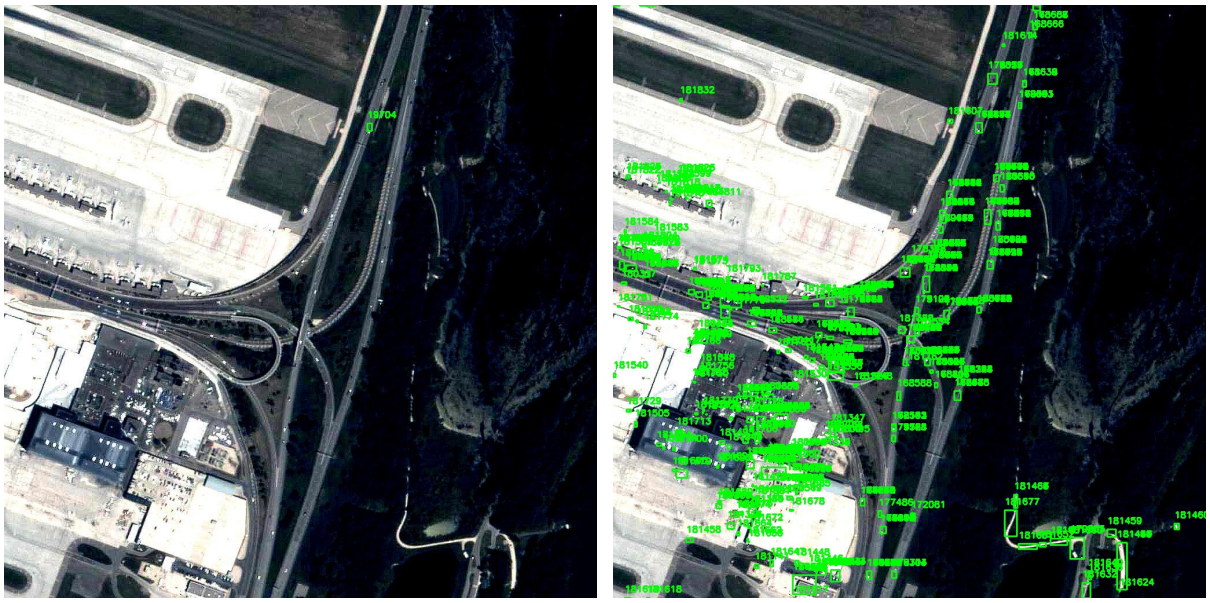
To evaluate the performance of Spot, the base metrics defined in Section 3.2 are combined to provide performance metrics. These metrics are defined below and correspond to metrics commonly used in this field of research.

3.3.1 Precision

Precision is a measure of the relationship between true positives to false positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.1}$$

This value goes to 1 if all regions detected by an algorithm are true positives. This means that everything detected is accurate but does not imply that all objects were detected. This metric



(a) Optimized Precision **(b)** Optimized Recall

Figure 3.2. Examples of optimized precision and optimized recall. Demonstrates that highly inaccurate algorithms can have either high precision or high recall.

can be manipulated by detecting a few regions and ensuring you have a few false regions.

3.3.2 Recall

Recall is a measure of the relationship between true positives and false negatives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.2}$$

This value goes to 1 if an algorithm detects all true regions. This means that all moving objects were detected but does not imply that objects not moving were not detected. This metric can be manipulated by detecting many regions, ensuring you hit as many regions as possible.

3.3.3 Precision vs. Recall

Precision and Recall are often trade-offs. For example, high precision, as in Figure 3.2a, can be achieved by accurately tracking only a few objects as long as it produces no false positives. An algorithm that maximizes this at the cost of recall will often miss many moving objects.

While, high recall, as in Figure 3.2b, can be achieved by ensuring that false negatives are avoided, even if the algorithm finds objects that are not moving. An algorithm that maximizes this at the cost of precision will find many moving objects that do not exist.

Thus, to truly evaluate Spot's performance, we must examine the two scores' trade-offs.

3.3.4 Precision-Recall (PR) Curve

A precision-recall curve measures the trade-off an algorithm has between precision and recall. Algorithms with a higher area under the curve (AUC), are considered better algorithms for a particular dataset. We refer to the AUC as the Average Precision-Recall (AP). These values can be visualized in Figure 3.3.

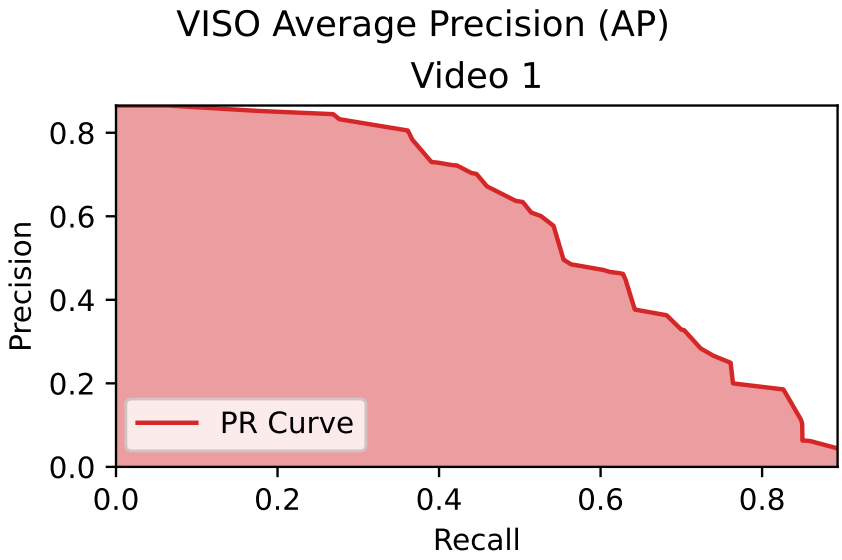


Figure 3.3. The red line represents a Precision-Recall curve while the shaded area represents the area under that curve, the Average Precision-Recall

3.3.5 F1-Score

Another method of approaching the trade-off is to consider what a balance of the two scores looks like. To capture the balance between precision and recall, we will also examine the harmonic mean of the metrics. This is known as the F1-score. It measures how accurate a particular algorithm is at finding only true positives. We can see this balance in Figure 3.4.

As the F1-score goes to 1, the algorithm has correctly found all the true positives and no false negatives or positives, meaning no objects were missed and nothing was incorrectly identified as a moving object.

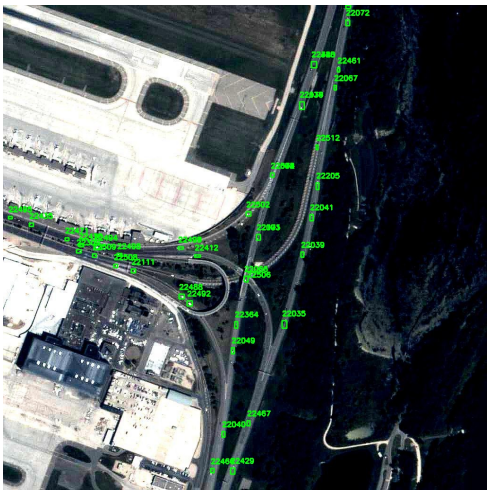


Figure 3.4. Example of optimized F1. By requiring the algorithm to optimize for both recall and precision, a balance that provides higher accuracy can be found.

$$\mathrm{F1} = \frac{2 \times \mathrm{Recall} \times \mathrm{Precision}}{\mathrm{Recall} + \mathrm{Precision}} \tag{3.3}$$

3.3.6 Mean Average-Precision (mAP)

To evaluate the performance of an algorithm across an entire dataset, we need a way of combining scores from individual videos within a dataset into a single score. As the AP score represents a measurement of an algorithm's effectiveness on a particular video, we will take the mean of all of the AP scores, referred to as the Mean Average Precision-Recall (mAP). Higher values for this metric mean that a particular algorithm is a better overall algorithm within the set of videos similar to those tested against.

Table 3.1. The parameters as defined in Table 2.1 and their respective value ranges.

Parameter Name	Minimum Value	Maximum Value	Step
k_p	3	7	1
h	3	8	1
s	5	9	2
ε	0.5	4.5	1
m	5	19	2
k_o	3	9	2 (must be odd)

3.4 Design Space Exploration

Before we can begin our discussion of exploring the Spot design space, we must first define the input and output spaces. This way, we can sample the input space and determine how it maps to the output space.

3.4.1 Considering the Design Space

To be exhaustive in our search, we would have to check every valid combination of parameters from the above algorithm, as seen in Table 2.1. Since we are sampling unique algorithm parameter combinations, considering them as continuous values do not make sense. Instead, we will approximate the continuous space by discretizing the possible values as seen in Table 3.1.

The total number of parameter sets described in Table 3.1 is 19,200 total parameter configurations and six input parameters.

We will focus on the recall and precision scores defined in Section 3.3 to evaluate Spot. Therefore, the Pareto front will compare precision and recall. For each parameter set, there will be one pair of values generated to evaluate algorithm performance.

Now that we have determined the metrics we wish to evaluate and how many possible evaluations there are, we need to discover which of those evaluations are on the Pareto front. One possible way is to explore the space exhaustively, calculating each value for the 19,200 com-

binations. Such an exploration is impractical since executing each input parameter combination may take more than ninety minutes. It requires fully running Spot on the input video, resulting in over three years of computing time per video. The following section will consider a better approach to the design space exploration problem.

3.4.2 Sherlock

Another approach to the design space exploration problem is to consider the problem to be a function f such that $f : \mathbb{R}^m \to \mathbb{R}^o$ where m represents the number of input knobs and o represents the number of metrics we are optimizing for. In this case, the input knobs are the parameters defined in Table 3.1. Therefore $m = 6$. The output space is the two metrics defined above, recall and precision, making $o = 2$. Considering Spot as a function f makes it possible to estimate the function using various existing design space exploration tools. One of those tools is Sherlock [8].

Sherlock [8] is a design space exploration algorithm with the goal of reducing the number of samples needed to approximate the Pareto front. Therefore, to use this algorithm, we must put our parameter search problem within the formulation of a design space exploration problem stated before. With the knowledge of the input and output spaces, Sherlock aims to approximate the function f and exploit its knowledge. It does so by alternating between exploration and exploitation. The exploration step allows Sherlock to improve its estimation of f. In contrast, the exploitation allows Sherlock to leverage the previously generated function to find the Pareto front and, therefore, focus its sampling around the Pareto front. Thus, it can calculate the ideal parameters to use on data for which the test video is representative.

We refer to each calculation of input space to output space mapping as a sample. Each of the samples requires fully executing Spot on a reference video. Since Sherlock aims at estimating f, Sherlock can focus its sampling either around unexplored areas or where it expects to find the Pareto front. This allows it to greatly reduce the number of samples needed to estimate the Pareto front.

The next question that needs to be answered is when to stop sampling values. Gautier et al. [8] demonstrated that in field programmable gate array (FPGA) applications, Sherlock converged in less than 30% of the space sampled. That would still be nearly 14 years of computing time for Spot, again impractical. Therefore, we examine the resultant Pareto front estimation after Sherlock discovers each new value. We assume convergence when the frequency of discoveries that improve the Pareto front is greatly reduced.

3.4.2.1 Objective Functions

The definition of the metrics has a major impact on the function mapping, f. The implementation of these metrics we will refer to as objective functions. The first two we will discuss will be labeled as problem objective functions as they contain false assumptions that must be corrected. As true positives, false positives, and false negatives form the basis for all other metrics, this section will focus on refining our definition of true positive, false positive, and false negative to address the aforementioned assumptions.

These faulty assumptions are brought to light here as part of an artifact of how Sherlock functions. As Sherlock does not know the problem it is trying to solve, it will blindly optimize its provided metrics, even if those metrics are faulty.

3.4.2.1.1 Problem Objective Function 1

The first objective function best matches the definitions defined in Yin et al. [22] and Section 3.2. Any overlap between a ground truth region and a detected region is a true positive. Intersection over Union (IOU)–a score that measures the overlap of two regions–is not considered part of the decision. That is to say, there is no minimum amount of overlap considered. There is no maximum size for a region detected; therefore, a detected region may bound the entirety of the frame. Such a large detected region may overlap more than one region in the ground truth, thus, allowing one detected region to result in multiple true positives.

When experimenting with this, it became clear that Sherlock would prioritize larger regions since they would artificially inflate the true positive score, improving precision and recall.

Table 3.2. Comparison of multiple different objective functions. A "-" means no explicit decision was made.

Objective Function	Max Width	Max Height	Allow Overlap	Recall	Precision	F1	AP
Problem Objective Function 1	-	-	-	1.0	0.86	0.92	0.86
Problem Objective Function 2	35/55	35/55	-	0.99	0.82	0.90	0.82
Final Objective Function	35/55	35/55	no	0.5	0.63	0.56	0.53

This can be seen in the Pareto front in Figure 3.5 where the graph corresponding to Problem Objective function 1 is focused at the right-most edge with a recall of near 1.

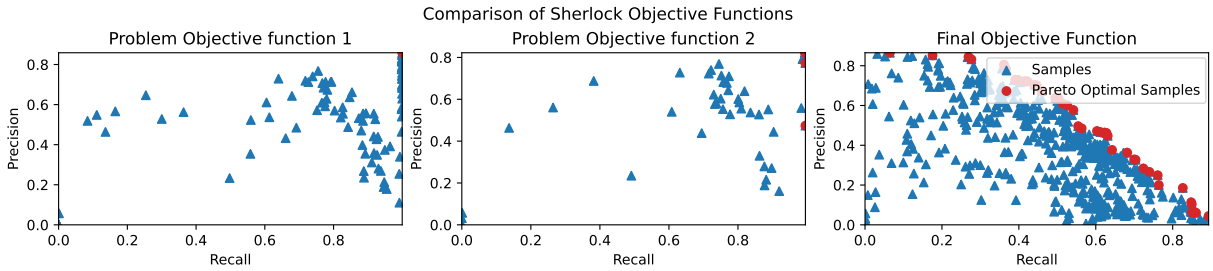


Figure 3.5. Pareto front of all three objective functions against a representative video.

3.4.2.1.2 Problem Objective Function 2

The second objective function corrects for the large regions seen with problem objective function 1. In one of the datasets we will consider later, the largest objects are below 35×35 pixels. On the other hand, the other dataset we will review has objects that are no larger than 55×55 pixels. Therefore, we have chosen to bound the regions to these sizes. Regions above this size are considered false positives. As seen in Table 3.2, this had the results of reducing the AP score. Also, in Figure 3.5, we can see that the Pareto front for Problem Objective function 2 no longer has large numbers of extremely high-valued recall. That said, values still skew towards a recall of 1.

Upon investigation, this was caused by an objective function that allowed multiple overlapping detection regions to be considered true positives. Before evaluating a frame, Spot does not know how many objects are expected to be within that frame. It may select multiple

regions for a single real object. Thus, if two of these detected regions overlap a single ground truth region, each is considered a true positive, potentially inflating our metrics and skewing Sherlock's search. This means that regardless of the number of errors, if an algorithm favored many, small area regions that overlap with ground truth, precision and recall could still tend toward 1.

3.4.2.1.3 Final Objective Function

To correct the issues presented in problem objective function 2, we introduced a rule that only allowed a single region with the largest IOU to be considered a true positive. All other regions are considered false positives. As seen in Table 3.2, this resulted in precision and recall avoiding the artifacts generated by Sherlock seen in the previous two objective functions. This is ideal because these artifacts do not represent true improvements in Spot's performance on the sample videos. This is the objective function used to evaluate the results presented in Chapter 5.

Chapter 4

Experiment Design

As stated previously, Spot is a generic moving object tracking algorithm. As such, we will evaluate it against two representative problems–tracking vehicles from satellites and tracking baboons from drone footage. Below we will discuss both problems and the data we aim to use to evaluate Spot as a solution to these problems.

4.1 Traffic Congestion Monitoring

As the resolution of satellites improves, they are being used more often for surveillance than before, particularly traffic density [12] and traffic flow [21] to aid in city planning. Information collected from satellites can be used to determine areas of high traffic or congestion.

These methods can be beneficial since they do not require sensors to be individually placed within locations that aim to be monitored. Alternatives may include road sensors and CCTV [12]. As these systems can be expensive, there have also been experiments to cover larger areas with aerial photography [21]. Ground-based systems can be expensive when placed over a large area. Likewise, tasked satellites can cover a much larger area than aerial photography, quicker and with less manpower, allowing satellites to cover approximately fifty-five square kilometers [22] within a single image. Since satellites do not require individual sensors or pilots to fly, evaluating regions with tasked satellite imagery can be cheaper.

Kopsiaftis and Karantzalos [12] use vehicle and traffic detection to estimate traffic density.

They perform this by dividing the scene into a "low-resolution grid". Each block then calculates the number of vehicles, yielding an estimated traffic density.

Yang et al. [21] use satellite footage to generate motion heat maps to estimate trajectory, allowing traffic flow tracking.

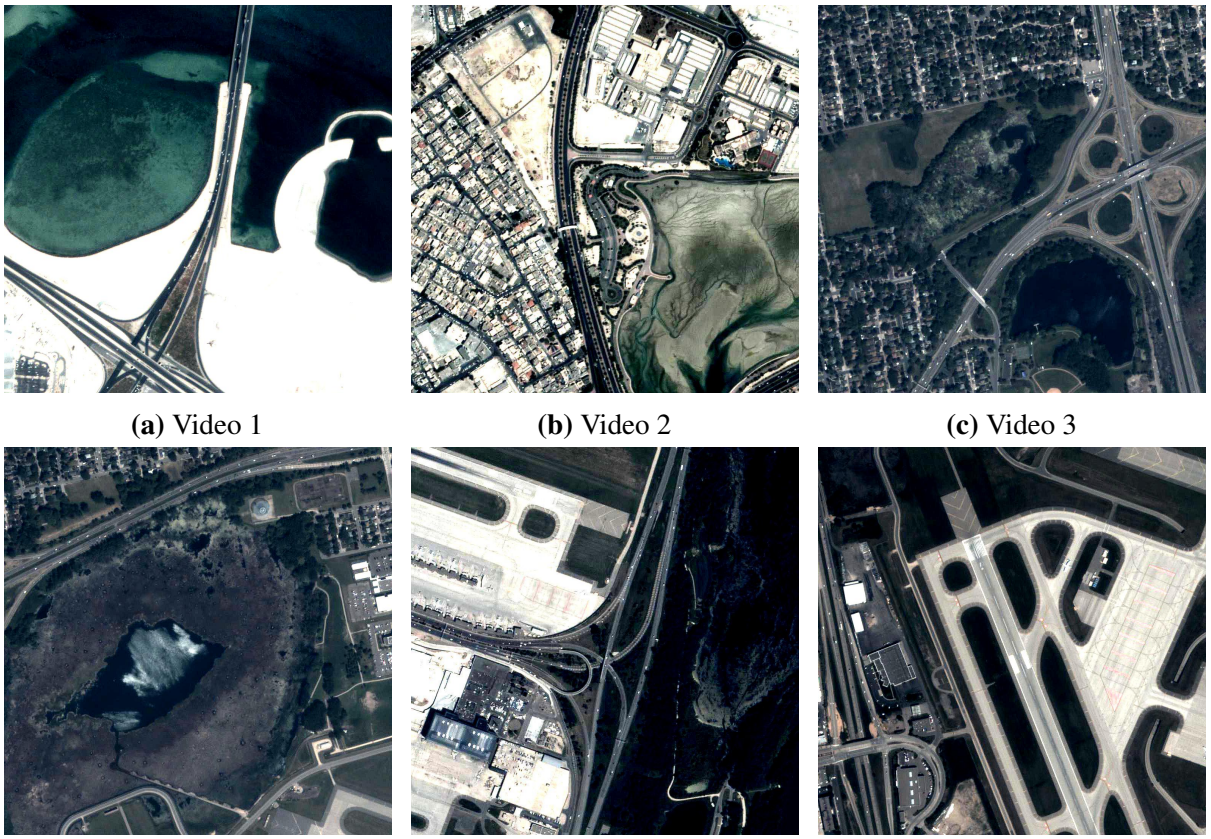
4.1.1 Dataset

The first moving object tracking problem we will examine uses satellite-based cameras to track moving vehicles, using **VI**deo **S**atellite **O**bjects (VISO) dataset [22]. This dataset aims to provide a baseline for research in the field.

Seven frames of the footage from the VISO dataset can be seen in Figure 4.1. These seven videos exhibit different potential complications a moving object detection problem may have. The videos aim to track moving vehicles within the images, including those in highway and cityscape conditions. While the satellites use relatively high-resolution RGB cameras, due to the height above the ground, the videos of an effective resolution of approximately 0.9m per pixel [22]. Given this low resolution, the typical size of vehicles to be tracked averages 6×7 pixels.

In addition to the low resolution of the target objects, it can be seen that the contrast many of the target objects have with the background is relatively low, having an average luminance value of 0.18, further complicating the tracking problem. Here, average luminance is calculated by (1) taking each ground truth region, (2) doubling its size so that it includes information from the background, (3) calculating the standard deviation of the pixel intensity within the expanded region, and (4) averaging the standard deviation across each detected region.

Other features of the videos make them hard for the moving object detection problem. The motion of the satellite, and thus by extension, the camera, results in the perceived motion of the objects, even if they are not moving in relation to the background. Other sources of this motion that does not represent moving objects are those triggered by elements in the scene, such as water, as can be seen in Figure 4.1a and Figure 4.1d. Throughout this write-up, we will refer



(a) Video 1 **(b)** Video 2 **(c)** Video 3

(d) Video 4 **(e)** Video 5 **(f)** Video 6

(g) Video 7

Figure 4.1. Videos represented from Yin et al. [22] in the VISO dataset.

to false motions induced by camera motion and false motion induced by background elements as pseudo-motion.

Another class of movement that adds additional complexity is that of objects that move faster than the target objects. As many algorithms depend on relative motion, too much change

from frame to frame can be perceived as multiple objects moving. For example, an object that may have too much motion frame to frame is a bird or a plane flying through the frame as it has faster motion than the objects we intend to track.

4.1.2 Video Selection

For this dataset, we aim to choose the same videos as selected by Yin et al. [22]. Unfortunately, it is not made clear what seven videos are chosen. Instead, we make assumptions based on the clues provided. Figure 8 of their paper provides screenshots for Video 1 and Video 7. Comparing these with the names of the files within the presented dataset, these videos are `car/003` and `car/009`. For the remainder of this document, we assume videos 2 through 6 are the five remaining video files between Video 1 and Video 7 in the files provided.

4.2 Baboon Tracking

Collective and distributed decision-making has long been a topic of interest in animal research since it is a complex process in many nonhuman animal species. Long-lived social mammals that interact within societies have much in common with humans. Within these particular societies, individuals and their connections within their social network critically impact group-level behavior. This is particularly true of nonhuman primates.

To understand group decisions and the context thereof, one would ideally be able to continuously (1) monitor identified individuals and their activities, (2) track the relational dynamics and social networks, (3) monitor group-level behavior and (4) monitor the environment.

To find and analyze the moment when a decision is made—for example, what direction a troop will head—it is necessary to track the entire baboons individually. The decision is a complex negotiation that can originate from both local dynamics and relational history between individuals. Since it is difficult to know all the variables involved in a decision, it is necessary to understand the moment a decision is made.

Currently, to understand the dynamics, researchers in the field take notes about obser-

vations. Unfortunately, these notes do not provide the full context, as a single researcher can only see a portion of the sleeping site. GPS radio collars have been used to augment the notes taken by field researchers. Still, they only allow for observing partial group membership and thus require attempting to fill in gaps left by having only partial results. Using drones would allow for the monitoring of individuals and small groups, but also the troop as a whole, in a way that is not achievable with GPS collars or field observations alone.

Aerial drone footage can help fill in some of these gaps, providing a complete view of the entire site. As there may be hours worth of drone footage to review, it is impractical to do it by hand. Instead, computer vision techniques can reduce the amount of video that must be reviewed.

The primary goal is to be able to use the processed footage to identify the moment of decision. Drone footage is less invasive than other methods (e.g. radio collars) and allows researchers to view the entire group. This footage provides additional context that collars cannot, as a collective decision may originate from an individual's agenda.

4.2.1 Dataset

Next, we will examine a similar problem from ecology, dubbed the baboon tracking dataset provided by photographer Neil Thomas.

In this section, we review the baboon tracking dataset. This dataset is several videos collected by photographer Neil Thomas flying a DJI Mavic 2 Pro over the Laikipia Plateau of Kenya.

Figure 4.2 shows an example frame from this dataset. In this figure, as with the VISO dataset, we see that our objects to track are rather small, having an average pixel size of 25×23.

In addition to the low resolution, just as with the VISO dataset, the baboon tracking dataset has targets that blend well with the environment, as expected, with animals well adapted to their environment. Targets within the dataset have an average luminance value of 0.13.

The baboon tracking dataset also contains examples of pseudo-motion. Examples can

Figure 4.2. Example from the baboon tracking dataset.

be seen in the blowing of trees and grass and the motion induced by the movement of the drone-mounted camera. These motions are similar to what is seen in the VISO dataset, with the shimmering water and satellite motion being analogs.

4.2.2 Video Selection

For this dataset, we chose a single video representative of the data that Spot was designed to process due to the difficulty of procuring data within the space.

4.3 Acknowledgements

Chapter 4 contains material originally published in International Conference on Methods and Techniques in Behavioral Research (Measuring Behavior), October 2020. Crutchfield, Christopher L.; Sutton, Jake; Ngo, Anh; Zadorian, Emmanuel; Hourany, Gabrielle; Nelson, Dylan; Wang, Alvin, McHenry-Crutchfield, Fiona; Forster, Deborah; Strum, Shirley C.; Kastner, Ryan; Schurgers, Curt. The thesis author was the primary investigator and author of this paper.

Chapter 5

Results

In Chapter 1, we introduced two representative problems in the moving object tracking space, vehicle monitoring from satellites in orbit and baboon monitoring from hovering drones. We chose to these as representative problems since they are ultimately the same class of problem, that of a moving-object tracking problem. In the next chapter, Chapter 2, we examined the algorithm we propose to solve the moving-object tracking problem. Then, in Chapter 3, we introduced our methods of selecting parameters for Spot. Finally, in Chapter 4, we discussed the datasets we intend to use to evaluate Spot. In this chapter, we intend to compare Spot against existing metrics for moving-object tracking problems presented through traffic congestion tracking. This data is provided by Yin et al. [22]. Finally, we will examine Spot as a solution to the baboon tracking problem.

5.1 Traffic Congestion

To evaluate Spot, we execute it on each of the seven videos within the VISO dataset, measuring the Precision, Recall, and F1-score generated by evaluating these videos.

Once we have these metrics, we can evaluate Spot many times with different parameters to determine the best parameter set to use. As stated in Chapter 4, we will use Sherlock [8] to perform this by finding the Pareto front for each video.

Table 5.1. Comparison of the total sampled point count vs. the Pareto optimal point count for the VISO dataset

Video Name	Sampled Count	Sampled Percent	Pareto Optimal Count
Video 1	555	2.89%	40
Video 2	546	2.84%	31
Video 3	809	4.21%	31
Video 4	1526	7.95%	41
Video 5	1298	6.76%	56
Video 6	1633	8.51%	42
Video 7	1637	8.53%	29

5.1.1 Estimation of the Pareto Front

To evaluate Spot, we run Sherlock [8] on several samples, observing the results each time. As each sample is expensive, we wish to execute the least number of samples possible to get an accurate Pareto front. Table 5.1 lists the number of samples used to arrive at the conclusions below. It also lists the number of Pareto optimal samples, which we will refer to as the Pareto optimal count. As can be seen, Sherlock sampled between approximately 3% and 9% of the total space. Contrasting this with Gautier et al. [8], which sampled 30%, it is necessary to determine the loss in accuracy. The testing performed by Gautier et al. demonstrated that with so few samples, we could expect an Average Distance to Reference Set (ADRS) within 10^{-1} from the true Pareto front [8].

When reviewing Figure 5.1, it becomes clear that Sherlock has spent most of its time around the Pareto front for the videos from the VISO dataset. Each graph focuses on samples around the Pareto optimal points, with few far away from the Pareto front. This suggests that Sherlock was able to estimate the function f accurately: $\mathbb{R}^m \to \mathbb{R}^o$ and there are unlikely to be any significantly better-performing configurations than the ones examined here.

The Pareto front is well explored in nearly all of the videos, having a near-contiguous collection of points. Exceptions to this are in the gaps in the Pareto front samples seen in video 3

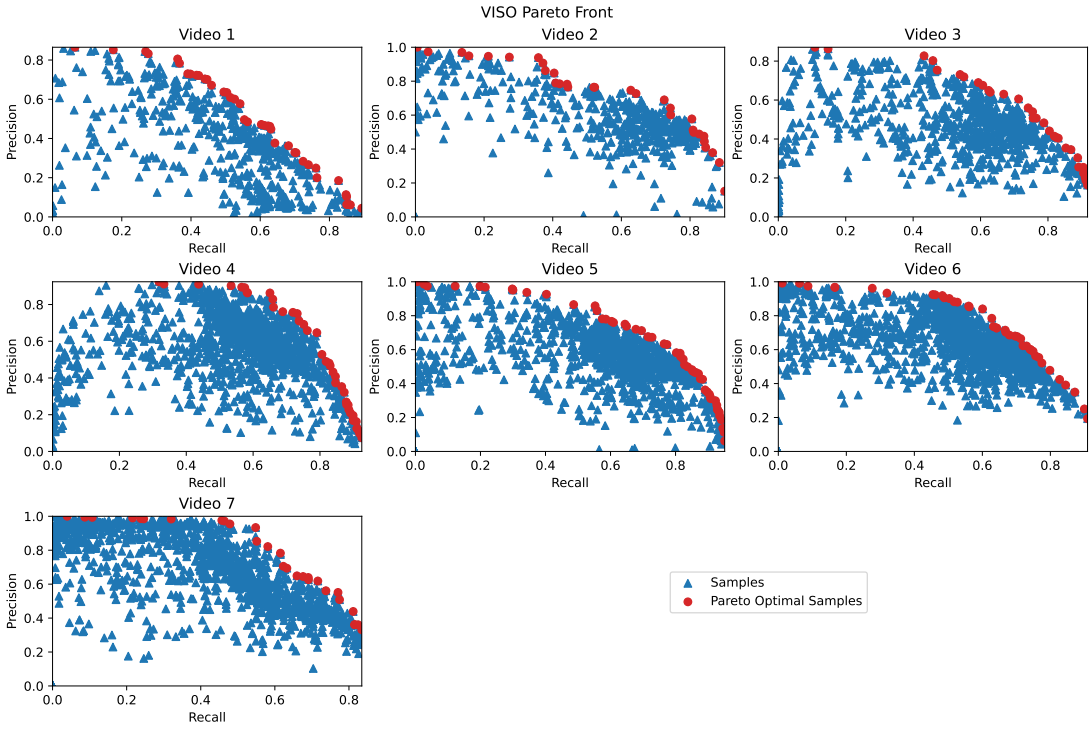


Figure 5.1. Plot of each of the design spaces for each video using Spot as sampled by Sherlock

and video 6. Even in these two cases, the areas of interest are well explored. Since the F1 score is a measure of balance, it is the score most interesting when reviewing these graphs; all videos are well explored around the middle point for their precision and recall.

Next, we will evaluate how effective parameter sets found for one video are on a different, similar video. Figure 5.2 takes the configuration sets from Video 4 and computes them for the remaining six videos. Here, an effective parameter set produces metrics close to the Pareto front. Looking at the results, we can tell that many of these Pareto optimal configurations for Video 4 are also Pareto optimal or near Pareto optimal for the remaining videos, giving confidence that it is possible to determine a parameter set for one video and then use it on like videos. This result is ideal for leveraging Spot as part of another automation pipeline.

5.1.2 Spot Results

Now we will compare Spot to the current state of the art as presented in Yin et al. [22]. The results of this comparison can be seen in Table 5.2. This is even more straightforward to

Table 5.2. Recall, precision, and F1 on seven videos from the VISO dataset using various algorithms [22]. Red represents the highest score in each column and blue represents the second highest

Method	Video 1			Video 2			Video 3			Video 4		
	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
FD [5]	0.58	0.19	0.29	0.79	0.25	0.38	0.80	0.25	0.39	0.69	0.22	0.33
ABM [9]	0.81	0.64	0.71	0.79	0.71	0.75	0.92	0.60	0.73	0.88	0.56	0.68
MGBS [11]	0.80	0.47	0.59	0.78	0.52	0.63	0.91	0.36	0.52	0.86	0.27	0.41
GMM [24]	0.37	0.63	0.47	0.49	0.53	0.51	0.45	0.53	0.49	0.64	0.36	0.46
AGMM [10]	0.72	0.56	0.63	0.80	0.77	0.79	0.93	0.65	0.76	0.87	0.62	0.72
VIBE [4]	0.61	0.34	0.44	0.82	0.61	0.70	0.68	0.59	0.63	0.65	0.52	0.58
FPCP [16]	0.39	0.80	0.53	0.62	0.46	0.53	0.82	0.27	0.41	0.68	0.22	0.34
GoDec [23]	0.92	0.51	0.65	0.73	0.81	0.77	0.93	0.53	0.68	0.72	0.38	0.50
DECOLOR [20]	0.24	0.92	0.38	0.77	0.88	0.82	0.89	0.83	0.86	0.44	0.93	0.60
FRMC [15]	0.55	0.68	0.62	0.57	0.21	0.31	0.61	0.21	0.32	0.63	0.17	0.27
ClusterNet [13]	0.75	0.67	0.71	0.66	0.81	0.72	0.90	0.72	0.80	0.50	0.70	0.58
DTTP [1]	0.74	0.66	0.70	0.67	0.84	0.74	0.71	0.85	0.77	0.64	0.86	0.74
D&T [2]	0.72	0.91	0.80	0.69	0.86	0.77	0.84	0.84	0.84	0.76	0.85	0.80
MMB [22]	0.83	0.84	0.84	0.83	0.89	0.85	0.94	0.88	0.91	0.85	0.86	0.86
Spot	0.50	0.63	0.56	0.72	0.69	0.71	0.71	0.61	0.65	0.65	0.86	0.74

Method	Video 5			Video 6			Video 7			Average		
	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
FD [5]	0.61	0.30	0.47	0.80	0.25	0.39	0.80	0.14	0.24	0.72	0.23	0.36
ABM [9]	0.77	0.61	0.68	0.83	0.50	0.62	0.03	0.13	0.04	0.72	0.54	0.60
MGBS [11]	0.74	0.39	0.51	0.78	0.24	0.36	0.03	0.13	0.05	0.70	0.34	0.44
GMM [24]	0.57	0.36	0.44	0.56	0.37	0.45	0.16	0.38	0.22	0.46	0.45	0.43
AGMM [10]	0.76	0.68	0.72	0.79	0.53	0.63	0.90	0.37	0.53	0.82	0.60	0.68
VIBE [4]	0.72	0.65	0.69	0.60	0.42	0.49	0.45	0.44	0.44	0.65	0.51	0.57
FPCP [16]	0.33	0.33	0.33	0.65	0.26	0.37	0.68	0.18	0.29	0.60	0.36	0.40
GoDec [23]	0.72	0.74	0.73	0.81	0.42	0.55	0.93	0.25	0.39	0.82	0.52	0.61
DECOLOR [20]	0.74	0.84	0.79	0.71	0.80	0.75	0.30	0.69	0.42	0.58	0.84	0.66
FRMC [15]	0.54	0.13	0.21	0.47	0.17	0.25	0.37	0.22	0.28	0.53	0.26	0.32
ClusterNet [13]	0.76	0.82	0.79	0.77	0.71	0.74	0.85	0.66	0.75	0.74	0.73	0.73
DTTP [1]	0.62	0.77	0.69	0.55	0.73	0.63	0.26	0.50	0.34	0.60	0.74	0.66
D&T [2]	0.63	0.81	0.71	0.65	0.76	0.70	0.83	0.43	0.56	0.73	0.78	0.74
MMB [22]	0.80	0.81	0.80	0.78	0.85	0.81	0.83	0.73	0.78	0.84	0.84	0.84
Spot	0.70	0.71	0.70	0.60	0.84	0.70	0.55	0.93	0.69	0.63	0.75	0.68

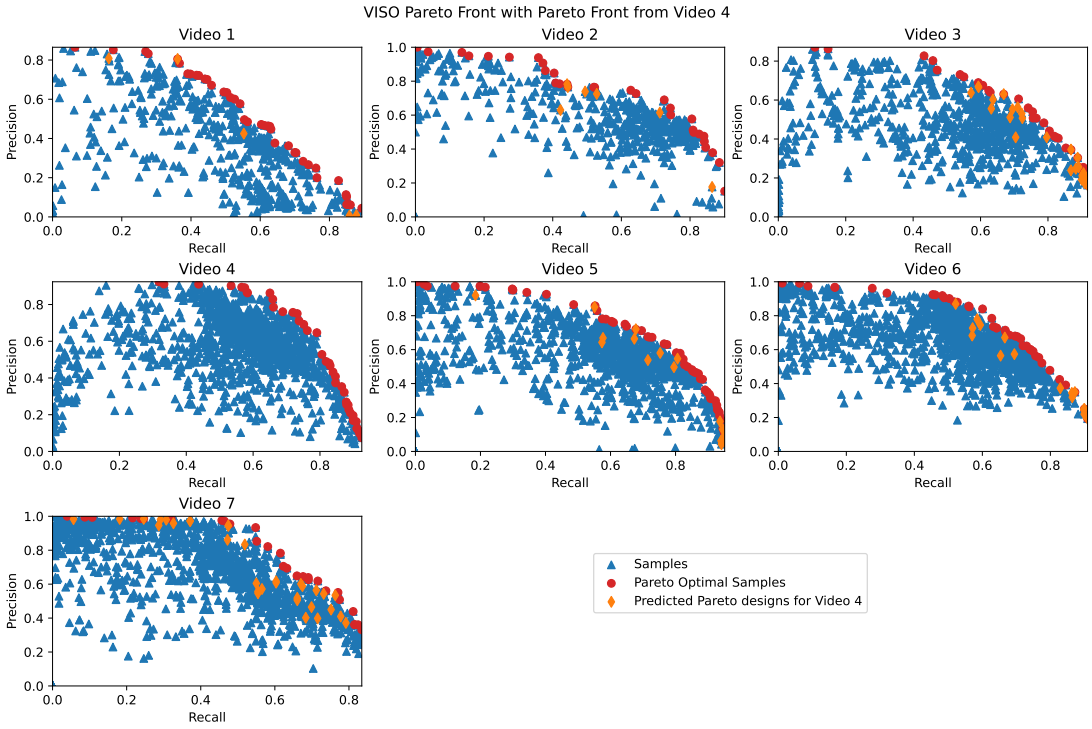


Figure 5.2. The Pareto optimal points for Video 4 compared against the Pareto front for the remaining six videos

see when comparing the average F1 scores. Spot's average F1 matches AGMM and beats many others, demonstrating that it can be a useful algorithm when it fits the constraints. Particularly, Table 5.2 demonstrates that the results are fairly consistent among the videos in the VISO dataset. Given the results previously shown in Figure 5.2 that parameter sets are portable, Spot may be a good algorithm for previously unseen videos and environments.

Table 5.3 looks at each algorithm examined in Yin et al. [22]. Since AP scores are the area under the curve, algorithms with higher AP values are more adaptable over a range of parameter values. As can be seen in Table 5.3, it can be seen that Spot produces the best AP score for Video 7. This would suggest that the precision and recall scores for Video 7 are better than most other algorithms over a wide range of parameter sets, ensuring better results than competing algorithms, especially not when extensively tuned.

Since the mAP score is an average of AP scores across many videos, the mAP score can be seen as a measurement of how well an algorithm can detect moving objects in a dataset. The

Table 5.3. AP and mAP for seven videos from the VISO dataset using various algorithms [22]. Red represents the highest score in each column and blue represents the second highest

Method	Video 1	Video 2	Video 3	Video 4	Video 5	Video 6	Video 7	mAP
FD [5]	0.26	0.55	0.51	0.47	0.44	0.55	0.28	0.44
ABM [9]	0.59	0.70	0.76	0.67	0.65	0.63	0.01	0.57
MGBS [11]	0.73	0.62	0.58	0.41	0.49	0.30	0.10	0.46
GMM [24]	0.41	0.41	0.56	0.39	0.35	0.36	0.16	0.38
AGMM [10]	0.64	0.71	0.75	0.71	0.61	0.60	0.47	0.64
VIBE [4]	0.55	0.73	0.56	0.54	0.67	0.50	0.30	0.55
FPCP [16]	0.54	0.45	0.42	0.28	0.27	0.35	0.20	0.36
GoDec [23]	0.56	0.75	0.72	0.64	0.74	0.68	0.30	0.63
DECOLOR [20]	0.42	0.73	0.81	0.56	0.75	0.72	0.52	0.64
FRMC [15]	0.51	0.22	0.23	0.20	0.11	0.14	0.13	0.22
ClusterNet [13]	0.77	0.73	0.74	0.61	0.77	0.75	0.57	0.71
DTTP [1]	0.75	0.72	0.76	0.63	0.65	0.69	0.42	0.66
D&T [2]	0.79	0.76	0.80	0.81	0.73	0.70	0.54	0.73
MMB [22]	0.88	0.84	0.95	0.89	0.81	0.77	0.73	0.84
Spot	0.53	0.71	0.65	0.74	0.76	0.74	0.72	0.69

mAP score for Spot suggests that the algorithm is better than approximately 75% of the other algorithms, again suggesting that Spot excels in adaptability.

This is impressive since Spot was not initially designed for or optimized for this scenario, unlike other algorithms examined here. Spot does not make any assumptions about the motion. It may perform better on this dataset if updated to take advantage of motion assumptions like other algorithms here. We choose not to add those assumptions to keep Spot more generic so that it can solve a broader range of problems.

5.2 Baboon Tracking

As we do not have data for the other algorithms presented in Chapter 5, we will only present on Spot for this dataset. As before, we will leverage Sherlock to determine the parameter sets to sample. Again, our goal here is to estimate the Pareto front.

Table 5.4. Comparison of the total sampled point count vs the Pareto optimal point count for the baboon tracking dataset

Video Name	Sampled Count	Sampled Percent	Pareto Optimal Count
Video 1	714	3.72%	52

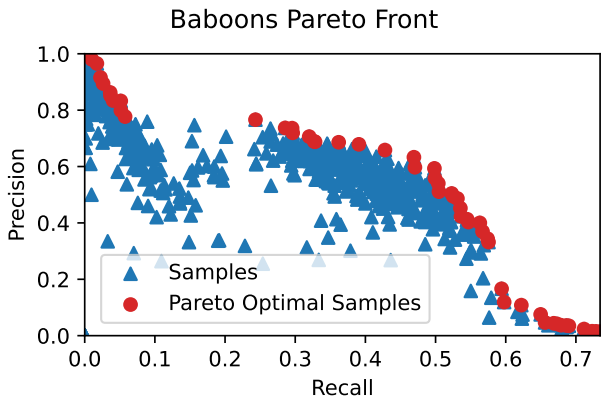


Figure 5.3. Plot of the design space for the dataset using Spot as sampled by Sherlock

5.2.1 Estimation of the Pareto Front

We ran Sherlock with Spot on the baboon tracking dataset and produced Table 5.4. This indicates how many samples Sherlock chose, the overall percentage those samples represent of the design space, and finally, the number of samples that represent the estimated Pareto front.

As previously stated in Chapter 4, data that presents unique challenges is difficult and expensive to collect. For the sake of the evaluation, we will focus on just a single video.

Comparing the results from Table 5.4 with Table 5.1, we can see that Sherlock sampled a similar number of points compared to the VISO dataset. Therefore, it likely has an ADRS within 10^{-1} from the true Pareto front.

In Figure 5.3, we can see that Sherlock is sampling near the estimated Pareto front, much like for the VISO dataset. This demonstrates that any potentially missed portion of the Pareto front will unlikely improve Spot's results substantially.

Table 5.5. Results from running Sherlock against a representative video from the baboon dataset. The recall, precision, and F1 scores were selected by choosing the largest F1 score found by Sherlock

Recall	Precision	F1	AP
0.5	0.59	0.54	0.42

5.2.2 Spot Results

In this section, we will discuss the performance of Spot for baboon tracking. As we do not have the same baseline metrics provided by the VISO dataset, we will not compare them against other algorithms.

Table 5.5 lists the results for Spot against the baboon tracking dataset. We can see that this Video appears to be similar to Video 1 listed in Table 5.2 and Table 5.3 from the VISO dataset. This suggests that, like Video 1, this video may be more difficult to process due to environmental pseudo-motion. This means that the algorithm sensitivity has to be turned down to ignore the pseudo-motion better.

Another possible reason it may be harder to track the animals within the video compared to vehicles from satellite footage is that animals are lower contrast. As the animals move, they blend in well with their environment. An additional complication is that moving animals may become occluded by large objects within the environment. This does not happen from the satellite footage as it is mostly highway.

Chapter 6

Conclusion

Throughout this write-up, we focused primarily on the introduction and evaluation of Spot. To this end, we evaluated it against the VISO and baboon tracking datasets in Chapter 5. To aid this analysis of Spot, we introduced the idea of using Sherlock [8], a design space exploration algorithm, to discover the best configuration parameters for a particular representative video. It estimates the function that maps Spot's input parameters to the metrics. When performing this optimization, we also analyzed the importance of calculating metrics when performing design space exploration. Our contributions around Sherlock were adapting away from its original, intended usage of exploring the design space of FPGA design and into the space of parameter tuning for software algorithms. We leveraged it to for finding the ideal parameter set for Spot.

We demonstrated that Spot tuned with Sherlock works well against the VISO dataset. We further examined Spot by demonstrating it as a potential solution to the ecology problem of tracking animals from a hovering drone as performed by Crutchfield et al. [6]. The main contributions of this write-up overall were the introduction and evaluation of Spot as a solution to the moving object tracking problem and Sherlock as a tool for parameter selection.

Comparisons with other algorithms performed in Chapter 5 suggest possible extensions of the research documented here. Evaluating other algorithms against the baboon tracking dataset would confirm that Spot is well-targeted for such a problem. In particular, we would like to evaluate AGMM [15], ClusterNet [13], D&T [2], and MMB [22] as these four algorithms

outperform Spot on the vehicle tracking problem. To further the evaluation of Spot, it would be ideal to collect additional information from the field with more variety.

Bibliography

[1] Seyed Ali Ahmadi, Arsalan Ghorbanian, and Ali Mohammadzadeh. Moving vehicle detection, tracking and traffic parameter estimation from a satellite video: a perspective on a smarter city. *International Journal of Remote Sensing*, 40(22):8379–8394, November 2019.

[2] Wei Ao, Yanwei Fu, Xiyue Hou, and Feng Xu. Needles in a Haystack: Tracking City-Scale Moving Vehicles From Continuously Moving Satellite. *IEEE Transactions on Image Processing*, 29:1944–1957, 2020.

[3] Oleksandr Bailo, Francois Rameau, Kyungdon Joo, Jinsun Park, Oleksandr Bogdan, and In So Kweon. Efficient adaptive non-maximal suppression algorithms for homogeneous spatial keypoint distribution. *Pattern Recognition Letters*, 106:53–60, April 2018.

[4] O Barnich and M Van Droogenbroeck. ViBe: A Universal Background Subtraction Algorithm for Video Sequences. *IEEE Transactions on Image Processing*, 20(6):1709–1724, June 2011.

[5] Yutian Cao, Gang Wang, Dongmei Yan, and Zhongming Zhao. Two Algorithms for the Detection and Tracking of Moving Vehicle Targets in Aerial Infrared Image Sequences. *Remote Sensing*, 8(1):28, December 2015.

[6] Christopher L Crutchfield, Jake Sutton, Anh Ngo, Emmanuel Zadorian, Gabrielle Hourany, Dylan Nelson, Alvin Wang, Fiona McHenry-Crutchfield, Deborah Forster, Shirley C Strum, Ryan Kastner, and Curt Schurgers. Baboons on the Move: Enhancing Understanding of Collective Decision Making through Automated Motion Detection from Aerial Drone Footage. page 7, October 2020.

[7] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. page 6.

[8] Quentin Gautier, Alric Althoff, Christopher L. Crutchfield, and Ryan Kastner. Sherlock: A Multi-Objective Design Space Exploration Framework. *ACM Transactions on Design Automation of Electronic Systems*, 27(4):1–20, July 2022.

[9] D. Gutchess, M. Trajkovics, E. Cohen-Solal, D. Lyons, and A.K. Jain. A background model initialization algorithm for video surveillance. In *Proceedings Eighth IEEE International*

Conference on Computer Vision. ICCV 2001, volume 1, pages 733–740, Vancouver, BC, Canada, 2001. IEEE Comput. Soc.

[10] Horng-Horng Lin, Jen-Hui Chuang, and Tyng-Luh Liu. Regularized Background Adaptation: A Novel Learning Rate Control Scheme for Gaussian Mixture Modeling. *IEEE Transactions on Image Processing*, 20(3):822–836, March 2011.

[11] David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Vladimir Reilly, Haroon Idrees, and Mubarak Shah. Detection and Tracking of Large Number of Targets in Wide Area Surveillance. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, volume 6313, pages 186–199. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. Series Title: Lecture Notes in Computer Science.

[12] George Kopsiaftis and Konstantinos Karantzalos. Vehicle detection and traffic density monitoring from very high resolution satellite video data. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 1881–1884, Milan, Italy, July 2015. IEEE.

[13] Rodney LaLonde, Dong Zhang, and Mubarak Shah. ClusterNet: Detecting Small Objects in Large Scenes by Exploiting Spatio-Temporal Information. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4003–4012, Salt Lake City, UT, USA, June 2018. IEEE.

[14] Kumar S. Ray and Soma Chakraborty. An Efficient Approach for Object Detection and Tracking of Objects in a Video with Variable Background. *arXiv:1706.02672 [cs]*, May 2017. arXiv: 1706.02672.

[15] Behnaz Rezaei and Sarah Ostadabbas. Background Subtraction via Fast Robust Matrix Completion. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 1871–1879, Venice, Italy, October 2017. IEEE.

[16] Paul Rodriguez and Brendt Wohlberg. Fast principal component pursuit via alternating minimization. In *2013 IEEE International Conference on Image Processing*, pages 69–73, Melbourne, Australia, September 2013. IEEE.

[17] Edward Rosten and Tom Drummond. Machine Learning for High-Speed Corner Detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, volume 3951, pages 430–443. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. Series Title: Lecture Notes in Computer Science.

[18] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571, Barcelona, Spain, November 2011. IEEE.

[19] Satoshi Suzuki. Topological Structural Analysis of Digitized Binary Images by Border Following.

[20] Xiaowei Zhou, Can Yang, and Weichuan Yu. Moving Object Detection by Detecting Contiguous Outliers in the Low-Rank Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(3):597–610, March 2013.

[21] Tao Yang, Xiwen Wang, Bowei Yao, Jing Li, Yanning Zhang, Zhannan He, and Wencheng Duan. Small Moving Vehicle Detection in a Satellite Video of an Urban Area. *Sensors*, 16(9):1528, September 2016.

[22] Qian Yin, Qingyong Hu, Hao Liu, Feng Zhang, Yingqian Wang, Zaiping Lin, Wei An, and Yulan Guo. Detecting and Tracking Small and Dense Moving Objects in Satellite Videos: A Benchmark. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–18, 2022. arXiv: 2111.12960.

[23] Tianyi Zhou and Dacheng Tao. GoDec: Randomized Low-rank & Sparse Matrix Decomposition in Noisy Case. page 16.

[24] Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27(7):773–780, May 2006.