Summer 6-12-2019

Obstacle and Change Detection Using Monocular Vision

Ryan Bluteau
University of Windsor

Obstacle and Change Detection Using Monocular Vision

By

Ryan Bluteau

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2019

Obstacle and Change Detection Using Monocular Vision

by

Ryan Bluteau

APPROVED BY:

_____

J. Wu
Department of Electrical and Computer Engineering

_____

I. Ahmad
School of Computer Science

_____

B. Boufama, Advisor
School of Computer Science

June 12, 2019

DECLARATION OF CO-AUTHORSHIP AND PREVIOUS PUBLICATION

I. Co-Authorship

I hereby declare that this thesis incorporates material that is result of joint research, as follows: Chapter 3 of this thesis contains materials in collaboration with Dr. Boubakeur Boufama and Dr. Pejman Habashi. In all cases, the key ideas, primary contributions, experimental designs, data analysis, interpretation, and writing were performed by the author, and the contribution of co-authors was primarily through the provision of guidance, proof reading and reviewing the research paper.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Previous Publication

This thesis includes 1 original paper that has been previously published/submitted for publication in peer reviewed journals, as follows:

Thesis Chapter	Publication title/full citation	Publication status
Chapter 3	Ryan Bluteau, Boubakeur Boufama, and Pejman Habashi. "Determining Location and Detecting Changes Using a Single Training Video"	Submitted

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Windsor.

III. General

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

We explore change detection using videos of change-free paths to detect any changes that occur while travelling the same paths in the future. This approach benefits from learning the background model of the given path as preprocessing, detecting changes starting from the first frame, and determining the current location in the path. Two approaches are explored: a geometry-based approach and a deep learning approach.

In our geometry-based approach, we use feature points to match testing frames to training frames. Matched frames are used to determine the current location within the training video. The frames are then processed by first registering the test frame onto the training frame through a homography of the previously matched feature points. Finally, a comparison is made to determine changes by using a region of interest (ROI) of the direct path of the robot in both frames. This approach performs well in many tests with various floor patterns, textures and complexities in the background of the path.

In our deep learning approach, we use an ensemble of unsupervised dimensionality reduction models. We first extract feature points within a ROI and extract small frame samples around the feature points. The frame samples are used as training inputs and labels for our unsupervised models. The approach aims at learning a compressed feature representation of the frame samples in order to have a compact representation of background. We use the distribution of the training samples to directly compare the learned background to test samples with a classification of background or change using a majority vote. This approach performs well using just two models in the ensemble and achieves an overall accuracy of 98.0% with a 4.1% improvement over the geometry-based approach.

ACKNOWLEDGEMENTS

I would like to take this time to express my sincere gratitude to both my advisor Dr. Boubakeur Boufama and to Dr. Pejman Habashi for their efforts and guidance throughout my masters. I would also like to thank Dr. Robin Gras for his guidance towards the end of my masters. Finally, I would like to thank the rest of my thesis committee, both Dr. Jonathan Wu, and Dr. Imran Ahmad, for their guidance and feedback.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AR	Augmented Reality
CCT	Correlated Colour Temperature
CNN	Convolutional Neural Network
EM	Expectation Maximization
FOE	Focus of Expansion
FOV	Field of View
FPS	Frames Per Second
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GPS	Global Positioning System
HCI	Human-Computer Interaction
HSI	Hue Saturation Intensity
HSV	Hue Saturation Value
IPM	Inverse Perspective Mapping
MOG	Mixture of Gaussian
MSE	Mean Squared Error
ORB	Oriented FAST and Rotated BRIEF
RANSAC	Random Sample Consensus
ROI	Region of Interest

SPD	Spectral Power Distribution
SSR	Surface Spectral Reflectance
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TCF	Two Consecutive Frames
ToF	Time-of-flight
WNN	Weightless Neural Network

CHAPTER 1
Introduction

1.1 Computer Vision

The field of Computer Vision aims at understanding images at a high level to perform vision-based tasks. Images can be captured using different sensors to acquire different information, such as, an RGB camera for colour, LIDAR for depth and many more. Information from lower level image processing methods can be utilized in order to extract higher level information, which in turn provides valuable information to perform related tasks in many fields.

Computer vision can be applied to many fields with a visual component. Computer vision does not aim to reproduce the biological vision system, but instead aims to reproduce the system's functions. As a result, there are a large number of fields where computer vision exists, such as motion detection, autonomous navigation, scene reconstruction and recognition, augmented reality (AR), object recognition, object tracking and many more high level vision-based tasks.

Motion detection aims to track the movement of an object or person. For example, background subtraction (to be discussed further in section 1.3) can be used in a surveillance camera to monitor a specific area to automate motion detection, see Figure 1. Motion detection is not limited to video surveillance, as it can be useful to track gestures, posture and actions for possible applications in many areas, such as, Human-Computer Interaction (HCI) and Augmented Reality (AR).

Many applications of motion detection can be incorporated into an augmented reality device. Gesture recognition can be useful for device interaction to allow a user the ability to react to a virtual scene and/or objects without the use of a controller, see Figure 2. Movement can be tracked for virtual object placement and occlusion. For example, a person might walk in front of a virtual object and occlude part of the view of said object, thus requiring the ability to understand a 3D environment.

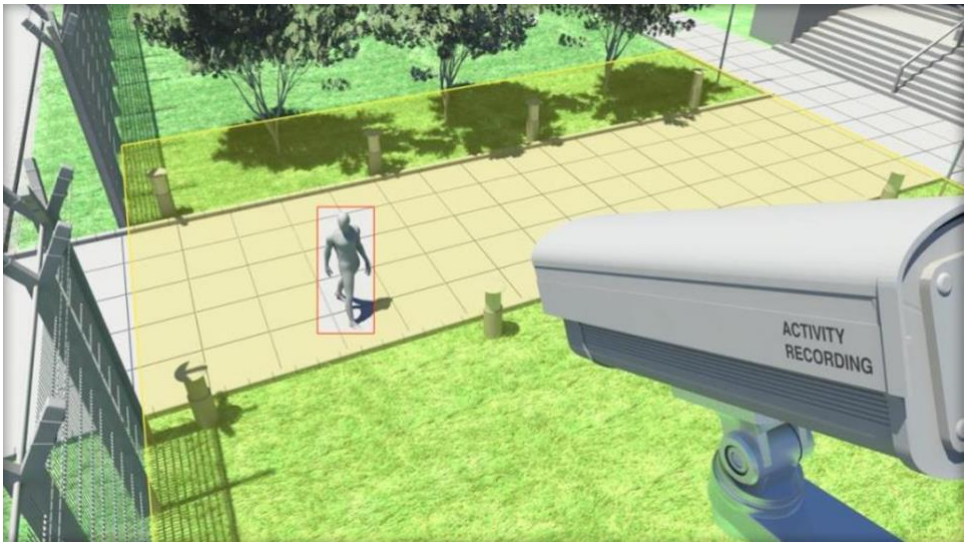

Figure 1. A surveillance camera monitoring a path and detecting a person's activity. (Image acquired from https://www.ips-analytics.com/en/products/ips-videoanalytics-new/server-based/ips-motion-detection.html).

Placement of virtual objects in augmented reality requires the ability to recognize and reconstruct the scene. Scene reconstruction requires information about the 3D space of the scene, including depth. An RGB camera can be used to recognize the scene, but an image does not have depth information. This is because the camera projects a 3D space into a 2D space to acquire the image, thus losing depth information through the loss of the third coordinate of each pixel. In computer vision, many solutions can be applied to this problem to acquire depth. For example, structure-from-motion can acquire depth

from many sequential images captured from a single camera, or more sensors can contribute information, such as a second camera for stereo vision, or other sensors like LIDAR and RADAR for depth coordinates from a time-of-flight (ToF) sensor. With depth information, the scene can be reconstructed virtually using estimated 3D coordinates of each pixel in order to project virtual objects into the scene.

Figure 2. An image from Microsoft's HoloLens 2 live demonstration. The user is interacting with virtual objects viewed from the created holograms within the HoloLens headset. The cameras on the headset detect the user's gestures and allows the user to adjust differently styled buttons and sliders (seen in the image) just by grabbing them in air. (Image acquired from https://www.youtube.com/watch?v=uIHPPtPBgHk).

In the last of our examples of fields in computer vision, we have applications to object recognition. The biological system uses object recognition for many tasks, such as reading the individual characters in a sentence or entire words at one time, understanding our environment from the objects that surround us, and immediately recognizing a person from their face. With recent developments in deep learning, object recognition has greatly advanced in computer vision through the use of training data and deep learning models.

We further explain deep learning in computer vision in section 1.2, background subtraction in section 1.3, obstacle detection in section 1.4, and change detection in section 1.5. We present the problem statement in section 1.6, possible applications in section 1.7, and finally the thesis organization in section 1.8.

1.2 Deep Learning

Computer vision has progressed rapidly through the use of deep learning. Specifically, we utilize deep learning models, such as the Convolutional Neural Network (CNN), to process images. At a high level, CNNs are special neural networks that use shared parameters in order to scan images. Computer vision also makes use of dimensionality reduction models, such as the autoencoder structure. An autoencoder aims to reduce the incredibly high dimensionality of an input image (for computer vision tasks) into a set of features that can be decoded to reproduce the original input image.

A neural network is a collection of nodes, called neurons, and weighted node connections, see Figure 3. An input is flown through layers of neurons in order to produce an output. In the case of computer vision, inputs can be images and the associated outputs are labels for classification or ranges (continuous values) for regression tasks. Each neuron is connected to several other neurons with an associated weight to each connection. Inside the neuron is an activation function that transforms inputs non-linearly. The neuron takes the dot product between the previous layer of neurons and the connected weights as input. The output of the neuron is a non-linear transformation of the dot product using the neuron's activation function, such as ReLU, Sigmoid, Tanh, and more. This is repeated for all the connected neurons through the network until an output node is reached. The final output is formatted to fit the criteria of

the problem in the case of classification or regression using functions, such as SoftMax, Sigmoid, and more.

The neurons of a neural network can be viewed as feature detectors where a given input is encoded into numeric values (features) as it is passed through the layers of the network. Each layer would ideally (not always the case) learn progressively more complex features of an input. The features are detected using a dot product to make a comparison of the weighted connections and the outputs of the previous layer of neurons (or the input itself). The neuron will output a value for the resulting comparison, which represents a more complex and high level feature composed of the previously detected features. At a high level, the network simply aims to learn its weighted parameters through training of various input and output examples to detect valuable features of the data. This then acts like a function relating the input examples to the output examples, and can be, ideally, generalized to a larger set of related unseen inputs and outputs.

A loss function and gradient descent algorithm are used to tune the weights of the neural network into producing the appropriate outputs. The loss function acts as a mechanism to measure the difference between the currently poor output of the network and expected (labelled/training) output of a given input. From there, the gradient descent algorithm calculates gradients of each weight and the series of connected neurons through back propagation based on the amount of loss described by the loss function. Gradient descent describes the estimated direction to tune the weights while back propagation is the process of tuning each weight as we backpropagate through the network nodes/weights. As a result, each weight is slightly tuned for better results with each training sample (or group of samples).

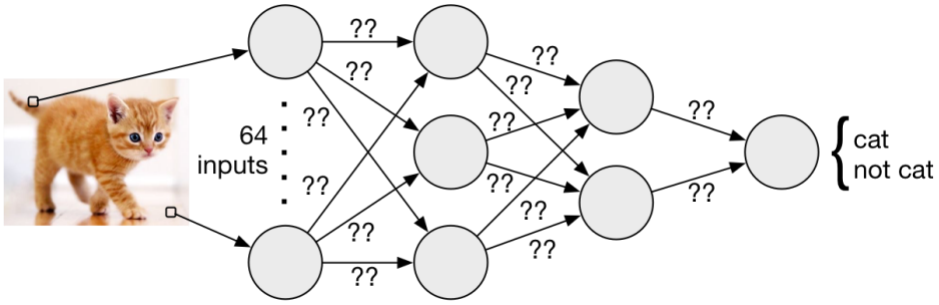

Figure 3. An image of a cat is used as an input to 64 nodes, called neurons. These neurons feed into several layers of neurons (in the case of deep learning) until an output neuron is reached for a final classification of cat or not cat. Each neuron compares the connected weights to the output of the previous neuron (or an input value) through a dot product. Training data, a loss function and gradient descent are used to tune the weighted parameters of the network (the "??" over the connections) to properly classify the image as a cat. (Image acquired from https://homes.cs.washington.edu/~bornholt/post/nnsmt.html).

CNNs are neural networks designed to be more efficient for images. Images are generally very large and costly to process. A neural network must have an input weight for every pixel of an image. This is unrealistic in most cases, especially high resolution images. To overcome this problem, CNNs use shared parameters to process and scan an image. A neural network, called a filter in this case, processes a very small portion of the image to produce a feature of the area. The filter is scanned using the same parameters (hence the term shared parameters) over the entire image to produce a feature for each region. There may be several filters processing the image and producing output, which together create a set of feature maps as an output. This process of scanning is known as a convolution. We can apply the convolution step to the produced feature maps of previous layers with new filters. This allows the CNN to obtain the desired growth in feature complexity the original neural network achieves by stacking layers of neurons. We may also process feature maps with pooling and activation functions to further focus processing on specific regions of an image using max pooling, or to process the image

non-linearly using an activation function. Once the feature map is small enough, we feed the feature map entirely to a normal fully connected neural network to finally classify the image. See Figure 4 for an example of a sequence of CNN steps. Training the CNN is similar to training the neural network where we update the weights using a loss function and gradient descent. The shared parameters of the filters are updated by accumulating the gradient of contributing inputs processed by the shared parameter.

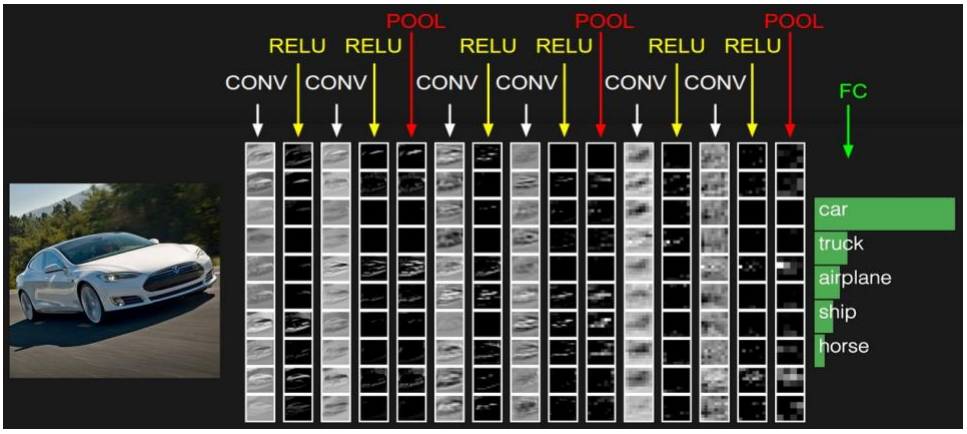

Figure 4. Here an image of a vehicle is processed by a convolutional neural network to recognize whether the image is a car, truck, airplane, ship or horse. Each column is a different step in the CNN process used (flowing from left to right). The boxes represent features produced in the current step of several filters. The first step is a convolution of shared parameters to generate a feature map. The next is the use of an activation function known as ReLU for non-linear processing. Occasionally pooling is used to focus processing on key information. Finally, a neural network is used to process the features and classify the image as a car. (Image acquired from http://cs231n.github.io/convolutional-networks/).

Finally, another important neural network is the autoencoder. The autoencoder simply aims to compress an input into a feature representation, known as dimensionality reduction. The network is built on two steps, an encoding of the input and a decoding of features. The relationship of both portions of the network can be seen in Figure 5. Previous networks discussed are generally used as supervised learning models. This means that the model is given

the opportunity to learn from labelled outputs, such as Cat or Not Cat. The autoencoder is an unsupervised approach where there are no associated labels to any input, because it simply learns the feature space of the inputs. The encoder portion of the network is a simple neural network with features as outputs, essentially skipping the final "output" layer with classification data. In order to train the network, a decoder is used to reproduce the original image. The decoder is just another neural network taking the features of the encoder as input and producing the original image as output, though it is expanding into outputs rather than shrinking into features. Together they make a large network. The images (or samples) are used as both the input and the training "label" of the network. The loss function compares the output image of the autoencoder to the original image and describes the amount information lost between the images. Finally, gradient descent adjusts the weighted connections to tune the network. The encoder and decoder portions of the network can be split and used individually to compress (encode) or decompress (decode) samples.

1.3 Background Subtraction

Background subtraction is a task where we aim to extract foreground information from a sequence of images obtained through a stationary camera. Ideally, the images will have each pixel labelled as black for background and white for foreground to visualize the segmentation of the foreground, see Figure 6. This information can be used in a variety of fields aiming to track motion, like video surveillance and gesture recognition.

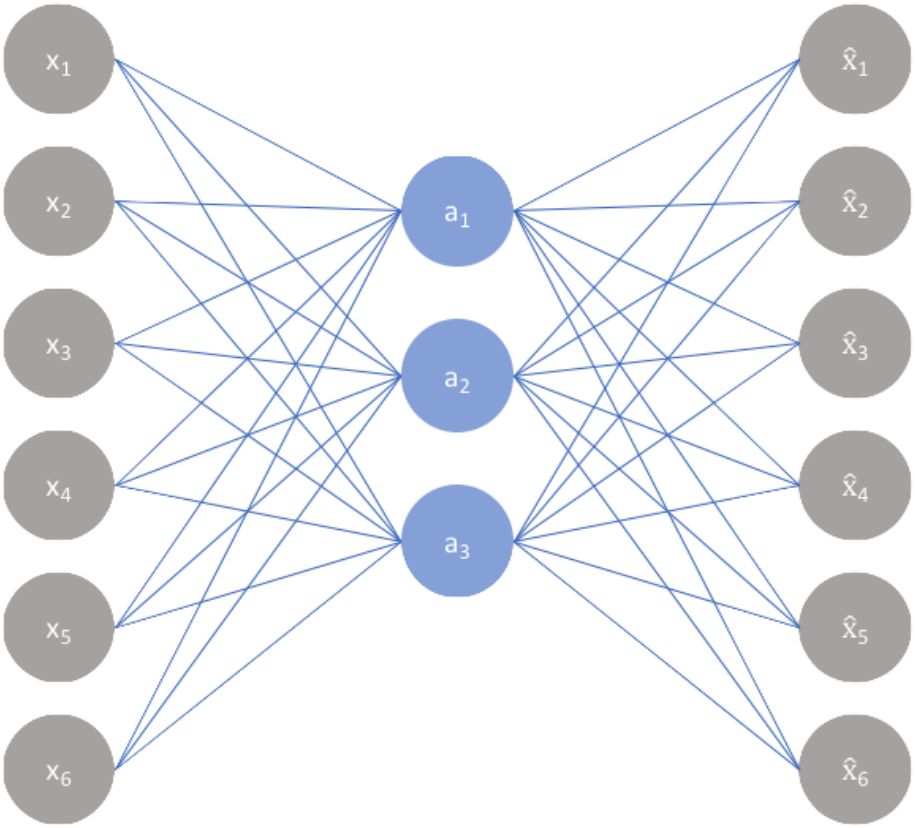

Figure 5. Example structure of an autoencoder with a single layer representing the features (neurons coloured in blue / placed in the center). The input is on the left, encoded into features in the middle and decoded into the original input on the right. The network can be detached to use the encoder and decoder individually. Many more layers can be incorporated into the autoencoder for a deeper network. (Image acquired from https://www.jeremyjordan.me/autoencoders/). (Best seen in colour).

Background subtraction has some areas of difficulty and limitations. The sequence of images must be stationary. Often, this is not the case as camera jitter can be present in the video feed, so methods must adapt to some movement. The background may have some background noise or circular movement such as waves, grass/tree leaves moving in the wind, and more. These motions should not be considered part of the foreground.

Figure 6. Example of extracting the foreground using a background subtraction technique, mixture of gaussian (MOG). (Images acquired from https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html).

Sometimes foreground movement may become part of the background, and other times the background can become part of the foreground, due to some movement. For example, a driving car is considered foreground, but when the car is parked it should merge into the background. At the same time, a parked car can be part of the background, but when the car is moved it should appear as the foreground. Another example would be a floor mat which is considered part of the background. When someone comes around and moves the mat in a different location, it should appear as foreground while being moved, but merge with the background once stationary. There's a related problem known as ghosting where the initial location of a moved background object might remain as a detected foreground, despite nothing being there. We are left with the "ghost" of the object since the new background appears different from when the background object was occupying that location. For example, in Figure 7 the mat that was moved will leave a new background in its initial location, and could be detected as foreground since the new floor is different from the mat. This would leave us with two mats detected, where one is the ghost of the original.

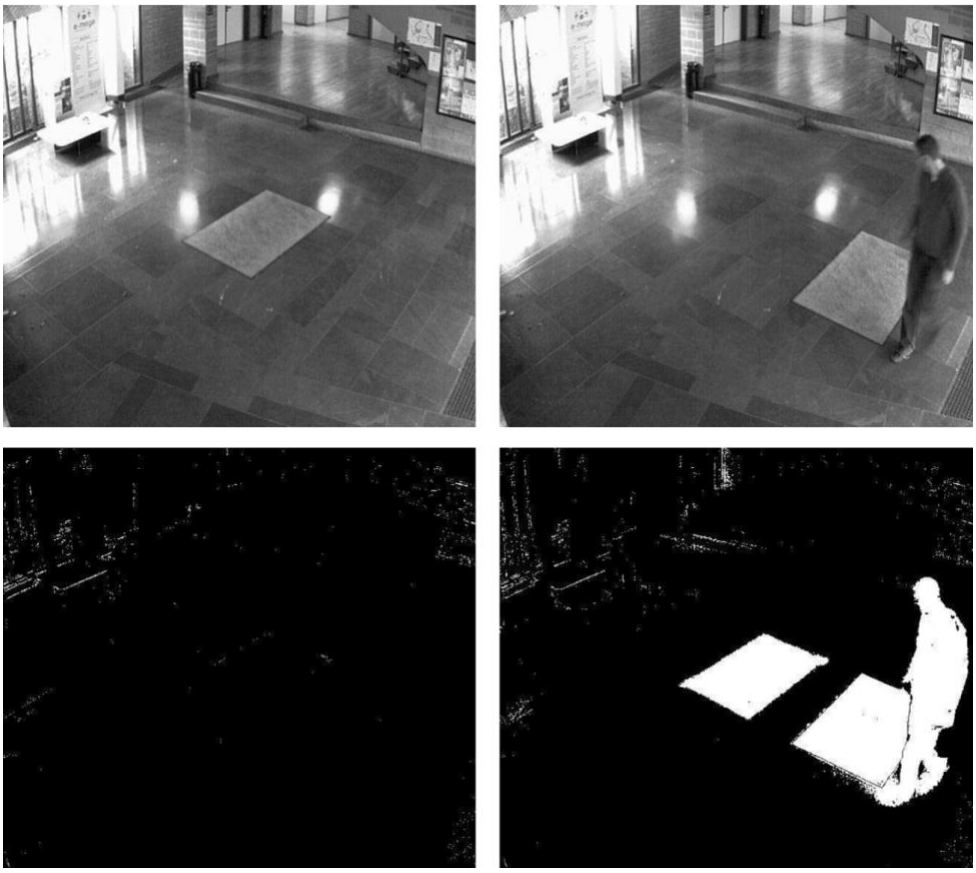

Figure 7. Example of the ghosting effect where a mat has been moved to another location leaving the ghost of the mat in its original position. (Image acquired from [12]).

Other difficulties can include lighting changes. For outdoor systems, light can shift from day to night, clouds can cause sudden lighting changes, and shadows can move with the sun. Lighting changes should be considered as part of the background and the background model should adjust for dynamic changes through time. Additionally, sudden bright areas, such as glares, can occur due to shinny surfaces and take up a large area of the image. These glares should have no affect on the background or foreground objects. Other lighting changes might be darker, such as shadows. Shadows caused by background objects must be modeled over time if any movement occurs. Foreground objects might also cause shadows creating difficulties in our foreground detection. We do

not want to include shadows in this case (though some algorithms include options to detect shadows), thus the background model must be flexible in lighting conditions.

1.4 Obstacle Detection

Obstacle detection aims to detect obstructions of a path being travelled by an automated vehicle (or to assist a person). For example, a robot travelling down a hallway might see potential obstacles, such as people, walls, and other objects, along the path. The robot does not have prior knowledge of the path's floor or structure and must determine where it is safe to travel. Obstacle detection relies on sensors to detect potential hazards, such as one or more of the following: RGB camera, LIDAR, RADAR, ultrasonic, and more. For example, a simple robot travelling a path might just need a single RGB camera, but a self-driving vehicle might require several RGB cameras and depth sensors such as RADAR and/or LIDAR (see Figure 8).

Obstacle detection has some areas of difficulty depending on the sensors used. LIDAR might detect depth and overall 3D structure with high precision, but is expensive and has low resolution. RADAR is generally less precise and slower than LIDAR, but can penetrate through weather such as rain and snow. RGB cameras are affected by lighting conditions, but are generally fast, precise, have high resolution, and are inexpensive. In that case, our focus will be on the use of a single RGB camera, known as monocular vision.

The main difficulty of monocular vision in obstacle detection is modelling the floor and pixel depth. Monocular vision in obstacle detection relies on detecting obstacles that appear different from the ground. Though depth can be estimated through a series a frames, this is a computationally expensive process. Trying to model the ground

appearance can lead to difficulties, such as sudden glares due to a shiny surface or a sudden change in floor textures and patterns. Other difficulties can include obstacles that appear similar to the floor creating false negatives.

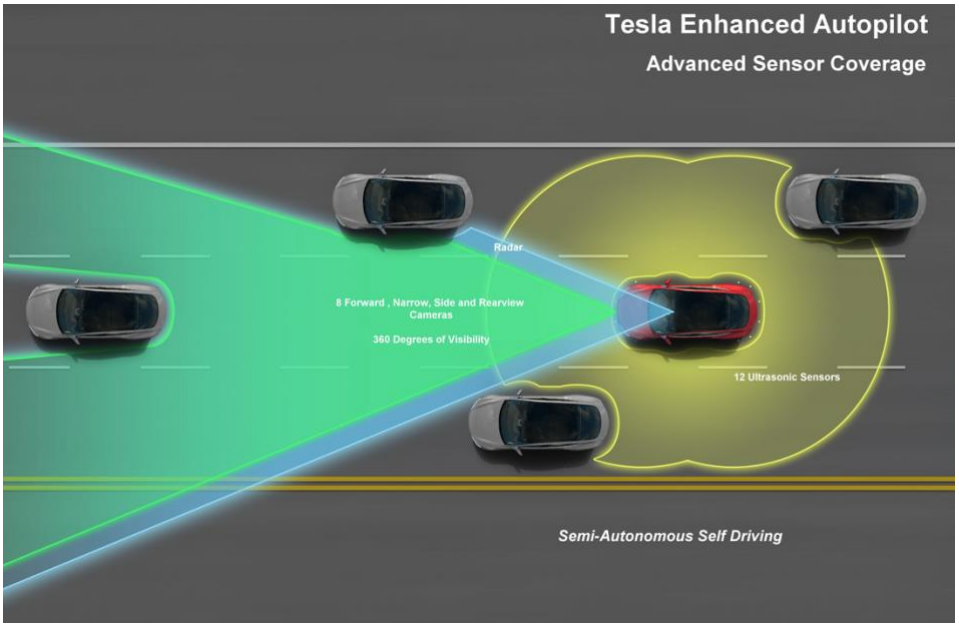

Figure 8. Tesla's autonomous vehicle sensors. Not shown are side facing and rear facing cameras. Sensor FOV is shown in this image and how each vehicle is detected using 8 cameras, RADAR and 12 ultrasonic sensors. (Image best seen in colour). (Image acquired from https://medium.com/self-driving-cars/tesla-enhanced-autopilot-overview-l2-self-driving-hw2-54f09fed11f1).

1.5 Change Detection

While obstacle detection aims at detecting obstructions and hazards to a path being travelled, change detection while travelling a path only considers a change in the path. Obstacle detection considers everything, apart from the ground, as a hazard or a detectable object. The main difference between change detection and obstacle detection is the background that is being modelled. Monocular vision obstacle detection methods generally aim to model the ground, but a change detection method (in the context of

change detection in a path) would ideally model everything that is part of the path, such as walls and other stationary objects.

Change detection aims at detecting a difference from prior knowledge. For example, change detection in the form of background subtraction would extract the foreground based on prior knowledge of the background (such as a video feed of the area), so that the segmentation only contains relevant information. In a similar way, applying change detection to obstacle detection would result in the detection of obstructions that differ from prior knowledge of the path. For example, the walls would be considered known and not a detectable change. At the same time, something missing from prior knowledge can be considered as a detectable change (such as a missing wall), but this might not be important for methods, such as background subtraction due to problems like ghosting.

1.6 Problem Statement

We aim to implement change detection in a travelled path using monocular vision. We are provided with a training video of the scene to be travelled. This video is ideally obstacle free and any objects included in the video are considered part of the path. The training video will show the path travelled in full. The robot that will be travelling the path is assumed to know the path it must travel and the directions it must take. When the robot travels the path during the test phase, it can utilize this training data to determine any changes along the path, such as unseen obstacles. The detectable obstacles of a path do not include elements such as walls and other objects within the training video. If a detection occurs, the robot will simply halt, and either wait for further instructions or wait for the change to be resolved (for example a person crossing the path).

1.7 Applications

This problem can be applied for robots travelling known paths, such as factory robots with repetitive tasks with a need to detect possible obstructions and changes in their path. This would allow for informed decisions about stopping or obstacle avoidance and other operations available. This problem can also be applied to automated robots, such as home cleaning robots where the environment is known. These robots can scan the environment to learn the path which can be used as the training video. This would allow for detection of new obstacles or movement of furniture to adapt to the environment. This could provide an advantageous cost reduction and easy environment training with just a single camera for detection.

1.8 Organization

This thesis is organized as follows: Chapter 2 reviews literature in background subtraction and obstacle detection. Chapter 3 will go over a geometry-based approach to solve the problem described in section 1.6. Chapter 4 will go over a second approach to solve the problem with the use of deep learning using an ensemble technique. Finally, chapter 5 will go over experimentation with results, and the thesis will be concluded in chapter 6.

CHAPTER 2
Previous Works

This chapter reviews literature related to change detection. We review work focused in two main areas: background subtraction and obstacle detection. For background subtraction in section 2.1, we focus on modelling background scenes using a stationary camera. Section 2.1.1 reviews geometry-based techniques, while section 2.1.2 reviews use of machine learning and deep learning techniques in background subtraction. We continue with obstacle detection in section 2.2 where we focus on monocular vision based obstacle detection methods. Finally, in section 2.3, we describe the relationship between our work and both background subtraction and obstacle detection.

2.1 Background Subtraction

2.1.1 Geometry-Based Approaches

Oliver et al. [1] perform segmentation using an eigen space of the scene. The eigen space is computed using several frames from history to build a mean-image and a covariance matrix. They decompose the covariance matrix using an eigenvalue decomposition. Finally, they use Principal Component Analysis (PCA) to reduce the space, and the eigenspace model is created using the eigen vectors. This method is beneficial for moving objects which do not appear in the same area for long. The model is unlikely to keep the moving object in the model after using PCA, and it allows for segmentation of the background.

Stauffer and Grimson [2] proposed the Gaussian mixture model (GMM) method. Each pixel is independently modelled using k gaussian distributions. This allows several

different events to be modelled when the pixel no longer views the same location in the real world due to jitter or background movement. The pixels belonging to the background will gain weight while lowering variance as more data is assigned to the associated distributions. Thus, any model the foreground produces will have a very low weight and high variance compared to the background models. Overtime, if the foreground remains in place, the data will accumulate in the foreground models by gaining more weight and lowering variance, while the background models lose weight from lack of data through a decaying process. In this case, the foreground will become the background which reduces ghosting. The method requires a history to build the gaussian distributions. Authors in [3] and [4] automate the number of components selected for the set of GMMs for each pixel through a derivation from the maximum likelihood of the multinomial distribution (a generalization of the GMM). Chen et al. [5] build on the GMM method by first using image segmentation and merging regions of consecutive frames. Gaussian Mixture Models are then used to model these regions, and foreground extraction is then performed.

Wang et al. [6] create the method Flux Tensor with Split Gaussian models (FTSG). They combine two techniques for background subtraction using K GMMs and a Flux Tensor approach [7], [8]. Flux tensor is a motion detection technique which is resistant to "complex scenes, lighting conditions and environmental variables" [7], but it requires infrared technology, which is not our focus. This paired with GMM allows them to "handle challenges such as shadows, illumination changes, dynamic background, stopped and removed objects" [6] for the purpose of motion detection. The GMM greatly

enhances this method in combination with other sensors, showing the importance of advancing monocular vision techniques.

In [9], [10], authors implement the sigma-delta method. First, they calculate the median of each pixel based on the given history of input images. Then, the difference is taken between the current frame and the median image to calculate a motion likelihood measure. The variance is computed in a similar way to the median for a measure of the temporal activity. Finally, each pixel is classified as background or foreground based on whether the motion likelihood outweighs the temporal activity. This allows classification of background pixels when there is a cyclic motion in the background such as grass or waves. With background noise cycling at a consistent rate the model will account for that motion in the variance and only classify this movement as foreground when the motion likelihood outgrows the variance.

Mandellos et al. [11] use background subtraction to track vehicles over highways. The method slowly builds a median image as the background model waiting for vehicles to pass through the highway. The assumption is that the background will be present more often than the vehicles over time. This allows the algorithm to isolate the background from the vehicles to create the background model. They further compare the median image to new frames and finetune the segmentation.

In [12], authors proposed the ViBe method. As they describe in their paper, “our proposed technique stores, for each pixel, a set of values taken in the past at the same location or in the neighborhood. It then compares this set to the current pixel value in order to determine whether that pixel belongs to the background, and adapts the model by

choosing randomly which values to substitute from the background model" [12]. The method requires a history of each pixel in order to build the background model.

St-Charles et al. [13], [14] proposed the SuBSENSE method. The method aims to automate parameter changes when considering different or dynamic scenes. They consider the built background model, the current frame, the frame distance, and pixel movement to automatically adjust parameters as needed.

Sedky at al. [15] propose the Spectral-360 approach. They model the background using a physics based dichromatic reflection model. The model assumes physical properties: "there is a single light source that can be a point source or an area source; the illumination has a constant [Spectral Power Distribution (SPD)] across the scene and the amount of illumination can vary across the scene" [15]. They use a history of 60 training frames to calculate the Correlated Colour Temperature (CCT) and the Surface Spectral Reflectance (SSR). They build their model and use the SSR and CCT to compare with new frames using an adaptive threshold to achieve a segmentation.

Wang and Dudek [16] use a history of frames to build K short-term templates and a long-term template in an extended version of the background subtraction algorithm AMBER [17]. These templates are composed of pixel values of the history using a background value and an efficacy counter. The templates are ordered by the efficacy. The long-term template is always first, as they place the values of highest efficacy in that template. To classify pixels, all are considered foreground until a pixel matches one of the K + 1 templates. If a match occurs, the pixel is considered background and they

proceed to update the templates. Ghosting is handled by updating the background model with foreground pixels that have gone unchanged for a period of time.

Zeng et al. [18] propose an algorithm for background subtraction called Background Subtraction with Real-Time Semantic Segmentation (RTSS). They run SuBSENSE [14] as the background subtraction algorithm and ICNet [19] as the semantic segmentation algorithm in parallel to perform background subtraction through a combination of their segmentations. SuBSENSE provides non-parametric background subtraction and they further merge ICNet's segmentation for enhanced results. SuBSENSE uses various functions to achieve a parameter-free design, see Figure 9 for a diagram of the approach. They ([18]) further incorporate ICNet's segmentation into the raw segmentation portion of SuBSENSE, rather than relying entirely on the foreground segmentation (see Figure 10). Combining these two segmentations allow for results exceeding some deep learning approaches.

2.1.2 Machine Learning Approaches

Cheng and Gong [20] propose a generalization of batch learning 1-SVM to classify each pixel as background or foreground. They focus on efficiency and use parallel programing to achieve real time processing at over 80 FPS.

Gregorio and Giordano [21] use weightless neural networks (WNN) (see [22] for original paper on the WNN method) for background subtraction. They use pixels to represent network nodes to classify the background and foreground. They maintain a history of pixels for better classification to account for lighting and other changes. [23] uses weightless neural networks to track motion to allow a robot to follow a leader. The network aims at predicting a window of the subject to follow within the frame.

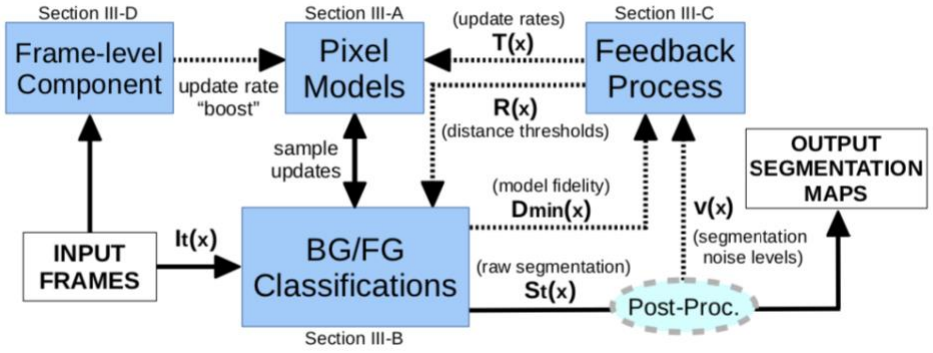

Figure 9. The SuBSENSE approach described in [14]. An input frame is compared to the maintained background model. The background model is maintained through several functions dynamically updating thresholds for segmentation, distance and noise levels. Finally a foreground segmentation is produced and post processing is performed on the resulting segmentation. (Image acquired from [14]).

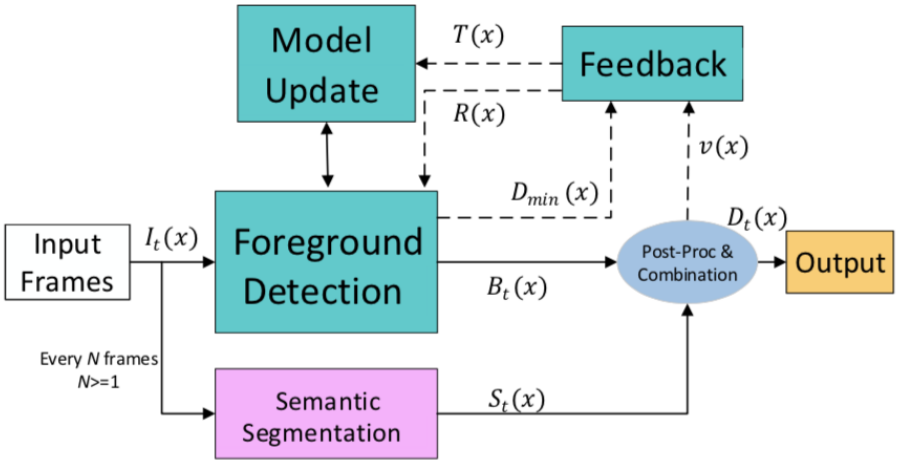

Figure 10. The proposed method in [18]. They alter the SuBSENSE approach in [14] (see also Figure 9) by using ICNet's segmentation [19] of the image along with a foreground segmentation produced by the algorithm to refine results in post processing. (Image acquired from [18]).

Braham and Droogenbroeck [24] perform background subtraction in their proposed method by using a Convolutional Neural Network (CNN). They use 150 frames of a video sequence to obtain a grayscale temporal median. They train the network using a ground truth of sequences labelled by either humans or generated labels using another

background subtraction algorithm. From there the CNN performs background subtraction using the current and median frame. Babaee et al. [25] also perform background subtraction using a CNN with additional post processing to smooth out the resulting background subtraction. They trained their network using the output of SuBSENSE.

Ciocca and Schettini [26] use an ensemble approach to perform background subtraction. They use several different background subtraction methods combined, using a genetic algorithm. They show that a combination of some of the simpler algorithms can perform better than any of the more complex algorithms they tested.

Sakkos et al. [27] use a convolutional network to perform background subtraction without a background model. They use 10 frames as input, but the frames are not concatenated as one input. They use four parallel convolution steps, each with 4 frames as input from a sequence of 10 frames (see Figure 11). The CNN further merges the convolution outputs. An upsampling of the output of the CNN at various steps are used to produce a full resolution background subtraction. Their model is trained on all scenes in the tested datasets, rather than each individual scene.

Authors in [28] use a convolutional autoencoder to produce a background subtraction. The input is the foreground segmentation of three other methods: SuBSENSE [14], FTSG [6], and CwisarDH+ [29] (based on weightless neural networks). The segmentations of these algorithms are encoded in order to extract features of the techniques (see Figure 12). The features are decoded into a segmentation of the foreground and trained on the ground truth segmentation. They show their approach performs better than the individual methods used in the network.

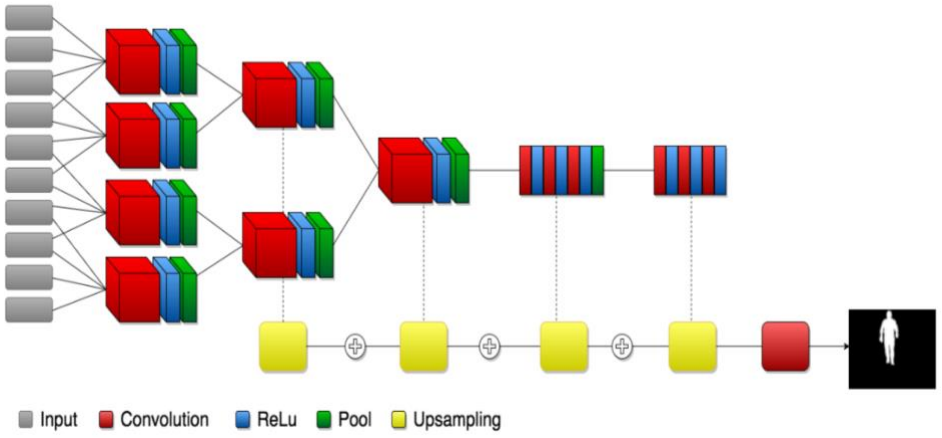

Figure 11. Convolution steps of the background subtraction network proposed in [27]. A series of images are fed into the CNN in parallel allowing for a 3D convolution for background subtraction. Yellow represents upsampling of the segmented foreground with connected outputs from the CNN. (Image acquired from [27]). (Best seen in colour).

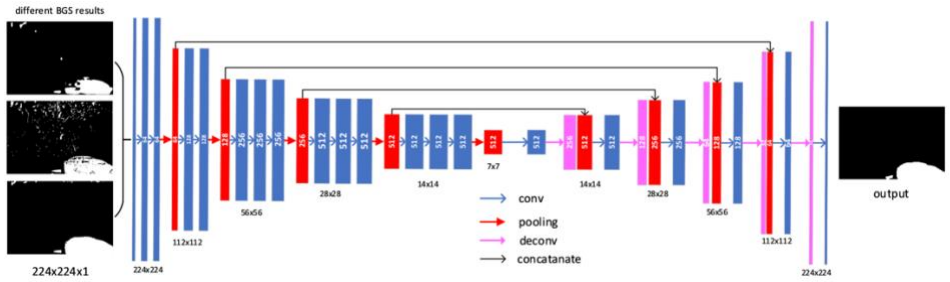

Figure 12. The structure of the proposed convolutional autoencoder used for background subtraction. This approach uses segmentations of three other background subtraction algorithms as input, then encodes them into features. The network decodes the features into a segmentation seen in the output. (Image acquired from [28]). (Best seen in colour).

2.2 Obstacle Detection

Lorigo et al. [30] use a three component system for obstacle detection. Each system uses a different method of detection through brightness, RGB, and HSV. They slide a window over vertical strips of the image to detect obstacle boundaries. Detected edge segments are combined to create a smooth boundary segmentation of the obstacle. Commands are generated for their robot based on the boundaries imposed by the detected edges. They tested this robot in different environments with the camera placed relatively close to the ground.

Ulrich and Nourbakhsh [31] use a ROI to convert the entire frame into a segmentation of the ground in black and possible obstacles in white. They use a histogram of the hue and intensity of the HSI colour space within a ROI covering the direct path of the robot. This models the ground and is then compared to the rest of the frame for classification. Obstacles cannot be present in the ROI and the ROI must be sufficiently large enough to model the ground. This method was tested using a robot in both indoor and outdoor environments. Similarly, Raj et al. [32] also use a histogram of the image to classify the image into ground and obstacle segmentations.

Michels et al. [33] performs obstacle detection in a forest a using remote controlled vehicle. They train a linear regression model to estimate depth of trees using both simulated and real data. They use windows scanned vertically along strips of the image to estimate tree boundaries and obtain controls for the remote car to avoid collision. The locations of trees are determined by their horizontal position in the image and a regression score for depth.

Li and Birchfield [34] train an SVM model to classify vertical and horizontal lines obtained through edge detection. This allows them to generate a segmentation through connected vertical and horizontal lines to determine the wall-floor boundaries. This provides the estimated safe region to navigate, though they do not directly attempt to detect obstacles aside from the walls. The method has difficulty with textured floor patterns, which creates confusion for wall boundaries due to the lines generated from the textures.

Jia et al. [35] perform detection on roads. Obstacles are detected in the bottom half of the image, up to the horizon of the roads tested. They use a two consecutive frames (TCF) approach to analyze feature points and calculate their change between frames. Then, they filter feature points that should be kept through a confidence measure based on the calculated height and distance of each feature point. They show good results in the KITTI dataset at detecting objects by differentiating obstacles from their shadows and other lane markings to avoid false positives through a sense of height and depth.

Souhila and Karim [36] use optical flow in a sequence of images to detect obstacles. Optical flow is the measure of 2D motion induced by a 3D motion in a 3D scene. They calculate several vectors over consecutive frames which describe the movement of pixels through their intensities, which in turn provides the 2D motion. They estimate the Focus of Expansion (FOE) which is the point where the motion vectors are directed away (or expanding from) in the 2D image plane, see Figure 13. They use the information from optical flow to estimate the time to contact and depth information. Finally, the robot will make a decision to turn away from larger flows and attempt to

balance out the amount of flow on the left and right sides of its view. The method relies on the flow of an obstacle to be greater than the flow of a safe region to travel.

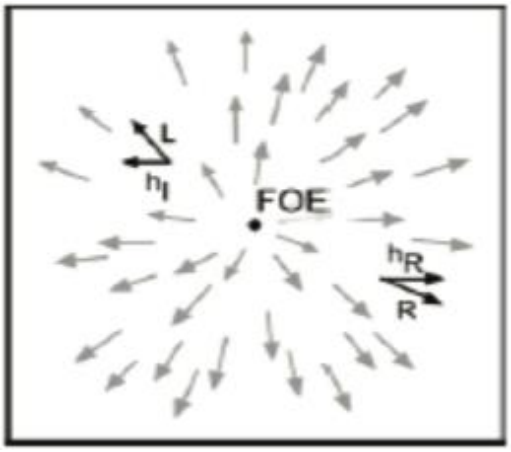

Figure 13. Example FOE given many motion vectors (arrows). The left (L / Hl) and right (R / Hr) motion vectors are estimated and are pointed left and right, respectively, from the FOE. (Image acquired from [36]).

Similarly to [36], authors in [37] use a simple corner detector to obtain feature points in order to reconstruct the scene from optical flow. They use this information to detect obstacles ahead of a vehicle. While driving the car, other cars and motions greatly affect the estimation of the estimated FOE using optical flow. Instead, they use straight lines from the edges of the lane lines of the road where the intersection of the straight lines are estimated to be the FOE, see Figure 14. Using this FOE, they estimate distance information and reconstruct the scene which provides the obstacle information through height and depth.

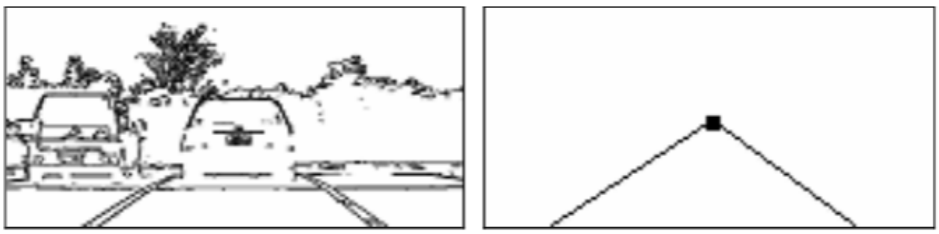

Figure 14. Parallel lane lines are interesting at a point infinitely far away due to camera projection. If we are travelling forward within the lane boundaries, we can expect the scene to expand (roughly) from this point. We can use this point as an estimate of the FOE. (Image acquired from [37]).

Authors in [38] use several images in a monocular camera to reconstruct the scene. The method aims at detection approaching obstacles within a vehicle's rear-view camera. The method has three key steps: first they detect feature points in the series of frames, then they reconstruct the scene, and finally classify points to detect obstacles. They use feature points with constant motion on the ground region and discard feature points with low quality disparity. These feature points are used to estimate vehicle motion. From there, they reconstruct the scene using matched feature points and triangulation equations for depth information. Triangulation is a process where we take a point of an image, and its identical point in another image (another view of the same scene), and calculate the depth and/or third coordinate of that scene point using calibration information of the camera. Each feature point is then labelled as an obstacle or part of the ground features. They use the height of the vehicle's camera as their threshold for obstacle classification where feature points below 20% of the camera's height are considered part of the ground. Finally, they report the nearest obstacle back to the driver.

Zhou and Li [39] assume that the ground plane will have the largest number of feature points. This assumption allows them to estimate the ground plane from the ground features using a ground plane homography. The robot is assumed to be moving on wheels

which mostly stabilizes the camera movement, thus creating a smooth movement in ground features. Then, they calculate a normalized homography of the ground plane using RANSAC. A homography is a matrix which describes the transformation between two views of a scene. In this case, they use a sequence of images from a monocular camera and estimate the ground plane with the homography. RANSAC allows them to search for the dominant plane (of the many possible ground planes) given the outlier obstacle points. A homography can be calculated using 4 points from each plane. As mentioned before, potential obstacles may cause error in the calculations of the homography. Thus, choosing any 4 points can result in a bad homography and a bad estimation of the ground plane. RANSAC searches for the best set of points (or a local minimum) to calculate the best estimate of the homography. Given many points of the ground plane (including obstacles), RANSAC searches for 4 points that result in a homography plane closest to all points, allowing for error and removal of outliers. The dominant ground plane is likely to be chosen, if it contains a significantly large amount of feature points, and the ground can be determined. Once the ground plane is determined, the feature points can be classified as part of the ground plane or part of an obstacle based on their distance from the plane.

Conrad and DeSouza [40] modify the Expectation Maximization (EM) algorithm with the ground plane homography in order to cluster feature points as part of the ground plane or part of an obstacle. EM is an unsupervised clustering machine learning algorithm. This means that there are no associated labels to any data point. The EM algorithm simply aims to find a relationship in the data through a grouping known as clusters. EM groups the data by finding a probability distribution for each cluster through

a two-step iterative process. EM depends on an initial guess of the samples and cluster distributions, then iterates in a two-step process. In the first step, the algorithm proceeds to fit a number of distributions to the labelled data, depending on the number of predefined clusters. In the second step, the samples are assigned to the nearest distribution and relabeled. This process will iterate until the distributions stabilize between the two steps. Finally, the algorithm is left with several clusters of similar samples. In the case of detecting obstacles, they cluster feature points with a modified EM algorithm. The algorithm is designed to cluster the data based on the ground plane homography between two images. Feature points that are part of the ground plane are grouped, while other points are grouped as obstacle points. See Figure 15 for an example of the resulting classification of feature points.

Kumar et al. [41] first calculate point correspondences between two frames using feature points. Then, they calculate the ground plane homography using optical flow and classify features as part of the ground plane, part of an obstacle, or ambiguous. Image segmentation is performed on the image and they further classify entire regions of the segmentation as an obstacle or part of the ground plane. The segmented regions are classified by the majority of labelled points in that region.

Kumar et al. [42] use ground plane homographies using two images and detect superpixels on the image to cluster similarly neighbouring pixels together forming "superpixels." They perform edge detection and filter lines that are not vertical. They further warp the image into a different perspective to filter out lines that are not part of the obstacles, since it is expected that floor lines will not deviate nearly as much as obstacle lines. They use a Markov Random field using the superpixels and edges to

finally classify each pixel as the floor or an obstacle. The Markov Random field is used as an undirected probabilistic graph where the superpixels are nodes, the edges (that were previously detected in the image) are the edges connecting the superpixel nodes, and the cost function is the resulting segmentation. The resulting segmentation outlines obstacles from ground plane.

Figure 15. Feature points classified as ground plane (black) or part of an obstacle (light blue). (Image acquired from [40]). (Best seen in colour).

Lee et al. [43] improve on the proposed method in [42] and use an improved Inverse Perspective Mapping (IPM) approach that is beneficial to camera placements that are low to the ground. Floor models are updated based on edge detections through time

and a ROI is used to classify each pixel as the floor or an obstacle. The system relies on the number of edges, so if at any point the number of edges become inconsistent, they further process the image to detect obstacles. They test a robot vacuum indoors in the same indoor scene for a total precision, false positive rate, and recall of 81.4%, 5.9%, and 74.4%, respectively. Their results improve on [42], which achieved (in the same data) a total precision, false positive rate, and recall of 57.4%, 14.2%, and 37.6%, respectively. However, they needed to modify the algorithm in [42] to fit low camera placements.

Xie et al. [44] use reinforcement learning to train a robot obstacle avoidance. They use two connected architectures to perform depth estimation and produce Q-values in the reinforcement learning Q-network. The Q-values provide information on the actions of the robot, such as linear action and angular action. The input to the reinforcement model is the output of a depth estimating CNN. This CNN takes an RGB image as input and produces a depth map of the scene as output. See Figure 16 for the overall structure of the model. Training takes place in a simulator and testing takes place in the real world, however, they needed to add noise, by blurring part of the simulated input images, for better results in the real world. They test the robot in a few scenes and show some examples of the robot circling around a room filled with randomly placed obstacles. Similarly, authors in [45] use 4 consecutive depth maps and a CNN trained through reinforcement learning. The CNN has two convolution steps and is finally fed into a fully connected neural network. The resulting outputs are 8 Q-values where 3 are actions for linear velocity and 5 are actions for angular velocity. The network is trained in simulation, though the depth maps are acquired through a depth sensor rather than a CNN, as in [44].

Authors in [46] use a horizon line of the image to detect obstacles for small flying vehicles that know their altitude. The idea is that the horizon line will intersect obstacles at the same height of the robot which will provide an uncertainty level applied to the obstacle above and below the horizon line. Note that this means obstacles completely below or above the horizon line will not be considered as obstacles. The ground and sky are less likely to intersect the horizon line and a higher certainty is labelled to those pixels. The uncertain pixels are classified as obstacle points and the certain pixels as non-obstacle points using a random forest classifier. See Figure 17 for an example segmentation of obstacles in the KITTI dataset.

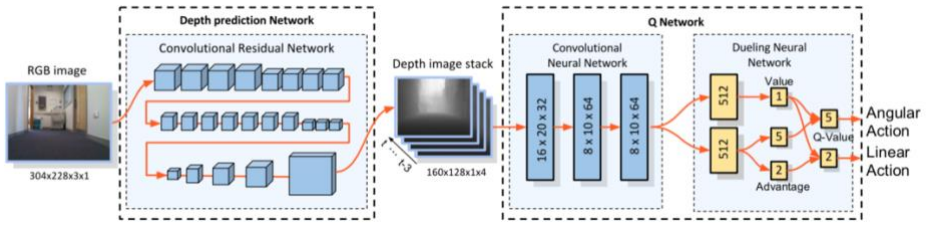

Figure 16. This is the architecture of the convolutional neural network described in [44]. They use a depth prediction CNN and a Q-network to produce actions based on the resulting depth. The network is trained through reinforcement in a simulation. (Image acquired from [44]).

Xue et al. [47] detect obstacles at a far distance, see Figure 18. They use a combination of an occlusion edge map and a regressor. The occlusion edge map is generated by using a far-to-near approach to estimate edges at different distances in the image/scene, with a fusion of edges in a near-to-far approach of the same image. This allows detection of edges around obstacles at many different distances in the scene. Finally, superpixeling is performed to completely fit edges around the small obstacles at different distance levels. They use random forest to generate a high regression value for obstacles based on four features: (1) Edge and Structure, (2) Pseudo Distance, (3)

Objectness Score (calculated likelihood a box contains an obstacle), and (4) Colour. Using these features they are able to predict obstacles at various distances in an image.

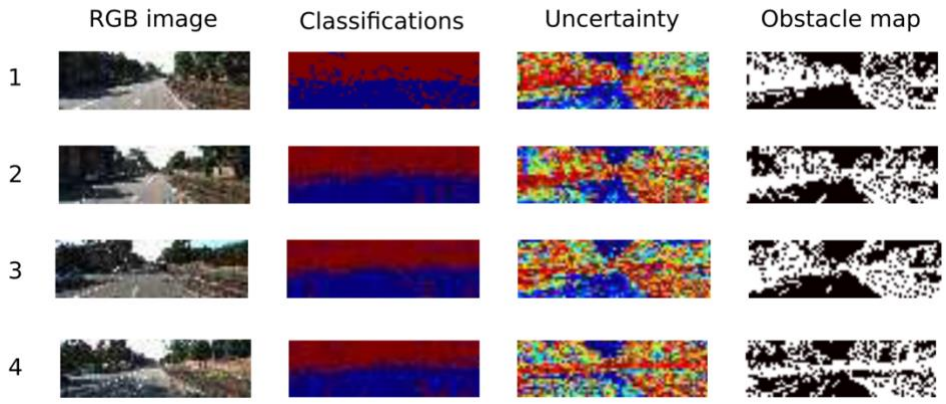

Figure 17. Example obstacle segmentation in the KITTI dataset using the horizon line approach described in [46]. They provide an example for each row. From left to right, columns show: an input RGB image, a classification of the horizon, an uncertainty map of potential obstacles that intersect the horizon line (red is obstacle, blue is non-obstacle) and finally an obstacle map where white is obstacle and black is non-obstacle. (Image acquired from [46]). (Best seen in colour).

Figure 18. Example of the type of obstacle the authors in [47] aim to detect. The obstacle is placed at a far distance and is relatively small in the image. The obstacle is highlighted with a red square as seen in the zoomed version in the bottom left corner of the image. The image is from the Lost and Found dataset. (Image acquired from [47]). (Best seen in colour).

2.3 Relation to our Work in Change Detection

Our work in change detection uses components from both background subtraction and obstacle detection fields. At the same time, our work is not completely part of either field. In this section, we will go over the differences and relationships of our work to both background subtraction and obstacle detection.

First, background subtraction methods aim to model the background of a scene using a stationary camera. We also aim to model the background of the scene (of the path we are travelling), though the camera is not stationary. This is a key difference in both approaches since background subtraction techniques have the opportunity to use hundreds of frames of the same portion of the scene and have a large description of the dynamic changes of each pixel, such as lighting and background motions. Our work uses a single video of the current path travelled resulting in just a few frames of information for a given portion of the scene.

Finally, monocular vision obstacle detection methods do not use a training video of the path. Using a video of the scene allows us to differentiate, with a much higher confidence, important changes from general obstacles that are part of scene. This is an advantage in robots that travel known paths multiple times to perform a task. The robot is assumed to know where it is travelling and we simply aim to detect sudden changes in the path, which could harm either the robot or the person and/or object in the robot's direct path. Another key benefit to the use of a training video is the ability to determine the robot's current location within the path. We can match the robot's current view of the scene to a location within the training video. This can alleviate the need for a Global

Positioning System (GPS) and allow positioning in environments lacking a GPS signal, such as basements and tall buildings.

CHAPTER 3
Geometry-Based Methodology

In this chapter we present our methodology for the geometry-based approach with the use of vision techniques. In section 3.1, we explain the motivation of the method with a slight overview of the approach. In section 3.2, we explain the details of the methodology of the geometry-based approach.

3.1 Motivation

In this approach, we use geometry-based techniques to detect changes along a path. The method is split into two main steps. First, we analyze a set of test frames in order to determine our current location within the training video. This provides us with a neighbourhood within the training video of possible matching training frames to the given test frame. The second step is to have the robot proceed and start detecting obstacles. Changes are detected through a comparison of the current testing frame and the best matching training frame. The search for the best matching training frame is found within a few seconds of the current location within the training video. See Figure 19 where we provide a visual for the setup of the comparison between both matching frames. The comparison uses histograms of a ROI to detect a change in the variance of the colours in both frames. The robot may halt if a detection occurs. Otherwise, we proceed with the next test frame and shift the neighbourhood to center around the previously matched training frame in search of the next match.

Figure 19. We visualize how two frames are set up for comparison. The test frame with changes (on the right) is registered onto the best matching training frame (on the left). This aligns the scenes in both frames. Then the ROI can be placed in the same area of both frames overlooking the same portion of the scene. (Best seen in colour).

3.2 Geometry-Based Approach

A high level pseudocode is presented in Algorithm 1. The algorithm only presents two procedures, one to determine the current location, as seen in section 3.2.1, and the other is the overall change detection approach, once a location is determined. The algorithm runs the first procedure only one time, then the second afterwards. We discuss first how the location is found in section 3.2.1 and how we match frames in section 3.2.2. From there, we explain how we compare matching frames through frame registration in section 3.2.3. Finally, the process to detect a change is explained in section 3.2.4.

3.2.1 Determining Current Location

Initially, the robot must determine its current location within the given training video, essentially positioning itself within the path. The robot may be halted during this process since we have no information on possible path obstructions. Once a location is determined, we can start change detection from the first frame. The process of determining the initial position of the robot is a one-time step and does not need to be performed once detection starts. The location will be updated as the robot travels through the path.

In order for the robot to position itself, we consider a set of N initial testing frames. In our experiments, 2 frames were enough to determine a reliable location. Choosing N greater than 2 allows for more confidence in the current location for dynamic and moving background elements. For each of the N frames, we perform a grid search along the entire training video to find their respective best matching training frame. We define the best matching frame as the most similarly positioned training frame through a compromise between real world location and camera angle. Neither of these metrics are available to us directly, so we perform frame matching through feature point registration which is further explained in section 3.2.2. Finally, once all frames have their respective matches, we consider the median position within the training video. This is the most frequently occurring position and is the probable area of the training video within the path on which we are currently positioned.

Algorithm 1 Algorithm

```
1: procedure DETERMINELOCATION
2: loop:
3:     TestFrame1 ← TestSet.getNextFrame()
4:     TestFrame2 ← TestSet.getNextFrame()
5:     [Note: Search entire training set, compare frames using feature points]
6:     MatchingFrame1 ← TrainingSet.findMatch(TestFrame1)
7:     MatchingFrame2 ← TrainingSet.findMatch(TestFrame2)
8:
9:     if distance(MatchingFrame1, MatchingFrame2) < Range then
10:        return MatchingFrame2.getPosition()
11:        close;
12:
13:    [Note: No valid match]
14:    TestFrame1.setDetection(true)
15:    TestFrame2.setDetection(true)
16:    goto loop.
17:
18:
19: procedure CHANGEDETECTION
20: loop:
21:    TestFrame ← TestSet.getNextFrame()
22:    [Note: Search training in smaller range]
23:    MatchingFrame ← TrainingSet.findMatch(TestFrame)
24:
25:    if matchingFrame.empty() then
26:        TestFrame.setDetection(true)
27:        goto loop.
28:
29:    [Note: Register frames using their feature points]
30:    RegisterFrames(TestFrame, MachingFrame)
31:    if AreaOfROI(TestFrame) is too small then
32:        TestFrame.setDetection(true)
33:        goto loop.
34:
35:    [Note: Compare histograms of the ROI of each frame]
36:    Result ← CompareROI(TestFrame, MachingFrame)
37:    TestFrame.setDetection(Result)
38:    goto loop.
```

Once a location is determined, this allows us to significantly reduce the search space to a very small neighbourhood of the training video centered on the determined location. Given this neighbourhood, we can perform frame matching starting from the

first testing frame. Once a match is found, in order to keep the neighbourhood moving with the robot, we center the neighbourhood around the new match location. This gives us the next search space for successive testing frames. The update of the search space reduces the probability of mismatching a frame, since most frames within the neighbourhood are expected to match the next test frame. The range of neighbourhood depends on the speed of the robot. For example, if the robot has a normal smooth pace, then it will only require a second of the search space centered around the determined location. However, at higher or inconsistent speeds (in relation to the speed of the training video) the range should initially remain 3-4 seconds and reduce or extend overtime. This essentially acts as a prediction of the next position based on the movement of previous positions of matched frames in the training video.

See Figure 20 where we show the change in match score for every single frame using just two consecutive frames. A large downward spike in the error is shown where the matches are correctly found, thus using a grid search algorithm with a low enough error threshold would allow us to avoid comparing every frame and limit the search to areas that are below the error threshold. We show in Figure 21 and Figure 22 how to obtain scene recognition using this approach using all of our tested scenes. The same spike in error is shown for the correctly matched frame within the correct scene. We repeated this test for every scene and each was correctly determined.

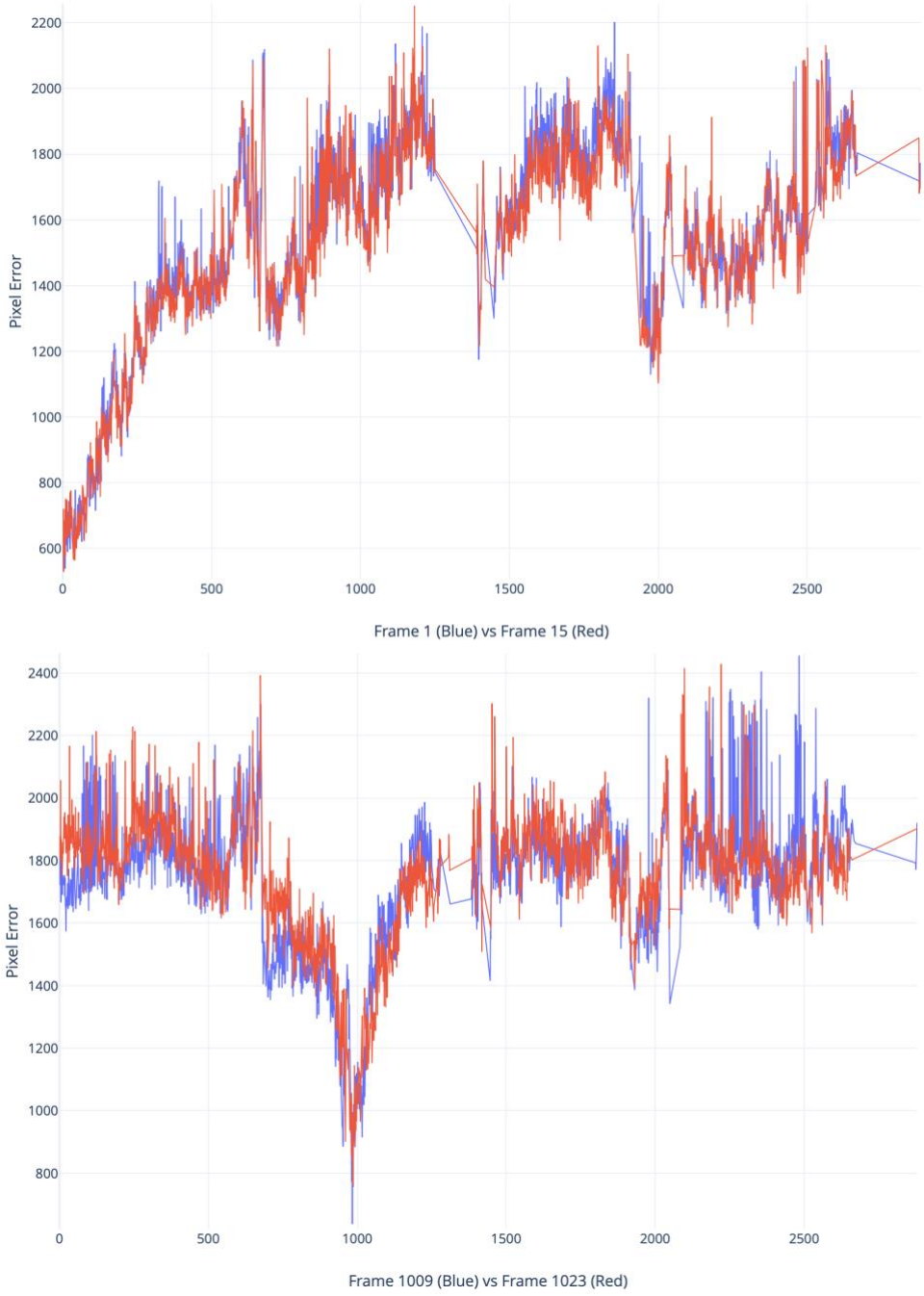

Figure 20. We plot the matching score (Y-Axis, pixel error) of two frames (red and blue lines) when compared to every frame in the training set (X-Axis, frame number). The top graph shows our search for a matching frame that would be located at the beginning of the training video. The bottom graph shows a similar comparison using the same video, but we search for a matching frame near the middle of the training video (at frame number 1000 roughly). (Best seen in colour).

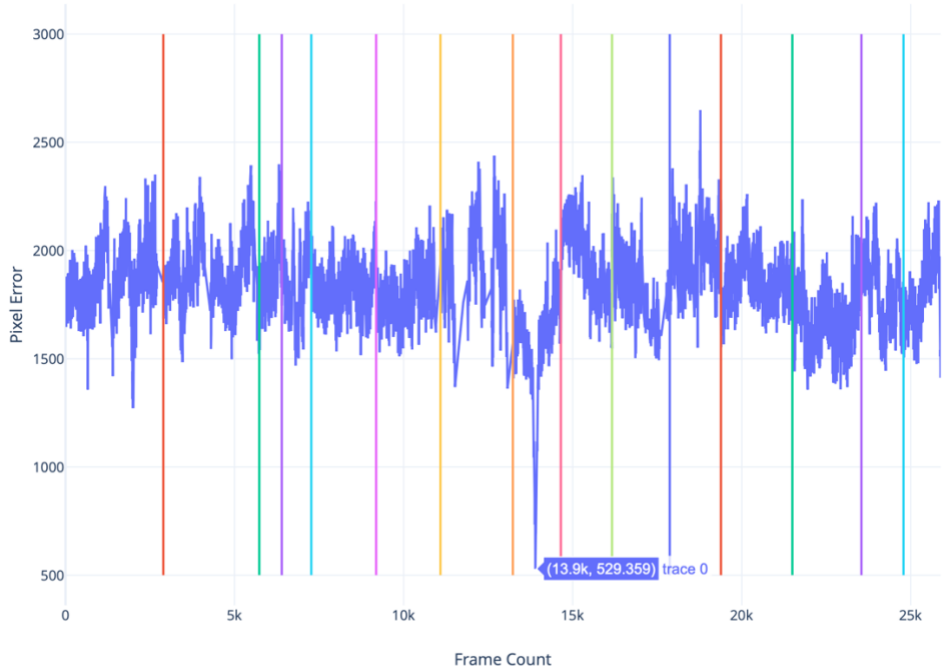

Figure 21. We combined all of our training videos of various scenes, each scene is separated by a vertical line. We search for the best matching frame roughly centered in the eighth scene. (Best seen in colour).

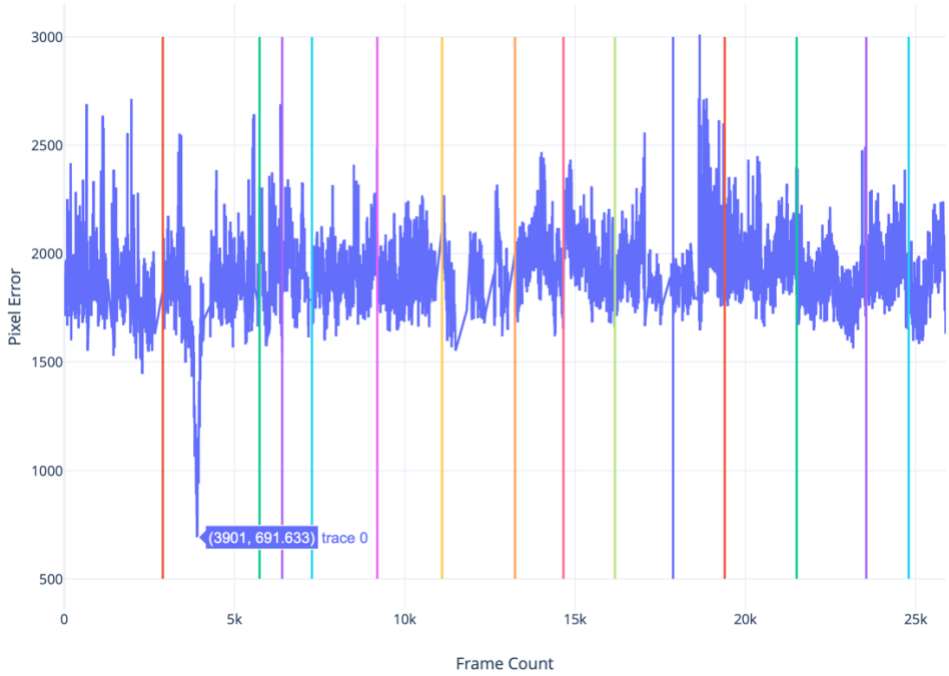

Figure 22. We combined all of our training videos of various scenes, each scene is separated by a vertical line, similarly to Figure 21. We search for the best matching frame roughly centered in the second scene. (Best seen in colour).

3.2.2 Frame Matching

We perform frame matching by feature point registration. We use the feature point detector ORB [48] to extract feature points from all the training frames and testing frames processed. Feature point extraction of the training set can be performed prior to testing through preprocessing.

We tested many feature point algorithms such as ORB, SIFT [49], and SURF [50]. ORB has no license restrictions, unlike SIFT and SURF, and is shown in their paper [48] to perform two orders of magnitude faster with similar results. In our experimentations, ORB performed significantly faster than both SIFT and SURF and allowed us to match frames with similar accuracy.

A feature point algorithm will generally use keypoints and descriptors to detect a feature point. The descriptors are used to describe a small region of the image based on the keypoints detected and the algorithm's methodology. A feature point is a point of interest in an image such as a corner where two lines meet at an angle. Thus the feature point descriptor will describe this in terms of features, ideally invariant of scale and orientation. The algorithm takes into consideration the descriptors and choses the best as feature points. ORB uses FAST [51], [52] to detect keypoints and BRIEF [53] for their descriptors. These algorithms are chosen for their speed. Unfortunately, FAST and BRIEF do not account for rotation, so the authors of ORB created versions of these algorithms to allow for rotational invariance. Finally, a brute force matcher can be used to compare feature point descriptors and allow for matched points between frames.

Given a testing frame and a training frame, we assign a score to how well they match. This score is calculated by the Euclidian deviation in matched feature point

coordinates (between both frames) after performing registration. In other words, both matching feature points have an X and Y coordinate within their respective frames. If the scene did not move between the frames, they would have the same (X, Y) coordinate and the Euclidean distance would be 0. Otherwise, error is introduced and the movement distance contributes to the match score. To do this, we use a brute force matcher (explained earlier) to match feature points between the two frames. We then remove large outlier feature point matches that have matched in a different region of their respective frame. We used a distance of 40% of the frame's diagonal length as a distance threshold. We do not expect feature points on the ground to match points on the ceiling, for example.

We now perform the feature point registration through an SVD routine. We can consider the feature points of each frame as a cluster of points in a 2D Euclidean space through their pixel coordinates. Each point will have a connection to their matching point in the opposite frame. We first translate the clusters so that their mean is positioned at the origin of their respective space. To find the best rotation of the clusters, we calculate a covariance matrix of the point connections. Note that we do not adjust for scale since this is one of our frame matching constraints where we want to keep a sense of positioning within the path. Using this covariance matrix, we can perform SVD to decompose the covariance matrix into three matrices where the outer decomposition is made up of eigenvectors and the inner matrix is a diagonal matrix of eigenvalues. By removing the diagonal matrix we can use the resulting matrix to find the best registration between the two clusters. Now that we have found the best translation and rotation of the clusters, we can register one cluster onto the other. The clusters will not match perfectly and will

leave some error due to feature points that have moved with the scene or potential obstacles, and changes mismatching with scene points. Given this new registration, we calculate an error by an average of the Euclidean distance of the matching feature points. Any error function may be used here such as MSE and other error based functions.

Finally, the best matching training frame can be determined using this score. The lowest score is considered the best match. The score will be at its lowest when the scene is in the best compromise between camera angle and scene location. This way we may perform frame registration which is explained in the next section.

3.2.3 Frame Registration

Once we find the best matching training frame for our given testing frame, we compare the frames for potential changes. Before performing change detection, we register one frame onto the other. That way, the scenes will be aligned and comparison can be performed though a ROI of the direct path, to be explained in more detail in section 3.2.4.

To perform frame registration, we can use the same feature point matches we calculated when we performed the frame matching process. Using the original points on the frames and their connection to points on the opposing frame, we can calculate their homography using RANSAC. A homography is a matrix that describes the transformation of one view of a particular scene to another view of the same scene. A homography requires just four matched points between both views to describe the transformation of every pixel in the view. A homography is built for the transformation of one plane to another given four matching points on the plane. It is not always possible to perfectly matched feature points, so RANSAC allows us to search for four candidate

points of an entire set of matched feature points. RANSAC will attempt to find a homography that best suits the majority of points in the space.

Here we consider the matching frames themselves as 2D planes (of pixels) and describe the transformation of the entire test frame to the training frame using previously matched feature points. We can use this homography to perform a perspective transformation on every pixel of one frame and register them onto the other frame. This will align the scenes between the frames where we can directly compare the frames at a pixel level, explained in the next section.

3.2.4 Change Detection

In the previous section, we registered the testing frame onto the training frame using a perspective transformation which will align the scenes of each frame. The resulting frame from registration will have the same dimension of the original frame with the transformed frame cut off where it moves beyond the frame borders and blacked out where no pixel information is available. This way, we can place a ROI in the exact same location of both the training and testing frame to overlook the same area of the scene. We used a quadrilateral ROI positioned in the direct path of the robot for detections near the ground level.

Comparison between the test frame and the training frame are now reduced to a comparison of the ROI of each frame. We use a histogram comparison of the HSV colour space between the ROI of both frames. A direct pixel comparison of both ROIs performs poorly. This is due to inaccuracies in the frame registration and slight movement in the scene. To overcome these inaccuracies, we create a histogram of both of the ROIs (using the pixels from the HSV colour space) and compare the histograms for possible changes.

This comparison works well for large changes, but smaller changes go undetected since they do not produce enough of a change in the histogram to overcome the background noise of the scene. We want to detect all changes, large and small, so we split the ROI into a grid system. That way, we compare a histogram of each grid region with the same region of the other frame's ROI. This allows for detection of both large and small changes since the regions are small enough to detect small changes while detection of large changes is unaffected by the change in size of the area.

If we consider a single grid region of the ROI from our grid system, we can compute two histograms: one for the training frame and the other for the testing frame. We compare the two histograms by first taking their absolute difference. This will subtract each entry of the histograms and take their absolute value, resulting in a new histogram with positive entries. We then compute the variance of the resulting histogram. We repeat this three times for the components of the HSV colour space: one for the hue, one for the saturation, and one for the gray scale component.

All three components will provide a variance for their respective histograms. To reduce noise, we consider a threshold for each type of histogram. After performing a comparison, if the value meets the histogram's respective threshold we keep the value unchanged. Otherwise, the value is reset to 0. Given three comparison values for a grid region, we must make a decision on whether we see a change. This is done by summing all three variance values and using a final threshold to make the classification.

In machine learning terms, the first three thresholds can be considered as feature detectors. We do not require these thresholds to be updated on every scene change as they

are meant to detect features, but can be optionally changed. The final classification threshold requires training to optimize the detection rates, thus requiring a validation set with a few test obstacles. Adding weights to the comparison values can extent the approach to a small neural network design, though we do not have any direct training data for additional weights, so this was not included.

Finally, each testing frame can be processed to perform change detection with their respective matched training frames. To sum up this approach, we first find the robot's current location. We reduce the search space to a small neighbourhood of this location and start to perform change detection. Change detection is performed in three steps: first, by finding the best matching training frame, second, by registering the frames and aligning the scene within the frames, and third, by a comparison of both frames through a direct comparison of their histograms within a ROI.

CHAPTER 4
Deep Learning Methodology

In this chapter we present our methodology for the deep learning approach with the use of an ensemble of deep learning models. In section 4.1, we explain the motivation behind the method and attempts taken to solve the problem. In section 4.2, we explain the methodology of the deep learning approach.

4.1 Motivation

In this approach we aim to solve the problem through the use of machine learning or deep learning methods. The idea is to simply extract feature points from the change-free training video and train a model to learn the samples as a one class classification. The model would classify feature points from test frames as either part of the learned class or part of an unseen class, such as an obstacle or a change. This type of approach would allow for training without labels and classification without seeing any obstacles or using a validation set for parameter tuning.

In our first attempt, we tried to use a 1-SVM, which is built to learn a single class and detect outliers as part of another unseen class. We first extracted feature points from the training video and trained the model using the feature point descriptors as the training samples. The difficulty with 1-SVM is a parameter that needs to be tuned in order to determine the sensitivity to outliers. This parameter was a threshold for the percentage of the training data to be considered outliers. We could consider the entire training set as part of the class, but this causes overfitting problems. Unfortunately, a threshold slightly larger than optimal will cause the model to learn a very large space as the class and detect

nothing during testing. Similarly, a threshold slightly too small will cause the model to detect the background as a change and perform more randomly than accurately. Considering the difficulty of finding the optimal threshold automatically, we decided to try other models.

In our next attempt, we considered using a deep learning model known as a Generative Adversarial Network (GAN). The feature points did not provide enough information about the area to be useful in training our deep learning model and did not provide us with any information on the quality of the generated samples. Instead, we took a small sample image for every feature point as the feature point neighbourhood. The sample images are centered at their respective feature point within the frame. These samples were used as training data for the GAN. Once trained, the model learns two important utilities, one through the generator and one through the discriminator. The first utility generates sample images with similar features to the training data through the generator. The second discriminates samples between real and generated images known as the discriminator. The generator provides us with validation that the model has learned some features from the image by comparing the generated images to real samples. If the generator has learned to generate similar images to the training data, it is likely that the discriminator has learned how to discriminate the learned features. We use the discriminator to perform the classification where each sample image of a given test frame will be labelled as generated or real. Ideally, obstacles would look different from the learned samples and cause the discriminator to classify them as generated, or in other words, as a change. Background samples would be classified as real since they should be indistinguishable from the generated samples. Unfortunately for this approach, training

GANs is known to be difficult and automating this process for each training video is not feasible. The instability of the network causes the generator to learn static noise more often than not, so we decided to try other methods.

In our final attempt we replaced the GAN with a simple autoencoder. The autoencoder was able to learn features automatically between scenes, but with the drawback of the missing discriminator for classification. Instead, we used the encoder portion of the network along with the generated features to make predictions. Classification decisions on testing samples is performed using a distance function. The distance function allows us to compare the features of the test sample to the features of a training sample. We compared a test sample to each training sample to find a "match" using the distance function and a threshold. If a match is found, we consider the test sample part of the background, otherwise it is a detected change. This method showed promising results where there was very few false positives, but with the drawback of a low true positive rate. Essentially, the model found some obstacles with a low precision. Unfortunately, we have a threshold which cannot be determined automatically, needing a validation set and either labels or a human expert.

In section 4.2, we will describe our final deep learning approach which solves these problems and alleviates the need for threshold tuning and training labels by using an ensemble of unsupervised models.

4.2 Deep Learning Approach

4.2.1 Ensemble of Unsupervised Models Introduction

Our deep learning approach incorporates several unsupervised dimensionality reduction models using a majority vote as the final decision for classification. This ensemble technique relieves us of the need for manual parameter changes through deep learning and the need for validation data for parameter tuning through the use of a final vote.

The overall approach is as follows. We must first choose the model(s) and build the approach, described in section 4.2.2. We choose the model(s) structure and parameters which will go unchanged in all future tests, so that the structure and parameters of all components will remain fixed (unless dynamically updated by the system) for all future training and testing, regardless of scene change. Next, in section 4.2.3, we describe the necessary preprocessing and training involved once a training video is provided. Finally, in section 4.2.4 we describe the testing process and how we detect changes.

4.2.2 Model Structure and Parameters

The deep learning method uses several unsupervised deep learning models which together provide information for a classification vote. We use a majority vote of all the built classifiers to determine a final decision for any change detections. The method avoids any parameter tuning or labelled validation data for scene learning and testing.

First, the structure of the method must be determined for future use. We must define the number of models, their structure, and decision thresholds. The method is adaptive, so additional models will not affect previously built models and will only contribute towards the final vote. A good model for an ensemble approach is a classifier

that is more accurate than random. We chose a model through a small test scene, unrelated to any other test. This test scene is not used for parameter tuning, but to evaluate its performance and determine if it is a good candidate for the ensemble technique.

For example, we just need to determine if the model is worth incorporating into the approach. In our specific experiments, we found two models which had a high true negative score, but did not detect all changes. Both models demonstrated a good ability to reproduce most large features from the images. Combining these two models will allow for a more precise detection of many more changes compared to individual use. The more models used, the more accurate the method is at predicting change detections, and the more precise detections become. In this case, model structure or type is not important, but rather its ability to detect meaningful features and be relatively accurate. We fixed two models, one is an autoencoder and the other is a convolutional autoencoder. Both models are small in the number of parameters used. Their decision thresholds will be based on a mean and standard deviation of both of the training samples and the learned features, not from the test scene.

4.2.3 Preprocessing and Learning

Once a training video is provided, our approach first extracts feature points viewed in a ROI from every frame, see Figure 23 for a visualization of the set up. The goal of the approach is to first learn the class of these feature points and to classify unseen points during the test phase (obtained in a similar ROI) as obstacle or part of the learned scene. This ROI is flexible and we can consider widened, shrunk and/or different positioning of the ROI. We could even consider multiple ROIs in different sections of the frame each

with different models trained and focused on those particular areas. Note that our experiments only consider the most simple case of using the bottom quarter of the frame as a ROI. If we considered using another ROI, an additional ROI on the bottom third quarter of the frame could be used for further validation of any seen obstacles. The second ROI would start at the midpoint of the frame and extend down to the start of the original ROI (at the horizontal white line in Figure 23), allowing for two detection systems at different distance levels.

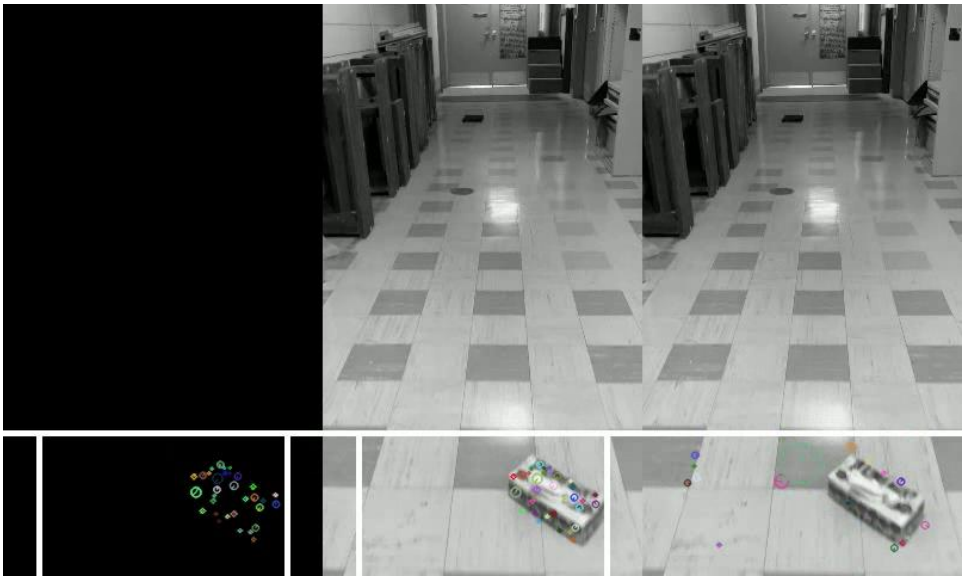

Figure 23. Visualization of the set up for the ensemble approach. We do not show a training frame here, but training samples are taken from the entire bottom quarter of the image as seen by the horizontal line on the test frames. During testing we reduce that area on the left and right edges shown with the vertical lines. Together they build the ROI of the direct path. We show three frames: the right shows feature points classified as background, the middle frame shows the feature points classified as a change and the black frame on the left helps visualize the same obstacle points which may be hard to see in the middle frame. (Best seen in colour).

Once feature points are detected in the training frames, we attract the training data from them. We extract a neighbourhood of each feature point as a small image of the frame. The size of this sample image would be configurable to each of the models for any

sample size. Note that the sample size is chosen during the model building phase seen in section 4.2.2, which will go unchanged once a final model is chosen. This is not a tunable parameter and will remain fixed with its respective model structure for the life of the approach or until the model is removed from the ensemble.

In the general case of the approach, we consider N dimensionality reduction models. These are unsupervised models and do not require labels, so the training data is enough to learn. In our design, we used N=2 models where one is a simple autoencoder and the other is a convolutional autoencoder. See Figure 24 for an example of the sample images regenerated from these models. Since we used two models, future models can be added without ever modifying the existing models and they will simply contribute another vote to the final decision mechanism. Finally, each model is trained on the extracted data and learns features through dimensionality reduction.

A classification of a sample is performed using a distance function. We aim to classify a test sample through a comparison of a set of automatically selected training samples. At a high level, if one sample from the training set is "similar" to the test sample, it is classified as part of the scene. Otherwise, if the test sample is "dissimilar" to every sample in the training set, then it is considered to be a change. Note that we actually reduce the training set to avoid large computations from these comparisons by removing redundant training samples.

We define the "similarity" between samples as the Mahalanobis distance of the models' encoded features, called a similarity score. Each model has their respective distance function which provide a similarity score between two samples. All that is

required for the Mahalanobis distance is a covariance matrix and two samples as vectors for comparison.

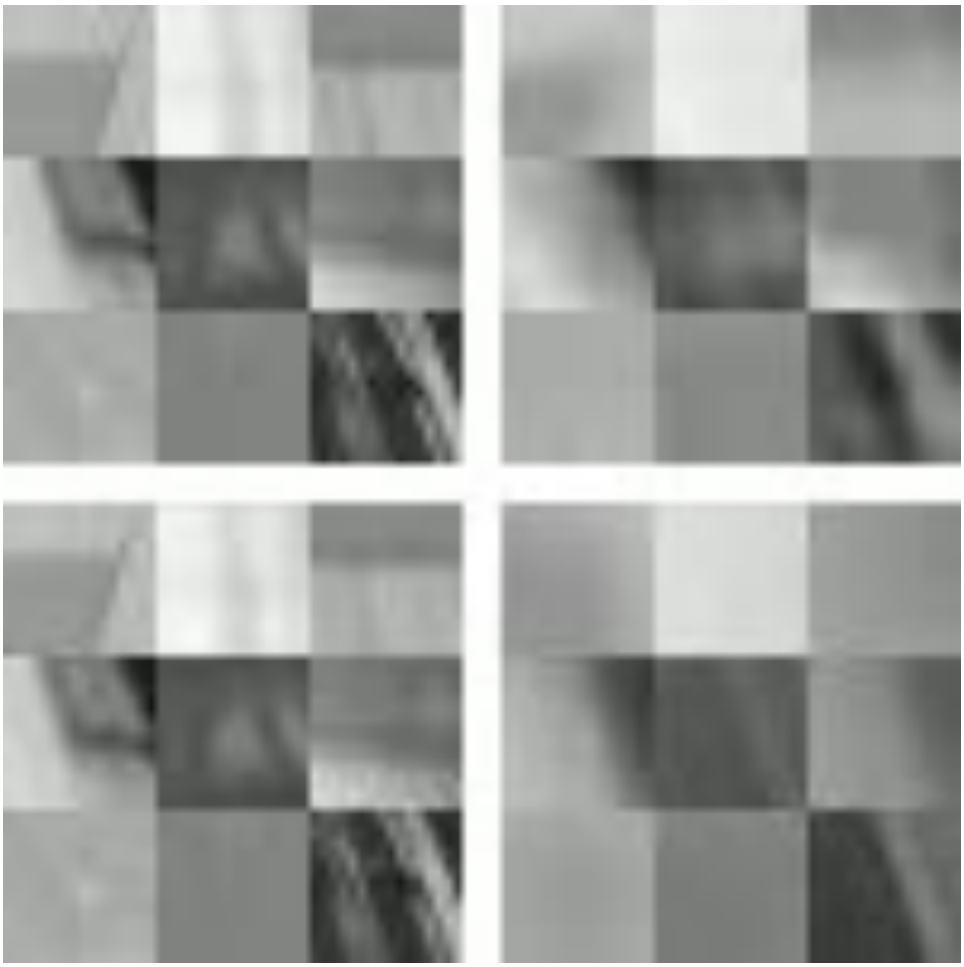

Figure 24. We show a grid of images and four 3x3 collections of samples. The top left and bottom left sections are exactly the same for visual comparison with the sections on the right side. They are the 9 training samples provided to the autoencoders. The area on the top right is the regenerated samples from the first autoencoder (after encoding each sample and decoding the features). The area on the bottom right is the regenerated samples of the convolutional autoencoder.

The vector samples are simple. We use the distance function's respective autoencoder to encode the frame samples into a vector of features. Now, we must build the covariance matrix for each distance function. For each model, we first encode all of the training samples into feature vectors. Then, the encoded samples are used to build a

covariance matrix. The encoded samples will describe the feature distribution of the path allowing for a description of the covariance of the model with respect to the path. Together, both the covariance of the feature space and the encoded samples are used to build the similarity score. Thus, each model will have to build a distance function by first training, then encoding the training samples as vectors, and finally calculating a covariance matrix of the sample features.

Once each model has a distance function built, we need to define a threshold over the similarity score to determine a proper classification decision (or particular vote of that model). With one model, we would usually need validation data and an expert with labels to find the optimal threshold. With an ensemble approach, the voting mechanism allows room for error and lenient thresholds for the individual models that make up the ensemble. To determine a threshold automatically, we use the encoded training set for this information.

First we calculate the mean of the feature space as a mean vector. This vector represents the average encoded sample. From there, we calculate the similarity score of each training sample to the mean vector. That way we end up with a list of how different each sample is from the average. We can compute the mean and standard deviation of the list of similarity scores to describe the expected variation of the data. We determine that two samples are relatively similar if their similarity score is below $T_J = \mu_J + \sigma_J$. T_J is the threshold of the model J which is calculated from μ_J (the mean) and σ_J (the standard deviation) of similarity scores. In other words, we can capture the majority of the space as part of the scene, by using the mean and standard deviation of the similarity scores, while outliers will be considered as obstacle. Ideally, this will allow us to reduce the

training set for each model to a few candidate training samples, which describe their portion of the feature space.

This distance threshold captures a large portion of the space, but not the whole space. In this case, the advantage of using an ensemble approach is the ability to capture many portions of the feature space with different models. However, this would require many more models to fit the whole space. So, rather than extending the approach with more models, we extending the range of each model. This is why we compare a given test frame to the many samples of the training set. Each training sample can represent a portion of a model's feature space. A test frame is likely to be similar to one of the many training samples when using the threshold T_J that captures a large portion of the space.

However, including many training samples for comparison is costly. We do not want to compare each test sample to every training sample, so we reduce the training space into a small set that we call a comparison set. We remove redundant training samples using the previously described similarity functions and respective threshold T_J. First, we pick a sample at random. Then, we remove all other samples from the set that match the chosen sample using the similarity function. We continue this process until either the set is exhausted or we reach the limit of M samples, configured to each model during the building stage described in section 4.2.2. In our experiments 200-1000 samples were sufficient for each model depending on the complexity of the floor pattern. The samples are ordered based on the number of redundant samples found and chosen from largest to smallest. Outdoor floor patterns have many more small features and inconsistent patterns, compared to indoor environments, which require more comparison samples to retain the large amount of information. Depending on the scene, an indoor

space will require 200-500 samples while an outdoor scene will require closer to 1000 samples. If even more samples are required, this would result from a lengthier video with a change in scene or floor patterns. In this case, it is best to add additional models rather than more comparison samples. The method will automatically adjust the number of samples needed, with a hard limit of 1000 samples, based on the reduction of the space as mentioned earlier. If more models are used, less comparison samples are needed for each model. In our case we used 2 models which require more comparison samples to retain the background model, especially in outdoor environments.

Finally, we will sum up the preprocessing stage. First, we extracted feature points from a ROI of each frame in the training video. We then extract frame samples around each feature point used as the training data. We train N=2 unsupervised dimensionality reduction models using the extracted frame samples. From there, we build the similarity function of each model using the model's encoded training samples. Finally, we remove redundant training samples using the built similarity function and reduce the comparison set to M samples (for each model). All of this is automated and done through preprocessing. In the next section, we will explain how we evaluate a test video with potential changes using our ensemble of unsupervised models, along with their respective comparison sets and similarity functions.

4.2.4 Evaluation and Postprocessing

Processing of obstacles and changes can begin from the first frame within a similar ROI to the training set. We reduce the left and right boundaries of the ROI to account for large horizontal movements. Similarly to the training frames, we extract feature points from the

test frames, and further extract the frame samples centered at each feature point. We want to classify each sample as a change or part of the background.

Now, we will explain how a single test sample will be classified. For each model, we will encode the sample into features. That way we are left with N vector representations of the sample. For each representation/encoding, we have a comparison set designed for the specific model responsible for the encoding. So, we compare the encoding to the M samples of the comparison set using the model's similarity function. Finally, the model will vote that the sample is part of the background if one of the model's M comparison samples match the test sample. Otherwise, all of the model's comparison samples are exhausted, and the model votes the sample as a detected change. Each model will make a prediction vote on their respective encoding of the sample. A final decision is made through a majority vote. This process is repeated for each sample within the test frame.

In our experiments, we chose N=2 models with 200-1000 comparison samples each, thus taking a maximum of N*1000*K comparisons per frame, where K is the number of feature points in the given frame. Processing time strongly depends on the density of feature points, especially the number of points detected as a change. A possible improvement on processing can be made by reducing the ROI to a PxQ grid. We evaluate a single feature point for every given region of the frame's grid if a feature happens to be detected in that region. Then, our computations will be reduced to a maximum of N*1000*P*Q comparisons where P*Q << K (significantly less than). Processing of the set of PxQ feature points is expected to be much more efficient and consistent compared

the entire set of K feature points. We processed all K points in our experiments since inconsistent processing times were not a problem in our setup.

Further false positives can be reduced with postprocessing. This involves using information from several frames to make a final prediction. We can consider a group of F frames, which are generally 3-7 consecutive frames depending on video FPS. In the first frame, we classify each feature point as described in our method. In the next frame, we first extract feature points and remove redundant points before extracting a frame sample for predictions. This is done by using a feature point matcher between the first and second frame's feature points. That way we have a likely prediction for each point in the second frame from the matched points of the first frame. We remove any point in the second frame that was matched to a point classified as the background in the first frame, and keep the label. Essentially, we reduce the majority of the computation of the frame and almost all of the computation if there are no existing changes. Feature points matched to points classified as a change will be processed again in our ensemble of models for a second prediction. We will also make a prediction for any remaining unmatched points. We continue this process between the second and third frame, the third and fourth frame, and so on until we build our group of F frames. In summary, if a group of matched points are classified as a change in all F frames, they will remain as a detection within their respective frame. If a point is classified as background in any of the F frames, all of the previously matched points are relabeled as background, and all future points within the group of F frames are not further processed and are labelled as background. This greatly benefits the approach in travelled path segments that are change-free. If there are no changes in the path, entire frames (except for a few unseen feature points) can be

processed without any model predictions. This way, most of the processing time only occurs on the first frame of each group or in frames with a detected change.

Finally, we will sum up the evaluation of test data. First, feature points are selected within a test frame's ROI. Frame samples are extracted and classified by each model using their respective comparison set and similarity function. Finally, a majority of the votes determines the final prediction for each sample. Further processing improvements can be made through the use of a grid to evaluate the frame evenly. Moreover, postprocessing is performed using a feature point matcher to reduce the need to evaluate points labelled as the background from previous frames and reduce potential false positives.

CHAPTER 5

Results and Discussion

In this chapter we explain our experimentation and present our results of both the geometry-based approach in section 5.1 and the deep learning approach in section 5.2 using the same data. Finally we compare both approaches in section 5.3. The data consists of 15 obstacle-free training videos and 17 test videos of varying lengths. There are 8 different scenes tested. Test videos of the same scene are either filmed in different directions or with the use of different obstacles. Training videos only contained the background of the path and no additional obstacles. The tests were filmed using a cellphone camera at 60 FPS. We walked down each path with the camera hand held at waist height.

5.1 Geometry-Based Results

5.1.1 Testing Methodology

The geometry-based approach needs parameter tuning for different scenes. In this case, we used the first test video of each scene as a validation set to tune the final threshold parameter. We also used the same parameter when testing the other test video(s) of the same scene. There are two outdoor scenes for a total of four different outdoor tests. The other scenes are filmed indoors incorporating different textures and floor patterns, as well as glare and shiny floors. Some scenes are filmed in cluttered backgrounds with many different objects laying in the background of the path. Some scenes will also change floor texture along the path.

We did not optimize the code for this approach or aim for real-time processing. It currently operates at 2 FPS on a laptop, but can reach much higher speeds with more efficient processes in frame management, histogram creation/comparison, and ROI segmentation. As a result we only tested 4 frames for every second of the input videos, which were filmed at 60 FPS. This is enough to have an overall understanding of the detection of changes within each test.

5.1.2 Testing Results

We display our results of our tests in Table I. We provide results for the number of obstacles detected, the number of mismatched frames, as well as the accuracy, TP rate and TN rate. The accuracy is the percentage of frames classified properly (using all frames tested). The TP rate is the percentage of frames containing changes that were classified properly. Similarly, the TN rate is the percentage of change-free frames that were classified properly.

Test ID 9 was a special case where we walked down three sets of stairs with a backpack laying in the middle of the stairs as a change. See Figure 25 for a variety of detections within different scenes. Unfortunately, this scene was only filmed with one test, so we used the beginning obstacle free portion of the video for parameter tuning. We did not include this scene in our overall results, but it is mentioned in Table I for comparison with the deep learning approach.

We consider only testing videos when calculating the overall accuracy, TP and TN result. In other words, we took the average over each metric using testing videos with ID 2, 4, 5, 6, 8, 11, 13, 15, and 17 as mentioned in the Scene column using "-Test" as the test label. We achieved an overall accuracy of 93.94%, an overall TP rate of 96.30% and

an overall TN rate of 93.05%. We were able to detect 35 out of 36 obstacles within the test videos.

Table I

Test results of change detection of paths in different scenes using the geometry-based approach. We mention the scene and it's use for training or testing under the column "Scene".

ID	Scene	Context	Obs. Detected	Mismatch	Acc.	TP	TN
1	S1-Train	Indoor	3 / 3	0	0.9524	0.9167	0.9667
2	S1-Test	Indoor	3 / 3	10	0.8730	1.0000	0.8049
3	S2-Train	Indoor	3 / 4	3	0.9455	0.9545	0.9444
4	S2-Test	Indoor	5 / 6	3	0.9606	0.9375	0.9649
5	S2-Test	Indoor	3 / 3	4	0.9668	1.0000	0.9643
6	S2-Test	Indoor	3 / 3	0	0.9951	1.0000	0.9947
7	S3-Train	Indoor	7 / 7	0	0.9855	0.9355	1.0000
8	S3-Test	Indoor	7 / 7	0	0.9856	0.9583	0.9913
9	S4-N/A	Indoor	1 / 1	19	0.9338	1.0000	0.9301
10	S5-Train	Indoor	3 / 4	0	0.9820	0.8571	1.0000
11	S5-Test	Indoor	4 / 4	0	0.7636	0.9375	0.7340
12	S6-Train	Outdoor	4 / 4	0	1.0000	1.0000	1.0000
13	S6-Test	Outdoor	3 / 3	0	0.9570	0.8333	0.9753
14	S7-Train	Outdoor	4 / 4	0	0.9759	1.0000	0.9728
15	S7-Test	Outdoor	4 / 4	0	0.9747	1.0000	0.9720
16	S8-Train	Indoor	3 / 3	0	1.0000	1.0000	1.0000
17	S8-Text	Indoor	3 / 3	0	0.9783	1.0000	0.9733

5.1.3 Discussion

This approach had an overall consistent score between the TP rate and TN rate. We detected almost every obstacle such as clear water bottles, small coins, and large items like a backpack. We processed many different scenes with varying floor patterns, some were homogeneous, some with inconsistent tile patterns, and some with occasional floor defects. We also processed scenes cluttered with different objects and backgrounds. Occasionally we had to turn corners where the entire ROI was on a wall and door frames. See Figure 26 for examples of difficult cases. Detection in all these circumstances performed well due to the ability to match frames with the dataset and tracking the

location within the scenes. Causes for classification errors were some glares and imperfect homographies for the perspective transformation.

Figure 25. Six examples of detections in different scenes. Top row detects a backpack, middle row detects water bottles, the bottom row detects an umbrella. (Best seen in colour).

Figure 26. Four examples of difficult cases are shown. The top row shows a corner turned with a doorway on the wall. The bottom row shows some examples of the variety of floor patterns tested. (Best seen in colour).

5.2 Deep Learning Results

5.2.1 Testing Methodology

In our deep learning approach, we processed each test without any parameter change. This means the models did not have any adjustments to any of their parameters such as learning rates, decision thresholds, and more. That way we can use each video in our data as a test instead of sacrificing one for validation and parameter tuning.

Processing time depends highly on the number of feature points. For a consistent processing frame rate, it was mentioned that we can choose to evaluate only one feature point for a given region of the frame, if a feature point exists in that region. Since we

filmed our test videos to simulate a robot, we are not affected by inconsistent processing times, so we processed all feature points. Processing frame rate was well above 30 FPS in change-free frames due to the low number of feature points in indoor scenes, but mostly due to the processing saved from matching feature points between frames. When obstacles are introduced, we saw a large number of feature points that must be processed in each frame, which reduces the speed to under 10 FPS. The processing rate of the change-free segments benefit greatly from our efficient handling of consecutive frames. We mentioned feature points were matched between frames to assign a guess for the labels of consecutive frames. We processed groups of 4 frames before forcing the models to re-evaluate each background feature point.

5.2.2 Testing Results

We display our results in Table II. Each test ID is exactly the same test ID and video from Table I. We also display exactly the same information aside from the number of mismatched frames, which does not relate to this approach.

Table II

Test results of change detection of paths in different scenes using the deep learning approach. The ID number of the test matches the ID number of the tests in Table I.

ID	Scene	Context	Obs. Detected	Acc.	TP	TN
1	S1	Indoor	3 / 3	0.9688	0.9778	0.9661
2	S1	Indoor	3 / 3	0.9483	0.9726	0.9401
3	S2	Indoor	4 / 4	0.9724	0.5862	0.9977
4	S2	Indoor	4 / 5	0.9545	0.4384	0.9977
5	S2	Indoor	2 / 4	0.9634	0.3455	1.0000
6	S2	Indoor	2 / 2	0.9915	0.8462	0.9990
7	S3	Indoor	7 / 7	0.9786	0.9390	0.9842
8	S3	Indoor	7 / 7	0.9645	0.9574	0.9055
9	S4	Indoor	1 / 1	0.9559	1.0000	0.9522
10	S5	Indoor	3 / 4	0.9533	0.7174	0.9765
11	S5	Indoor	3 / 4	0.9588	0.6304	0.9914
12	S6	Outdoor	4 / 4	0.9391	0.8125	0.9527
13	S6	Outdoor	4 / 4	0.9322	0.8958	0.9649
14	S7	Outdoor	3 / 4	0.9703	0.8182	0.9819
15	S7	Outdoor	4 / 4	0.9918	0.8750	1.0000
16	S8	Indoor	3 / 3	0.9710	1.0000	0.9665
17	S8	Indoor	3 / 3	0.9279	1.0000	0.9179

We processed 100% of the frames and considered all test videos for our overall results since we did not need any parameter adjustments. This gives us a total of roughly 32,500 testing frames. We achieved an overall accuracy of 96.31%, a TP rate of 81.25% and a TN rate of 97.03%. We were able to detect 60 of 66 obstacles. We placed a water bottle in almost every scene tested, some of the time it went undetected, other times the label or cap was detected. The low TP rate was due to these instances and frames containing portions of the water bottles that were undetected, such as the see-through bottom. See Figure 27 for an example of test ID 9, see Figure 28 for examples of various detections and see Figure 29 for a high glare environment.

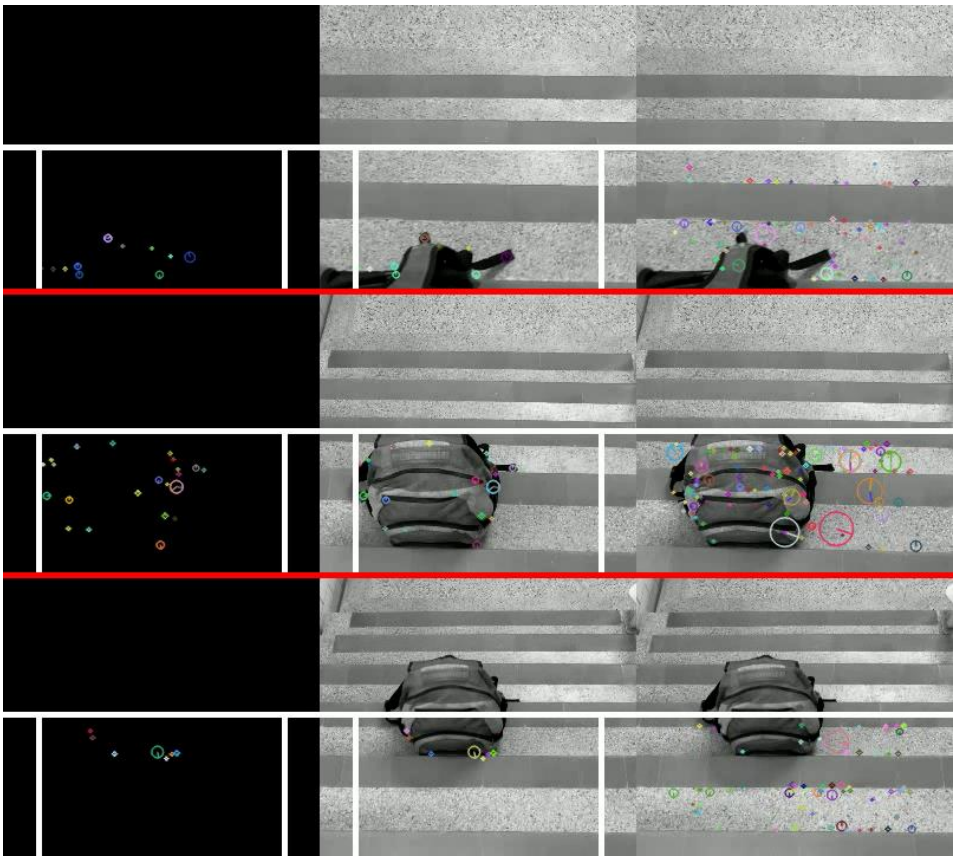

Figure 27. We show three rows of frames separated by the red lines. We removed the top 50% of the frames to save space. From bottom to top, we move past the backpack and show the detected changes in the left and middle frames. We show feature points classified as background in the rightmost frame. (Best seen in colour).

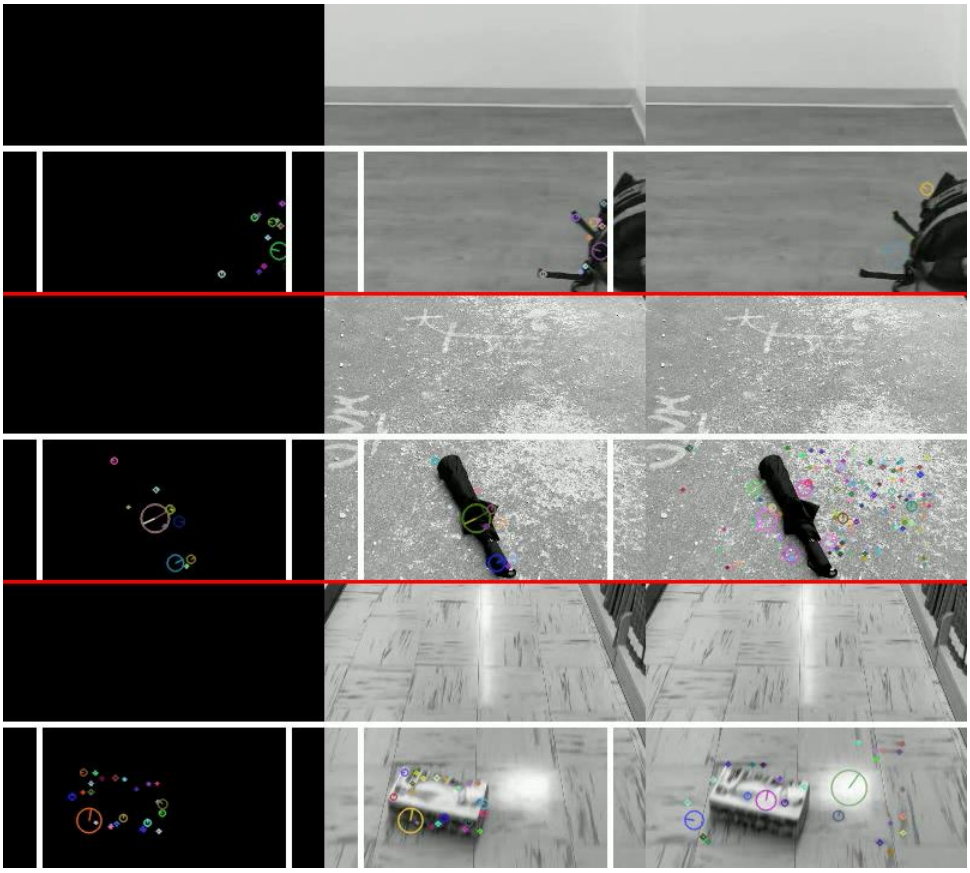

Figure 28. Three example detections in different scenes. (Best seen in colour).

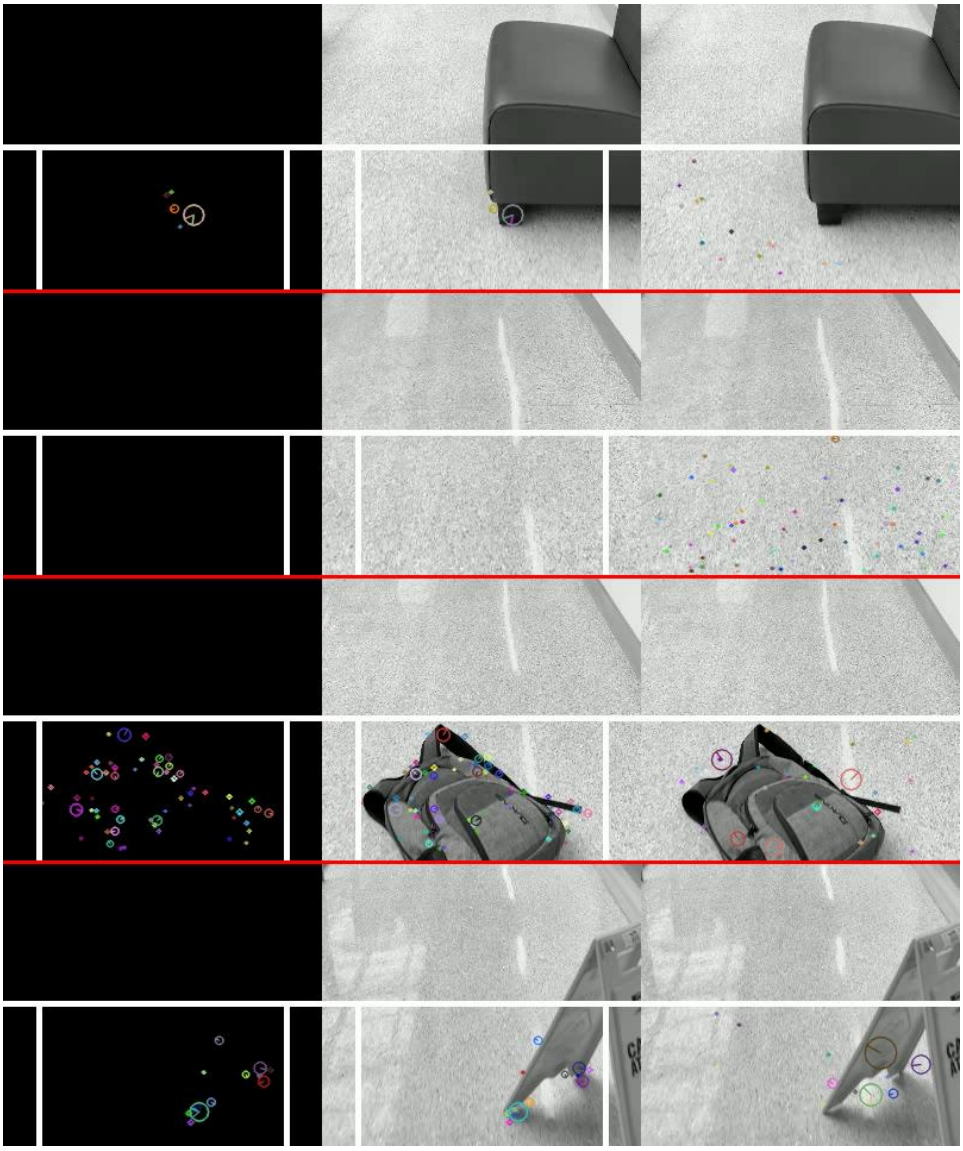

Figure 29. Detection on a reflective, high glare floor. (Best seen in colour).

5.2.3 Discussion

We used only two models in our ensemble approach to show a worst case performance. The models were constrained to a maximum of 70 epochs and a maximum of 45,000 training samples (with 5000 validation samples). Some scenes generated a significant amount of samples at over 200,000 training samples, thus showing a need for more models to capture the majority of space. While we detected 60 of the 66 obstacles, precision in the TP rate was low. For example, in Figure 27, many feature points are still classified as background while they are clearly on an obstacle. This is alright for this circumstance since we simply aim to detect the change, even at a low precision. This is where the ensemble approach can really improve detection by having more feedback of a given feature point and improving overall precision. Generally in ensemble approaches, the more models that contribute to the final vote, the better the accuracy (and precision in this case) of the classification, as long as the models are more accurate than random.

5.3 Comparison

In Table III, we present a table containing the results of both approaches on a single row (for each test). Note that the geometry-based approach required training, so tests 1, 3, 7, 9, 10, 12, 14 and 16 were used for training purposes. The results of these training tests are included for a direct comparison with the deep learning approach, which ran without parameter tuning. There is a large variety in the performance of both approaches. The key difference is that the geometry-based approach is appearance-based, while the deep learning approach is feature-based. Detection in the geometry-based approach relies on the colour distribution of a given change to stand out from the background of the scene. This is advantageous in the detection of homogeneous obstacles which appear differently from the background using colour alone. This is reflected in the results where the

geometry-based approach is able to detect smooth obstacles, like a laptop case, and difficult to see obstacles, such as water bottles.

Table III

Direct comparison of the tests displayed in Table I and Table II. The test ID numbers match the ID numbers of the tests in Table I and Table II. We present the results of the geometry-based approach on the left and the results of the deep learning approach on the right.

		Geometry			Deep Learning		
ID	Scene	Acc.	TP	TN	Acc.	TP	TN
1	S1	0.9524	0.9167	0.9667	0.9688	0.9778	0.9661
2	S1	0.8730	1.0000	0.8049	0.9483	0.9726	0.9401
3	S2	0.9455	0.9545	0.9444	0.9724	0.5862	0.9977
4	S2	0.9606	0.9375	0.9649	0.9545	0.4384	0.9977
5	S2	0.9668	1.0000	0.9643	0.9634	0.3455	1.0000
6	S2	0.9951	1.0000	0.9947	0.9915	0.8462	0.9990
7	S3	0.9855	0.9355	1.0000	0.9786	0.9390	0.9842
8	S3	0.9856	0.9583	0.9913	0.9645	0.9574	0.9055
9	S4	0.9338	1.0000	0.9301	0.9559	1.0000	0.9522
10	S5	0.9820	0.8571	1.0000	0.9533	0.7174	0.9765
11	S5	0.7636	0.9375	0.7340	0.9588	0.6304	0.9914
12	S6	1.0000	1.0000	1.0000	0.9391	0.8125	0.9527
13	S6	0.9570	0.8333	0.9753	0.9322	0.8958	0.9649
14	S7	0.9759	1.0000	0.9728	0.9703	0.8182	0.9819
15	S7	0.9747	1.0000	0.9720	0.9918	0.8750	1.0000
16	S8	1.0000	1.0000	1.0000	0.9710	1.0000	0.9665
17	S8	0.9783	1.0000	0.9733	0.9279	1.0000	0.9179

Water bottles were present (one or more, either empty or full) in almost every scene as a difficult case. The deep learning method is a feature-based approach, so features were analyzed on the cap, the label, and the base of the water bottles that meet the ground features. See Figure 30, the method classifies one feature point on the water bottle as an obstacle in the left/middle frames. The deep learning method does not have enough understanding of the scene to fully differentiate the features of the water bottle

compared to the background of the scene, which is the source of many errors. For example, later on in the scene shown in Figure 31, the features of the ground are very similar to the features of the water bottle. The darker label of the bottle is similar to the darkened sidewalk and the cap is similar to the white patches of the salt distributed on the path (filmed in the winter).

A way forward to prevent misclassifications of appearance based features, such as colour, would be to introduce colour and lighting noise to the training samples. This would force the models to learn features of the path rather than colour based features. Moreover, we used just two models in our ensemble approach. This is not enough to fully understand the scene and differentiate the background from the water bottle at this level. For example, we could generate many more feature points on the path (see the lack of feature points in the background frame (right side) of Figure 30) and increase the number of models trained on the path. Training is important for this approach and a variety of training methods can be taken.

We currently have two models. We can extend the ensemble approach by two more models (the same models even) and train the additional models on misclassified samples from the training video. We can repeat this process until we are satisfied that the scene is learned or until there are no large improvements to be made. This would allow for a detailed understanding of the scene and a focus on hard-to-classify samples.

Another approach is training by sections of the training video. We have an algorithm capable of determining the robot's current location, which can be applied to the deep learning method. Here we can train on similar sections of the scene and create an

ensemble of models for each portion of the scene. During the testing phase, the location detection algorithm would allow us to determine in which section of the scene the robot is currently positioned. This would allow us to use the specific set of ensemble models we used to classify the samples from that particular section of the scene. This could prevent features in later portions of the scene from interfering with earlier portions of the scene, such as the salt and darkened sidewalk compared to the water bottle. Training as well as the number of models used is the limiting factor of the currently tested approach, both can expand the method's ability to learn the scene.

For current results, the geometry-based method performed the best when considering the number of obstacles detected properly. The deep learning approach shows its worst-case performance using just two models. Note that training was limited to 50,000 samples for both models in the deep learning approach (45,000 training, 5,000 validation). Some scenes generated over 200,000 training samples depending on the ground features (such as outdoor scenes) and video length. As a result, better use of the training data gives an opportunity for the deep learning approach to exceed the geometry-based approach.

Figure 30. Detection of one point on the water bottle. There are a low number of background feature points present in the background of the path (see the frame on the right). The middle and left frames show points classified as obstacle. Many of the feature points are not classified properly and results in a misclassified obstacle. (Best seen in colour).

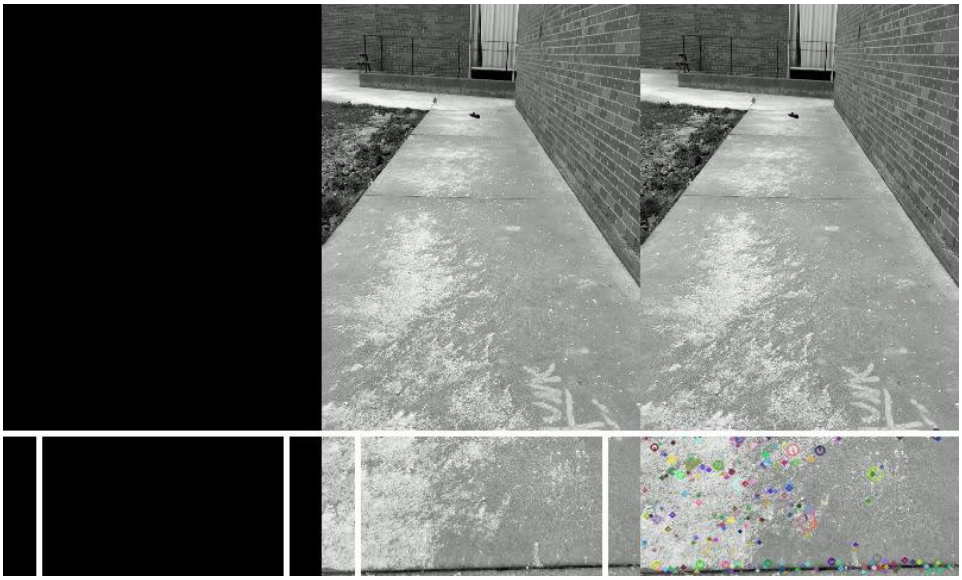

Figure 31. Another frame from the same scene shown in Figure 30. The sidewalk is darkened resembling the label of the water bottle in Figure 30 while the salt on the path resembles the white portions of the water bottle. (Best seen in colour).

CHAPTER 6

Conclusion and Future Work

In conclusion, we presented two approaches to detect changes along a path: one geometry-based and the other based on deep learning. Our geometry-based approach has a clear advantage at detecting various different obstacles while filtering out the various background information of the path using a single camera. Additionally to those benefits, our deep learning approach is free of any parameter tuning and has a clear path of improvement. We achieved an overall accuracy of 93.94%, a TP rate of 96.30%, and a TN rate of 93.05% in our geometry-based approach. We also achieved an overall accuracy of 96.31%, a TP rate of 81.25%, and a TN rate of 97.03% in our deep learning approach.

Both approaches can have false detections due to both dynamic lighting changes and glares, while also missing detections of homogeneous obstacles and more difficult obstacles, such as water bottles. In the geometry-based approach, it is a balance between filtering out glares and detecting homogeneous obstacles (in relation to the ground features). Future work can improve on the histogram comparison to account for lighting changes. Further improvements can focus on automated parameter tuning and dynamic parameter tuning over the course of a test. We can use this information in conjunction with a better precision of detected changes within the ROI to allow the robot to make informed decisions on given tasks or obstacle avoidance.

In our deep learning approach, the use of the training samples can be greatly improved. Several models can focus on different data and even difficult to classify

training samples. To build more models into the approach, we could try classifying the training data using the first models trained. It is expected that all samples are part of the background, so we can focus training of new models (with the same structure) on misclassified samples, since the scene has not been fully learned. Models can also be trained to ignore lighting changes through the use of lighting noise. Samples can be adjusted many different ways (even differently on both input and output) to force the model to learn features rather than colour distributions.

By improving the models and their training data, more work can be done to build a precise detection method that could improve accuracy in distinguishing changes from the background. Moreover, the method's ROI can be expanded or even several ROIs can be considered for redundancy or detection in other portions of the frame. For example, several ROIs at ground level placed along the path could detect obstacles at different distance levels.

Additionally, the method already has a precise location of a detected change. We can use the feature point locations of detected changes, and we could perform clustering on groups of points to create an overall structure of potential obstacles present in the frame. This can benefit the robot in automated tasks or obstacle avoidance with the use of type, size, and location of a given obstacle.

REFERENCES/BIBLIOGRAPHY

[1] Oliver, N., Rosario, B., Pentland, A.: A bayesian computer vision system for modeling human interaction. vol. 22, pp. 255–272 (01 1999).

[2] Stauffer, C., E. L. Grimson, W.: Adaptive background mixture models for real-time tracking. *Proceedings of IEEE Conf, Computer Vision Patt. Recog.*, vol. 2 (01 2007).

[3] Zivkovic, Z.: Improved adaptive gaussian mixture model for background subtraction. vol. 2, pp. 28–31 (09 2004).

[4] Zivkovic, Z., van der Heijden, F.: Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, vol. 27, 773–780 (1 2006).

[5] Chen, S., Zhang, J., Li, Y., Zhang, J.: A hierarchical model incorporating segmented regions and pixel descriptors for video background subtraction. *IEEE Trans, Industrial Informatics*, vol. 8, pp. 118–127 (02 2012).

[6] Wang, R., Bunyak, F., Seetharaman, G., Palaniappan, K.: Static and moving object detection using flux tensor with split gaussian models. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 420–424 (06 2014).

[7] Bunyak, F., Palaniappan, K., Nath, S.K., Seetharaman, G.: Flux tensor constrained geodesic active contours with sensor fusion for persistent object tracking. *Journal of multimedia*, vol. 2(4), pp. 20–33 (08 2007).

[8] Bunyak, F., Palaniappan, K., Nath, S.K., Seetharaman, G.: Geodesic active contour based fusion of visible and infrared video for persistent object tracking.

2007 IEEE Workshop on Applications of Computer Vision (WACV '07), pp. 35–35 (02 2007).

[9] Manzanera, A., C. Richefeu, J.: A robust and computationally efficient motion detection algorithm based on sigma-delta background estimation. pp. 46–51 (12 2004).

[10] Lacassagne, L., Manzanera, A., Dupret, A.: Motion detection: Fast and robust algorithms for embedded systems. pp. 3265–3268 (12 2009).

[11] Mandellos, N.A., Keramitsoglou, I., Kiranoudis, C.T.: A background subtraction algorithm for detecting and tracking vehicles. *Expert Syst. Appl.*, vol. 38(3), pp. 1619–1631 (03 2011).

[12] Barnich, O., Droogenbroeck, M.: ViBe : A universal background subtraction algorithm for video sequences. *Image Processing, IEEE Transactions on*, vol. 20, pp. 1709–1724 (07 2011).

[13] St-Charles, P.L., Bilodeau, G.A., Bergevin, R.: Flexible background subtraction with self-balanced local sensitivity. (06 2014).

[14] St-Charles, P.L., Bilodeau, G.A., Bergevin, R.: SuBSENSE : A universal change detection method with local adaptive sensitivity. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 24 (12 2014).

[15] Sedky, M., Moniri, M., Chibelushi, C.C.: Spectral-360: A physics-based technique for change detection. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 405–408 (06 2014).

[16] Wang, B., Dudek, P.: A fast self-tuning background subtraction algorithm. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 401–404 (06 2014).

[17] Wang, B., Dudek, P.: AMBER: Adapting multi-resolution background extractor. pp. 3417–3421 (09 2013).

[18] Zeng, D., Chen, X., Zhu, M., Goesele, M., Kuijper, A.: Background subtraction with real-time semantic segmentation. (11 2018).

[19] Zhao, H., Qi, X., Shen, X., Shi, J., Jia, J.: ICNet for real-time semantic segmentation on high-resolution images. CoRR **abs/1704.08545** (2017).

[20] Cheng, L., Gong, M.: Realtime background subtraction from dynamic scenes. pp. 2066–2073 (11 2009).

[21] De Gregorio, M., Giordano, M.: Change detection with weightless neural networks. (06 2014).

[22] Aleksander, I., Thomas, W., Bowden, P.: WISARD a radical step forward in image recognition. *Sensor Review*, vol. 4, pp. 120–124 (12 1984).

[23] Staffa, M., De Gregorio, M., Giordano, M., Rossi, S.: Can you follow that guy? (04 2014).

[24] Braham, M., Droogenbroeck, M.V.: Deep background subtraction with scene-specific convolutional neural networks. *2016 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 1–4 (2016).

[25] M. Babaee, D. T. Dinh, G. Rigoll. A deep convolutional neural network for background subtraction. CoRR **abs/1702.01731** (2017).

[26] Bianco, S., Ciocca, G., Schettini, R.: How far can you get by combining change detection algorithms? CoRR **abs/1505.02921** (2015).

[27] Sakkos, D., Liu, H., Han, J., Shao, L.: End-to-end video background subtraction with 3d convolutional neural networks. *Multimedia Tools and Applications*, vol. 77(17), pp. 23023–23041 (09 2018).

[28] Dongdong Zeng, Ming Zhu, A.K.: Combining background subtraction algorithms with convolutional neural network. *Journal of Electronic Imaging*, vol. 28(1), pp. 1–6 (2019).

[29] De Gregorio, M., Giordano, M.: CwisarDH+ : Background Detection in RGBD Videos by Learning of Weightless Neural Networks. pp. 242–253 (01 2017).

[30] Lorigo, L., Brooks, R., Grimsou, W.: Visually-guided obstacle avoidance in unstructured environments. vol. 1, pp. 373–379 (10 1997).

[31] Ulrich, I., Nourbakhsh, I. R.: Appearance-based obstacle detection with monocular color vision. *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 866–871, AAAI Press (2000).

[32] Kompella, V., V Bidargaddi, S., Kaipa, K., Ghose, D.: A tracked mobile robot with vision-based obstacle avoidance. pp. 12–13 (01 2008).

[33] Michels, J., Saxena, A., Ng, A.Y.: High speed obstacle avoidance using monocular vision and reinforcement learning. *Proceedings of the 22nd International Conference on Machine Learning*, pp. 593–600. ICML '05, ACM, New York, NY, USA (2005).

[34] Li, Y., T Birchfield, S.: Image-based segmentation of indoor corridor floors for a mobile robot. pp. 837–843 (11 2010).

[35] Jia, B., Liu, R., Zhu, M.: Real-time obstacle detection with motion features using monocular vision. *The Visual Computer*, vol. 31(3), pp. 281–293 (03 2015).

[36] Souhila, K., Karim, A.: Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, vol. 4 (03 2007).

[37] Naito, T., Ito, T., Kaneda, Y.: The obstacle detection method using optical flow estimation at the edge image. *2007 IEEE Intelligent Vehicles Symposium*, pp. 817–822 (06 2007).

[38] Lalonde, J., Laganire, R., Martel, L.: Single-view obstacle detection for smart back-up camera systems. *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8 (06 2012).

[39] Zhou, J., Li, B.: Robust ground plane detection with normalized homography in monocular sequences from a robot platform. *2006 International Conference on Image Processing*, pp. 3017–3020 (10 2006).

[40] Conrad, D., DeSouza, G.N.: Homography-based ground plane detection for mobile robot navigation using a modified EM algorithm. *2010 IEEE International Conference on Robotics and Automation*, pp. 910–915 (05 2010).

[41] Kumar, S., Dewan, A., Krishna, M.: A bayes filter based adaptive floor segmentation with homography and appearance cues. (12 2012).

[42] Kumar, S., Karthik M, S., Krishna, M.: Markov random field based small obstacle discovery over images. (05 2014).

[43] Lee, T., Yi, D.H., Dan Cho, D.I.: A monocular vision sensor-based obstacle detection algorithm for autonomous robots. *Sensors*, vol. 16, pp. 311 (03 2016).

[44] Xie, L., Wang, S., Markham, A., Trigoni, N.: Towards monocular vision based obstacle avoidance through deep reinforcement learning. CoRR **abs/1706.09829** (2017).

[45] Wu, K., Esfahani, M.A., Yuan, S., Wang, H.: Depth-based obstacle avoidance through deep reinforcement learning. *Proceedings of the 5th International Conference on Mechatronics and Robotics Engineering*, pp. 102–106, ICMRE'19, ACM, New York, NY, USA (2019).

[46] Croon, G., De Wagter, C.: Learning what is above and what is below: horizon approach to monocular obstacle detection. (06 2018).

[47] Xue, F., Ming, A., Zhou, M., Zhou, Y.: A novel multi-layer framework for tiny obstacle discovery. CoRR **abs/1904.10161** (2019).

[48] Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: an efficient alternative to SIFT or SURF. pp. 2564–2571 (11 2011).

[49] Lowe, D.: Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, vol. 60, pp. 91 (11 2004).

[50] Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded up robust features. vol. 3951, pp. 404–417 (07 2006).

[51] Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. vol. 3951 (07 2006).

[52] Rosten, E., Porter, R., Drummond, T.: Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, pp. 105–19 (01 2010).

[53] Calonder, M., Lepetit, V., Strecha, C., Fua, P.: BRIEF: Binary robust independent elementary features. vol. 6314, pp. 778–792 (09 2010).

VITA AUCTORIS

NAME:	Ryan Bluteau
PLACE OF BIRTH:	Leamington, ON, Canada
YEAR OF BIRTH:	1995
EDUCATION:	Master of Science in Computer Vision, University of Windsor, Windsor, ON, 2019
	Honors Bachelor of Science in Mathematics and Computer Science, University of Windsor, Windsor, ON, 2017