Text Localization for Unmanned Ground Vehicles

Allan Richard Kirchhoff

Thesis submitted to the faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

Alfred L. Wicks, Chair

John P. Bird

Devi Parikh

June 20, 2014

Blacksburg, VA

Keywords: Optical Character Recognition, Unmanned Ground Vehicles, Stroke Width Transform, Stroke Filter, Adaptive Thresholding, Machine Vision, Robotic Perception, Machine Learning, Support Vector Machine

Text Localization for Unmanned Ground Vehicles

Allan Richard Kirchhoff

Abstract

Unmanned ground vehicles (UGVs) are increasingly being used for civilian and military applications. Passive sensing, such as visible cameras, are being used for navigation and object detection. An additional object of interest in many environments is text. Text information can supplement the autonomy of unmanned ground vehicles. Text most often appears in the environment in the form of road signs and storefront signs. Road hazard information, unmapped route detours and traffic information are available to human drivers through road signs. Premade road maps lack these traffic details, but with text localization the vehicle could fill the information gaps. Leading text localization algorithms achieve ~60% accuracy; however, practical applications are cited to require at least 80% accuracy [49].

The goal of this thesis is to test existing text localization algorithms against challenging scenes, identify the best candidate and optimize it for scenes a UGV would encounter. Promising text localization methods were tested against a custom dataset created to best represent scenes a UGV would encounter. The dataset includes road signs and storefront signs against complex background. The methods tested were adaptive thresholding, the stroke filter and the stroke width transform. A temporal tracking proof of concept was also tested. It tracked text through a series of frames in order to reduce false positives.

Best results were obtained using the stroke width transform with temporal tracking which achieved an accuracy of 79%. That level of performance approaches requirements for use in practical applications. Without temporal tracking the stroke width transform yielded an accuracy of 46%. The runtime was 8.9 seconds per image, which is 44.5 times slower than necessary for real-time object tracking. Converting the MATLAB code to C++ and running the text localization on a GPU could provide the necessary speedup.

Acknowledgments

I am very fortunate to have had the opportunity to study at a world-class university for mechatronics research for the past five years. It has been an incredible learning experience and I hope to continue my education after my departure from Virginia Tech.

I would like to thank my advisors Dr.Wicks, Dr.Bird and Dr. Parikh for their help and guidance. Dr.Wicks and Dr.Bird initially introduced me to the field of mechatronics that lead to much of my graduate work.

Furthermore I would like to thank all my fellow graduate students, especially James Burns, Chris Del Barga and Marcus Thoreson for their help and for the experience of going through graduate school together. Working together we learned and accomplished more than I could have imagined from the beginning. All dataset photos were taken by the author in 2014 with the help of Robert Burgess.

Finally, thank you mom and dad for your lifetime of love and support. I will be forever grateful for your love.

Contents

List of Acronyms and Definitions

Acronym	Definition
NDVI	Normalized Difference Vegetation Index
OCR	Optical Character Recognition
SVM	Support Vector Machine
SWT	Stroke Width Transform
UGV	Unmanned Ground Vehicle

List of Figures

List of Tables

Chapter 1

1 Thesis Summary

The purpose of this thesis is to create a text localization algorithm optimized for Unmanned Ground Vehicles (UGVs). When combined with existing UGV hardware and Optical Character Recognition (OCR) technology, text localization allows for reading road signs and similar text in complex scenes. Human drivers heavily rely on road signs for navigation and hazard awareness, but current UGVs are blind to text information. Text localization can be used on military vehicles for improved hazard awareness/navigation and on commercial vehicles for improved navigation in unpredicted or unmapped scenarios. Text localization also has non-UGV applications, such as, allowing blind persons to read and text translation for tourists. This thesis describes the process of surveying existing text localization algorithms, selecting the most promising for experimentation in a UGV environment, implementing the algorithms using MATLAB and testing their performance on multiple datasets. Text localization is an image processing step necessary for text recognition in natural scenes. Text localization segments a natural scene into text and non-text regions. The text within the regions can then be identified using text recognition. Text localization for natural scenes poses a significant problem because of the variety and complexity of the scene and the text. Background textures often exhibit patterns similar to that of text. Text within natural images also varies significantly in its size, color, font and orientation.

This thesis proposes a potential solution to text localization on UGVs; however, more work needs to be done before it can reliably be used on UGVs. The best performing algorithm on individual images was able to identify approximately half of the text in challenging scenes containing road signs and storefront signs. The use of temporal data was also experimented with. A proof-of-concept test indicated that utilizing temporal data can reduce false-positives dramatically while preserving true-positives. Combining the best performing text localization method tested with an improved temporal tracking algorithm has the potential to produce a reliable text localization method for UGVs.

1.1 Background

OCR, the ability for a machine to identify text and figures in an image, has been an ongoing area of research since the 1920's [1]. Applications of OCR include document digitization, manufacturing quality assurance, and autonomous vehicle navigation. Many of these applications have structured and predictable environments which make OCR less difficult. OCR in these structured environments has been fairly well solved. For example, Google's Tesseract OCR

engine has 98-99% character recognition accuracy and 95-97% word recognition accuracy for printed binary texts [3]. However, many environments where OCR is needed are not structured. Specifically, in autonomous vehicle applications the text is often unstructured in its orientation, color & location. Obstructions and environment are highly variable for these applications and thus increase the difficulty of identification. Text localization is a necessary image preprocessing

step to OCR for complex scenes. An increasing variety of text localization methods are being explored to handle these unstructured OCR applications.

1.1.1 Previous Work in Text Localization

Previous work in text localization has tested a myriad of methods with steadily increasing success; however none have performed well enough to be used in practical applications [49]. Leading algorithms achieve approximately 60% accuracy where practical applications are cited to require at least 80% accuracy [49].

Several text features can be used to isolate text from the background. Text features commonly used for segmentation include edges, pixel intensity and texture. Different features perform better in different environments. Text localizing algorithms are typically optimized for a type of environment. Relatively little work has been done on optimizing algorithms for UGV specific environments. This is because natural environments are far more challenging. Complex backgrounds have many potential background objects that may have textures and edges similar to text.

UGV text localization in the past has focused on reading licenses plate and road signs. The best two performing methods used in UGV-like scenes are the Stroke Width Transform (SWT) and Maximally Stable Extremal Regions (MSER). SWT relies on edge information while MSER relies on intensity information. SWT finds continuous groups of pixels with similar intensities whose edges are all equidistance from each other. In other words, the width of an object must be constant throughout the object. The logic behind SWT is that natural objects vary in width while printed text characters do not. MSER also finds continuous groups of pixels with homogenous intensities, but identifies text based only on intensity information. Once regions are identified, MSER segments pixel groups that have a high intensity difference from its surrounding background. Both SWT and MSER are currently able to identify about 60% of text in simple natural scenes. That level of accuracy is not enough for practical applications, but for simpler cases they have been used successfully. MSER has been used for license plate detection in natural scenes. License plates are relatively easy to read because the text style is consistent. Reading license plates using MSER has been done with 83.3% accuracy [30], which is good enough for practical applications. This technology is already being used for practical applications, such as, automated vehicle ticketing. Many toll roads allow for drivers to maintain speed while driving though tolls by giving each driver an RFID tag that is identified as they drive on the toll road. Drivers without the RFID tag are identified using automated license plate reading software. Google maps also uses license plate identification to ensure privacy by blurring license plates in images taken for Google maps. Practical use of text localization in natural scenes is limited to license plate detection, but has steadily been improving since the first international text localization competition held in 2003 where the top performers only identified 40% of text [59].

1.1.2 Text Localization for Autonomous Vehicles

Navigation and hazard awareness of autonomous vehicles can be improved using text recognition. OCR could augment information gathered by other perception technologies to provide a clearer understanding of the environment. A plethora of information is available to a human driver while navigating via signs and text. Highway signs indicate heading and location, warning signs alert drivers of hazards unique to a stretch of road, buildings identify themselves and unique vehicles can be identified with license plate information. This information is often unattainable through prior knowledge and helps the vehicle react to a dynamic and unpredictable environment.. The ability to identify non-standard signs and text is also valuable for reading a larger variety of text, such as store front signs.

1.1.3 Text Localization in Unstructured Environments

Structured text, such as the text on license plates, is clear and has predictable positions, fonts and orientations. These environments make text localization easier as general rules can be followed and fewer unknowns are present. Less structured environments contain non-standard signs where size, font, color and orientation are highly variable. The text that UGVs would want to read is often embedded in these complex environments. Outdoor environments also introduce a greater potential for false positives since the non-text objects in the image are numerous and unpredictable. Handling the unknowns in the environment requires extra steps to get from the image to a binary character set that can then be identified with more proven methods. Because there are so many unknowns a “second pass” method can be used where information about the characters that were identified with high confidence is used to identify the characters with low confidence.

1.2 UGV Applications that Could Benefit from Text Localization

UGVs offer advantages over manned vehicles in military and civilian automobile applications. Military UGVs take the human driver out of dangerous environments, thus saving lives. Military UGVs also free up a driver to focus on combat or more important tasks [56]. Civilian UGVs have huge potential to improve quality of life by freeing up time spent by drivers in transit, allowing vehicles to transport themselves, increasing safety and reducing traffic. With major automobile manufactures predicting fully autonomous commercial vehicles by 2020 opportunities are ripe for developing algorithms to assist in vehicle autonomy [55]. Text localization has potential for use in both military and automobile UGV applications.

1.2.1 Text Localization for Commercial Vehicles

Text localization on autonomous commercial vehicles could provide navigation assistance by reading road signs and store front signs. In unmapped areas text localization would identify road names and other traffic information. However, autonomous vehicles will likely use premade maps where reading road signs is unnecessary [55]. Premade maps are inadequate in some scenarios where the traffic signs were added recently or where unmapped information is needed. Text localization can allow vehicles to respond appropriately in these scenarios.

Unpredicted traffic flow changes, such as detours or temporary road hazards, would require autonomous vehicles to adapt. Text localization could improve the vehicle's response by gathering text information about the change. Human drivers are alerted to temporary changes in traffic via signs that, with text localization, autonomous vehicles could also read.

Premade maps are also likely to fail in scenarios where detailed information is required. Less critical information, such as signs indicating parking spots, fire lanes and store information, is unlikely to be included in premade maps. Autonomous parking would require that the vehicle distinguish appropriate parking spots from restricted spots. Text localization could read signs restricting parking and avoid those spots. Additionally, store front signs contain useful information for drivers, but are unlikely to be available in premade maps. For example, the vehicle could identify whether the destination is open or closed. Figure 1 shows examples of signs that are unlikely to be in premade maps, but could be read using text localization [57].

Figure 1. Examples of road signs that are not available in premade maps, but contain valuable information.

1.2.2 Text Localization for Military Vehicles

Autonomous military vehicles stand to benefit from text localization where maps are unavailable or hazardous conditions are present. Maps of combat areas may be outdated or non-existent. Text localization could assist in navigation by reading road signs. Signs indicating hazardous terrain or combat areas would also be of interest to military vehicles.

Military vehicles are likely to encounter foreign languages when deployed. Reading and translating text in foreign languages is not only beneficial to the vehicle, but also to the operators. Fortunately, text in all languages shares the low level traits that text localization looks for. Language independence allows for one text localization algorithm to identify foreign languages without needing to edit the algorithm.

1.3 Other Text Localization Applications

Text localization used on UGVs could be used for any application where text is found in complex natural scenes. Potential applications include assisting the visually impaired and translating written language. Visually impaired persons can get printed text information without using brail. In one study the blind persons wore glasses with an embedded camera. Text

identified in the image was read to the person audibly [58]. The text could also be read at a distance as opposed to brail which can only be read up close.

Translating written language is another application for text localization. Tourists in foreign countries can translate text written in the foreign language without typing the text. Currently, an Android and iPhone application called Word Lens performs written text translation using only an image. Figure 2 shows the software in action.

Figure 2. Written text translation uses text localization to help translate foreign languages.

1.4 Autonomous Vehicle Perception

An autonomous vehicle's perception sensors provide it with all the information it has to orient itself and find a path to its destination, possibly utilizing priori information and/or maps. Regardless of how that information is processed, an autonomous vehicle can only understand its environment to the extent that its sensors allow it to. A tremendous amount of information is available via several mediums all with different detection devices needed. Common examples are LIDAR which actively emits infrared light and measures the time of flight to estimate distance to the object while a visible light camera passively images the ambient light. Light outside of the visible spectrum also provides information useful for object identification, although those systems are less common. Nature provides many mediums for information to be passed through. This is clear when identifying perception techniques used by animals. Some animals use electric fields or the Earth's magnetic field for orientation [10]. In both nature and machines, all sensing techniques fall into one of two categories: passive or active. Understanding how UGVs currently collect information on their environment is critical to understanding how text localization can best be integrated with current UGV perception systems. It is preferable for text localization sensors to be the same as perception sensors already present on UGVs.

1.4.1 Active Sensing

Active sensing is where a signal is emitted from the vehicle/animal and the reflection is used to measure the environment. A bat's echolocation is an example of this in nature. The bat emits pulses of high frequency sonic waves and listens to the reflections. The sound waves time of flight and angle of reflection give the bat a picture of the environment. LIDAR works on the same principle, except that an infrared wave is used. Figure 5 shows a false color point cloud generated by a multi-plane LIDAR.

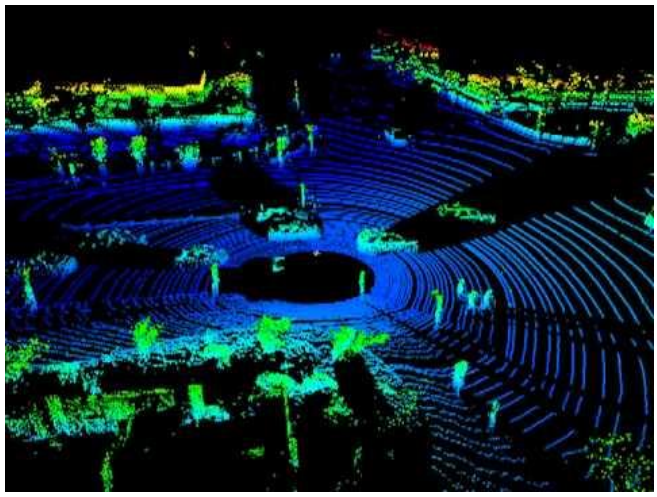


Figure 3. A false color point cloud generated by a multi-plane LIDAR. The road, pedestrians and other vehicles reflect the LIDAR's infrared emissions. Time of flight calculations yield a 3-D point cloud. Used under fair use, 2014.

Citation: *The Velodyne High Definition LiDAR (HDL) Grabber*. PCL

All MLVT autonomous vehicles used active sensing for the majority of their perception. Specifically, LIDAR was used to map the terrain and detect obstacles. Similar active sensing technologies are Sound Navigation and Ranging (SONAR) and Radio Detection and Ranging (RADAR). Active sensors have a reliable history, have long range capabilities and return dense data. Unfortunately, active perception does have some drawbacks. The data returned only provides a point cloud in 3D space, which does not help in differentiating similarly sized and shaped objects. For example, a prone human and a log of the same size cannot be differentiated between because there is only size and shape information provided by active sensing. Additionally, the signals emitted by an active sensor are easily detected by outside observers. While this is irrelevant in civilian applications it is disastrous in a military scenario. Imagine ground troops attempting to move unseen through hostile territory with a vehicle that lights up the infrared band with its LIDAR. The enemy would simply have to observe the infrared spectrum to see the bright LIDAR beacon. Active sensors are also expensive when compared to the vehicle. Google's autonomous civilian vehicle uses $75,000 of LIDAR equipment on a Toyota Prius that costs $25,000 [11]. For these reasons, passive sensing has been increasingly the focus of perception systems for autonomous vehicles. OCR also depends on passive sensing systems.

1.4.2 Passive Sensing

Passive sensing measures a wave or signal that is present in the ambient environment without any input from the system. Traditional visible light cameras work on this principle by recording the visible light that is already present in the environment. The light reflected on the various objects carry information about that object, such as, color and intensity. In nature, passive sensing is the most common with vision. Hermit crabs live on the ocean shore where waves often pick up the animal and make the force of gravity a poor indication of up and down. To combat this the hermit crab measures the Earth's magnetic field, which is independent of the force of the waves, to determine which way is south. While visible light (400nm to 700nm) is the most common passive sensing method others provide additional information, such as, infrared and ultra-violet. Passive sensing does not suffer the drawbacks that made active sensing undesirable, but at the same time introduces different problems. Visible and infrared cameras cost orders or magnitude less than active sensors, are completely covert and provide high density detailed color and intensity information. However, this plethora of data is more difficult to decipher and is more susceptible to noise than active sensing methods.

Successful applications of passive sensing to identify objects include Normalized Difference Vegetation Index (NDVI), stereo vision, object classification & OCR. NDVI is a technique that identifies vegetation from non-vegetation. NDVI requires a visible and infrared camera and minimal processing. The difference between the infrared intensity and visible red intensity generates a ratio that is unique vegetation. NDVI is a very convenient method for object identification. GUSS implemented a basic NDVI system to reduce false obstacle detection with good results. NDVI succeeds where LIDAR does not and requires few computational resources; however, broader object identification methods require more information and more complex algorithms.

Stereo vision estimates an objects distance, similar to LIDAR, but using the disparity between two visible cameras rather than light's time of flight. Stereo vision takes two images simultaneously from adjacent points. Each camera sees the same objects from a slightly different perspective. The difference in these perspectives is called disparity. Figure 4 shows how two cameras experience disparity. The magnitude of the disparity is inversely proportional to the objects distance. With many sets of images two visible cameras can create data similar to what is collected by a LIDAR. Like LIDAR, a 3-D point cloud giving object size, shape and distance is produced with stereo vision. Unfortunately, stereo vision only works for short ranges. In nature most animal's stereo vision is only good up to 2 meters while machine stereo vision can extend this range, but beyond 17 meters the disparity is too small to detect even at high resolution and high camera separations [12]. Also, stereo vision is susceptible to noise which reduces the accuracy of the distance calculations.

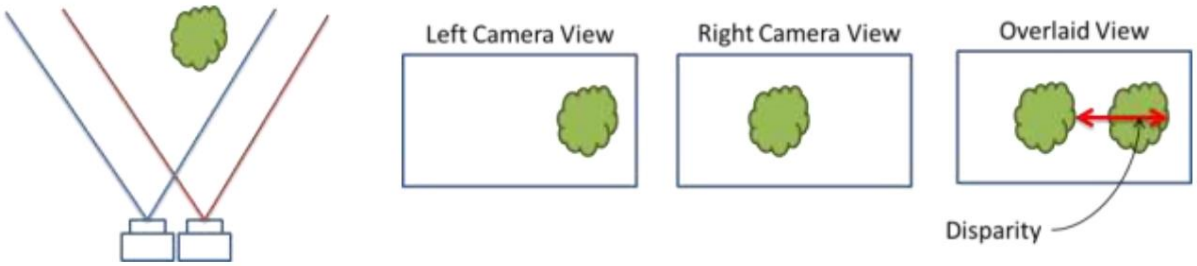


Figure 4. Two adjacent cameras see the same object from different viewpoints which creates disparity between the two images. That disparity is inversely proportional to the objects distance.

Object classification, something LIDAR provides only a limited solution to, can be done with visible cameras. LIDAR is limited in its object classification with size, shape and orientation information. Visible cameras can detect color, texture and small objects not seen by the LIDAR. This additional information can determine if an obstacle can be driven through, or must be avoided. Object recognition using passive sensing often employs machine learning. Machine learning takes human labeled examples of objects and "learns" the differences between objects. The object is broken down into basic quantitative components (descriptors) which can be compared to those of unknown objects for identification.

Integrating text localization on existing hardware is important because it minimizes the cost of adding text localization to vehicles. Text localization relies on visible cameras which already exist on autonomous vehicles using passive sensing. This means that no new hardware will be needed to implement text localization on most existing vehicles. Machine learning algorithms currently used for object detection on UGVs can also be used for text localization. Text localization can be "taught" what text regions look like. OCR descriptors differ from object descriptors, but the matching algorithms are similar. Text localization can benefit from previous work on machine learning rather than creating entirely new algorithms.

1.5 Thesis Problem Statement

Building on perception methods for autonomous vehicles, text localization will be explored and implemented, specifically for UGVs to read street signs and other text within their environment. The text will be identified, but the comprehension of the text is another problem entirely and will not be discussed in this thesis. MLVT has set a starting point for how the data will be collected and how environmental information can be used to assist in autonomous vehicle navigation. The focus of the application of this text localization system will be tweaked specifically for UGVs similar to MLVT's UGVs.

1.6 Thesis Content/Overview

Chapter 2 is a literature review of the OCR software called Tesseract, which is currently the best open source OCR software available. It also covers a breadth of existing text localization methods that could be applicable to UGV environments. The necessity of pre-processing natural

scenes with text localization before applying OCR will be explained. A brief history of OCR will also be covered.

Chapter 3 goes into the specifics of how the tested text localization algorithms work and how they were implemented using software. It also explains what software was used for implementation and why.

Chapter 4 explains the experimental setup. It explains the selection and creation of the trial datasets. It also covers how the performance of different text localization methods are quantified and compared. Lastly, it explains the proof-of-concept test which examines the usefulness of temporal data.

Chapter 5 covers the results from the experiments and trials explained in Chapter 4.

Chapter 6 discusses the conclusions drawn from the results and what future works needs to done before the text localization algorithm can be used reliably on a UGV.

Chapter 2

2 Literature Review

This chapter reviews current literature on how OCR and text localization have been solved in the past and how they have been applied previously. It will also explain the necessity of text localization before OCR can be applied.

2.1 OCR – A Brief Overview

The earliest genuine attempt at OCR was in 1914 [13]. The Optophone was designed to allow blind people read individual printed characters without using Braille. The Optophone projected light onto a character and a selenium photosensor would detect the magnitude of light and match that with a character. A sound would indicate which letter was read. Initially it was exceedingly slow at 1 word per minute, although later models achieved up to 60 words per minute. [14]

The first commercially successful OCR system was developed in 1953 by David Shepard. Called "Gismo", the system could read all 26 English characters by brute force character-by-character template matching. It was used to read credit cards printed with the Farrington B numeric font, which was invented by Shepard to increase recognition reliability. Although credit cards have long moved on to magnetic strips to carry information, the Farrington B font is still in use today. [15]

The first successful alphanumeric OCR system was the IBM 1428 invented in the 1960's, which was based on a technology similar to Shepard's. The applications of this technology began expanding to automated banking and mail sorting. This technology was limited to a fixed font where any change in font could not be handled. Industrial applications where multiple fonts needed identifying drove the technology to be developed to achieve omnifont recognition.

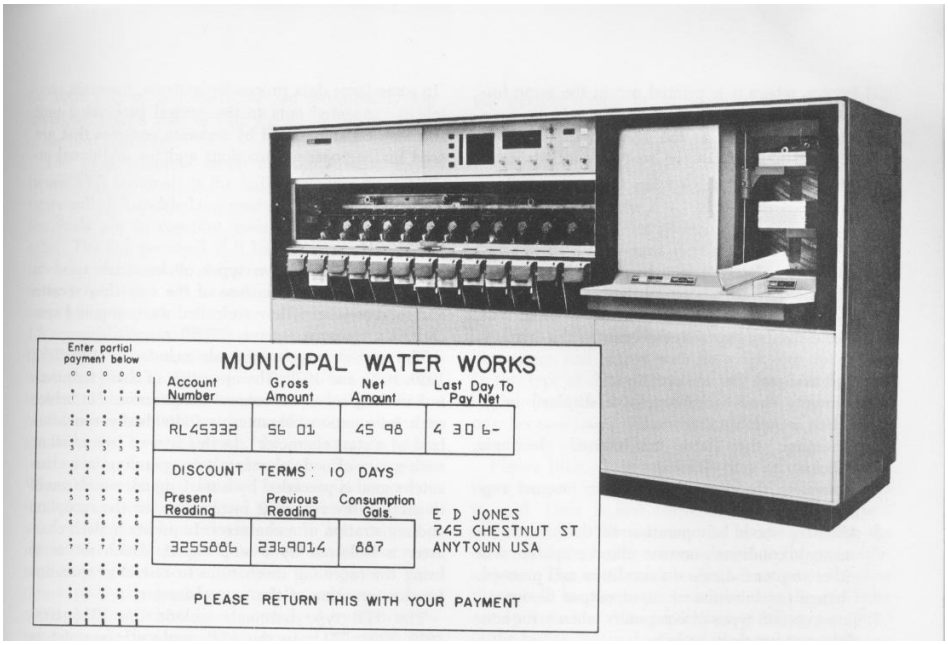


Figure 5. IBM 1428 is the first alphanumeric OCR machine developed in the 1960's. Shepard's Farrington B font, still in use today, is recognizable in the printed document. Used under fair use, 2014.

Citation: *Introduction to IBM data processing systems*. Washington: [U.S. G.P.O.], 1966.

The brute force method could not handle more than a few fonts because of the computational requirement needed. More advanced feature extraction was needed to identify characters regardless of font. These features that identify characters are the same regardless of the font. The most common features are line segments, concavities and loops. All characters are made up of some unique combination of these features and feature extraction is relatively cheap computationally.

Ray Kurzweil's Reading Machine was released in 1976 [16] and used feature extraction that had to be trained to identify characters of any font. The training process took several hours and only nine fonts could be read at any one time. The technology was primarily used for converting printed records to a digital format. Kurzweil's approach of pattern recognition is still the basis of OCR today, although it is more sophisticated.

2.2 Omnifont Feature Detection

In order to recognize letters of any font, the features of the letter must be characterized. Line segments, concavities and loops are the most commonly used because of usefulness and ease of detection. Extracting these traits require an image that is binary and free of non-character objects. It is important to understand how the OCR software identifies text so that the text localization can output something readable by the OCR. The next two sections will explain how OCR software deals with objects in the image. The third section will discuss why those OCR methods fail in natural scenes and how pre-processing images can allow OCR to work for natural scenes. There are many methods of OCR; however, currently the best performing free OCR software is

Google's Tesseract. It is the most likely OCR software to be used in research and in the MLVT. For this reason Tesseract's methodology will be the focus.

2.2.1 Text Base Line Detection

Most OCR systems can only read characters that are upright and facing perpendicular in the image. If the perspective changes even slightly the OCR system can fail. Printed text and especially outdoor signs will be viewed from a varying perspective and need to be altered before the OCR system attempts to recognize it. A skewed baseline is the most common perspective change for text reading OCR, but for outdoor OCR the problem is amplified by a third dimension of possible perspective change. Additionally, the characters identity often depends on its relative position on the baseline. This makes the baseline of the sting of characters critical to identify before the rest of the OCR system can proceed.

For baseline detection OCR generally starts be identifying the connected components (CCs), or the individual characters that make up a word. This is done by identifying the centroids & sizes of contours and matching similar contours together. Once grouped the baseline of the CCs is created based on contour centroids and bounding box dimensions. The lowest points on the line are fitted with a quadratic spline. A quadratic spline is a polynomial function that approximates a smooth curve and is useful for quantifying the distortion of a baseline. Figure 6 shows an example of a line of characters being fit by multiple splines for baseline approximation.

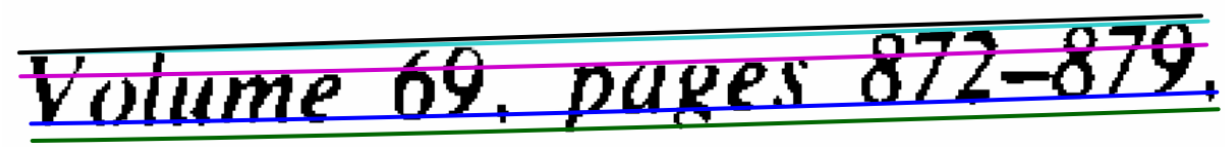


Figure 6. Quadratic splines approximating a skewed baseline of a line of text. This quantifies the characters skew and is taken into account for character identification. Used under fair use, 2014.

Citation: Smith, Ryan. "An Overview of the Tesseract OCR Engine"

Tesseract's baseline skew detection has a 98.5-99% success rate for two dimensional skews up to ±30 degrees. However, baseline skew in the third dimension is not accounted for by OCR. Depth information is not collected in a single image and therefore skew in the third dimension is not accounted for [22]. Work has been down to account for 3-D skew using information from a single image, but it is only successful in well-structured environments [23].

2.2.2 Character Detection

Tesseract employs multiple processes for character detection [3] each building off the other. If the image is clear and each character is a discrete contour then Tesseract proceeds directly to character classification. However, less clear images with joined or broken characters need different processing before character identification can begin. Initially Tesseract attempts to identify if the characters are fixed pitch (characters are equally space apart). Figure 7 shows a fixed-pitch word and where Tesseract separates each character.

Figure 7. A fixed pitch word that is separated into individual characters using Tesseract. Used under fair use, 2014.

Citation: Smith, Ryan. "An Overview of the Tesseract OCR Engine"

If a word is not fixed pitch and characters are not discrete the joined characters must then be separated for identification. Concave vertices of a polygonal approximation become candidate separation points between characters. The chosen separation point starts near where the character is estimated to end. Then the characters are identified and if the identification confidence is low, a different separation point is chosen until a high confidence is yielded. Figure 8 shows an example of separation point candidates on joined letters.

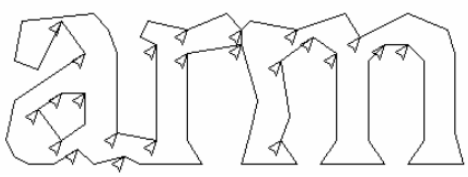


Figure 8. Letters bleed together in the word 'arm'. Candidate letter separation points are marked at letter concavities. The separation points that yield the highest confidence letter identification are chosen. Used under fair use, 2014.

Citation: Smith, Ryan. "An Overview of the Tesseract OCR Engine"

If any of the above mentioned methods fail to generate high confidence characters, then each contour is assumed to be part of a character, not a full character. These partial characters are then searched for possible combinations that yield high confidence letters. Figure 9 shows an example of a word with broken characters which are associated with each other to build full letters out of discrete contours.

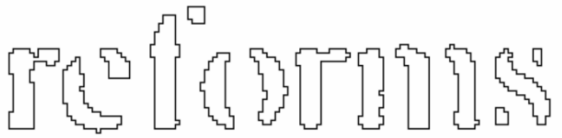


Figure 9. An example of broken letters. Tesseract identifies these by searching for contours that when combined generate high confidence letters. Used under fair use, 2014.

Citation: Smith, Ryan. "An Overview of the Tesseract OCR Engine"

2.2.3 Feature Based Character Classification

Font-invariant and size-invariant features are extracted from each letter for characterization. Tesseract uses straight edge features that are defined by their height, width, position and angle.

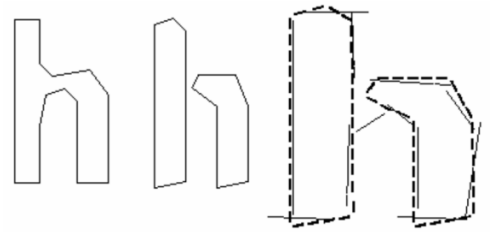


Figure 10. The edge features that make up an ideal (left) letter being matched to an unknown letter's features (middle). Each feature difference contributes to the letter's classification confidence. Used under fair use, 2014.

Citation: Smith, Ryan. "An Overview of the Tesseract OCR Engine"

An unclassified character typically contains 50-100 features [19]. The set of features is then compared to a known letters feature set where the letter with the least difference between features is selected. Tesseract also has limited linguistic analysis and an adaptive classifier; however, those are beyond the scope of this research. Text localization methods can be easily optimized for static classification, but adaptive classification is much more difficult to implement.

2.2.4 Pre-Processing Required for Complex Scenes

The need for preprocessing natural scenes before using OCR software is made apparent by OCR performance on natural scenes. Figure 11 shows Tesseract's output given an image of a street sign. The large amount of non-text objects in the scene cause Tesseract to output gibberish,

Tesseract OCR Output:

‘ Y ‘ ‘ -
% U“”‘§4wJ*ty amr :\:;r\4
L~:J§1\<”!" 1 ‘ . ’ W“‘J~*9-‘»€
9» -
*-ff.“/@lIJih:<3;@ - 1; r‘-
%'e%"’a
-"-*r—-r....,,,,_, ‘J ‘J1 - V,/—/, , .
"""""”"3/ » , .
,K/"“/ "’ J '7'!) 5:’ E! “‘W«‘
F _ r E *7 / -
I K \ a u , “'7 J
/.9'%% mt!‘ 3 j’ _ _
V. A:// H .- > A I/ .>- V
I I ‘

Figure 11. OCR software typically assumes the image contains only text and a constant background dissimilar to the text. When applied to natural scenes the text detection fails. Here Tesseract returns gibberish given a picture of a street sign.

As discussed in Section 2.2, OCR software has strengths and weaknesses that should be taken into account when developing a text localization solution. Since OCR is robust to 2-D rotation, broken characters and joined character the text localizer does not need to address these issues in pre-processing. However, OCR software cannot handle non-character contours in an image. OCRs typically attempt to classify all contours or blobs as characters [19]. This is not an issue for reading text from scanned documents where non-character contours were rare; however, natural scenes often have more non-character contours than character contours. Before being sent to the OCR software, the image must be reduced down to its text-only components. To get the best result from the OCR software any non-text objects should also be removed from the text regions.

Figure 12 shows a cropped, binary image containing the word ‘Trail’ from the image in Figure 11. Where Tesseract previously failed, it now succeeds because of pre-processing.

Tesseract OCR Output:

Trail

Figure 12. Tesseract OCR successfully identifies the same text after pre-processing.

The goal of text localization is to identify the bounding box around each text region and eliminate any non-text objects in that bounding box without reducing the quality of the

characters. Inputting the simple pre-processed image into Tesseract will allow for high word recognition rates similar to the high recognition rates of simple scanned text.

2.3 Text Region Detection in Natural Scenes

For natural scenes, such as what a UGV would experience, filtering the text containing regions from the rest of the image presents an additional challenge to OCR. As discussed in Section 2.2, OCR software like Tesseract needs a binary input where non-text objects have been removed. Scale, orientation and joined/broken letters do not need to be addressed because Tesseract already handles those problems. Identifying text regions in a natural scene is non-trivial. Repetitive textures and strong edges in the image appear similar to text. Many text features which remain constant in a controlled environment are unpredictable in natural scenes. Text orientation, font, size, lighting and obstruction are often greatly varied between texts in the same scene.

2.3.1 Text Features

Text features typically have a unique combination of texture, edges, corners and Connected Component Analysis (CCA) [24]. These features are often partly shared with background objects. For example the busy, high energy texture of text is similar to vegetation or a brick wall where a busy texture is repeated. No single trait accurately isolates text from the background for all scenes, but the combinations of these features are unique to text. Figure 13 shows how the combination of features allows for a more accurate text descriptor.

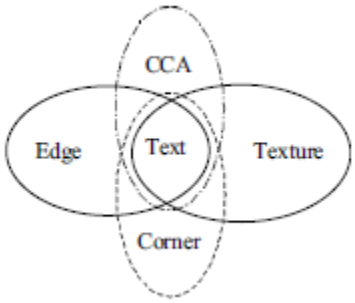


Figure 13. No single trait accurately isolates text from the background, but the combinations of these features are unique to text. Used under fair use, 2014.

Citation: Liu, Qifeng. *Stroke Filter For Text Localization In Video Images,* 2014

2.3.1.1 Texture

Text can be identified as having a unique texture. The texture is unique because of text's tendency to have dense horizontal and vertical edges. Text texture typically has high energy [25] response to texture filters compared to the background which has a lower edge density. A high energy response indicates that there are many edges, or changes in intensity, in a region. Text produces high energy because it has a very different intensity from the background. Generally texture identification strategies first employ a set of linear filters, such as, the Gabor filter to identify regions where the edges are dense and strong [26]. The linear filter is followed by a non-

linear transformation which is applied to each pixel. A popular example is $tanh(\alpha t)$ [26]. The local energy of the response is used as a feature to describe the pixel in the original image. Each filter creates a new matrix of the size of the original image where each pixel corresponds to the original image pixel. Each pixel in the original image is characterized by a feature vector that contains *n* features, where *n* is the number of linear filter responses calculated.

A machine learning approach is then used to classify each pixel as text or non-text based on its feature vector. Texture filtering typically has a low false-negative classification rate, but has a very high false-positive classification rate. Figure 14 shows an example of how a text texture approach highlights the text in two different scenes. When applied to the image with a low energy background (a) the response (b) is strong for the text, but not for the background. However, when applied to the image with a highly textured background (c) the text is still highlighted, but so is a significant level of noise from the textured background.

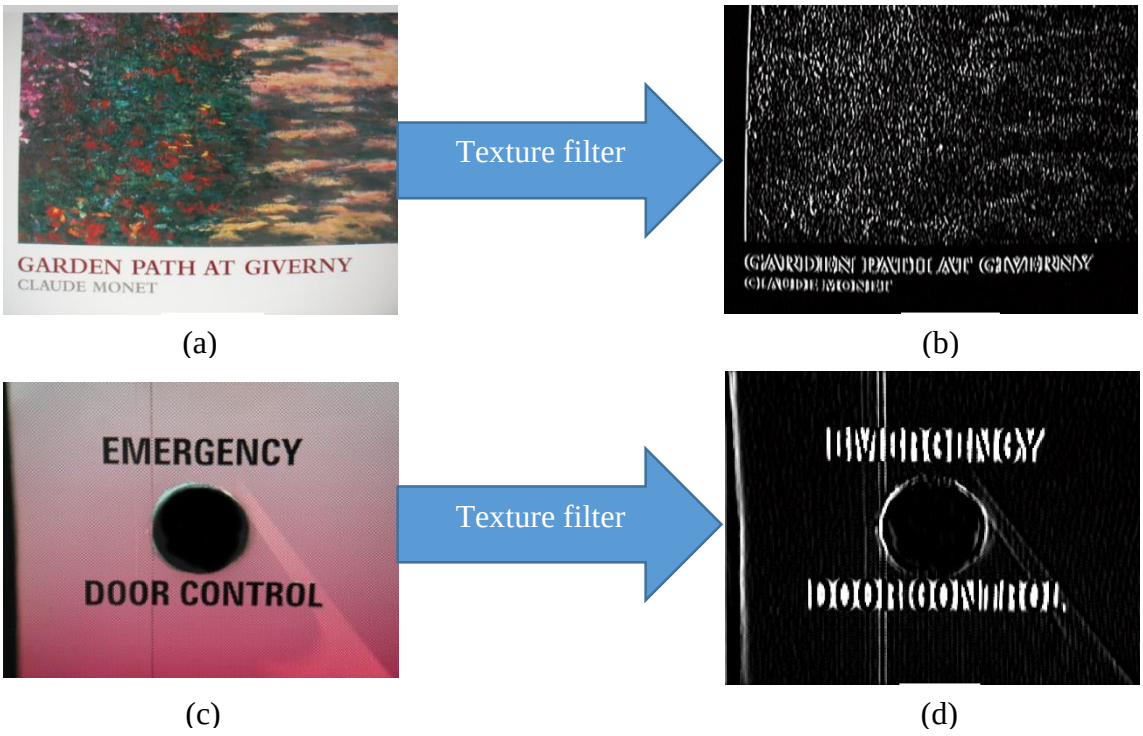


Figure 14. Text has a high response to texture filter, but background textures can introduce false-positives. Depending on the scene a texture filter alone can fail (b) due to many false-positives or succeed (d).

2.3.1.2 Edge/Corner

Text generally has strong edges, especially in the horizontal and vertical directions. The density of the corners created by the edges is greater than most of the background. A simple horizontal filter and vertical filter can be used to find edges by searching for sudden changes in pixel intensity in a direction. Figure 15 shows an example of a simple horizontal (a) and vertical (b) edge filter. The center pixel intensity is the sum of the intensities on one side of the filter box

minus the sum of the intensities on the other side of the filter box. When the intensities are the same the result will be nearly zero. Conversely, if an edge exists the response will be large.

-1	0	1
-1	0	1
-1	0	1

(a)

-1	-1	-1
0	0	0
1	1	1

(b)

Figure 15. Simple edge filters for horizontal (a) and vertical (b) edge detection in an image.

Corners where horizontal and vertical edges come together is an indicator that text is present. Many methods exist for corner detection; however, a Harris corner detector is one of the more common methods. The Harris corner detector takes a window, noted by $w(x, y)$, and slides it throughout the image. Each window is compared to windows shifted adjacent to it. The difference between the two windows, noted by $E(u, v)$, is calculated using Equation 1 where u is the shifted windows displacement in the horizontal direction and v is the displacement in the vertical direction. $I(x + u, y + v)$ is the intensity of the shifted window and $I(x, y)$ is the intensity of the original window.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \quad [1]$$

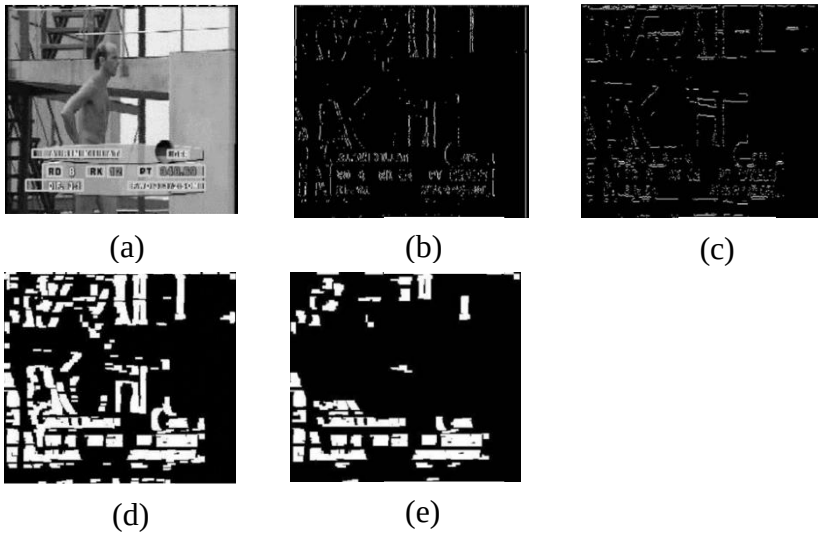
A large difference indicates a significant change in that region which is likely caused by a corner in the image. Equation 1 can then be written in matrix form, see Equation 2.

$$E(u,v) \approx [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$
$$M = \sum w(x,y) \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix} \quad [2]$$

The eigenvalues of the Harris matrix (M) indicate the amount of change in either of the directions. If neither eigenvalue is large then there is no change in that region of the image. If one is large and the other small it indicates that an edge was found. Both eigenvalues being large indicate change in all directions, which is likely caused by a corner [27].

Edge and corner detection methods generally suffer from the same high false positive rate at texture filters. Natural scenes often contain edges and corners in the background. The many false positives make it difficult to separate text regions out from the background. Figure 16 illustrates

previous work done using a corner detector. In (b) and (c) the vertical and horizontal edges are extracted. Regions where those edges create corners are highlighted in (d). The regions of interest are then filtered based on size and shape to eliminate some false positives, but many still remain (e).



(a) (b) (c)

(d) (e)

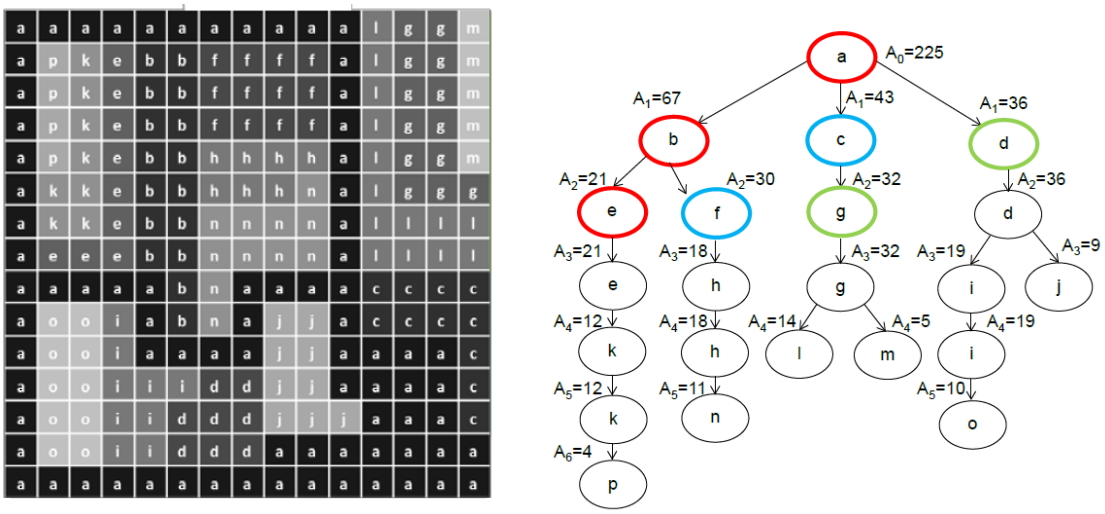
Figure 16. Corner detection identifies text regions; however, the many false positives make it a poor standalone solution to text localization in natural scenes Used under fair use, 2014.

Citation: Bourlard, Hervé. "Text detection and recognition in images and video frames." *Pattern Recognition*: 595-608

2.3.1.3 Connected Component Analysis

Connected Component (CC)-based methods group small areas with common features (typically intensity) into successively larger components until all regions are identified in the image [28]. Each region is then compared spatially for components that could be text regions. Many methods exist for generating the original components and results vary significantly. A more straightforward approach is binarization based on local thresholding. The contours are then analyzed as character candidates [43]. CC-based methods have also been used given edge information to CC generation [44]. Color has also been used to quantize the CCs, although this makes the assumption that the text takes up a large amount of the image, which is not the case in natural scenes [45]. RGB information has also been used by clustering color histograms in an RGB space and using supervised machine learning to identify unique regions [46].

One of the best performers against natural scenes in the CC category is the Maximally Stable Extremal Regions(MSER) method. MSER takes a grayscale image and identifies objects using intensity information. Regions of similar intensity are grouped into a single object [29]. Once every object in the image has been labeled, MSER attempts to group character-like objects to form a word using a hierarchy structure. Figure 17 shows an example of how regions are separated and how an object structure is formed.



(a) **(b)**

Figure 17. An example of an image where continuous regions of similar intensity are grouped using MSER (a). The regions are then categorized in a hierarchy of objects (b).

MSER has many advantages when used in natural scenes which accounts for its high performance. Invariance to affine (perspective) transformations of pixel intensities allows MSER to identify text at various perspectives. Size invariance is also a benefit of MSER. MSER does not need multiple passes on the same image to find various sized text, which improves computation time. Lighting changes also do not greatly affect MSER and its ability to detect objects. This stems from its use of relative local intensities for component generation. MSER also has a high component density with very small components being detectable. This is an advantage is text is very small. MSER's greatest disadvantage is its sensitivity to blurred images. Blurred edges make it difficult to separate local regions based on edge information [29].

MSER is a popular approach to text localization because it is perspective and lighting invariant. License plate detection and general text localization has been done with high accuracy using MSER. License plate detection was particularly successful with an 83.3% detection rate given a complex background [30]. Figure 18 shows examples of MSER outputs when used for license plate detection in natural scenes.

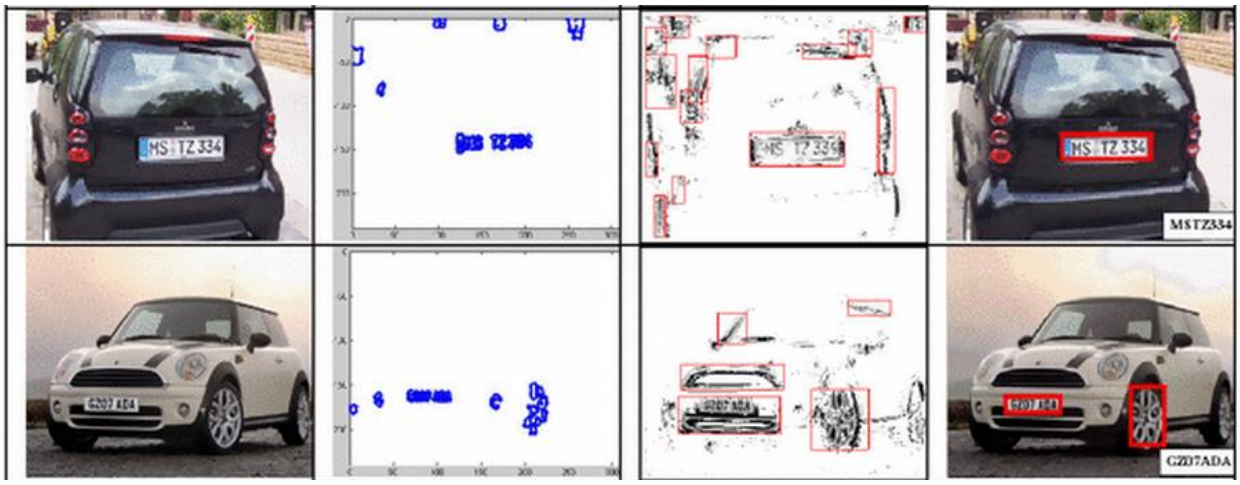


Figure 18. An example of MSER and component filtering used in license plate detection. Used under fair use, 2014.

Citation: Wang, Wei. "Car License Plate Detection Based on MSER." *IEEE*

Another high performance CC approach is called the Stroke Width Transform (SWT). SWT takes advantage of the tendency for text characters to have consistent width throughout the character. Components are generated using a dark-on-light and light-on-dark threshold. The components minimum width is then measured throughout its length. For each character the standard deviation in the stroke width indicates the stroke width consistency. Components with high stroke consistency are more likely to be characters. An advantage of SWT, like MSER, is that character information is extracted so additional information is available to group regions into words. SWT is also robust against background noise because background textures are rarely consistent width while being spaced like text.

2.3.1.4 Intensity Thresholding

Within an image the intensity of the text generally is significantly different from the background intensity. Very early text localization approaches used global thresholding of an image to extract the text while removing the background. Global thresholding converts a grayscale image, which contains pixels of intensities ranging from 0 to 255, into a binary image where pixels are either 1 or 0. Global thresholding takes every pixel below some pre-defined threshold, say 127, and sets those to 0, while setting all the pixels above 127 to 1.Global thresholding works well in controlled environments where the text intensity is always different from the background and lighting is constant. When applied to natural scenes global thresholding performs very poorly because the background intensity varies significantly and the lighting is not constant. However, more recently (2014) thresholding has been revisited [35].

Adaptive thresholding is a different approach that sets its threshold based on the intensities of a region within the image. Typically in an image the background represents many pixels of high intensity and the text represents pixels of low intensity. The gap between those intensities is where the threshold value would ideally be set so that the text could be extracted from the background. Adaptive thresholding attempts to set the threshold to that ideal gap.

A study noted that for natural scenes adaptive thresholding performed better than global thresholding [35]. The advantage of adaptive thresholding in natural scenes is its speed. Texture based methods and especially CC methods are very slow compared to adaptive thresholding.

2.3.1.5 Image Compression

A lesser used approach uses an image's compression information to determine if any text is in the frame or not. The advantage of this approach is that it requires very little processing per frame. The disadvantage is that it does not return the text region. It only returns whether or not the image contains text. Gargi et al. [37] considered the number of intra-coded blocks in P- and B- frames of an MPEG compressed video. The number of blocks usually increases when text is present, because the text is a different color than the background. Use on a UGV; however, is impractical because of the very noisy backgrounds and the small size of the text regions in the images.

2.4 Non-text Features

The difficulty of text localization in natural scenes stems from the difficulty of separating the non-text background from the text. The categorization of image regions is called segmentation. For text localization there are only two categories: text and non-text. Backgrounds with few edges and little texture are easier to distinguish from text because text have many edges and is highly textured. Backgrounds with non-text objects or textured regions are more difficult to distinguish from text because they share many of the same features. Horizontally repeating textures increase the odds of misclassification. Unrepeated text-like objects will be filtered because there are not enough characters to form a word; however, roughly horizontal repeating textures will contain enough objects for a word. Different text localization methods tend to misclassify different types of non-text objects; however, common misclassifications include vegetation, bricks and printed non-text characters. A few examples are shown in Figure 19. In (a) multiple patches of limbs were extracted using SWT because limbs are roughly of even width. In (b) printed non-text symbols are identified as text because of the similar sizing, texture and spatial features.

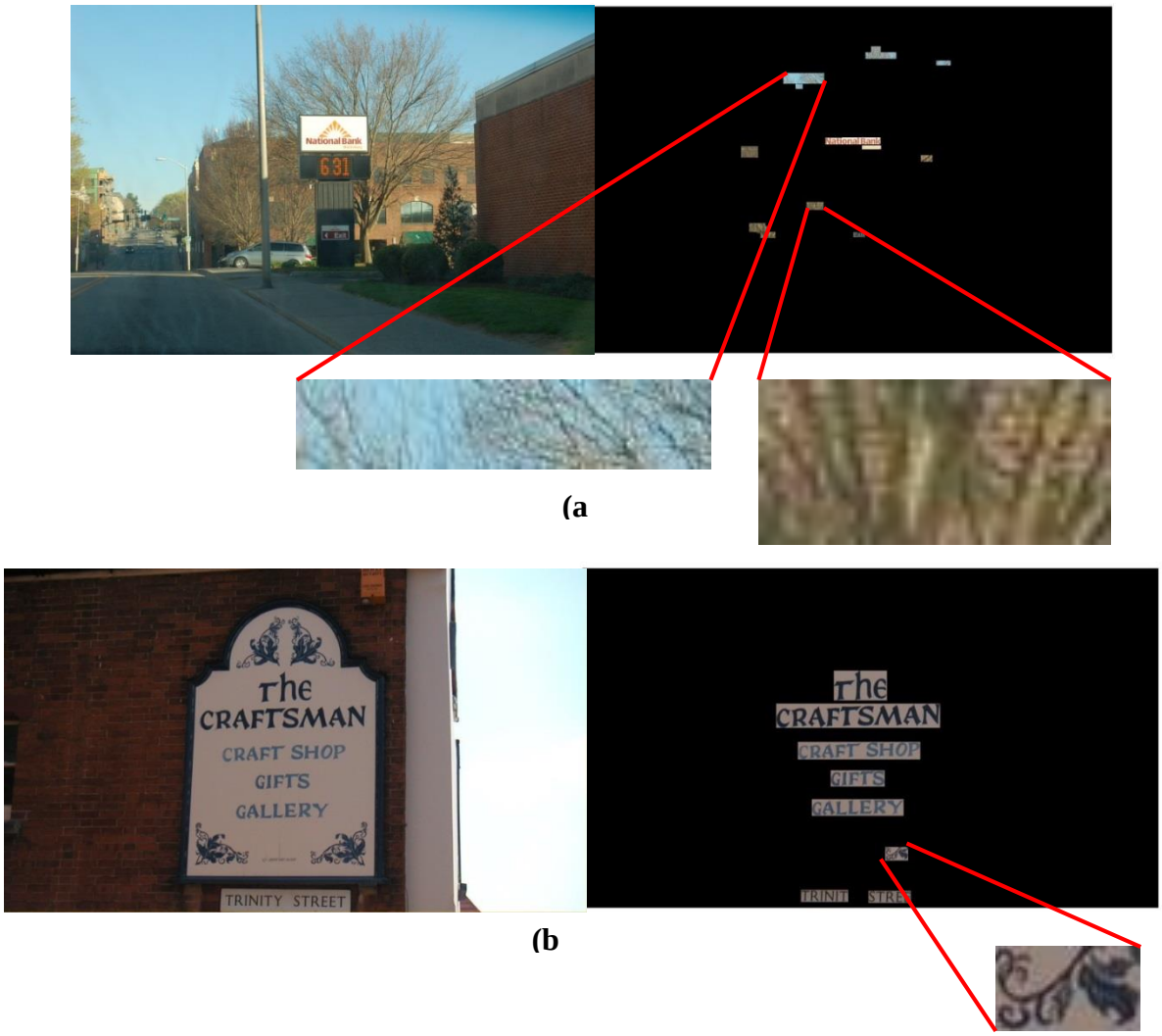


Figure 19. Examples of objects that are often misclassified as text due to their texture and spatial features.

2.5 Machine Learning/Adaptive Approach

Machine learning is often used in text localization because of its adaptability. There are two general methods of machine learning/adaptation in text localization. The first is used to 'teach' the software what is text versus non-text. The user will select regions in the image that are text and that region is classified as text. The region's features are used to create a classifier for unknown images. This method has the advantage of being trainable to find a certain type of text. It is easier to train the software rather than tuning many thresholds to the new text. This is called 'supervised machine learning'.

An example of supervised machine learning is the Support Vector Machine (SVM). Given a set of data labeled by a human as objects and a set labeled as not-objects, the SVM will find the best separation between the two. The SVM separates the feature space of the objects into two regions:

objects and not-objects. Later, when an unknown case is considered, its features will place it in either the object or non-object region.

The second method is an adaptive threshold that uses information from high confidence identifications to help identify less confident objects. Machine learning could be used here, but a computationally less expensive method is to use an adaptive threshold. An example is using color information from characters identified with a high confidence to assist in identifying less confident characters.

2.6 Previous Work Comparison

Significant previous work has not been done for optimizing text localization for UGVs. Previous work has focused primarily on either less complex scenes or general natural scenes. Simple background methods include identifying text overlaid in a video or identifying text on book covers. In these types of applications the text takes up a significant amount of the image and the background has relatively simple textures. Text in these type of images are usually clearly printed in standard fonts. Text localization methods designed for these simpler scene applications generally fail with many false positives in natural scenes.

Significant work has been done for text localization in natural scenes, but they are for general outdoor and indoor scenarios.

The only work done specifically for cameras mounted on vehicles is license plate detection [31]. These systems typically use MSER with geometric localization methods to find license plate text in natural scenes very similar to what UGVs would experience. However, license plate text is very standardized in its font and has high contrast locally (dark text on light license plate). License plates are also always rectangular and on the rear of vehicles. UGVs may be interested in non-standard fonts and signs, such as store front signs, and other less standardized text. Unfortunately, widening the search criteria to include non-standard fonts also increases the chances of false positives.

The ‘best’ text localization method varies with scene. In natural scenes two general approaches outperform the rest. The texture approaches and the CCA approaches produce the best results, but have different outputs. A promising texture approach is called the stroke filter. The stroke filter is a filter optimized for finding text strokes. For identifying text from videos clips the stroke filter was reported to have an accuracy of 95.8% and a recall of 91.1% [34]. In natural scenes MSER and SWT out-perform other CC localization techniques. The reported f-measures of MSER and SWT are comparable although SWT slightly outperforms MSER [33]. Figure 20 shows the results of SWT and MSER across two standard datasets of text within natural scenes. One study attempted to combine SWT and MSER for improve detection results, but the f-measure was the same as SWT alone [32].

Method	p	r	$f(\alpha = 0.5)$
Proposed	**0.59**	**0.59**	**0.59**
Neumann [5]	0.59	0.55	0.57
Zhang [11]	0.67	0.46	0.55
Liu [2]	0.66	0.46	0.54
Zhou[13]	0.57	0.50	0.53
Ashida [3]	0.55	0.46	0.50

(a)

Method	p	r	$f(\alpha = 0.5)$
Proposed	**0.59**	**0.62**	**0.61**
Neumann	0.69	0.53	0.60
TDM_IACS	0.64	0.54	0.58
LIP6-Retin	0.63	0.50	0.56
KAIST AIPR System	0.60	0.46	0.51
ECNU-CCG Method	0.35	0.38	0.37
Text Hunter	0.50	0.26	0.34

(b)

Figure 20. Results comparison between SWT(Proposed) and MSER (Neumann). (a) is performance against the ICDAR 2003 dataset and (b) is against the ICDAR 2011 dataset . Used under fair use, 2014

Citation: Chen, Huizhong. "ROBUST TEXT DETECTION IN NATURAL IMAGES WITH EDGE-ENHANCED MAXIMALLY STABLE EXTREMAL REGIONS." *Department of Electrical Engineering, Stanford University*

2.7 Methods Well-suited for UGVs

Scenes that a UGV would experience have complex backgrounds with many non-text objects and textures. Methods well-suited for UGVs were identified so that they could be investigated further. Edge and corner based methods were expected to perform poorly in natural scenes because natural objects introduce many edges and corners. Differentiating the false positives and the real text would be difficult purely using edge and corner information. Texture based methods; however, generally suffer less in natural backgrounds. Texture approaches generally use edge information in addition to a region's energy and spatial information. Promising results for complex backgrounds were achieved by one study using a stroke filter [34]. This method is faster than CC methods, although it does not provide any character specific information.

Of the CC methods MSER and SWT lead the pack, with SWT outperforming MSER slightly. MSER was first used in 2002 and at the time out performed other methods and was widely adopted and implemented across many languages. For example the MATLAB Computer Vision Toolbox has included a MSER function since 2011. SWT, however, was first used in 2010 and has had less time for implementation across different languages. Less work has been done with post processing and machine learning approaches for SWT than MSER. SWT's youth makes it more likely to see further improvements than MSER. SWT's novelty, increased potential for improvement and superior performance makes it a more promising candidate for UGVs.

Adaptive thresholding produced promising results in a study [35] which used the ICDAR 2003 robust reading dataset. Adaptive thresholding is font invariant although it can be sensitive to

complex backgrounds. It is also faster than both SWT and the stroke filter methods. Because of its speed adaptive thresholding was of interest for application for UGVs.

General ground vehicle specific text localization has little previous work so a breadth of methodologies was selected to move forward into testing. The stroke filter, SWT and adaptive thresholding were all tested as the initial feature extraction method. Post-processing for each method varied and was edited to improve the detection results.

Chapter 3

3 Methods and Fundamental Science

This section goes further in depth about how the more promising text localization methods work. All of these methods were tested and the results are discussed in this paper. In the selection of the methods to test for UGV applications, the net was cast wide in order to take many types of approaches into account. This was done because little previous work has been done for UGV scenes and it is not clear which methods would perform best. Adaptive thresholding was selected because of its speed and promising results in natural scenes [35]. The stroke filter was selected because it was a leader in the texture based methods. SWT was selected because it was the leader of CC methods and had increased potential for improvement. Each method will be discussed in detail to the point where they can be reproduced.

SWT and the adaptive thresholding have five general steps. (i) The first is the processing which produces the features that describe each area of the image, such as, stroke width or relative intensity. (ii) The second is character extraction. Based on the feature information in the image, character-like objects are identified while non-character objects are eliminated. (iii) Then characters are grouped into words based on the spatial information of the characters. (iv) Words are then filtered based on character specific and word specific features. (v) Finally, a 2^{nd} pass attempts to detect characters near high confidence words which may have been missed initially. This last step prevents words from having missing characters.

The stroke filter is identical except it does not have steps two, three or five. This is because the stroke filter does not return character specific information.

3.1 Adaptive Thresholding

Adaptive thresholding is an approach similar to global thresholding that sets its threshold based on the intensities of a region within the image. Typically in an image the background represents many pixels of high intensity and the text represents pixels of low intensity or vice versa. The gap between those intensities is where the threshold value would ideally be set so that the text could be extracted from the background. Global thresholding labels all pixels of values higher than a threshold as background pixels and all other pixels to text pixels. Global thresholding uses a predefined intensity threshold value entered by the user and is applied to every image. Each image is different and thus has a unique ideal threshold. Adaptive thresholding attempts to set the threshold to that ideal gap [35]. Figure 21 depicts the concept of adaptive thresholding.

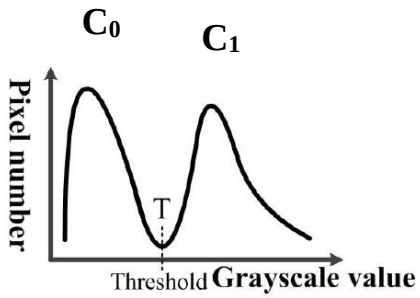


Figure 21. Adaptive thresholding attempts to separate the foreground from the background based on pixel intensities. Used under fair use 2014.

Citation: Huang, Chuen-Min. "Apply Adaptive Threshold Operation and Conditional Connected-component to Image Text Recognition." *Computer Science and Information Technology* 2: 87-94. *HRPub*.

C_0 and C_1 represent the class of the foreground and the background respectively. Trayscale intensity distribution of C_0 is 0 to T and C_1 is T+1 to L where T is the adaptive threshold value and L is 255, the maximum intensity. Calculating T is done through an iterative process where the probability of an intensity of T being the ideal cutoff is calculated. In the case of an 8-bit image the threshold can be any intensity from 0 to 255. Equations 3 through 7 show how T is calculated.

N is the total number of pixels in the image and n_i is the number of pixels of ith intensity. P_i is the rate of occurrence of the ith gray intensity. W_0 and W_1 are the sum of the occurrence rates of all pixels on either side of T. The means (m_0 , m_1) and variances (σ_0^2, σ_1^2) of each T value are summed to yield the total variance (σ_w^2) within both the background and the foreground. Adaptive thresholding minimizes σ_w^2 in order to find the best threshold.

A study [35] improved its text localization performance by applying adaptive thresholding on

$$P_i = \frac{n_i}{N} \quad [3]$$

$$W_0 = \sum_{i=0}^{T} P_i \ , \ W_1 = \sum_{i=T+1}^{L} P_i \quad [4]$$

$$m_0 = \frac{\sum_{i=0}^{T} i * P_i}{W_0} \ , \ m_1 = \frac{\sum_{i=T+1}^{L} i * P_i}{W_1} \quad [5]$$

$$\sigma_0^2 = \frac{\sum_1^T (i-m_0)^2 * P_i}{W_0}, \sigma_1^2 = \frac{\sum_{T+1}^L (i-m_1)^2 * P_i}{W_1} \quad [6]$$

$$\sigma_w^2 = W_0 \sigma_0^2 + W_1 \sigma_1^2 \quad [7]$$

horizontal strips in the image, rather than the image as a whole. The idea is that text in natural scenes is usually horizontal and does not take up the entire image. By breaking the image into smaller pieces the text represents more pixels and is easier to identify. A failure mode of this method can occur if the text happens to be along the edge of a stripe and the text is cut in two. In

this case the text usually does not represent enough pixels in either stripe to be identified. In some cases the lower bits of text are highlighted, but the upper parts are threshold out. The number of stripes used is a balance between text representation and text loss. The study cited three stripes as being the best performer [35]. In qualitative testing, images in the ICDAR 2003 dataset were used to determine the optimum number of stripes for the image. One, three, five, ten, fifty and one-hundred stripes were tested. As the number of stripes increased so did both the chance of splitting a word and the chance of extracting the text. Above five stripes the text was usually split into chunks which were misidentified. At three chunks the text was rarely split. One chunk performed poorly because the text was generally too small to be found by the adaptive threshold. Based on these qualitative results it is reasonable that best performance is achieved using three chunks.

Breaking the images into stripes may cause a problem when the image height is not evenly divisible by the number of stripes. Preventing errors requires some processing and the elimination of the remaining information. The number of image rows is divided by the number of stripes to find the width of each strip. The remainder is then cropped out of the image before the rest of the image is reduced into stripes. The lost information is very small and always at the bottom of the image.

3.2 Stroke Filter

One approach of text region identification is to apply filters to an image where the response highlights text edges, corners and textures. Previous research [17] has compared different filters in their success on text detection. Canny, Gabor, Harr and Ratio filters had previously been used for text detection; however, their responses were not ideal for finding the unique text response. The research tested a new filter called a stroke filter which specifically highlights text-like pixel regions. A stroke filter takes more region characteristics into account than other filters. The interval between the central and lateral areas of the region I_{cl} is added to ignore blurred edges around the text. The intensity difference between the central and lateral regions is denoted as S_{cl}, which should be large for text regions. The similarity between the two lateral regions is denoted as S_{ll}, which should be small for text regions. The homogeneity of the central region is denoted as H_c, which should be small for text regions. R_s indicates the relative processing speed with 1 being slowest and 5 being fastest.

Filter	I_{cl}	S_{cl}	S_{ll}	H_c	R_s
Canny	×	√	×	×	5
Gabor	√	√	×	×	1
Haar	×	√	×	×	4
Ratio	√	√	×	×	3
Stroke	√	√	√	√	4

Figure 22. Filter effectiveness for text detection. The check mark means that factor is taken into account by the filter. Used under fair use 2014.

Citation: Kim, Joongkyu. "A stroke filter and its application to text localization." *Pattern Recognition Letters*: 114-122.

The stroke filter was also tested against a test image and the output compared to other filter outputs. This comparison is shown in Figure 23. (a) is the original image. Notice that the 4 lines on the left are thin strokes similar to the stroke of text characters. The other changes in intensity are edges, but not strokes so they cannot be text. (f) shows that the 4 strokes on the left are highlighted, but not the edges on the right, which is the desirable output for identifying text. The stroke filter results in a more pronounced and precise output between similar strokes, while outputting nothing for dissimilar strokes. This is superior to the other filters which identify all the strokes, just with lesser intensity for the dissimilar strokes. Ignoring the dissimilar strokes is important for natural scenes because vegetation and background textures often contain dissimilar strokes that need to be eliminated.

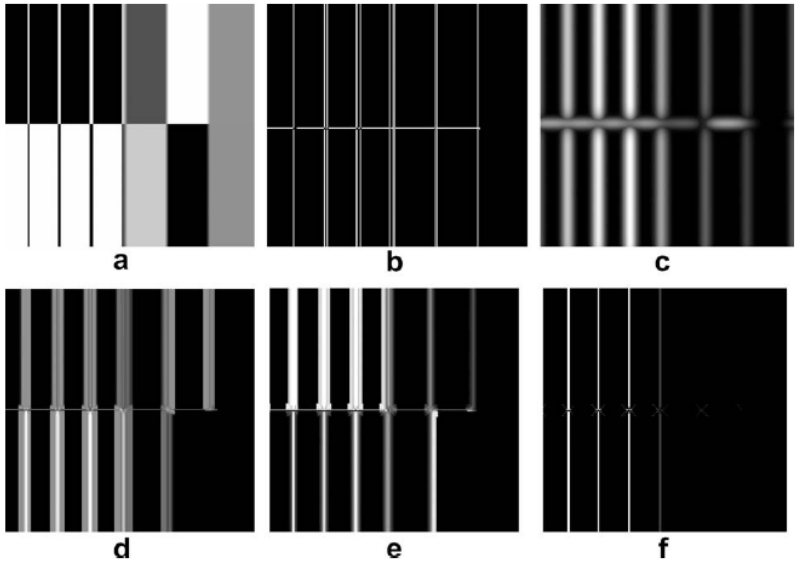


Figure 23. Comparison of filter responses on original image (a) of the Canny (b), Gabor (c), Haar line (d), Ratio edge (e) and Stroke filter (f). Used under fair use 2014.

Citation: Kim, Joongkyu. "A stroke filter and its application to text localization." *Pattern Recognition Letters*: 114-122.

3.2.1 Generating Stroke Maps with Stroke Filter

This section will cover how the stroke filter works and how it generates a Stroke Map. Each pixel in the original image gets a computed stroke filter response. An illustration of the filter is shown in Figure 24.

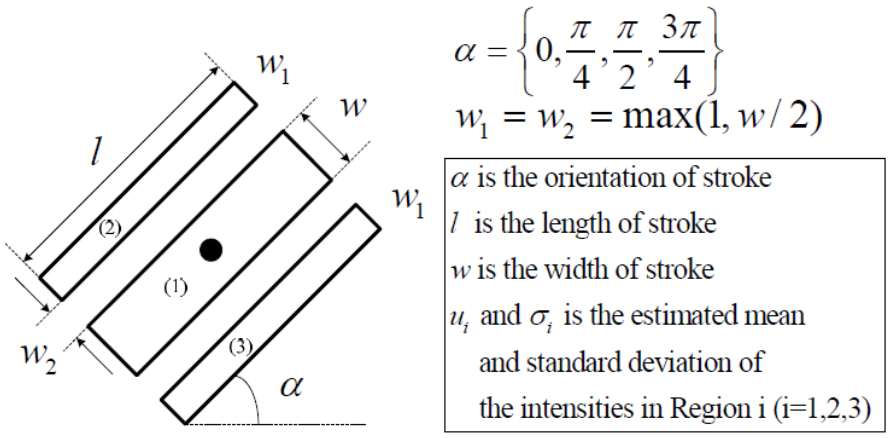


Figure 24. Illustration of a stroke filter. Used under fair use, 2014.

Citation: Kim, Joongkyu. "A stroke filter and its application to text localization." *Pattern Recognition Letters*: 114-122.

The processed pixel is at the center point of the filter, around which there are three rectangular areas of the filter. The central area is denoted as (1) and the adjacent areas are denoted as (2) and (3). The size of the stroke is denoted as (l) and (w) being the length and width of the filter. The variable α represents the orientation of the filter in radians. The stroke filter response is shown in Equation 8 where $R_{\alpha,l,w}(x,y)$ is the pixel response.

$$R_{\alpha,l,w}(x,y) = \frac{|u_1 - u_2| + |u_1 - u_3| - |u_2 - u_3|}{\max(\sigma_1, 2)} \quad [8]$$

This formula was found to yield the best results [17] with an α of 0 (horizontal), π/2 (vertical), 3π/4 (left-diagonal) and π/4 (right-diagonal). The length and the width of the stroke are equal. Equation 1 yields four different stroke maps; one map for each value of α. The size of the stroke filter can be changed as well to create more features. This is done to accommodate text strokes of different thicknesses. Rather than resizing the stroke filter the entire image can be resized to gain the same effect. Each stroke map is the same resolution as the original image.

3.2.2 Candidate Text Blocks Detection

Each stroke map is processed with a W (width) * H (height) discrete sliding window. The window is characterized with 24 features, which are then used to classify the window as text or non-text with a Support Vector Machine (SVM). Best results were achieved with a square sliding window [17]. The size of the window should ideally be half height of the text trying to be identified, although the sliding window is somewhat robust against varying text sizes. Larger window sizes risk missing text if the window does not align on the text. Smaller window sizes risk falling within the text stroke and losing information.

Text varying size across images or within images prevents a single sized window from being used on each image.In order to find text of all sizes a constant window size is used;

however, the same image will be processed multiple times at different sizes. While this is a performance hit, it allows for very large and very small text to be found in the same image with the same sliding window classifier.

The sliding window is characterized with a 24-dimensional feature vector. Twelve of these features are the mean (m_α), variance (v_α) and weighted energy (e_α) of the window across the four different stroke maps.

$$m_\alpha = \frac{1}{W * H} \sum_{(x,y)\in B} R_\alpha(x,y) \quad [9]$$

$$v_\alpha = \frac{1}{W * H} \sum_{(x,y)\in B} (R_\alpha(x,y) - m_\alpha)^2 \quad [10]$$

$$e_\alpha = \frac{1}{W * H} \sum_{(x,y)\in B} \frac{1}{1 + (x - x_c)^2 + (y - y_c)^2} R_\alpha^2(x,y) \quad [11]$$

The rest of the features characterize the spatial distribution of the window, both horizontally and vertically. Four features describe the horizontal accumulation profile (HAP) and eight describe the vertical accumulation profile (VAP). Each window in the horizontal stroke map is divided into four equally sized horizontal rectangles to create the HAP. The mean horizontal intensity of each subsection (S_i) is calculated as a feature (Equation 12). Each window in the vertical stroke map is divided into eight equally sized vertical rectangular regions to create the VAP. The mean vertical intensity of each subsection is calculated as a feature (Equation 13). The HAP and VAP characterize the spatial intensity distribution of the window, which is unique to text.

$$VAP[i]_{i=1,2,\ldots 8} = \sum_{(x,y)\in S_i} R_{\frac{\pi}{2}}(x,y) \quad [12]$$

$$HAP[j]_{j=1,2,3,4} = \sum_{(x,y)\in M_j} R_0(x,y) \quad [13]$$

The result is that each sliding window is characterized by a 24-dimensional feature vector which represents the mean, variance, weighted energy, vertical distribution and horizontal distribution. This 24-dimensional feature vector is what is used to train the SVM to identify text regions and non-text regions.

3.3 Supervised Machine Learning – Support Vector Machine

Supervised machine learning is often used to classify text and non-text regions in images based on image region features [17]. Humans give the algorithm examples of text and non-text regions to train the machine learning algorithm. The algorithm takes the training data and attempts to find a mathematical way of best segmenting the text and non-text regions. The Support Vector Machine (SVM) is a popular supervised machine learning method used to

segment text regions because it requires relatively few training examples. This section covers the history of SVMs and how they work.

SVMs are a binary classification method created in 1992 by Boser, Guyon, and Vapnik [18]. It is useful in text region classification because of its ability to deal with high-dimensional data and has a flexible kernel. Compared to other classification methods, like the neural network, a SVM requires fewer training samples and has superior generalization ability. The SVM uses supervised data classification to learn and classify future datasets [18].

First it is important to understand the details of how a SVM classifier works. A SVM takes the inputs of a set of objects characterized by an n-dimensional feature vector and a set of non-objects characterized in the same way. It then attempts to find a binary classifier with the greatest amount of margin between the objects and non-objects in the feature space. Figure 25 shows how a SVM creates a simple 2-D linear classifier given a set of objects and non-objects. Figure 25 also shows the support vectors which are the closest points to the decision boundary and determine the margin with which the two classes are separated.

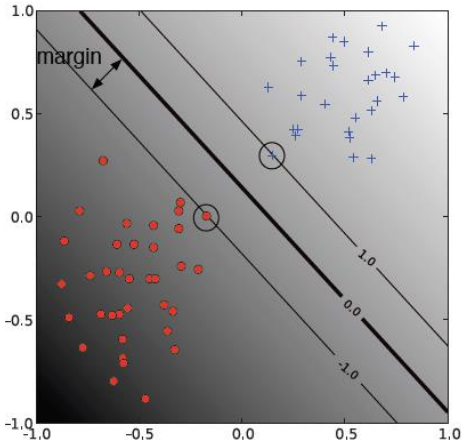


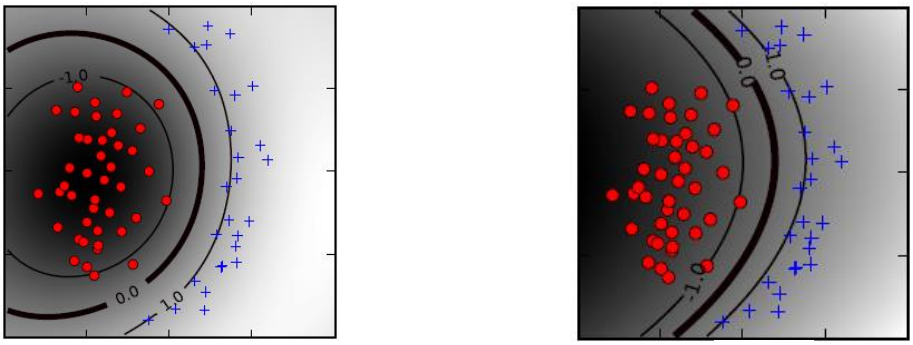
Figure 25. A simple 2-D linear classifier. The circled data points are the support vectors. Support vectors are closest to the decision boundary and determine the margin with which the two classes are separated. Used under fair use 2014.

Citation: Ben-Hur, Asa. "A User's Guide to Support Vector Machines."

The line that divides the space up between the two classes is called the decision boundary. In three or more dimensions the decision boundary is called a hyperplane. This hyperplane is not limited to being linear and can be tuned for an application for best results.

3.3.1 Customizable SVM Kernel

An advantage of an SVM is that the kernel the SVM uses is very flexible and can be tuned to fit the shape of the data. In many cases, and in the case for text localization, a linear decision boundary is not the best solution. The shape of the training data may not be separable using a linear decision boundary. In this case non-linear decision boundaries are used for better classification. Three non-linear decision boundaries are popular and are supported by MATLAB: quadratic, polynomial and Gaussian Radial Basis Function (RBF). Figure 26 shows an example of RBF and polynomial decision boundaries.



(a) **(b)**

Figure 26. Examples of non-linear decision boundaries used to create a binary classifier for 2-D data. (a) is a Gaussian or RBF decision boundary and (b) is polynomial. Used under fair use 2014.

Citation: Ben-Hur, Asa. "A User's Guide to Support Vector Machines."

3.3.2 SVM Kernel Selection & MATLAB Implementation

The best decision boundary shape depends on the shape of the data. Because the data from the stroke filter was 24 dimensional it is impossible to 'see' the shape of the data. Finding the best decision boundary for the data was done experimentally. Several different decision boundary types were tested and the one with the best result was determined to fit the data the best. The original creator of the stroke filter [17] suggests using RBF as the kernel. The results of the SVM kernel selection are in the results chapter of this paper.

Creating a SVM is unnecessary as it has been implemented across many languages. MATLAB provides two simple functions that allows the user to train and classify their data using a SVM. The SVM kernel can also be specified among other criteria. Note that MATLAB defaults to a maximum number of iterations while trying to get the classifier to converge using svmtrain. If the number of maximum iterations is met before the level of convergence is the MATLAB will quit the script and throw an error. With large amounts of data or with data that is very mixed it may be necessary to either increase the maximum iterations or the convergence threshold to avoid errors. Increasing the maximum iterations (Default 15000) will increase processing time while preserving classifier quality. Increasing the convergence threshold (Default 1e-3) will decrease iterations and thus processing time, but the quality of the classifier

may be reduced. In this paper only the maximum iterations were increased, but the training of a single dataset took many hours.

3.4 Stroke Width Transform (SWT)

One text feature that separates it from background textures and objects is consistent stroke width. SWT takes advantage of this tendency for text characters to have consistent width throughout the character. Components are initially generated using a dark-on-light and light-on-dark threshold. SWT is a two pass process since it is unknown whether the text is light and the background is dark or vice versa. Two stroke width maps of the same size as the input image are generated and text is found in each independently of each other. Stroke width is calculated for each pixel inside of a component where the pixel value is the length of the ray passing through it.

The process of calculating a single stroke width map is as follows. Initially every pixel value is set to ∞. A Canny edge detector is applied to the input image and returns edge information. The gradient direction at each edge pixel (p) is then considered. In theory, if the gradient direction of that pixel (d_p) is followed using a ray ($r = p + n * d_p, n > 0$) it should intercept another edge pixel (q) which has a gradient (d_q) approximately opposite that of $d_p (d_q = -d_p \pm \pi/6)$. If the gradient d_q does not have an opposite direction then the ray is not considered to be part of a stroke and the ray is discarded. Conversely, if d_q has an opposing gradient, every pixel along ray r is assigned the width $|\overrightarrow{p-q}|$ of the ray, which is the stroke width. The pixel's stroke width value will only be overwritten if it contains a value larger than the stroke width of the current ray. Every pixel of the output contains the minimum stroke width that occurs for a pixel. Figure 27 shows a stroke being processed with this methodology.

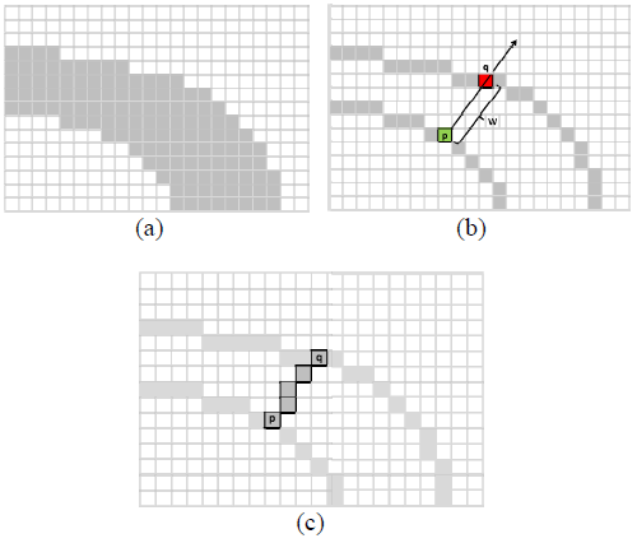


Figure 27. (a) is a depiction of a stroke, (b) is the ray drawn between the stroke edges. (c) is the pixels on the final output ray. The ray's length is the stroke width. Each pixel along the ray has its value set to the stroke width. Used under fair use, 2014.

In more complex scenarios, such as corners, the above mentioned first pass will not produce an accurate result. For this reason a second pass is done over each non-discarded ray. Each ray's median SWT value (m) is calculated and then all pixels in the ray with SWT values greater than m are set equal to m. After the second pass only corners should be effected and their stroke width values will be set to the true stroke width rather than the stroke length output by the first pass [36].

A significant advantage to using SWT is that character information is very well preserved. This is beneficial in the next post-SWT processing step where characters are extracted. Information on individual characters, such as color, can be accurately extracted.

3.5 Character & Word Extraction for CC Methods

The initial processing of the image using adaptive thresholding or SWT produces a binary image where presumably characters and background noise are both visible. Sorting out the non-character objects from the characters is the next step in text localization. Because adaptive thresholding and SWT have similar outputs both use similar character and word extraction methods. SWT outputs more features describing objects so it has additional character extraction steps.

A three step multi-pass method was used for the best results. First all individual character candidates were filtered. Then the spatial relationships between all the remaining potential characters were analyzed for characters that could be groups into words. Words identified with high confidence are then revisited with a relaxed set of filters. The second pass uses characteristics of the word's letters, such as color, to identify nearby characters that were initially filtered out. The logic is that a word is more likely to have more characters nearby with similar characteristics. This allows characters that were originally filtered to be picked up. The following sections will go into detail on how these filters were applied.

3.5.1 Character Candidate Extraction

The goal of this step is to eliminate objects that cannot possibly be characters. Individual objects are filtered by aspect ratio, stroke width to height ratio, stroke width standard deviation and the number of non-character objects within a characters bounding box. Adaptive threshold outputs do not have stroke width information so they had a more limited set of filters. Also, the original SWT output has many long stroke widths that look like cobwebs going between natural objects in the scene. Unless the text takes up a very large part of the image, which is unlikely in a natural scene, very long stroke widths are not text. For this reason the longest 20% of stroke widths are removed before character extraction begins. This can be very beneficial in character extraction because the 'cobwebs' sometimes connect characters with a thin individual stroke which makes the object identification merge letters.

All objects in the image were initially identified as continuous objects or blobs with a high response in the binary image. In MATLAB the function regionprops identifies the objects and returns information on the region, such as, area, bounding box, and centroid. The order in which the filters are applied can improve processing time. It is advantageous to start with a filter that is computationally simple and eliminates many objects. Objects eliminated early on will not take up additional processing time late for other filters.

Aspect ratio is the simplest and first filter applied. Aspect ratio is the width of the bounding box divided by its height. Letters fall into a limited range of aspect ratios. The letter 'l' has a very low aspect ratio, approximately 0.14, while the letter 'w' has a large aspect ratio, approximately 1.7. The range 0.14 to 1.7 captures a vast majority of text characters while eliminating some non-character objects. Extremely rarely (~0.1%), very non-standard or stylized fonts will have letters that fall out of this range and are erroneously filtered. However, these are individual characters within the word so during the second pass with more relaxed filters, these letters are properly identified.

The second filter, which takes more time, but eliminates many non-characters is the stroke width standard deviation filter. For each character, the standard deviation of the stroke width within the object is calculated. Text generally has low standard deviation while natural objects do not. Experimental results indicated that approximately 80% of characters will have a stroke width standard deviation of less than 2 pixels. Approximately 95% of non-text characters will be above this threshold. Text filtered at this stage still has a standard deviation less than most background objects and will be identified as text on the relaxed second pass.

Text stroke width is also proportional to character height. Although it varies with font the stroke width to height ratio was experimentally determined to be between 3.6 and 20. This filter eliminates a significant portion of background objects.

The final filter removes characters that have many objects previously identified as non-characters within their bounding box. Text usually has homogenous backgrounds right around the letters. Background scene textures; however, have lots of noise and are likely to have non-character objects within their bounding box.

3.5.2 Word Candidate Extraction

Word candidates are formed using the set of characters produced by the character extraction filters. Characters are joined into words based on several criteria. Two characters are compared at a time. If they meet the criteria to be joined and neither character is already assigned a word, a new word is created. However, if one character is already associated with a word the other character becomes part of that word as well. If both characters are already associated with different words then the two words will be joined.

In order for characters be become a word the horizontal displacement between characters must be greater than the sum of approximately half the widths of the characters. In other words the character's bounding boxes cannot overlap. This criterion is shown in Equation 14. The difference in the character heights must not be greater than the minimum character height. [16]

Characters in the same word share the same height scale. This criterion is shown in Equation 15. The vertical difference between the characters must be less than 0.7 times the minimum character height. This makes the assumption that the word's skew is less than ~45 degrees from the horizontal and that characters follow a straight line. This criterion is shown in Equation 16. Lastly, the horizontal displacement between characters must be less than 1.75 times the minimum character height. This criterion prevents non-text from being included to the beginning and end of a word. Note that 1.75 times the minimum height is rather larger than the distance between each character. This was done so that if a character in the middle of the word did not pass the character extraction filters, the characters on either side of it could still be joined. Unfortunately the long reach allows for separate words to be joined and increases the amount of non-text included. The benefit of better word cohesion is worth that loss. This criterion is shown in Equation 17. All of these criteria were tuned experimentally for best results; however, depending on the type of scene the ideal thresholds could change slightly. Below are all four criteria for character joining in equation form where x is the horizontal displacement between characters, y is the vertical displacement between characters, i and j are the two characters being considered, w is character width and h is character height.

$$x > \frac{w_i}{2.15} + \frac{w_j}{2.15}$$

$$|h_i - h_j| < min(h_i - h_j)$$

$$y < 0.7 * min(h_i - h_j)$$

$$x < 1.75 * min(h_i - h_j)$$

After the word joining process the word candidates must be confirmed. First the words are filtered based on their average character characteristics and then word characteristics.

For the first filter, the mean character's aspect ratio is considered. Each character in the word already had to pass an aspect ratio filter, but it was relaxed so that the thinnest and fattest characters could make it through. In a word; however, the average aspect ratio cannot be that extreme because a word cannot be made only of very thin or very fat characters. The average aspect ratio of letters within a word must be between 0.3 and 1.25. Otherwise the word is eliminated. For comparison, the character aspect ratio allowed characters of aspect ratio of 0.14 to 1.7.

The second filter eliminates non-text objects within the word that were initially picked up, but do not follow the word's centerline. Figure 28 illustrates this filter.

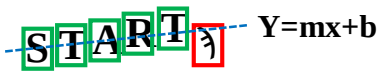


Figure 28. Characters that do not follow the words center line are eliminated.

For implementation in MATLAB the function polyfit was used to generate a line that best fits the x and y values of each characters centroid. Any character with a vertical distance from the line greater than 0.15 times its height was eliminated from the word.

An additional character filter considers the stroke width of each character within the word. Characters of the same word typically all have the same stroke width. A character with a stroke width different from the median stroke width of the word is likely not a character. These characters are eliminated from the word. Note that text in the image that is very small may have a stroke width of only a few pixels. The stroke width may have significant variance between pixelated characters, making it difficult to tell the difference between characters and non-characters. From experimental results, characters with a stroke width of more than approximately 8 pixels do not suffer from this problem.

Words are also filtered based on characteristics of the entire word. The simplest is an aspect ratio filter on the word's bounding box. A word is rarely taller than it is wide so any words with an aspect ratio less than one is eliminated. The word must also be at least three characters long. This is done because background textures generate many false-positive words with only two characters. The odds of a random background texture have three consecutive false-positive characters is much lower than having only two. Unfortunately there are a few English words with only two characters and road signs often use two letter abbreviations, such as, "St" and "Dr". Ignoring two letter words is not ideal and if character detection was improved to the point that false positives were rare, two letter words should not be ignored.

The final word filter considers the number of non-character objects within the words bounding box. Earlier a character filter eliminated characters with many non-character objects within its bounding box under the logic that text is not surrounded by noise, but non-text textures often are. This same logic is applied to each word as a whole. This reduces the number of false positives significantly without eliminating true words. It should be noted that the dots on characters like 'i' and 'j' will be considered non-text objects. This is because the 'i' and 'j' are not yet identified as such. The dots cannot be associated with the letters until after OCR, which is not part of text localization. These text dots prevent this filter from having a very low threshold.

3.5.3 2nd Pass Post Processing with Adaptive RGB Threshold

The initial pass for character and word extraction eliminates approximately 99.5% of false-positives and correctly extracts approximately 50% of text. Large portions of the missed texted are individual letters in or around an identified word. A second pass near the word's bounding box can pick up previously missed characters. This is done using an RGB adaptive threshold based on the high confidence characters and more relaxed filters

The size of the region of the second pass depends on the word being considered. Using the confident characters, a line of best fit is created with their centroids. The 2nd pass region will follow this line because characters that would have been missed are expected to be on the same line as the rest of the word. The average confident character height in then used to determine

how far away the 2nd pass should look along the line for potentially missed characters. The average confident character height multiplied by 1.15 is the distance along the word's line-of-best-fit where the 2nd pass is executed. The height of the 2nd pass box is the average confident character height times 0.5. Figure 29 illustrates an example of the 2nd pass box. Characters with a centroid within this box are reconsidered as potentially being part of the word.

Figure 29. The 2nd pass box (shown in red) where any previously rejected character in the box will be reconsidered as a potential part of the word.

Not all previously rejected characters within this region are accepted. Although this simple approach was tested and did improve word cohesion, it often picked up non-text objects as well. A superior method leverages adaptive RGB thresholds and relaxed filters.

Approximately 95% of false negatives from the first set of filters were from the stroke width standard deviation filter. This filter removed objects whose stroke width had high variance. It was successful in removing much of the background, but also removed some text characters as well. False negatives were often due to pixelated strokes and environmental variation. Stroke width is calculated by an approximation using a pixelated image where the text stroke often is less than ten pixels wide. Discretizing a thin stroke makes it difficult to accurately calculate the stroke width, especially at curves. Pixilation can cause gradient directions to misalign, which in turn causes stroke width variation. Blurry edges or strong lighting variations can also cause variation in the stroke width calculation. The 2nd pass reassesses these initially removed characters with a relaxed stroke width variance filter, if the character is near the ends of a word. It performed well in reinstating false positives. Words became more cohesive and were less likely to drop a letter within the word.

Unfortunately, relaxing the SW variance filter occasionally added non-text objects which happened to be near the ends of the word. An adaptive thresholding approach was implemented to reduce the number of non-text objects being picked up in the 2nd pass because of relaxed filter.

The SWT has the advantage of preserving character information. The original pixels in the image which made up the character are known. Leveraging the RGB information of the high confidence characters supplies an additional feature which can filter false positives on the 2nd pass. The assumption is that characters within the same word all have nearly the same RGB values. The average color information of the high confidence characters was used to create an additional 2nd pass filter. For an object to be added to the word in the 2nd pass the object had to have similar RGB values as the word. Results from this were promising. False positives were significantly reduced without incurring false negatives. This adaptive threshold approach solved the issue of picking up non-text objects in the 2nd pass.

3.6 Stroke Filter Word Extraction

The stroke filter outputs a binary image where each 16x16 pixel box is "high" for being classified as a text or "low" for background. The stroke filter reduces the block information to a 24 feature vector which makes it impossible to extract character information. Words can be identified be searching for groups of "high" blocks oriented similarly to a line or word of text. Removing false positive blocks starts with the removal small block groups that are too small to actually represent words and are likely noise from the background textures. Groups of blocks smaller than 4 were removed using the MATLAB function bwareaopen.

Often regions of text would have a background noise that would appear to be part of the text region, but would only be attached by one or two blocks of noise. True text regions were usually joined at several blocks. Without separating the noise from the true character blocks, regionprops would return large areas where no text existed. Simply applying the morphological open to the image successfully separated the noise from the true text regions, but would also remove too much appropriately identified text regions. A custom 3x3 filter was created to separate the noise from the text regions. The filter was based on the fact that noise regions were only attached by one or two blocks while real true text regions were attached by many. The filter would set a pixel from white to black if it was mostly surrounded by black pixels, i.e. it was noise connecting separate regions. The filter would set the pixel from black to white it was surrounded by mostly white pixels. The filter increased the accuracy of text region bonding boxes. This helped to fill in gaps in a text region. Equation 18 shows the filter below where x is the 3x3 region.

$$round(\frac{mean2(x)}{4} + 0.389 * x(2,2)) \quad [18]$$

The rest of the processing is similar to the CC word extraction method. Regionprops was used to extract bounding boxes of the text region candidates. The aspect ratio and the size of the bounding box was used to filter regions. Regions taller than they were wide were eliminated and very large regions were eliminated. Regions where the original contour filled less than 1/5 of the bounding box were eliminated. The logic behind this filter, also called the extent filter, was that text should fill most of a rectangular box. Very thin, long or steeply angled regions are unlikely to be text. Bounding boxes that passed these filters were considered text regions.

As mentioned in the literature review the stroke filter is not size invariant. It depends on the stroke being roughly the same width as the center section of the filter. One way to take size into account is to vary the size of the filter during training and the block during classification. Another way is to apply the same training data to differently sized versions of the image in question. This effectively achieves the same result while being easier to implement. Based on previous work [17] the best scaling sizes for the image are 100%, 75%, 50% and 40%. The smallest text will be identified in the 100% sized image while the largest text will be found in the 40% sized image. All the identified regions are combined together in the end to form a single binary text and non-text image.

3.7 Software Selection

There are several computer vision (CV) software languages to choose from when developing CV software solutions. For most applications the best languages are OpenCV, MATLAB and LabView. For this project the language selected was based on two criteria: ease of implementation on a UGV and the development time.

LabView CV is less flexible than OpenCV and MATLAB. It can have very low development time for simple, widely used tasks, but custom CV tasks would be difficult to program. LabView is not significantly implemented on UGVs.

MATLAB offers the customizability necessary for creating experimental CV software. Development time using MATLAB is low because MATLAB handles many tasks automatically. It also has many useful CV functions built into the language. Other languages need more detailed code to handle small tasks. This makes MATLAB ideal for testing new CV approaches because of its low development time. Unfortunately MATLAB is not an executable language and needs and environment to run in. MATLAB code can be converted into an executable language, but it requires expensive software from MathWorks. This means that implementation on a UGV is difficult. Once the new method has been tested and optimized in MATLAB it can be converted to a more implementable language, like C/C++.

OpenCV using C and C++ represents most of the CV code used in the past for MLVT UGVs. C or C++ is easy to implement on a UGV, but it is not as fast for code development as MATLAB. The text localization methods being tested are experimental and many of them may never actually be used on a UGV. It is more reasonable to reduce the development time for experimental methods and once finalized convert it to C/C++ for implementation. For this reason MATLAB was chosen for text localization.

Chapter 4

4 Experiment

Three of the more promising text localization methods were tested. Adaptive thresholding uses local intensity information and is very fast. The stroke filter uses texture information and is slower than adaptive thresholding. The stroke width transform uses edge and stroke width information and has the longest processing time. Experimental setup for each of these methods is discussed in this chapter.

4.1 Trial Dataset Selection

Quantifying the performance of text localization algorithms is a difficult task. Performance varies significantly with scene and type of text. For this reason standard datasets are used for comparison. The most widely used benchmark for text localization is the ICDAR datasets. They feature a variety of scenes where text font, lighting, size, and background vary significantly. They are designed to require robust text detection. The most commonly used of the ICDAR datasets is the ICDAR 2003 dataset. This dataset was chosen so that a universal comparison would be available.

The ICDAR 2003 dataset includes scenes similar to those a UGV would experience; however, many scenes are not applicable, such as, indoors or close ups. Indoor background textures do not accurately represent the backgrounds experienced by UGVs. Many close ups of text are also in the ICDAR dataset. Images where the text takes up large portions of the image are not similar to UGV scenes where the text is generally small relative to the size of the scene. Different methods may perform better on different scenes. The ICDAR 2003 dataset may not indicate how well an algorithm performs in scenes more applicable to UGVs. For this reason a custom dataset was needed that contained scenes that a UGV would be more likely to experience. Figure 30 below shows two sample images from the ICDAR 2003 dataset.

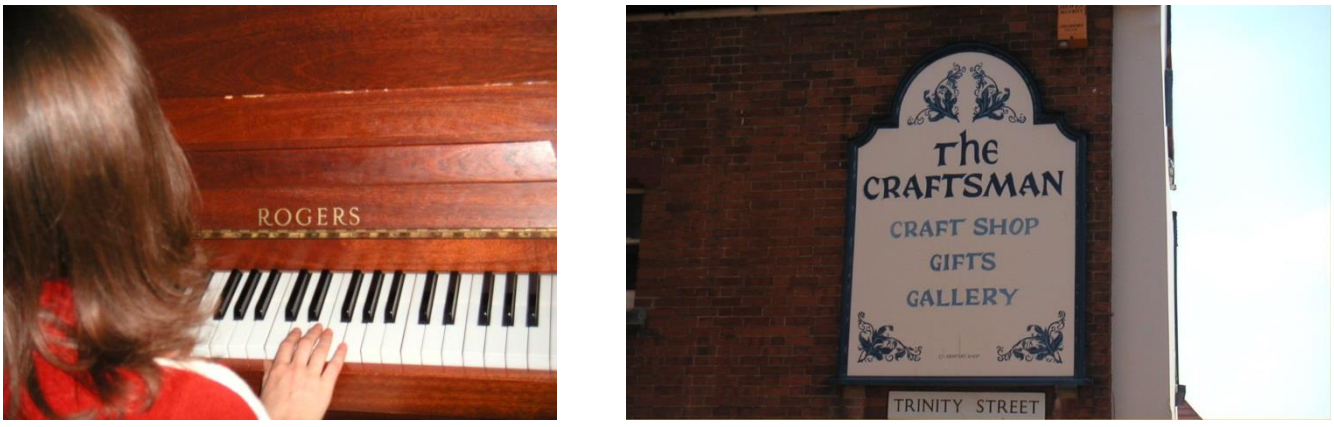


Figure 30. Two sample images from the ICDAR 2003 dataset. Note that the image on the left is a scene that is not applicable to a UGV. Approximately half the ICDAR dataset is made up of similar images of indoor and/or close up scenes.

The dataset includes 53 images taken at a resolution of 3008 x 2000. Scenes are taken from the dash of a vehicle driving on public roads in Blacksburg, Virginia. Scenes include road signs and store front text. Images in the UGV dataset contain a much smaller text-to-background ratio. The text region is generally very small in the scene because the images encompass the driver's entire field of view. Backgrounds are also more detailed and contain more objects than the ICDAR dataset. Performance on this dataset cannot be compared to existing algorithms; however, it will provide a better picture of how the algorithms would perform on a UGV.

Figure 31. The UGV dataset uses images similar to what a UGV would experience. Note that the text is small relative to the scene. Backgrounds are more complex and varied compared to the ICDAR dataset.

4.2 Performance Quantification

Text localization algorithms are typically compared using an f-measure which is proportional to two measures called precision and recall [32]. Precision and recall are calculated as follows [32]. The output of any algorithm is a set of rectangles prescribing the bounding boxes of each detected word. This output set is called the estimate. A set of ground truth boxes, regions indicated by a human, are called targets. The targets come with most datasets. For a match m_p of the target rectangle and the estimate is the area of intersection divided by the area of the minimum bounding box that encompasses both rectangles. The match will return a value of one if the boxes perfectly overlay and will return zero is they do not overlay at all. Matches between rectangles are made by finding which matches are closest to overlaying each other. The best match is defined as $m(r, R)$ where r is in a set of rectangles R.

$$m(r, R) = \max m_p(r, r') | r' \in R \quad [19]$$

Precision refers to how close the estimate bounding box is to the target bounding box. Recall refers to how much of the target bounding boxes are filled with the estimate bounding boxes. Note that it is possible to correctly identify all text and not score a perfect recall or precision because of slight differences between the human drawn bounding box and the estimated bounding box. Equations 20 and 21 show how precision and recall are calculated respectively.

$$p' = \frac{\sum_{r_e \in E} m(r_e, T)}{|E|} \quad [20]$$

$$r' = \frac{\sum_{r_t \in T} m(r_t, E)}{|T|} \quad [21]$$

The combination of recall and precision indicates a performance level, called f-measure. Precision and recall can be weighted differently with the variable $\propto$; however, it is typically set to 0.5 for an equal weighting of precision and recall [36].

$$f = \frac{1}{{}^{\propto}/_{p'} + (1-\propto)/r'} \quad [22]$$

4.2.1 Performance Context

Although a perfect *f*-measure score is 1, careful human recognizers score an average of 0.92 [47]. Different humans will draw slightly different bounding boxes and occasionally miss valid text [59]. The ICDAR 2013 competition report suggests that for practical applications, text localization needs an *f*-measure of approximately 0.80 [49].

4.3 Training SVM Classifiers

Creating a SVM classifier requires training examples of objects and non-objects. More training examples correlate with better identification rate at the cost of processing time. The ICDAR dataset comes with a XML file which describes the ground truth text regions in each image. Features are extracted from these regions and used to train the SVM.

An SVM was only used for the stoke filter method. For the stroke filter, each predefined text region in the dataset was used for training. And equal amount of non-text regions from the same images were manually cropped and used as training. The training data file size for each was approximately 15 MB. Using MATLABs `svmtrain`, the processing time was estimated to be 10+ hours per classifier. Since several classifiers were needed for testing, the training data was reduced in size to half the provided amount. At 7.5 MB per training set, each training process required approximately three hours runtime on a 4-core 3.2GHz processor.

4.4 Temporal Processing

Text localization can benefit from information across multiple frames of the same scene. Multiple frames of the same text from slightly different viewpoints may be able to be combined to better localize the text and filter out false-positives. Temporal data could greatly assist in text localization, but it effectively tracking text between frames is a challenging problem. Significant progress in object tracking algorithms has already been made and the same technology could be applied to tracking text between frames.

Two general methods could be used to merge text localization and object tracking. The first is to initially identify text objects and use the bounding boxes to show the object tracking algorithm what to track. Text localization could be done intermittently rather than on every frame. The

object tracking software could identify the text without additional text localization in the frames where no text localization is done. The text localization algorithm would have to have authority in deciding what to track. The second method is to apply text localization to each frame while simultaneously tracking the text with an object tracking algorithm. The two methods combine their results in order to check each other and remove false positives. The combination of the object tracking results and the text localization results is non-trivial and would require more work than the first method; however, is likely to produce superior results.

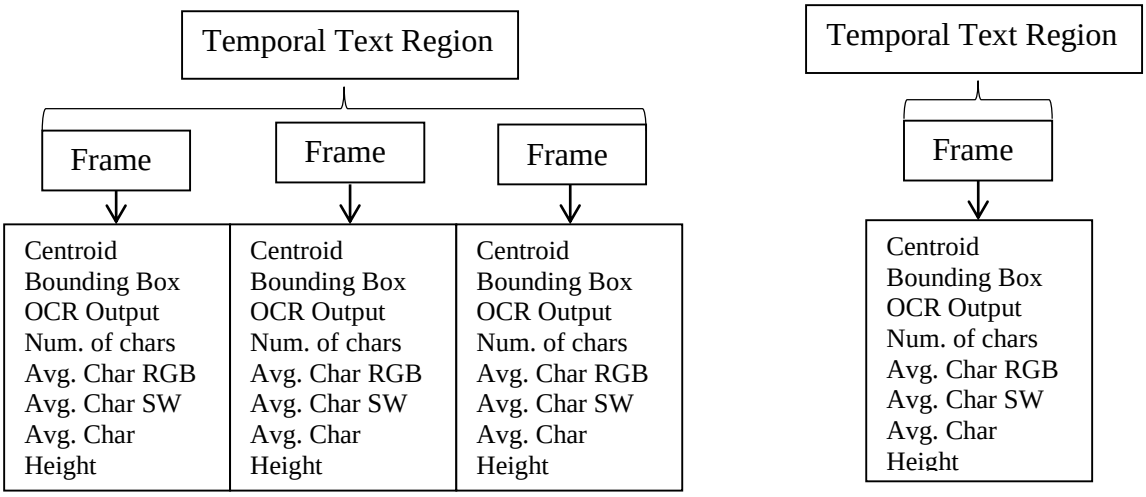
In both methods the iterative text localization could reverse false-negatives and remove false-positives. Regions containing the same content that were identified across many frames could be considered words with higher confidence. Regions identified only once among many frames could be considered noise and removed. This would improve recall and precision significantly.

Note that processing time would increase if both object tracking and text localization is performed on every frame. If the text localization process is much slower than object tracking than the first method could improve the processing time by limiting the number of frames the text localization is needed for.

Implementing a fully functional temporal processing method is outside the scope of this paper, but a proof of concept was tested. The goal of this proof of concept is to show that using temporal data can reduce false-positives and can increase the number/quality of true positives. It may also mitigate the problem where partial lines of text are extracted.

The implementation for the proof of concept used a simple object tracking algorithm that connected text regions which were identified in each frame using SWT. SWT output regions within a frame where text was found. Between two consecutive frames an object in the first frame would be matched to the closest object in the second frame. The second object's centroid was required to fall within a bounding box around the first centroid location. The bounding box accounts for the object movement between frames. If the same text region did not occur in at least one set of consecutive frames it was considered to be a false-positive and was removed. Frames after the second would be processed in a similar way; however, matching regions are also associated with the matched regions in all previous frames. In this way the same text would be considered as the same word across all the frames.

The output of the tracking method is a list of matched text regions. The output from a set of frames is a 3 dimensional array. The first dimension is the temporal text region number, the second is a list of frames where the text region was found and the third dimension contains the information about the text region, such as, the centroid, bounding box, OCR output, number of characters, average character color, average character stroke width and average character height. This word information could assist in tracking the text in later frames. Figure 32 shows a model of a temporal text region. In future iterations it may be necessary to store temporal features of a text region. This could be stored on the second tier along with the list of frames.



(a) **Probable true-positive** (b) **Probable false -positive**

Figure 32. The data structure of a temporal text region. Information from separate frames can be used to track a text region (a) and identify false positives (b).

The temporal proof-of-concept experiment was setup using seven consecutive images of a road sign. The camera approaches the road sign and translates slightly. Three different text colors, backgrounds and sizes are in the image. Eleven lines of text are in the image. The variety of text was chosen to see if it had any effect on the temporal processing.

The results of the temporal proof-of-concept are in the Results chapter.

Chapter 5

5 Results

This chapter goes into detail of the results obtained with adaptive thresholding, the stroke filter, and SWT. The performance of each of the kernels used for the stroke width SVM is also discussed with RBF performing the best. In summary, SWT out performed both adaptive thresholding and the stroke filter on the UGV dataset. SWT did not perform as well on the ICDAR 2003 dataset because of the non-standard fonts in the dataset. Failure modes of each algorithm in each dataset are examined and explained. A proof-of-concept test indicated promising results when using temporal data and SWT together. The temporal information allowed for drastic false negative reduction while preserving the true positives.

5.1 SVM Kernel Comparison

The SVM that classified the stroke filter blocks uses a 24 feature vector. Four types of SVM decision boundaries are supported by MATLAB: linear, quadratic, polynomial and Gaussian Radial Basis Function (RBF). The ideal decision boundary depends on the shape of the data in the feature space. Above three dimensions it becomes difficult for a human to see which type of decision boundary to choose. The performance of the decision boundary types was determined experimentally. In the interest of time, a small dataset was used for training and testing. The dataset was trained and then tested on itself. This was done so that the only difference in the test was the SVM decision boundary. Precision and recall of each decision boundary type was calculated.

Table 1 – Kernel Performance

	Recall	Precision	f-measure
Linear	0.08	0.02	0.03
Polynomial	.21	.11	0.14
Quadratic	.22	.19	0.20
RBF	.26	.23	0.24

RBF performed the best. The results support the original stroke filter author's findings [17]. They found that an RBF decision boundary outperformed the others. This is not surprising because an RBF filter has the ability to fit the data more closely compared to the others. Other methods require the data to have a certain shape while RBF fits to the shape of the data.

5.2 UGV Dataset

The UGV dataset contains 53 images (see Appendix) where the text regions are tagged in an .xml file. The scenes are similar to those a UGV would experience where the target text is on road signs and store fronts.

5.2.1 Algorithm Performance Comparison

The performance of the three algorithms on the UGV dataset are shown below in Table 2. *n* denotes the number of horizontal stripes the image is broken into when performing adaptive thresholding on each strip. As a comparison, the performance of randomly selecting pixels for this dataset is also listed. The random precision and recall were calculated by dividing the mean number of text pixels in an image by the total pixels in the image.

Table 2 – Performance on UGV Dataset

	Precision	Recall	*f*-measure
Random	0.015	0.015	0.015
Adaptive Thresholding (*n*=1)	0.03	0.16	0.06
Adaptive Thresholding (*n*=3)	0.04	0.16	0.06
Adaptive Thresholding (*n*=5)	0.04	0.18	0.07
Adaptive Thresholding (*n*=10)	0.04	0.21	0.07
Adaptive Thresholding (*n*=20)	0.04	0.21	0.07
Adaptive Thresholding (*n*=50)	0.04	0.22	0.07
Stroke Filter	0.03	0.01	0.02
Stroke Width Transform	0.12	0.46	0.19

The ICDAR 2013 competition report suggests that for practical applications, text localization needs an *f*-measure of approximately 0.80 [49], with the average human scoring 0.92 [47]. SWT is not close to an f-measure of 0.8; however, temporal tracking showed promising results in raising SWTs precision to a usable level. Section 5.5 outlines the results of temporal tracking.

5.2.2 Adaptive Thresholding

Testing the adaptive thresholding algorithm on the UGV dataset yielded a recall of 0.22 and a precision of 0.04, f-measure of 0.07. Several variations were used to test the impact of the number of horizontal image stripes (*n*) on performance. An *n* of one, three, ten, twenty and fifty were tested. At low values of *n* the text intensity did not make up a large enough portion of the image and the text was not highlighted. For large values of *n* the text represented a large portion of the image, but lines of text could be split.

Increasing *n* produced an improvement in results until *n*=10. Increasing *n* to values greater than 10 did not have a significant effect on the performance. This result contradicts previous work which indicated that the ideal value for *n* was three [35]. The most likely cause for this difference is the dataset. The cited study used the ICDAR 2003 dataset where text generally takes up larger portions of the image. Larger text does not need to be as isolated as small text in order to be identified.

Adaptive thresholding performed poorly overall, especially in precision, for several reasons. Adaptive thresholding depends on the text intensity to be a large enough portion of the image to be identifiable from the background intensity variation. In the UGV dataset the images cover most of a driver's field of view. This means that the text is small compared to the background

and does not provide enough of a presence to be considered by the threshold. Text that is successfully segmented is done more by a coincidence of the background intensities and is not reliable. Figure 33 shows a zoomed in section of an image containing a sign. The upper text is not threshold out well while the lower text is. This indicates the text is too small to be considered.

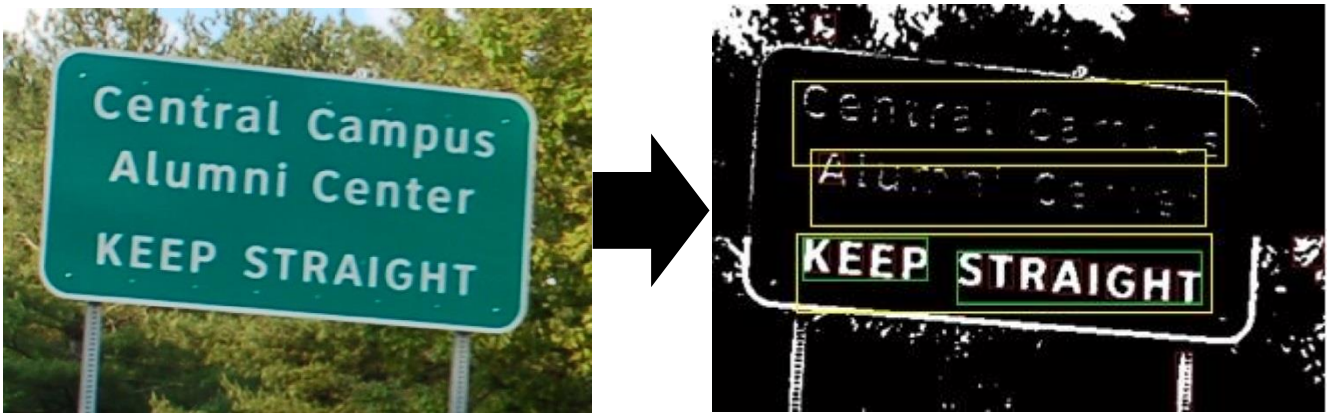


Figure 33. Adaptive thresholding on images where the text is small performs poorly due to the text intensity not being considered. The green boxes indicate detected words. Yellow are human specified regions.

Adaptive thresholding also suffers from a lack of information on character candidates. Size and spatial information is available, but there are no additional features, such as stroke width, to help distinguish characters from non-characters. This results in a high false positive rate and thus the low precision. Figure 34 illustrates this problem where horizontally repeating textures can generate false positives.

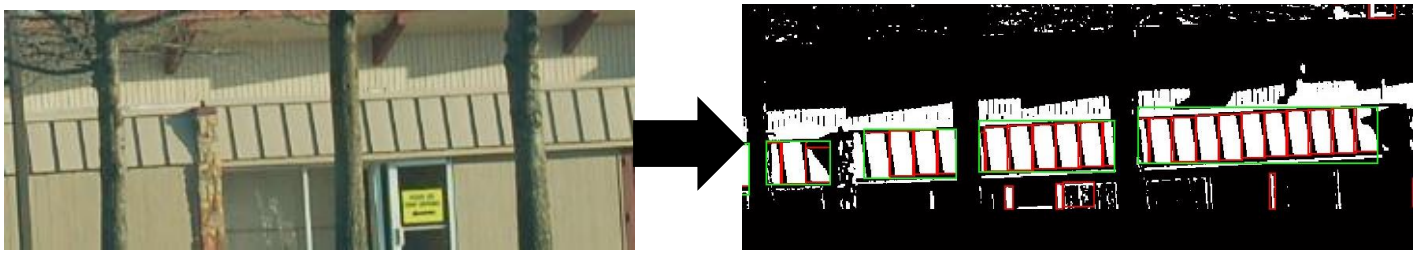


Figure 34. Horizontally repeating textures cause adaptive thresholding to fail. The lack of character features makes it difficult to remove non-character objects.

Overall adaptive thresholding alone is not a robust solution to text localization in UGV scenes. However, it does perform well if the text represents a significant portion of the image. In conjunction with other methods it could offer a solution to character extraction.

5.2.3 Stroke Filter

The stroke filter algorithm on the UGV dataset yielded a recall of 0.03 and a precision of 0.01, f-measure of 0.02. The stroke filter did not work well in any image of the UGV dataset. Outputs from the SVM were too inconsistent to create regions of high enough confidence for identification. Regions were rarely classified as text because of the inconsistent SVM output. The likely cause is that the low level feature space used to classify blocks is not distinct between text and non-text regions. Background textures, in terms of lower level information, look similar to text regions. In the original study the text was larger and more pronounced than in the UGV dataset. That distinctiveness is not present in the text of the UGV dataset. For example, the stroke filter looks for high energy areas caused by the large number of high contrast edges in text. In natural scenes vegetation and other textures have an identical energy and cannot be separated. The SVM fails in creating an accurate classifier because the features are poor. Essentially, the SVM is not able to find a difference between the background and the text when there is none based on the features used.

Overall, low level image information is not useful in classifying text regions in UGV scenes because of the similarity between text and non-text regions given only low level information.

5.2.4 Stroke Width Transform

SWT localized text in the UGV dataset with a recall of 0.46 and a precision of 0.12, f-measure of 0.19. Of the methods tested SWT performed the best in both recall and precision. The recall performance of 0.46 is on the level of state-of-the-art algorithms. The original implementation of SWT achieved a recall of 0.42 on a similar dataset [42]. The precision performance was considerably poorer compared to their precision of 0.54. False-positive failure modes were common and were caused by background textures despite the extensive filtering. Several methods of filtering the large number of false positives could be done after the text localization process. False-negatives were much less common. The 2^{nd} pass completed words and significantly reduced the number of split words or missing characters.

The most common false-positive failure mode occurred on background textures. Background textures that were horizontally repeating and had constant stroke width created the most failures. The most common texture to cause failures was vegetation. Tree limbs and grass often occur in the background and have horizontally repeating texture. Blades of grass or limbs often have a consistent stroke width and therefore evade the stroke width filter. Road pavement and architecture were the other often misclassified textures although they were much less frequent. False-positives were grouped into categories based on textures in Figure 35.

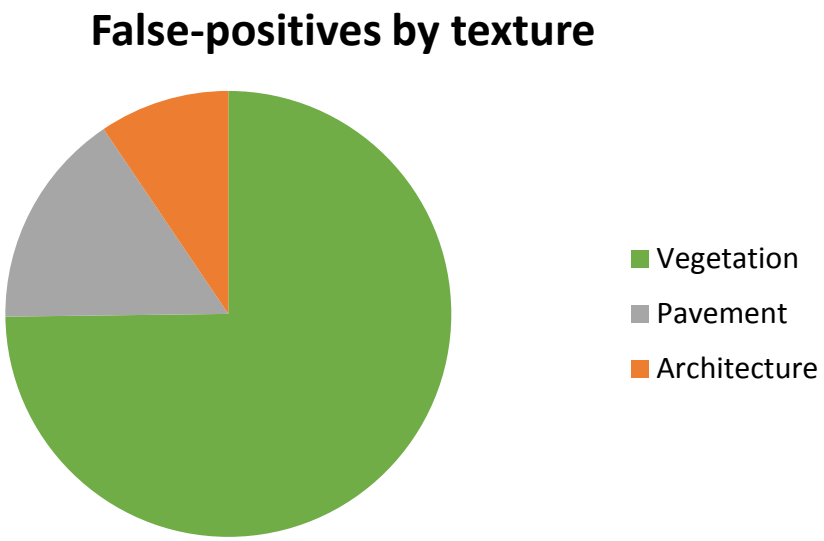


Figure 35. Categorization of textures that cause false-positives using SWT.

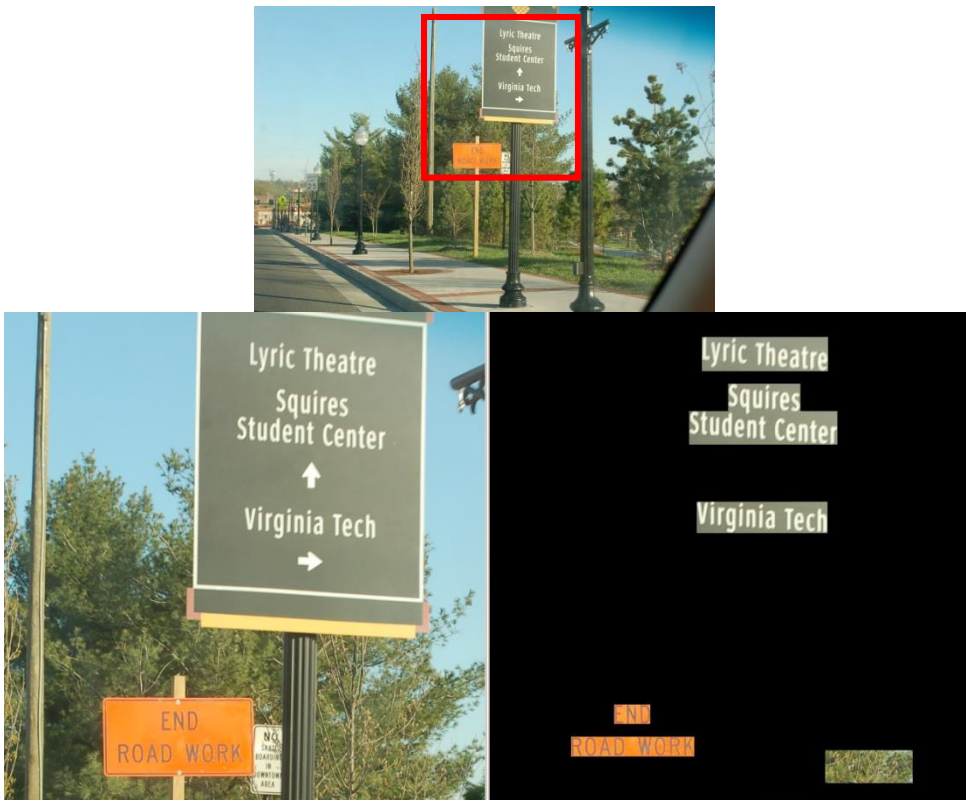
Examples of each category are shown below in Figure 36. Each of the boxes were identified as words using SWT. Misclassified textures are usually horizontally repeating and have consistent stroke width.

(a) Vegetation (b) Architecture (c) Pavement

Figure 36. Examples of image regions that were misclassified as words using SWT. All these textures are similar in that they are horizontally repeating and have consistent stroke width.

SWT did have a very high recall rate, meaning that much of the text in a scene was identified, especially for road signs. Store front signs typically have less standard fonts and therefore had a lower recognition rate than road signs. Figure 37 shows examples of results on road signs and store front text. In (a) the words ‘must’ and ‘lane’ are missing a single character at either the start or end of the word. The 2nd pass on each word should have picked up characters like this, but did not because the relaxed color intensities or stroke width filter was too liberal. Relaxing the 2nd pass could increase the chances of including missed characters, although it would also increase the chance of non-text being added to the start or end of a word. In (d) store front text using a standard font is successfully extracted. The blue text, however, is not fully segmented. This is because the characters ‘FR’ were not initially identified as characters and then in the 2nd pass they were not similar in color to the white text associated with it. This failure is because of the assumption that text on the same line will be the same color. Also note in (d) that the spatula symbol on the left of ‘Cookout’ is included in the word. Symbols near the ends of words are occasionally considered to be part of the word because of their proximity.

(a)


Lyric Theatre
Squires
Student Center
Virginia Tech
END
ROAD WORK

(b)


Central Campus
Alumni Center
KEEP STRAIGHT

(c)

(d)

Figure 37. Examples of SWT results on road signs and store front text.

The primary false-negative failure mode of SWT occurs on very small or pixelated text. Small or distant text will end up pixelated by the camera. Pixilated text makes it difficult to accurately calculate the stroke width. For example, a pixelated stroke may vary from 2 to 3 pixels wide because the true edge is within a pixel. The same stroke at a higher resolution only varies from 10 to 11 pixels. The edge is still within a pixel, but that single pixel is relatively small compared to the rest of the stroke. The pixelated stroke variance is much higher relative to its size and may be incorrectly rejected for having a high variance stroke width. Note that the camera resolution is directly proportional to text resolution. A cameras' field of view is inversely proportional to text resolution. Increasing the camera resolution or decreasing the camera's field of view can increase the text resolution and thus mitigate this failure mode. Keep in mind that increased resolution requires longer processing times. Figure 38 illustrates the problem with using stroke width on pixelated strokes.

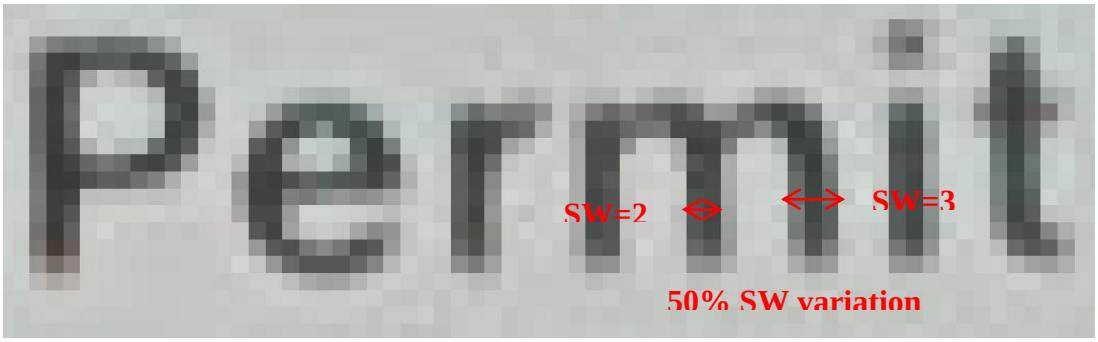


Figure 38. Stroke width calculations have high variance on pixelated text even if the true stroke is of constant width. High variance will cause characters to be misidentified as background.

A rarer failure mode for SWT occurs on characters that have varied lighting. SWT assumes a character does not have internal gradients or strongly varied lighting. Edges within a character produce inaccurate stroke widths for a character making it difficult to identify. Figure 39 shows an example of this failure mode.

Figure 39. SWT failure mode where characters have internal gradients causing incorrect stroke width calculation.

Occasionally false-negatives were due to partial character obstruction by non-text. For example, street signs often have a bolt head that goes through a character. This can join letters in some cases, or cause the character to be missed in other cases.

5.3 ICDAR 2003 Dataset

The full 251 images from the ICDAR 2003 dataset were used for evaluating the performance of each text localization method. The ICDAR 2003 dataset was chosen because it is widely used to compare different text localization methods. The ICDAR dataset was used as a reference and comparison to other methods, but does not necessarily reflect the performance in a UGV environment. It contains many different fonts, sizes and orientations of text. It is a general robust reading set with no focus on type of scene. This makes it a non-ideal candidate for testing a text localization algorithm focused on UGVs. Many scenes in the ICDAR 2003 dataset are not

applicable to a UGV environment. Handwritten text, non-standard fonts and indoor backgrounds are present in the ICDAR dataset, but are not relevant to a UGV environment. The performance on this dataset should be regarded as a point of comparison, rather than an overall measure of performance in a UGV environment. The UGV dataset better represent scenes experienced by a UGV.

5.3.1 Algorithm Performance Comparison

The performance of the three algorithms on the ICDAR 2003 dataset are shown below in Table 3.

Table 3 – Performance on ICDAR 2003 Dataset

	Precision	Recall	*f*-measure
Adaptive Thresholding (*n*=3)	0.34	0.31	0.32
Stroke Filter	0.19	0.07	0.10
Stroke Width Transform	0.37	0.23	0.28

5.3.2 Adaptive Thresholding

When applied to the ICDAR 2003 dataset, adaptive thresholding performed the best of any of the algorithms with an *f* -measure of 0.32.

The scene type greatly affects the performance of adaptive thresholding. Text that had a strong intensity difference from its background was easily identified. The strong difference allows for text extraction even if an inaccurate threshold is set. Additionally, images where large text represented a significant amount of the image were easily identified. Adaptive thresholding takes the text intensity into account and if the text is too small the threshold cannot be made accurately. Constant lighting improved detection rates as well. Adaptive thresholding is very sensitive to lighting conditions because of its reliance on intensity information. Images with inconsistent lighting had a lower probability of being identified.

Figure 40 shows an example of an image that was accurately segmented because of its text and lighting characteristics. Green boxes indicate the identified text regions.

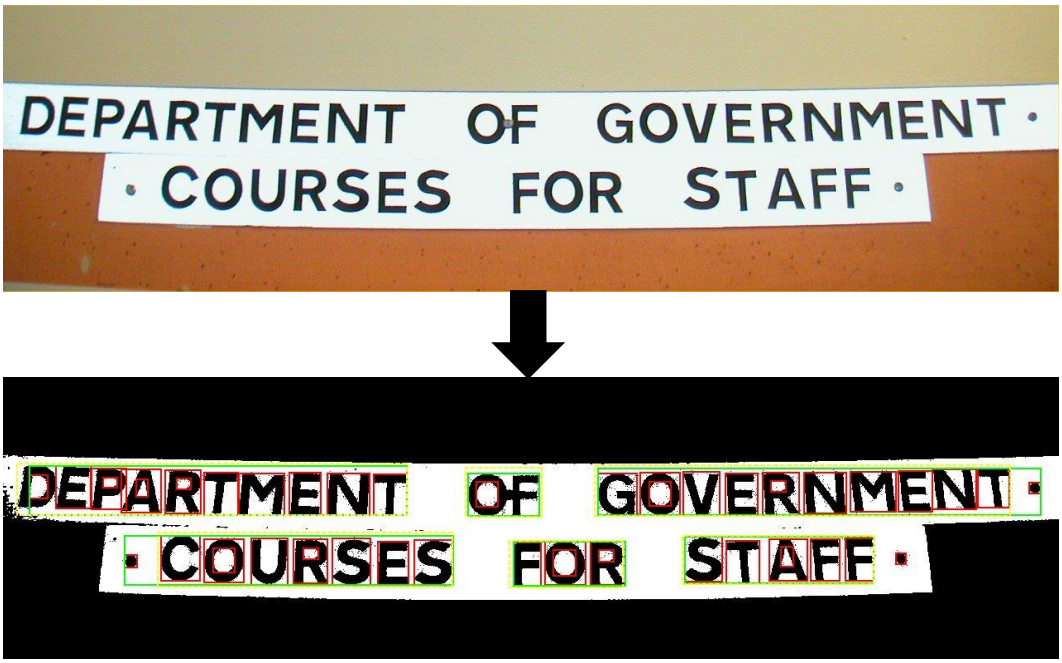


Figure 40. Lighting and text characteristics greatly affect adaptive thresholding performance. Above is an example of successful text segmentation due to consistent lighting and text intensity/size.

Failure modes usually had their roots in too much intensity variation or too little. The most common false-negative failure mode is caused by the text having a similar intensity as the background. Without a strong difference the threshold does not separate the two. Another false-positive failure mode was caused by strong lighting variation. Varied lighting makes it difficult for one blanket threshold to succeed on all parts of the image. Characters in the text will either be blacked out or whited out by the background because of the lighting variation.

Backgrounds with strong variations in intensity can also cause failures. This is because the text intensity is between the two and is not considered by the threshold. Figure 41 shows an example of this failure mode. The dark buildings are segmented from the light sky. The text's intensity is too small and has too little variation from the background to be identified.

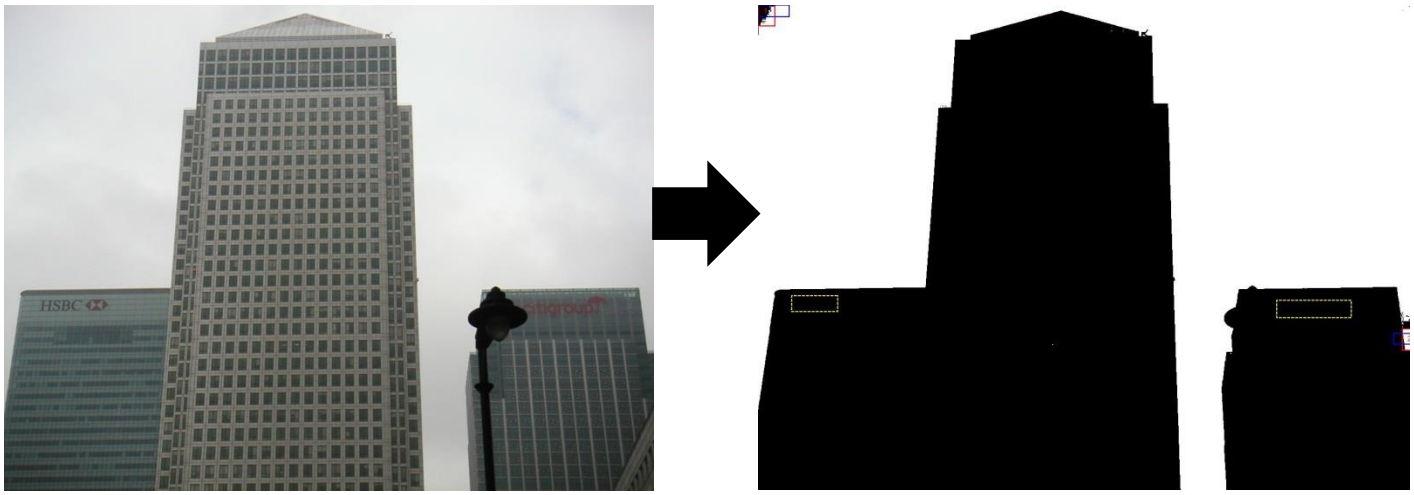


Figure 41. Adaptive thresholding failure mode where text is too small and has too little intensity variation to be identified. The yellow boxes are human labels.

Reducing the image into horizontal stripes and then applying the adaptive threshold mitigates this problem somewhat, but introduces failures of its own. If the horizontal strip splits a string of text it is possible that half the text is highlighted while the other half is not. Splitting the text's presence in each strip decreases its chance of being considered by the threshold. Chances of the OCR identifying partial letters are unlikely.

Adaptive thresholding is prone to false positives because of background textures. Background textures that vary in intensity can create character-like objects which can be misidentified. Adaptive thresholding only generates spatial and size information which makes character and word filtering difficult. Overall, adaptive thresholding has significant potential in scenes where text size and lighting can be controlled. Unfortunately in UGV environments this is not always the case.

5.3.3 Stroke Filter

The stroke filter algorithm on the ICDAR 2003 dataset yielded a recall of 0.07 and a precision of 0.19, f-measure of 0.10. The root cause of the poor performance was the same as in the UGV dataset, but it manifests itself to a lesser extent because of the relative simplicity of the backgrounds in the ICDAR dataset. The output from the SVM was not consistent enough to accurately identify text regions from the background. The likely cause is that the low level feature space used to classify blocks is not distinct between text and non-text regions. Background textures, in terms of lower level information, look similar to text regions.

An advantage of the stroke filter is that it is largely font invariable. Text in many fonts that were missed by the stroke width transform, which requires standard fonts, were identified with a stroke filter. Figure 42 shows a group of images from the ICDAR 2003 dataset and the corresponding identified text. Note that the font invariance in performance.

Figure 42. Samples of the text localization using the stroke filter on the ICDAR 2003 dataset.

5.3.4 Stroke Width Transform

SWT localized text in the ICDAR 2003 dataset with a recall of 0.23 and a precision of 0.37, f-measure of 0.28. These results underperform state-of-the-art algorithms which achieved an f-measure of 0.47 on the same dataset [42]. The lower performance is due to the ICDAR dataset containing scenes that are not applicable to UGVs. The SWT algorithm was tuned to perform on road signs and store front signs, which generally use standard fonts with consistent stroke widths. Many images in the ICDAR dataset do not have consistence stroke width and/or do not follow the algorithm's spatial assumptions. However, SWT performed well against most road signs and signs with standard text.

5.3.4.1 SWT Common Success Modes

SWT did perform well against most road signs and store front signs with standard text. Two sample images below show an example of results against a road sign and a store front sign. The red box indicates the text bounding box found using SWT. The green bounding box shows the human selection. In both cases the text is cropped closely and accurately. Both images also show a region of the background that was erroneously identified as text. In both cases the texture has strong edges and is repeating. Note that the recall for these images is approximately 0.9 and the precision is approximately 0.75. Scoring a recall or precision above .9 to .95 is unlikely, even if all the text regions are identified correctly. This is because of how the scoring is calculated and

how the human drawn boxes are slightly arbitrary. Exactly where the bounding box is drawn can make up to 10% of the error in an image.

Figure 43. Red bounding boxes indicate text identified with SWT. Green boxes indicate human identified regions. SWT accurately identifies text in street signs and store front signs with standard fonts. Background textures with strong edges and repeating patterns

5.3.4.2 SWT Common Failure Modes

Several failure modes were made apparent when using SWT on the ICDAR dataset. Non-standard fonts, horizontally repeating textures and occluded text failed to be recognized the most often. The cause of the misclassification stems from how SWT finds characters. SWT relies on consistent stroke width for character identification. Characters without consistent stroke width are often erroneously ignored. Figure 44 shows two images where fonts without consistent stroke widths are missed.

Figure 44. Examples of non-standard fonts that were missed because of their inconsistent stroke width. The red box marks text found with SWT. The green boxes are labeled by a human.

False positives were most common on horizontally repeating textures, especially textures like vegetation, fences and architecture. Repeating strokes rarely have the spatial similarity necessary for a misclassification, but the prevalence and variety of these textures make them difficult to filter. Using SWT alone may not provide enough information to eliminate these textures without performing character recognition. Figure 45 shows an urban image where the repeating horizontal windows of the building are erroneously detected.

Figure 45. Examples of architecture being identified as false positives because of its horizontally repeating pattern. The red box marks text found with SWT. The green boxes are labeled by a human.

5.4 Processing Time Comparison

Processing time was measured for each of the three algorithms. Each algorithm was tested by processing a series of 902x600px images. The average time per image was measured. No software acceleration, such as parallelizing the processing on a CPU or GPU, was used in these tests. Table 4 shows the result of the test.

Table 4 – Algorithm Processing Time Comparison

Algorithm	Speed (sec/image)
Adaptive Thresholding	0.83
Stroke Filter	8.0
Stroke Width Transform	8.9

SWT and the stroke filter have long processing times that, if not reduced, could not be used on a UGV. Implementing SWT on a UGV will require a much faster processing time than 8.9 seconds. See the Future Work section of this paper to see the details on how fast of a speed up is necessary for practical applications.

It is possible to reduce the processing times by using a GPU or a multi-core CPU. This can be implemented in MATLAB. The current code is written so that steps are sequential and would take a significant time investment to convert to parallelizable code.

5.5 Temporal Proof-of-Concept

A simple text region tracking algorithm was tested as a proof-of-concept. The goal was to determine if temporal information could reduce false positives and increase the number/quality of true positives. The occurrence of the same text line across multiple frames is called a temporal word. A sequence of seven frames containing a road sign was tested. Figure 46 shows the first and last frame of the sequence.

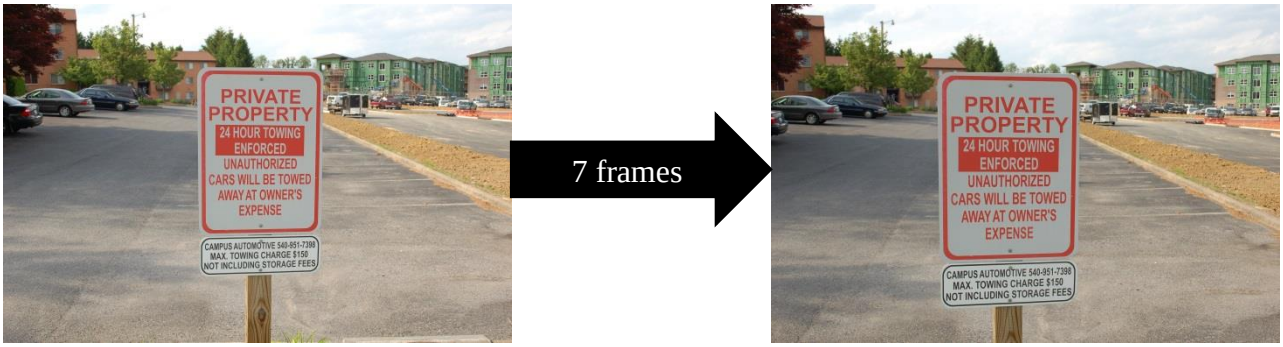


Figure 46. Above are the first and last of the seven frames in the temporal test. Text regions are tracked in each frame in order to supplement identification.

last frames with the background removed and only text regions remaining visible. Notice that the false-positives in the image do not correspond to the same real world objects across frames. The true text regions; however, are identified in both frames. This makes it easy to remove false positives without removing true-positives.

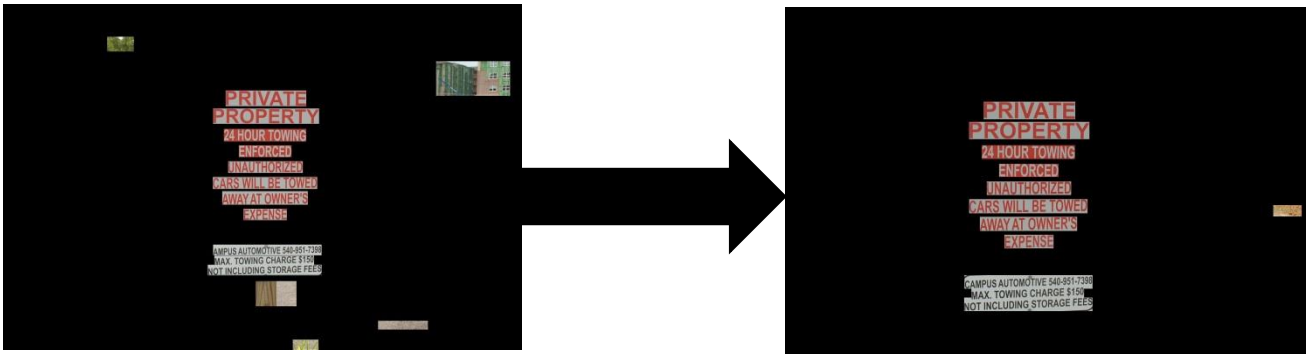


Figure 47. Above are the first and last of the seven frames in the temporal test after SWT is applied. The visible regions are text regions and the background is black. Notice how the false-positives do not reoccur across frames.

Temporal analysis eliminated 100% of false-positives in the image sequence. Originally 17 false-positives were identified in the sequence, or about 2.4 per frame. After isolated text regions were eliminated, based on adjacent frames, no false positives remained. This is very promising considering SWT's tendency to have a low precision for natural scenes. Temporal analysis could be used to significantly improve SWT's precision.

The preservation of true-positives was also promising. Ten of the eleven words were tracked well enough that the true text region made up a majority of the temporal word. An alternative metric is the number of estimated connected text regions divided by the number of true connected text regions. Using this metric 79% of text regions were linked to the correct temporal word. There were two common failure modes. Either the text region in a frame would not be associated with the temporal word or the wrong text region would be associated with the temporal word. An example of a wrong association is in Figure 48. Temporal word #4 was associated with the wrong text region in frame 1, but was correctly associated for the rest of the frames. The cause of this failure is the simple object tracking algorithm used. A more advanced tracking algorithm that uses more information to link text regions could do much better.

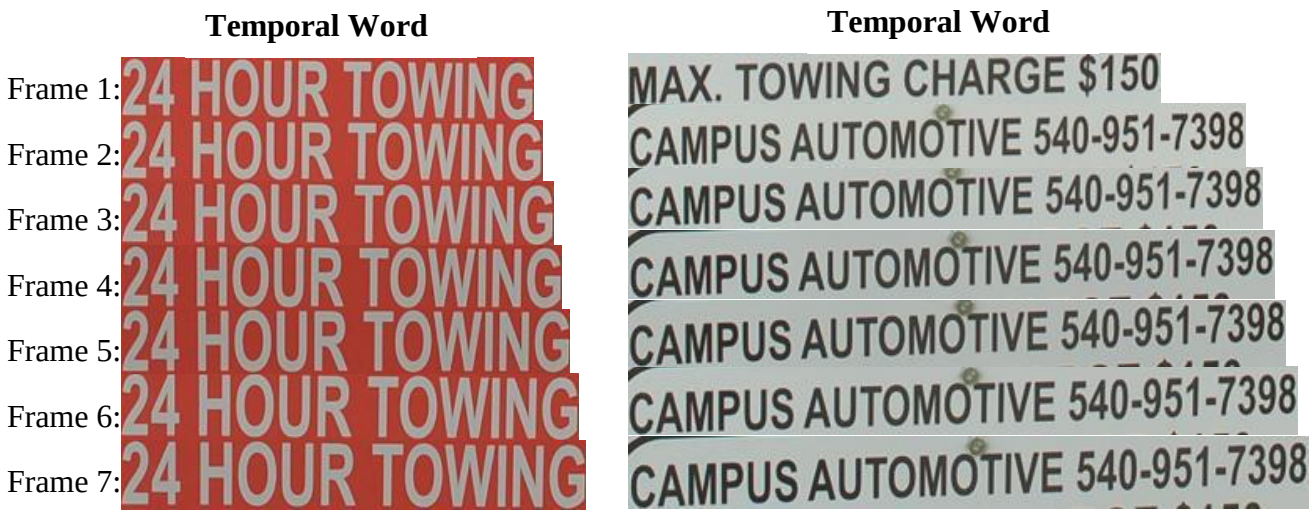


Figure 48. Above is an example of two text regions tracked through the frames. Notice how in temporal word #4's frame 1 the wrong region is found. Improving the tracking algorithm is necessary for preserving true-positives.

Overall, the results using temporal information are very promising. False positives were greatly reduced. The basic text tracking algorithm used was only successful 79% of the time and needs to be improved. Improving the tracking algorithm is non-trivial, but very possible. Temporal information has great potential in improving precision without sacrificing recall.

Chapter 6

6 Conclusion and Future Work

SWT performed the best on the UGV dataset and has potential to become a reliable solution to text localization if temporal data is utilized. SWT had a very promising recall rate, but a low level of precision. The low precision could be increased drastically based on the experiment with temporal data. Future work is required to build on this thesis to create a more reliable solution. Temporal text tracking needs to be improved in order to fully preserve the true-positives. SWT's processing time also needs to be reduced by either using a GPU or multi-core CPU. The current processing time is too slow for real time implementation especially if temporal data is going to be used.

6.1 Stroke Width Transform

SWT's recall was high (0.46), but the precision was low (0.12). The low precision was caused by background textures with similar patterns as text. Horizontally repeating strokes with constant width were the cause of most of the false-positives. Vegetation represented 3/4th of such false-positives. It is unlikely that the low precision rate could be improved using information from a single isolated image. The textures are too similar to text after applying SWT. Potential solutions exist for increasing SWTs precision. One is to use temporal data to help remove false-positives. Another is to use another method along with SWT to help in identification.

6.2 Adaptive Thresholding

Adaptive Thresholding did not perform well on the UGV dataset because the text was small relative to the scene. The text must represent a significant portion of the scene in order to be reliably extracted. For this reason it performed better on the ICDAR 2003 dataset where the text was larger relative to the scene. Although adaptive thresholding is not a viable solution on its own it could be used in tandem with a different method for verification. For example, if an algorithm could identify regions in the image with text, but not the individual characters, adaptive thresholding could be applied to the reduced region for character extraction. Using adaptive thresholding with SWT is unlikely to improve results because SWT already has character information.

6.3 Future Work

The results of using SWT for text localization on UGVs is promising, but needs further work to become more reliable. Specifically, the precision, accuracy and processing time need to be improved.

6.3.1 Temporal Tracking

The results using temporal information are very promising. False positives were reduced by 100% in the proof-of-concept test. The basic text tracking algorithm used was only successful in

preserving true-positives 79% of the time and needs to be improved. Temporal information has great potential in improving precision without sacrificing recall. Improving the tracking algorithm is non-trivial, but very possible. Existing object tracking software could be used, such as Predator, but they are not intended for text tracking and may not integrate well. If developed, an important decision to make early on will be what features of the text is tracked. I tracked each text region's bounding box, but since pixel level detail is available, other features could be tracked. SIFT features are commonly used for object tracking [54]. SWT provides pixel level detail so the text color is known and could assist in the tracking of text in scenes where multiple text colors exist.

6.3.2 Improving SWT Accuracy and Precision

The ICDAR 2013 competition report suggests that for practical applications, text localization needs an *f*-measure of approximately 0.80 [49]. Without temporal processing SWT only achieves an *f*-measure of 0.19, but the low score is primarily due to low precision. The temporal tracking proof of concept indicated that precision could be increased to human levels. Careful human recognizers score an average of 0.92 [47]. Implementation of a successful temporal tracking algorithm would be a big step in creating a practical text localization algorithm. If the precision was improved to human levels using temporal tracking and accuracy stayed the same it would yield an *f*-measure of 0.73, which is much closer to the practically usable threshold.

As noted in Section 5.2.4, SWT's accuracy is much better on high resolution text. Increasing the resolution of the text would result in a significant improvement in accuracy. The image resolution could be increased, however, this would also increase processing time. Another solution is to first identify regions of interest in the whole image and only apply SWT to those regions of interest. Sign detection could be used to find regions of interest. This would not account for store front text, but would encompass road signs. The regions of interest would then be processed using SWT, but at a much higher resolution without sacrificing processing time.

Adaptive thresholding did poorly on images where the backgrounds were complex and the text was small. This makes it a poor candidate for identifying text in the scene as a whole, but not if signs were extracted. Signs have simple backgrounds and the text is larger relative to the sign than it is to the scene. If signs or similar regions of interest were extracted, adaptive thresholding could perform just as well as SWT, but much faster.

Combining SWT and adaptive thresholding has potential in improving precision, but not accuracy. Text regions returned using SWT could be adaptively thresholded to confirm that text is present in the text region. Adaptive thresholding performs well when the text is large relative to the background so it should identify the same characters as SWT given a cropped text region. Regions where SWT disagrees with adaptive thresholding likely indicate that SWT returned a false positive. Improving accuracy using adaptive thresholding and SWT is unlikely because both methods are prone to missing small text. Both would miss the same text in the image.

6.3.3 Software Acceleration

Temporal tracking in real time requires that the text localization be done in the time between frames. The higher the frame rate the easier it is to track objects, but the less time is available for processing each frame. SWT's processing time is very high. It takes 8.9 seconds per 902x600px image. Accurate object tracking in video requires between 5 and 30 frames per second (FPS) [50, 51, 52]. Even at 5 FPS, SWT has 0.2 seconds of processing time per frame. Using real time temporal tracking with SWT will require at least 44.5 times faster processing per image.

The current processing time could be reduced significantly, although not likely to the extent needed, if the code was rewritten so that it could be done in parallel on a GPU. MATLAB can use a GPU to improve processing, but the current sequential nature of the code does not allow MATLAB to effectively use a GPU. Rewriting the code to run in parallel on a GPU would decrease its run time significantly.

Additionally, implementation of text localization on UGVs requires the code to be written in C/C++ which is faster than MATLAB. A processing speed comparison for large datasets showed that C++ is approximately 500 time faster than MATLAB [53]. Results are likely to vary depending on the application, but may offer a solution to real time temporal processing.

6.3.4 Text Localization Integrated with Vehicles Perception System

Text localization will not be an isolated perception system on a UGV. UGVs have a wealth of other object classification systems which can provide information that assists in text localization. Image regions that score high in text and a non-text object are more likely to be a false positive than text regions that do not score high as non-text regions. Vegetation was the most common false positive identified by SWT. UGVs use vegetation detection to assist in autonomy. The vegetation detection could assist in eliminating false positives from text localization. Regions identified as text that also scored high as vegetation could be eliminated because it is likely a false positive caused by vegetation.

Vegetation detection has the most potential to assist in text localization because UGVs typically already have vegetation detection and vegetation causes many false positives in text localization. It is possible that other perception systems could be used as well. As mentioned earlier, sign identification could assist in identifying regions of interest in the image where text is more likely to occur. Sign detection could be assisted by stereo vision, which is common on UGVs. Stereo vision returns the distance to objects in the scene. Road signs are likely to stand out against the background because they are closer to the vehicle than the landscape behind the sign.

6.3.5 OCR Feedback

Using feedback from the OCR software could allow for an improvement in precision and accuracy. Text regions identified using SWT could be processed with OCR and if the result is not similar to a word then the text can be eliminated. Accuracy could not be improved if the entire word is missed by the text localization algorithm, but if a text region contains a partial word it could guess at the full word. OCR feedback could be used as an unsupervised method of

machine learning where the text localization algorithm learns what it identified correctly and incorrectly based on OCR feedback.

6.3.6 Adaptive Thresholding Sliding Window

Adaptive thresholding performs well on images where the text is large relative to the scene, but not if the text is small. This thesis improved results by slicing the image into horizontal strips and applying adaptive thresholding to each strip, thus making the text larger relative to the image region. However, in many images this was not enough and the text would still be washed out. Another problem that arose from slicing an image into strips is that the text could be cut into parts and segmented differently in each strip. There are potentially better ways to break up the image for adaptive thresholding. Sliding a small square region over the image could allow the text to represent enough of the square to be identified. Also, the squares could partially overlap and overlay results so that text is not slip between squares.

References

[1] Yamamoto, K.. "Historical review of OCR research and development." *Proceedings of the IEEE*: pp. 1030. Web. 30 May 2014. Available: http://www.handwritten.net/mv/papers/mori92historical_review_of_ocr_research_and_development.pdf

[2] "OCR Applications." *OCR Applications*. N.p., n.d. Web. 30 May 2014. <http://www.cvisiontech.com/reference/general-information/ocr-applications.html?lang=eng>.

[3] Smith, Ryan. "An Overview of the Tesseract OCR Engine." : n. pag. *Google Inc.*. Web. Available: http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/33418.pdf

[5] Umansky, Mark. "A Prototype Polarimetric Camera for Unmanned Ground Vehicles." Print.

[6] "DARPA Grand Challenge (2004)." *Wikipedia*. Wikimedia Foundation, 21 May 2014. Web. 30 May 2014. <http://en.wikipedia.org/wiki/DARPA_Grand_Challenge_(2004)>.

[7] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, D. Hong, A. Wicks, T. Alberi, D. Anderson, S. Cacciola, P. Currier, A. Dalton, J. Farmer, J. Hurdus, S. Kimmel, P. King, A. Taylor, D. Van Covern, and M. Webster, "Odin : Team VictorTango's Entry in the DARPA Urban Challenge," Journal of Field Robotics, vol. 25, no. 8, pp. 467–492, 2008.

[8] "DARPA Urban Challenge." *DARPA Urban Challenge*. TORC Robotics, n.d. Web. 30 May 2014. <http://www.torcrobotics.com/case-studies/darpa-urban-challenge>.

[9] "Ground Unmanned Support Surrogate." TORC Robotics, n.d. Web. 30 May 2014. <http://www.torcrobotics.com/case-studies/guss>.

[10] Keeton, W. T.. "Magnets Interfere with Pigeon Homing." *Proceedings of the National Academy of Sciences*: 102-106. Print.

[11] Fisher, Adam. "Inside Google's Quest To Popularize Self-Driving Cars." *Popular Science* 18 Sept. 2013: n. pag. Web. Available: http://www.popsci.com/cars/article/2013-09/google-self-driving-car

[12] Palmisano, Stephen. "Stereoscopic perception of real depths at large distances."*Journal of Vision* 10: n. pag. Web. 1 May 2014.

[13] F. d'Albe. "On a Type-Reading Optophone," *Proc. Roy. Soc., Lond.*, 1914, pp. 373-375.

[14] Fish, R.M. (1976), "An audio display for the blind", *IEEE Transactions on Biomedical Engineering* (IEEE) (2): 144–154

[15] D. Shepard, "Apparatus for Reading," United States Patent Office. 2,663,758, Dec. 22, 1953.

[16] A. Kleiner, and R. Kurzweil, "A Description of the Kurzweil Reading Machine and a Status Report on Its Testing and Dissemination," *Bulletin of Prosthetics Research*, vol. 27, no. 10, Spring 1977, pp. 72-81

[17]. Kim, Joongkyu. "A stroke filter and its application to text localization." *Pattern Recognition Letters*: 114-122. *yaroslavvb*. Web. 12 May 2014. Available: http://www.yaroslavvb.com/papers/jung-stroke.pdf

[18] Ben-Hur, Asa. "A User's Guide to Support Vector Machines." : n. pag. Web. 1 May 2014. Available: http://pyml.sourceforge.net/doc/howto.pdf

[19] Smith, Ray. "An Overview of the Tesseract OCR Engine." *Google Inc.*: n. pag. Web. 1 May 2014.

[20] Smith, Ray. "A Simple and Efficient Skew Detection Algorithm via Text Row Algorithm." : n. pag. Web. 1 May 2014. Available: http://www.hpl.hp.com/techreports/94/HPL-94-113.pdf

[21] Breuel, Thomas. "High-Performance OCR for Printed English and Fraktur using LSTM Networks." : n. pag. Web. 1 May 2014. http://iupr1.cs.uni-kl.de/~shared/publications/2013-breuel-high-performance-ocr-for-english-and-fraktur-using-lstm-networks.pdf

[22] Jirasuwankul, N. "OCR Output Error from Reading Skew Text and Anti-Skew by Using Projective Transform Technique." *Automation and Control Research Unit*: n. pag. Web. 1 May 2014. Available: http://www.kmitl.ac.th/lej/PDFjournal54/Volume28_No4_DEC2554_(4)1418.pdf

[23] Lu, Shuian. "Camera Document Restoration for OCR." : n. pag. Web. 1 May 2014. Available: http://imlab.jp/cbdar2005/proceedings/papers/O1-3.pdf

[24] Liu, Qifeng. "STROKE FILTER FOR TEXT LOCALIZATION IN VIDEO IMAGES." : n. pag. Web. 1 May 2014. Available: http://funkybee.narod.ru/misc/0001473.pdf

[25] BEN SALAH, Ahmed. "Adaptive Detection of Missed Text Areas in OCR Outputs: Application to the Automatic Assessment of OCR quality in Mass Digitization projects." *Document Recognition and Retrieval*: n. pag. Web. 1 May 2014. Available: http://hal.inria.fr/docs/00/82/05/64/PDF/Adaptive_Detection_of_Missed_Text_Areas_in_OCR_Outputs_Application_to_the_Automatic_Assessment_of_OCR_quality_in_Mass_Digitization_projects_pr_2_.pdf

[26] Wu, Victor. "Finding Text in Images." : n. pag. Web. 1 May 2014. Available: http://maroo.cs.umass.edu/getpdf.php?id=311

[27] MIKOLAJCZYK, KRYSTIAN. "Scale & Affine Invariant Interest Point Detectors."*International Journal of Computer Vision* 60: 63-86. Web. 1 May 2014. Available: http://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/mikolajczyk_ijcv2004.pdf

[28] Jain, Anil K.. "Text information extraction in images and video: a survey." *Pattern Recognition*: 977-997. Web. 30 May 2014.Available: http://www.cse.msu.edu/prip/Files/TextDetectionSurvey.pdf

[29] Matas, J. "RobustWide Baseline Stereo from Maximally Stable Extremal Regions."*Center for Machine Perception, Dept. of Cybernetics*: n. pag.*http://cmp.felk.cvut.cz/*. Web. 1 May 2014. Available: http://cmp.felk.cvut.cz/~matas/papers/matas-bmvc02.pdf

[30] Wang, Wei. "Car License Plate Detection Based on MSER." *IEEE*: n. pag. Web. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5768335

[31] Kim, Jong-Bae. "MSER and SVM-Based Vehicle License Plate Detection and Recognition System." *Communications in Computer and Information Science* 310: 529-535. Online.Avaiable: http://link.springer.com/chapter/10.1007%2F978-3-642-32692-9_66

[32] Chen, Huizhong. "ROBUST TEXT DETECTION IN NATURAL IMAGES WITH EDGE-ENHANCED MAXIMALLY STABLE EXTREMAL REGIONS." *Department of Electrical Engineering, Stanford University*: n. pag. Web. 1 May 2014. Available: http://www.stanford.edu/~hchen2/papers/ICIP2011_RobustTextDetection.pdf

[33] Li, Yao. "Scene Text Detection via Stroke Width." *21st International Conference on Pattern Recognition*: n. pag. Web. 1 May 2014. available: http://cs.adelaide.edu.au/~yaoli/wp-content/publications/icpr12_strokewidth.pdf

[34] Li, Xiaojun. "FAST AND EFFECTIVE TEXT DETECTION." *Graduate University of Chinese Academy of Sciences*: n. pag. *www.google.com*. Web. 1 May 2014.

[35] Huang, Chuen-Min. "Apply Adaptive Threshold Operation and Conditional Connected-component to Image Text Recognition." *Computer Science and Information Technology* 2: 87-94. *HRPub*. Web. 1 May 2014.

[36] Ofek, Eyal. "Detecting Text in Natural Scenes with Stroke Width Transform."*Microsoft Corporation*: n. pag. *research.microsoft.com*. Web. 1 May 2014.

[37] U. Gargi, D. Crandall, S. Antani, T. Gandhi, R. Keener, and R. Kasturi, A System for Automatic Text
Detection in Video, Proc. of International Conference on Document Analysis and Recognition, 1999,
pp. 29 –32.

[38] E. Svoboda, “PopSci’s Darpa Grand Challenge Preview: Update #3,” *Popular Science*, 2005. [Online]. Available: http://www.popsci.com/scitech/article/2005-10/popscis-darpa-grand-challenge-preview-update-3.

[39] "The Velodyne High Definition LiDAR (HDL) Grabber.” PCL, n.d. Web. . <http://pointclouds.org/documentation/tutorials/hdl_grabber.php>.

[40] *Introduction to IBM data processing systems*. Washington: [U.S. G.P.O.], 1966. Print.

41] Bourlard, Hervé. "Text detection and recognition in images and video frames." *Pattern Recognition*: 595-608. Web. 12 May 2014. [Online]Available: http://www.idiap.ch/~odobez/publications/ChenOdobezBourlard-PatternRecognition2004.pdf

42] Ofek, Eyal. "Detecting Text in Natural Scenes with Stroke Width Transform."*Microsoft Corporation*: n. pag. *research.microsoft.com*. Web. 1 May 2014.

[43] J. Ohya, A. Shio, and S. Akamatsu, Recognizing Characters in Scene Images, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16 (2) (1994) 214-224.

[44] C.M. Lee, and A. Kankanhalli, Automatic Extraction of Characters in Complex Images, International Journal of Pattern Recognition Artificial Intelligence, 9 (1) (1995) 67-82.

[45] Yu Zhong, Kalle Karu, and Anil K. Jain, Locating Text In Complex Color Images, Pattern Recognition, 28 (10) (1995) 1523-1535.

[46] H. K. Kim, Efficient Automatic Text Location Method and Content-Based Indexing and Structuring of Video Database, Journal of Visual Communication and Image Representation 7 (4) (1996) 336-344.

[47] Su, Bolan. "Character Recognition in Natural Scenes using Convolutional Co-occurrence HOG." : n. pag. Web. 23 June 2014. Available: http://www.comp.nus.edu.sg/~tians/papers/ICPR2014_Convolution-CoHOG_Su.pdf

[49] Karatzas, Dimosthenis. "ICDAR 2013 Robust Reading Competition." : n. pag. Web. 24 June 2014. Available: http://dag.cvc.uab.es/icdar2013competition/files/icdar2013_competition_report.pdf

[50] Li, Yuan. "Tracking in Low Frame Rate Video: A Cascade Particle Filter with Discriminative Observers of Different Lifespans." : n. pag. Web. 25 June 2014. Available: http://media.cs.tsinghua.edu.cn/~ahz/papers/YuanLi-cvpr2007-0231.pdf

[51] Korshunov, Pavel. "Reducing Frame Rate for Object Tracking." : n. pag. Web. 25 June 2014. Available: http://infoscience.epfl.ch/record/176953/files/pavelkorshunov-mmm10-paper.pdf

[52] Boroujeni, Hamidreza. "Tracking Multiple Variable-Sizes Moving Objects in LFR Videos Using a Novel genetic Algorithm Approach." : n. pag. Web. 25 June 2014. Available: http://link.springer.com/chapter/10.1007%2F978-3-642-32826-8_15#page-2

[53] Andrews, Tyler. "Computation Time Comparison Between Matlab and C++ Using Launch Windows." : n. pag. Web. 29 June 2014. Available: http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1080&context=aerosp

[54] CHAOYANG, ZHO. "Video Object Tracking using SIFT and Mean Shift." : n. pag. Web. 27 June 2014. Available: http://publications.lib.chalmers.se/records/fulltext/136882.pdf

[55] LAVRINC , DAMON. "Nissan Promises to Deliver Autonomous Car by 2020." . Wired, n.d. Web. 1 July 2014. <http://www.wired.com/>. Available: http://www.wired.com/2013/08/nissan-autonomous-drive/

[56] "Development and Utilization of Robotics and Unmanned Ground Vehicles." : n. pag. Web. 1 July 2014. Available: http://www.ndia.org/Divisions/Divisions/Robotics/Documents/Content/ContentGroups/Divisions1/Robotics/JGRE_UGV_FY06_Congressional_Report.pdf

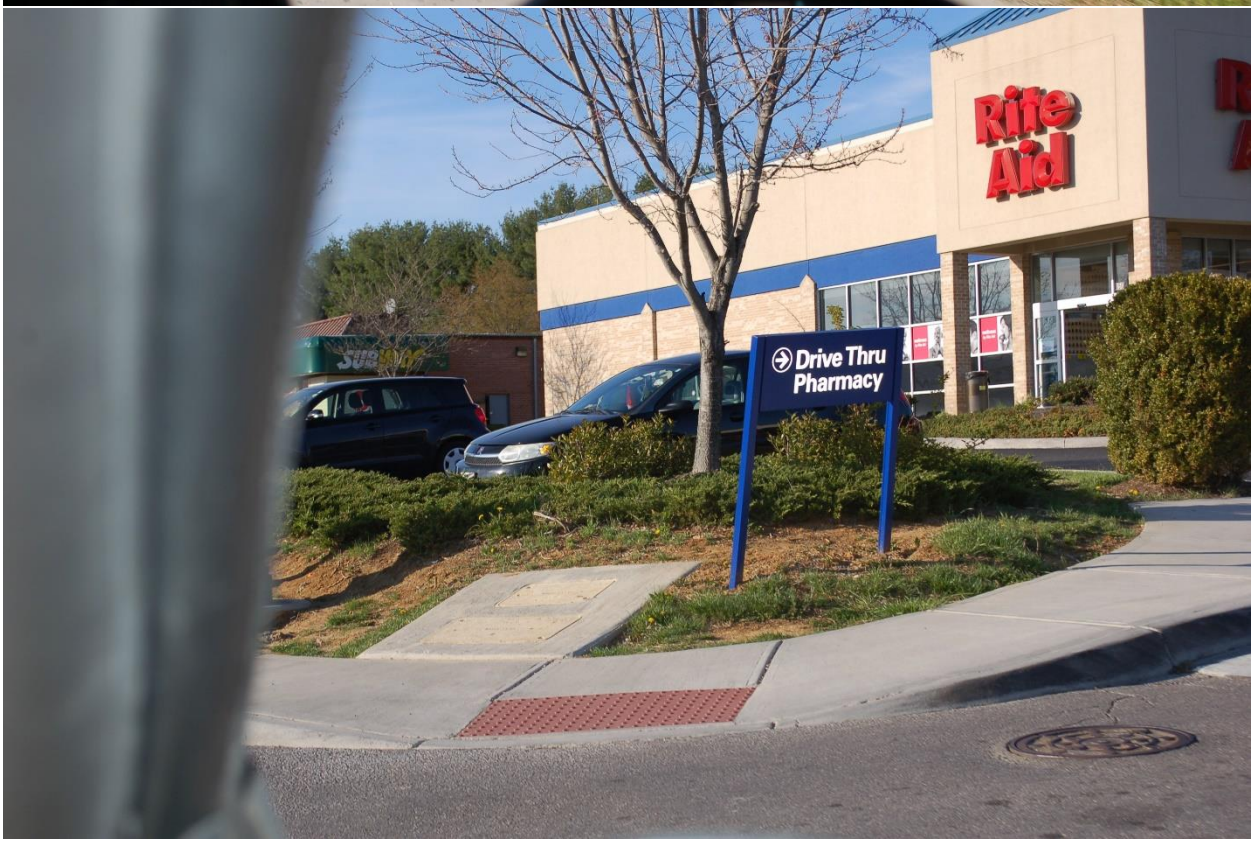
[57] Appelqvist, Pekke. "Mechatronics Design of an Unmanned Ground Vehicle for Military Applications." : n. pag. Web. Available: http://www.intechopen.com/books/mechatronic-systems-applications/mechatronics-design-of-an-unmanned-ground-vehicle-for-military-applications

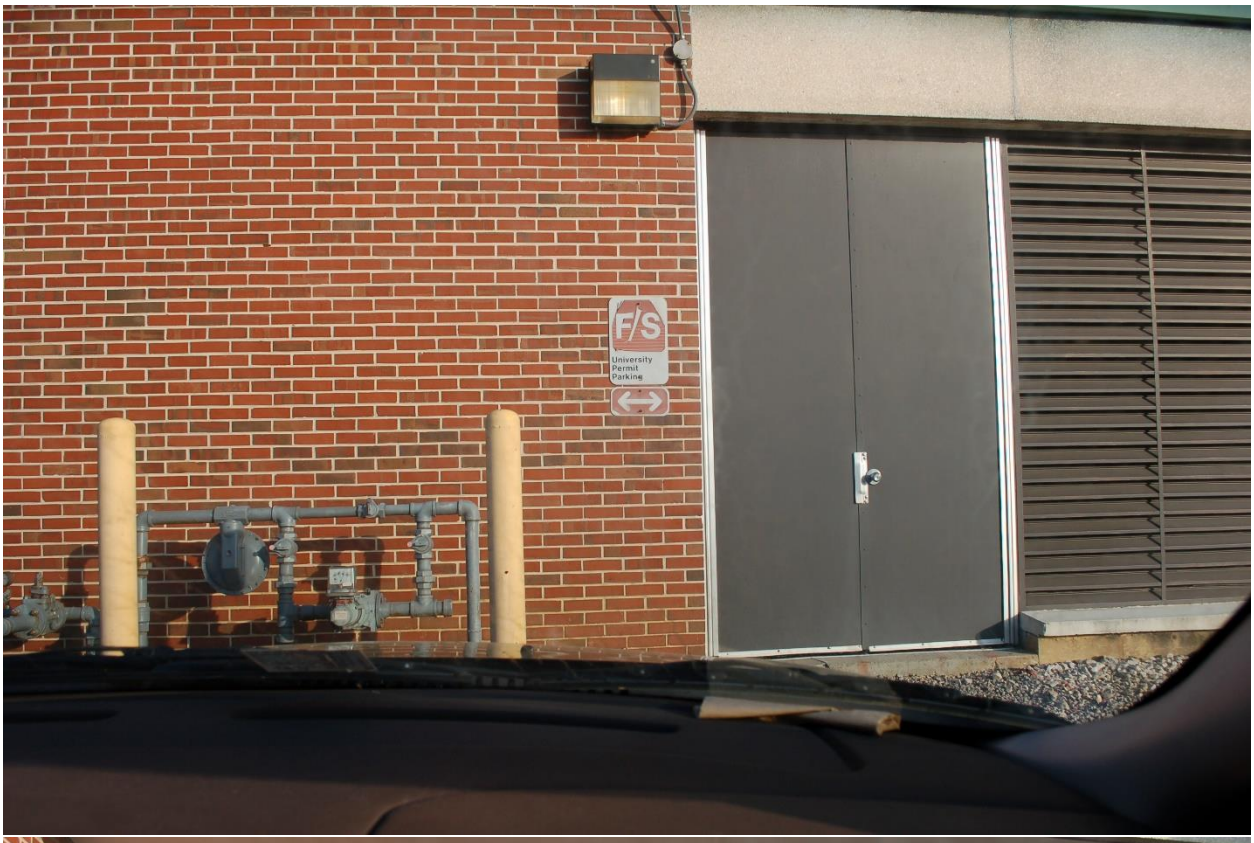
[58] Yi, Chucai. "Assistive Text Reading from Complex Background for Blind Persons." : n. pag. Web. 10 July 2014. Available: http://www-ee.ccny.cuny.edu/wwwn/yltian/Publications/CBDAR11-Yi-Tian.pdf

[59] Lucas, S.M.. "ICDAR 2003 Robust Reading Competition." *IEEE*: 682 - 687. Web. Available: http://algoval.essex.ac.uk/rep/textloc/IjdarSpecialFinal.pdf

Appendix A: UGV Dataset (53 Images)


Rite Aid
UCB
830
Kroger
UNLEADED
3.29 9
we're game
Pharmacy


Rite Aid
Drive Thru Pharmacy

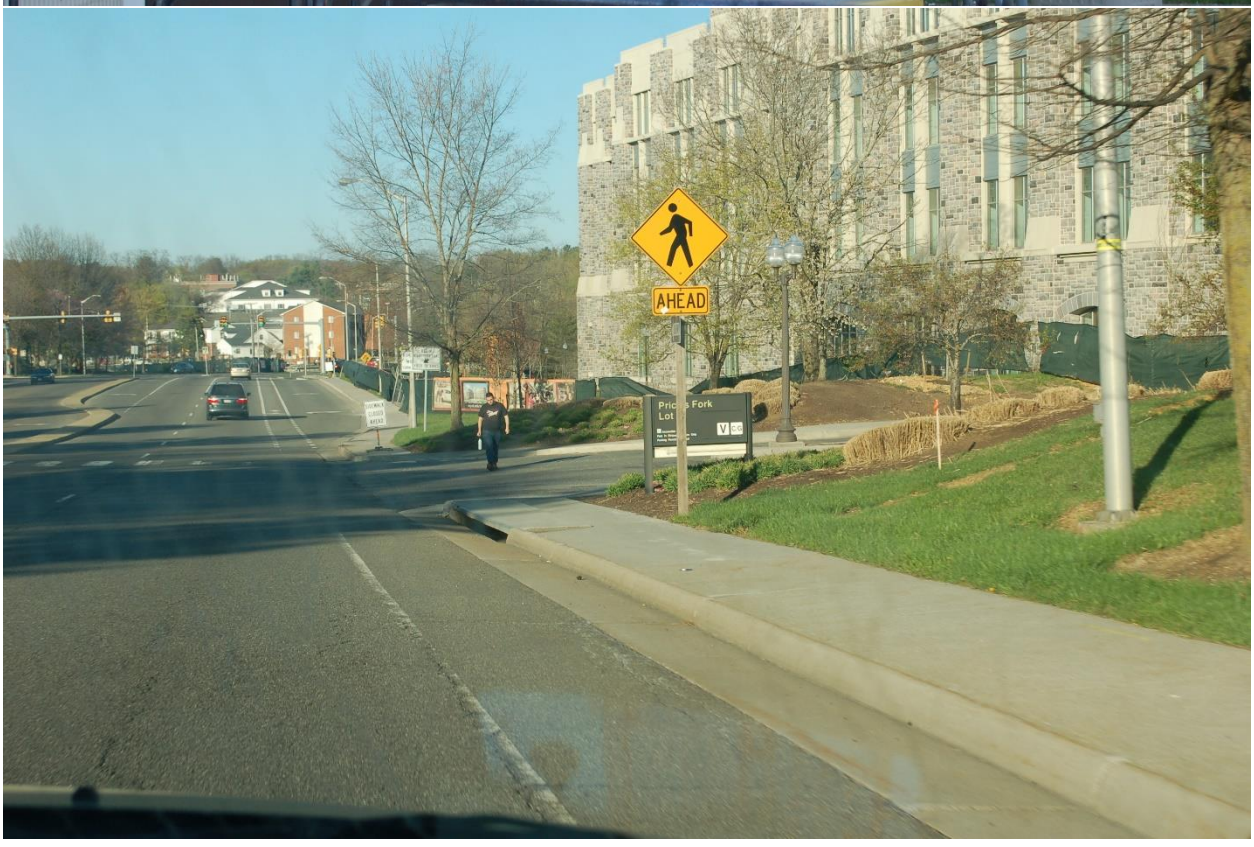

F/S
University
Permit
Parking


NO
PARKING
LOADING
ZONE
DRIVERS
SHUT OFF ENGINE
WHILE PARKED
AT DOCK


ONE WAY


ONE WAY
DO NOT
ENTER
The Real Estate Book


Blacksburg
Main Post Office
Blacksburg, VA
909
Select it.
Stamp it.
Send it.


AHEAD


RIGHT LANE MUST TURN RIGHT
BEGIN RIGHT TURN LANE YIELD TO BIKES
SIDEWALK CLOSED AHEAD


Historic Downtown
Moss Arts Center


Parks & Recreation
Historic Downtown
Municipal Complex


BIKE LANE
ENDS
LEFT LANE
RIGHT LANE


708
Valley
EATS
Main St
Downtown
YIELD TO
TRAFFIC
IN CIRCLE

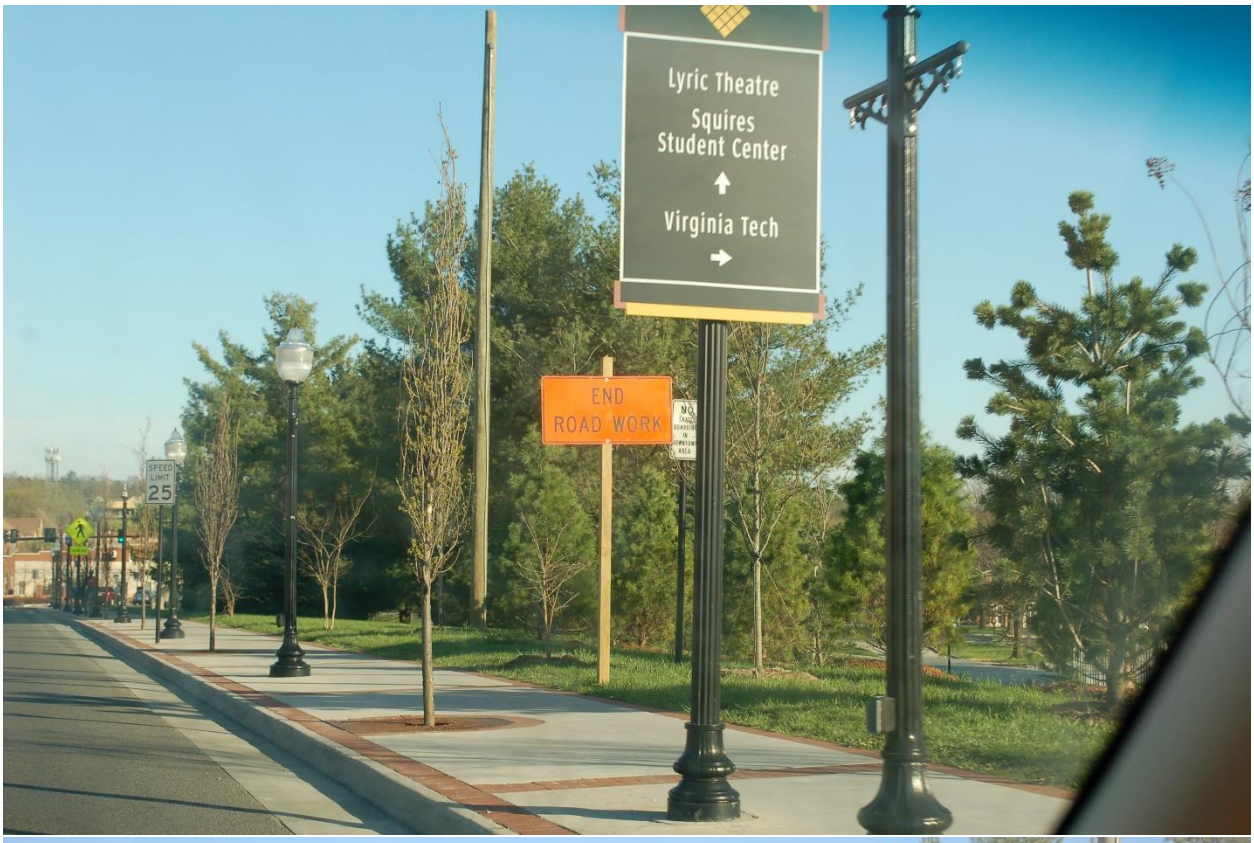

SCHOOL
BUSES
ONLY


Tall Oaks Dr
Cambridge Rd
STOP

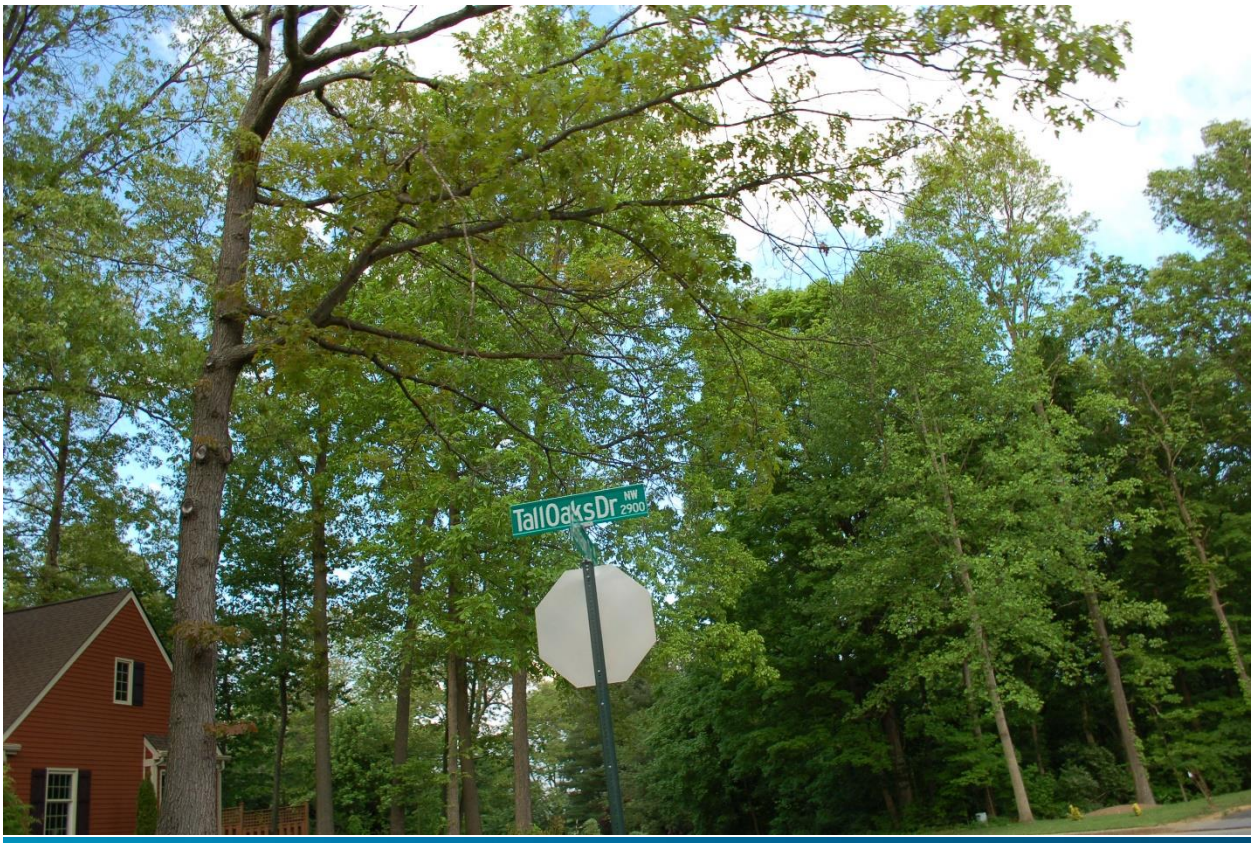

END
CENTER LANE
ONLY
NEW TRAFFIC PATTERN AHEAD
Turner St


NO
OUTLET
STOP
Cambridge Rd NW
600


Turner St


Lyric Theatre
Squires
Student Center
Virginia Tech
END
ROAD WORK
SPEED
LIMIT
25

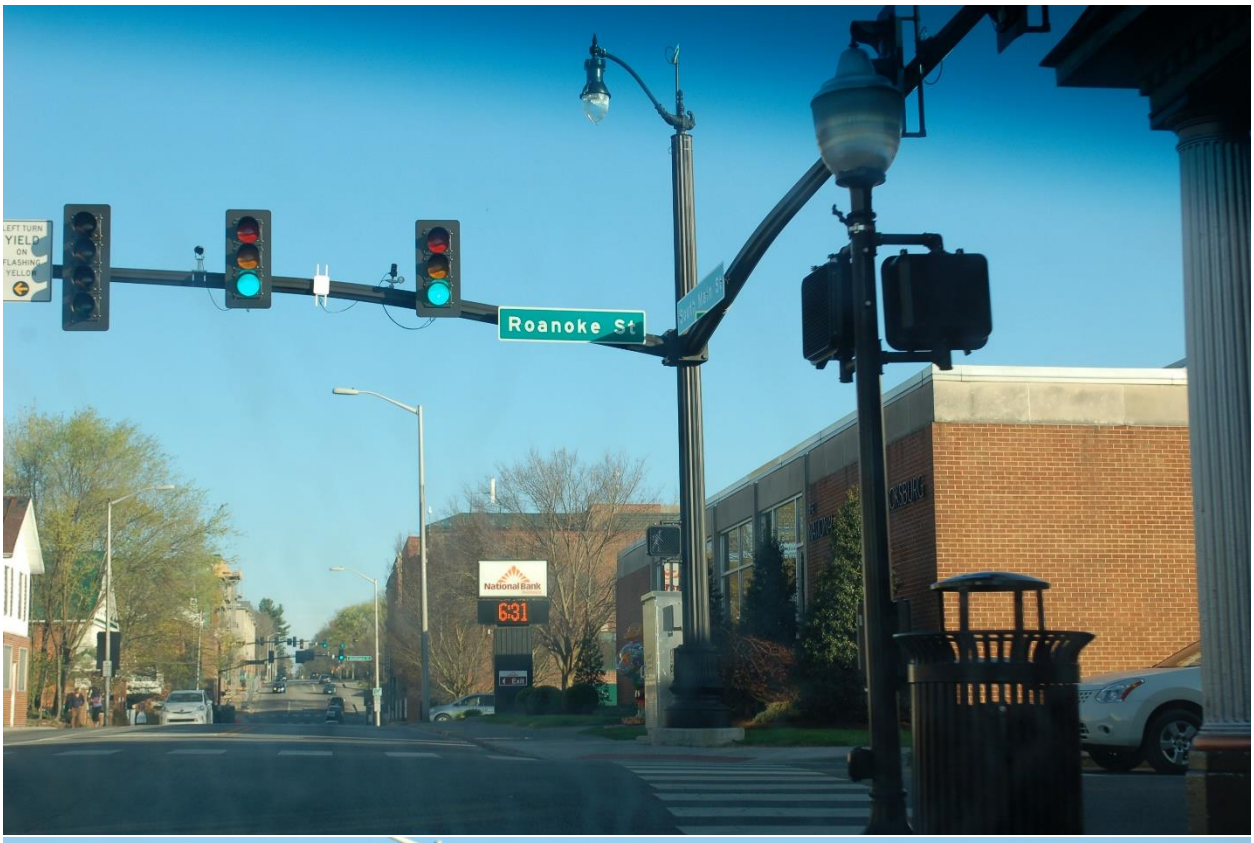

SPEED
LIMIT
25


TallOaksDr
NW
2900

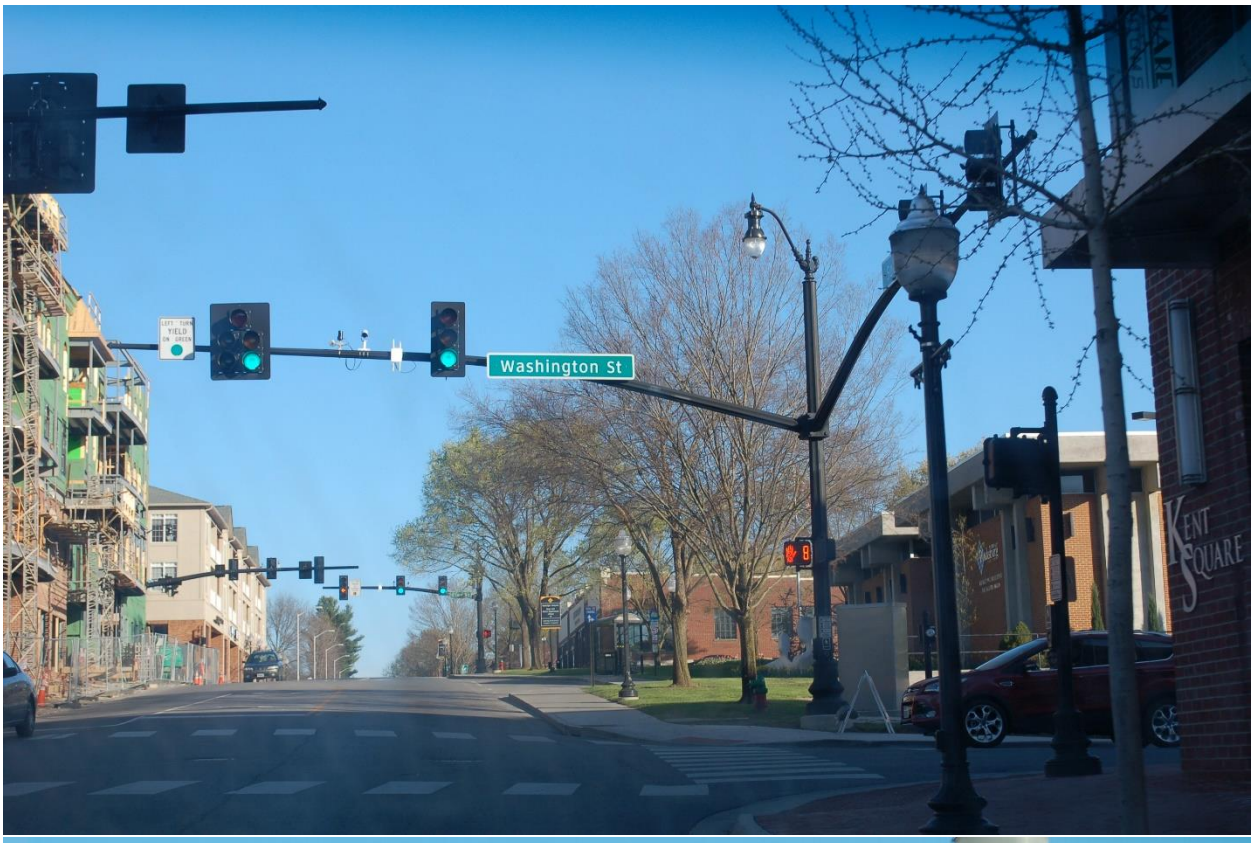

WELLS FARGO
College Ave
MOE'S


Farmers Market
Municipal Complex
Entering Blacksburg's
Original 16 Squares
P
953 • 2200
OPEN
Zone of the Day
$5


SPEED
LIMIT
25


Roanoke St
National Bank
6:31


National Bank
63°
Exit


Washington St
KENT SQUARE

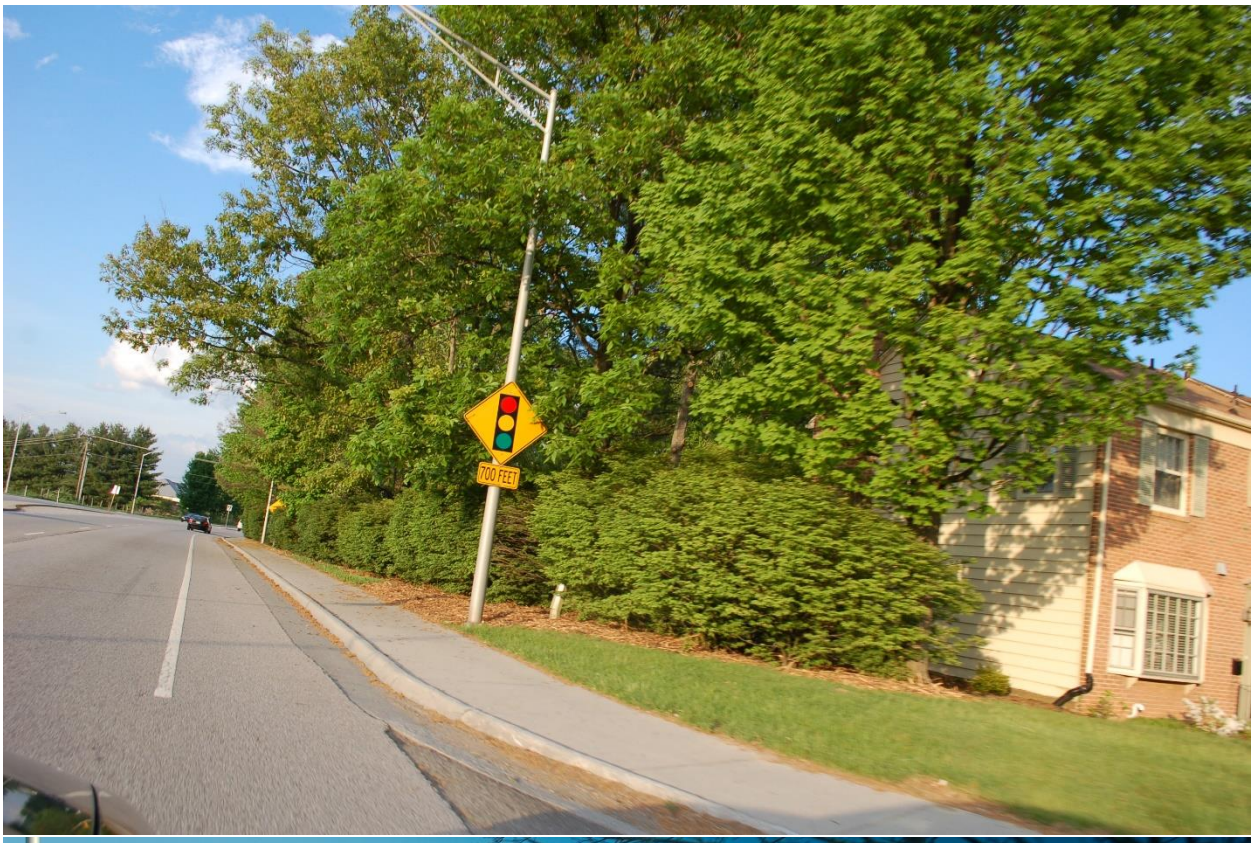

Huckleberry Trail
P

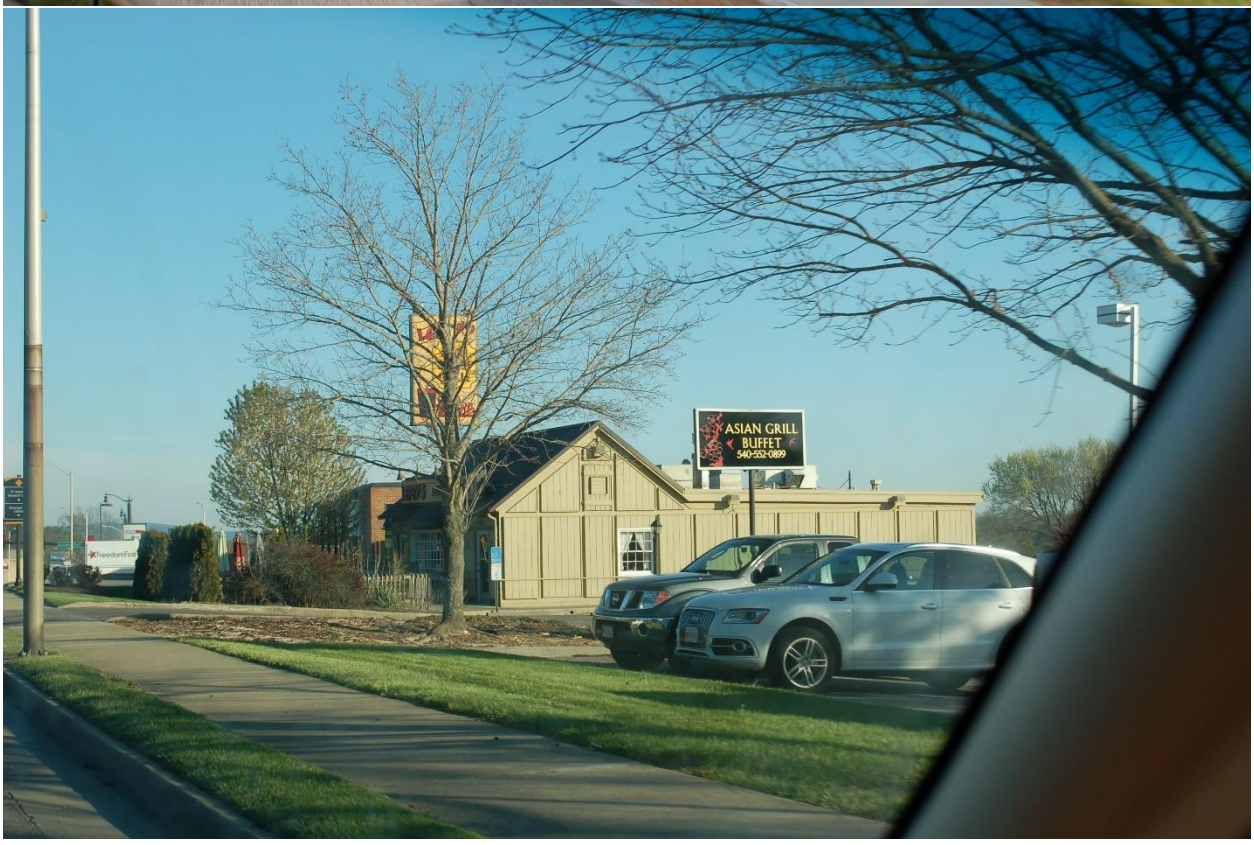

HANDICAPPED
PARKING
ONLY
PENALTY
2210

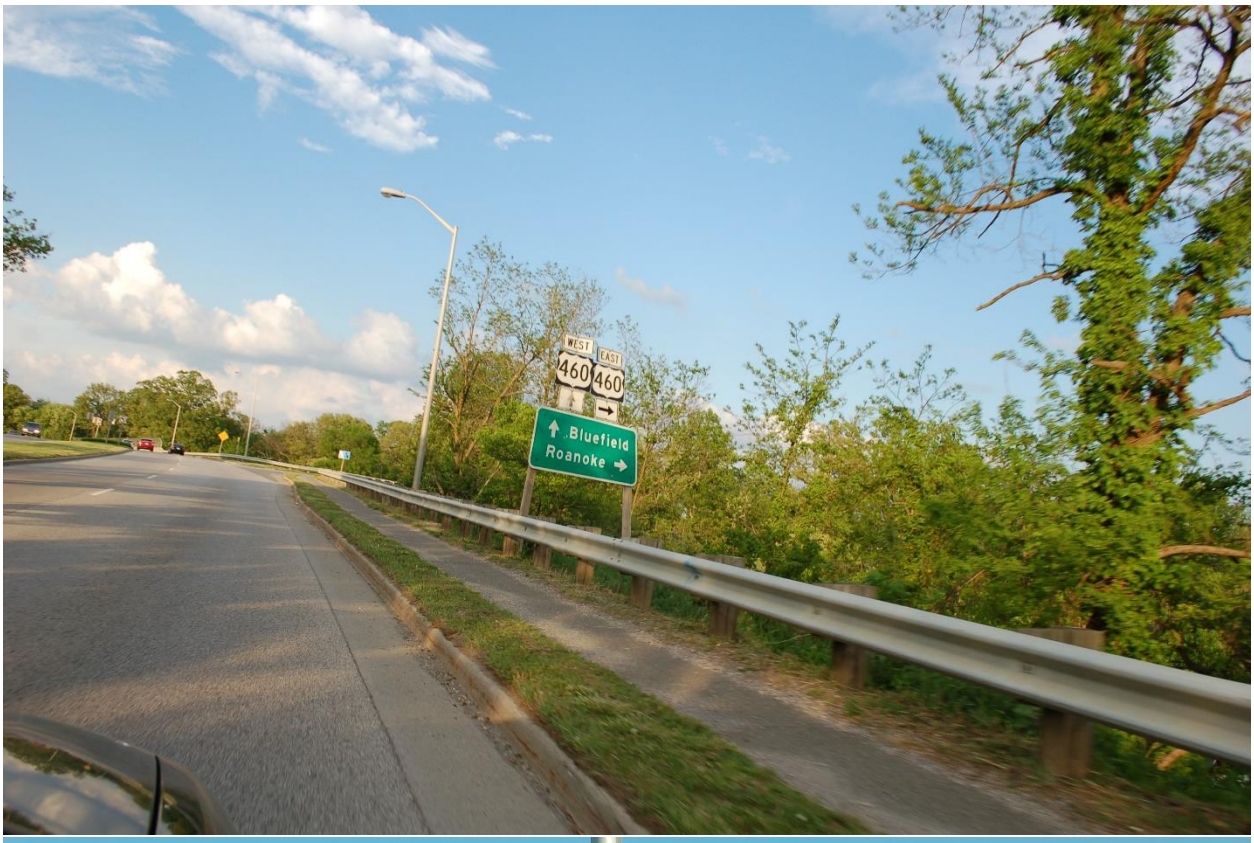

NO
PARKING


NO
PARKING
BETWEEN
SIGNS
SPEED
LIMIT
35


NO
TURN
ON
RED

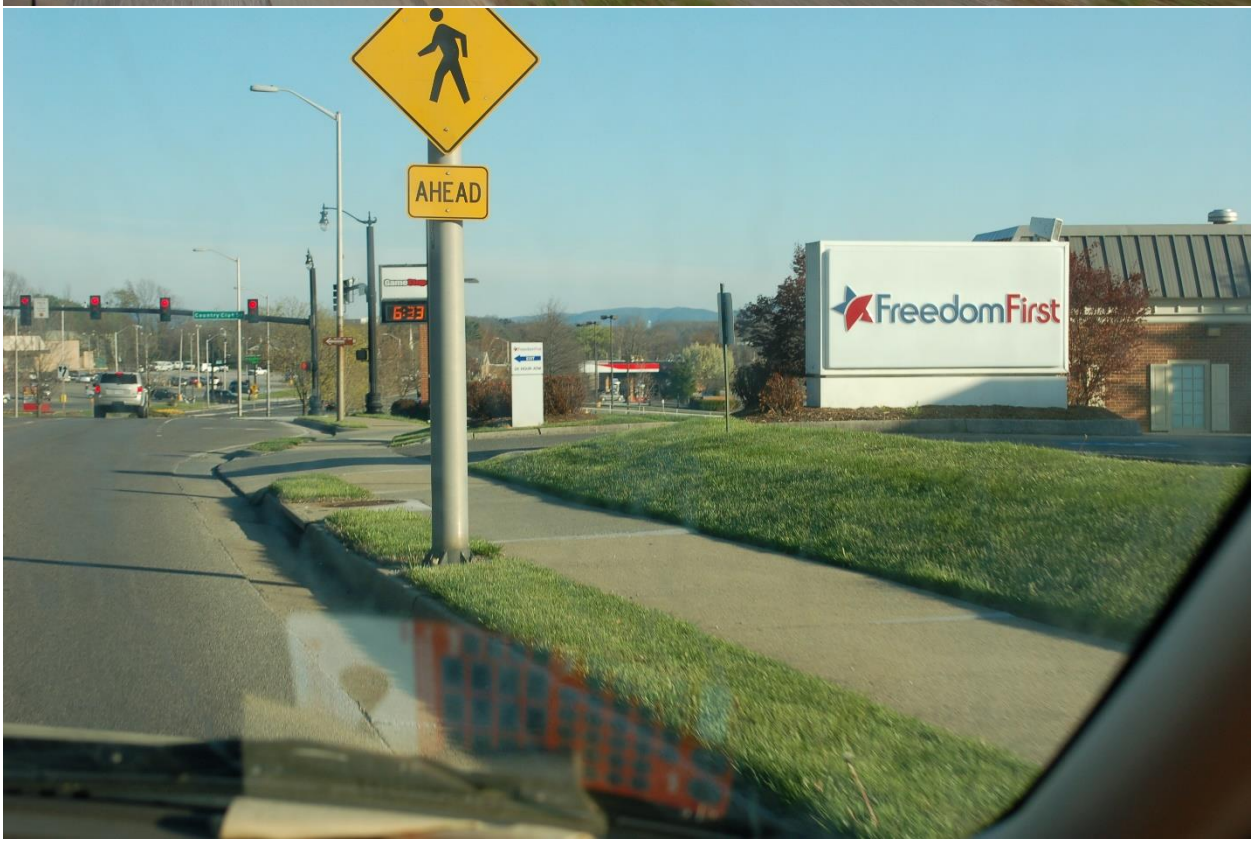

700 FEET


ASIAN GRILL
BUFFET
540-552-0899

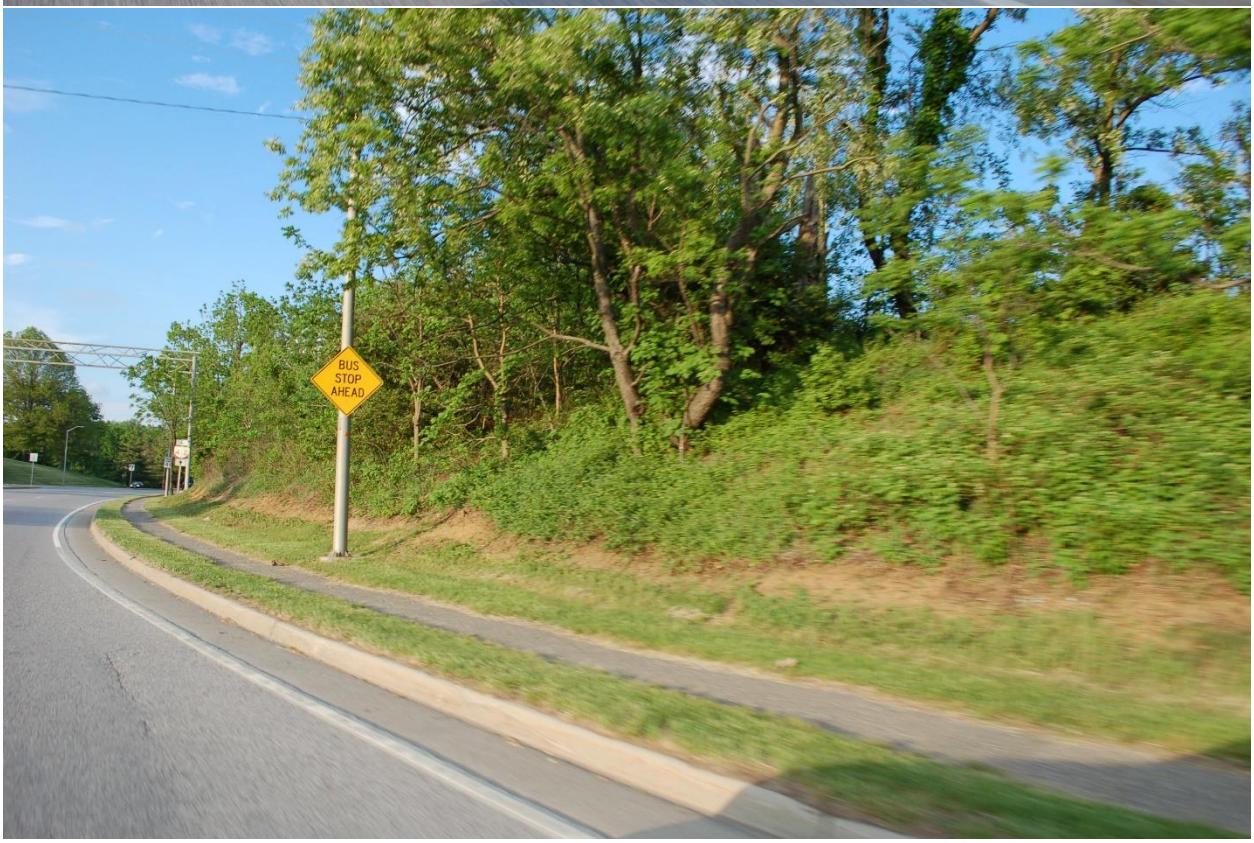

WEST
460
EAST
460
Bluefield
Roanoke


NO
PARKING
VT Corporate
Research Center
Municipal Golf
Course


ICE FREEZES
ON BRIDGE
BEFORE
PAVEMENT


AHEAD
FreedomFirst


Bluefield


BUS
STOP
AHEAD


Virginia Tech
Visitor Center
Conference Center
USE RIGHT LANE


Central Campus
Alumni Center
KEEP STRAIGHT


SPEED
LIMIT
35
SUBWAY


BUS
STOP
AHEAD

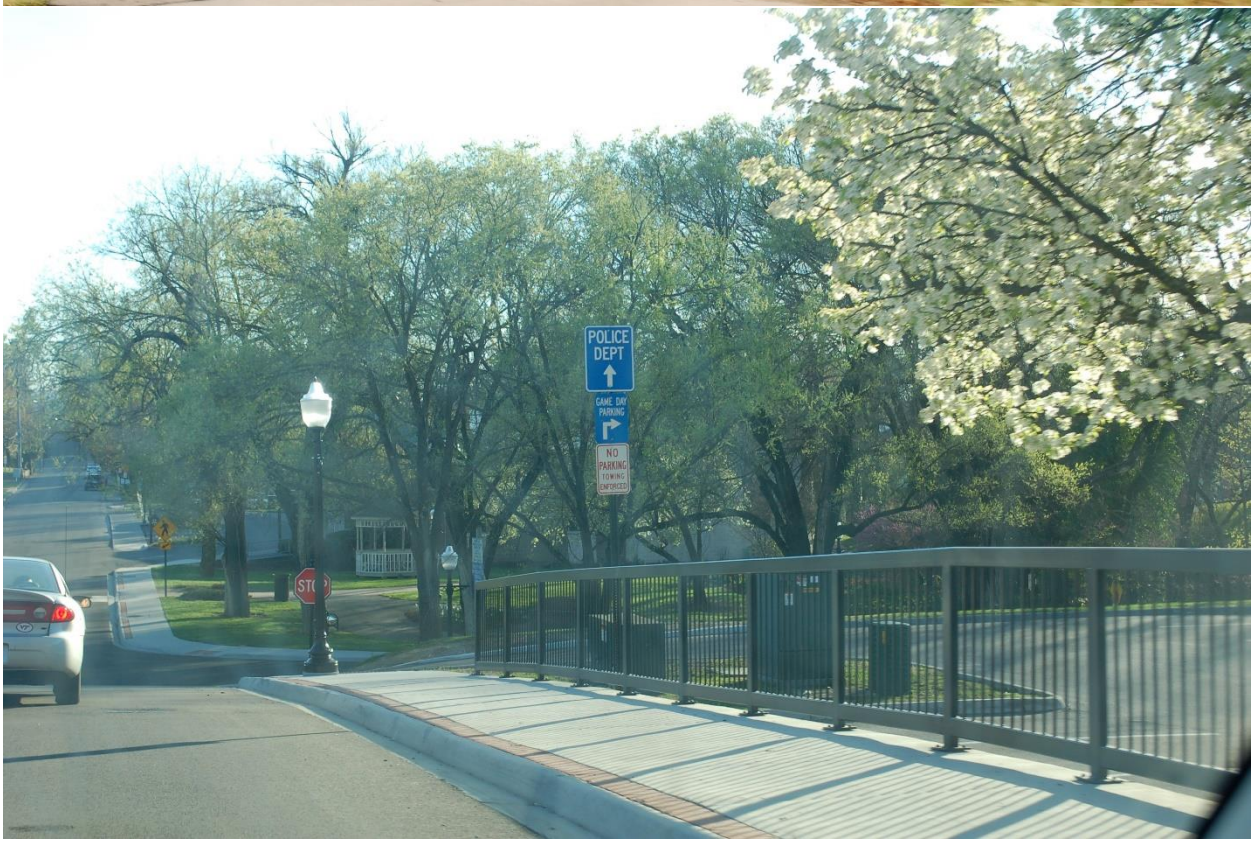

Central Campus &
Alumni Center
Visitor Center &
Inn at Virginia Tech


University City Blvd


CELL PHONE REPAIR
HERE
540-961-4400
Henebry's Jewelers
WE BUY GOLD


Welcome Parents!
NOW
LEASING!
theEDGE
apartment homes
NOW LEASING!
TURN ON
TOM'S CREEK
TO
LEASING
OFFICE
BRAND NEW!
• 2, 3 & 4 BEDROOM
APARTMENTS & TOWNHOMES
• INDIVIDUAL LEASES
(540) 552-EDGE (3343)
www.cmgleasing.com
theedge@cmgleasing.com
NATIONAL
RENT-A-FENCE
1-800-352-5675
KEEP OUT
STOP
CONSTRUCTION
PERSONNEL
ONLY


COOKOUT
FRESH BURGERS • BBQ • HOT DOGS • SHAKES
Breakfast


POLICE
DEPT
GAME DAY
PARKING
NO
PARKING
TOWING
ENFORCED


PARKING FOR
MUNICIPAL BUILDING
BUSINESS ONLY
Mon-Fri
7:30am-5:00pm
20 MINUTE LIMIT
TOWING ENFORCED
PAID PARKING
ON VIRGINIA TECH
FOOTBALL
GAME DAYS
TOWING
ENFORCED