VEHICLE TRACKING AND SPEED ESTIMATION FOR TRAFFIC SURVEILLANCE

KAIRIL FARIQ BIN CHAIROL MOHD FEROZ

This project report is submitted in partial fulfillment of the requirement for the award of the Degree of Master of Electrical Engineering

Faculty of Electrical and Electronic Engineering
Universiti Tun Hussein Onn Malaysia

JULY, 2014
ABSTRACT

Vehicle tracking is one of the critical applications of object detection and tracking. Traffic surveillance has become crucial in this day and age where the number of vehicles on the road has risen considerably. To preserve the safety of motorists, traffic law enforcement assign speed limits at different locations throughout the country. However, irresponsible motorists still exceed the speed limit since they know it is unlikely that they will get caught. In this paper, a system is developed which is capable of detecting moving vehicles in a video and display the vehicles speed as it goes. Should a vehicle exceed the allowed speed limit, it will be displayed in the video alongside the vehicle so that traffic law enforcers will be able to take necessary action based on the displayed speed. The system uses Matlab/Simulink as a simulation platform as it provides comprehensive tools for thresholding, filtering and blob analysis. Optical flow was the image processing technique used to determine the moving vehicles. A median filter was used to remove salt and pepper noise from the thresholded image. Combinations of several morphological operations were used to rectify whatever that is left. Blob analysis produces rectangles around the moving objects. The centroid of the rectangle is used to determine the location of each vehicle at a given frame. To make up for the absence of depth perception, the camera’s height and angle from the road is fixed so that the rate of which a vehicle approaches the camera can be determined. The results show that the system successfully detects vehicles and displays its speed, though there is a relatively small margin of error for the displayed speed. The displayed speed is set to only change once every couple of frames so that it would be easier to see.
ABSTRAK

TABLE OF CONTENTS

TITLE i
DECLARATION ii
DEDICATION iii
ACKNOWLEDGEMENT iv
ABSTRACT v
TABLE OF CONTENTS iv
LIST OF FIGURES x
LIST OF TABLES xii
LIST OF ABBREVIATIONS xiii

CHAPTER 1 INTRODUCTION

1.1 Introduction to Vehicle Tracking and Speed Estimation 1
1.2 Problem Statement 2
1.3 Aim and Objectives 2
1.5 Scope 2

CHAPTER 2 LITERATURE REVIEW

2.1 Case Study 4
2.1.1 Tracking an Unknown Moving Target from UAV
2.1.2 Real-time Tracking of Round Object.
2.1.3 Video Stabilization and Motion Detection using Matlab Video Processing Toolbox.
2.1.4 Comparison of Optical Flow Algorithms for Speed Determination of Moving Objects.
2.2 Noise filtering
2.3 Image segmentation
2.4 Closing (Morphology)

CHAPTER 3
METHODOLOGY

3.1 Flowchart of Overall Project
3.2 Software
3.3 Converting from RGB to intensity
3.4 Optical flow computation
3.5 Noise filteration
3.6 Object detection and tracking
3.7 Calculation of object’s velocity
3.8 Display speed of object in output video.

CHAPTER 4
RESULT AND DISCUSSION

4.1 Simulink model of overall project
4.2 Speed comparison
4.3 Shadow effect

CHAPTER 5
CONCLUSION
REFERENCES

5.1 Conclusion 38
5.2 Recommendation 38
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Lucas-Kanade optical flow</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Simulink Model of Real-Time Ball Tracking</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Motion Detection using Simulink</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Focus of expansion</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>Flowchart of Overall Project</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>Simulink model for RGB to intensity conversion</td>
<td>12</td>
</tr>
<tr>
<td>3.3</td>
<td>The range of intensity values</td>
<td>12</td>
</tr>
<tr>
<td>3.4</td>
<td>Output on the left displays the original video</td>
<td>13</td>
</tr>
<tr>
<td>3.5</td>
<td>Simulink model for Optical Flow (Lucas-Kanade)</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>computation</td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Configuration of the optical flow block</td>
<td>14</td>
</tr>
<tr>
<td>3.7</td>
<td>Motion vectors represented by lines</td>
<td>14</td>
</tr>
<tr>
<td>3.8</td>
<td>Simulink model for noise filtration</td>
<td>15</td>
</tr>
<tr>
<td>3.9</td>
<td>Image after Median filter</td>
<td>15</td>
</tr>
<tr>
<td>3.10</td>
<td>Simulink model for blob analysis</td>
<td>16</td>
</tr>
<tr>
<td>3.11</td>
<td>Blob analysis block statistics</td>
<td>16</td>
</tr>
<tr>
<td>3.12</td>
<td>Blob properties</td>
<td>17</td>
</tr>
<tr>
<td>3.13</td>
<td>Draw shapes block that uses the matrix of the bounding box</td>
<td>17</td>
</tr>
<tr>
<td>3.14</td>
<td>Output video with bounding boxes around tracked cars</td>
<td>18</td>
</tr>
<tr>
<td>3.15</td>
<td>Displacement between current frame and previous frame</td>
<td>19</td>
</tr>
<tr>
<td>3.16</td>
<td>Blob analysis block for determining centroid</td>
<td>20</td>
</tr>
<tr>
<td>3.17</td>
<td>Simulink model for current frame centroid and previous frame centroid</td>
<td>21</td>
</tr>
<tr>
<td>Figure 3.18</td>
<td>Speed formula represented in Simulink blocks</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3.19</td>
<td>Scope of detected object’s speed</td>
<td>23</td>
</tr>
<tr>
<td>Figure 3.20</td>
<td>Both frames have the same number of blobs</td>
<td>23</td>
</tr>
<tr>
<td>Figure 3.21</td>
<td>Check to make the output signal a variable</td>
<td>24</td>
</tr>
<tr>
<td>Figure 3.22</td>
<td>Simulink model for checking the dimensions</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.23</td>
<td>Scope of detected objects speed after dimension checking</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3.24</td>
<td>The testing location of the image processing system</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3.25</td>
<td>A simple mathematical representation of the testing location</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3.26</td>
<td>Rate of which an object accelerates as it moves towards the camera</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3.27</td>
<td>Simulink model for creating a bounding box for the speed</td>
<td>30</td>
</tr>
<tr>
<td>Figure 3.28</td>
<td>Simulink model for displaying speed bounding box</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3.29</td>
<td>Simulink model for noise filtration</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3.30</td>
<td>Simulink model for displaying speed in the video</td>
<td>32</td>
</tr>
<tr>
<td>Figure 3.31</td>
<td>Insert text block is used to display the speed</td>
<td>32</td>
</tr>
<tr>
<td>Figure 3.32</td>
<td>Speed displayed in the black bounding box</td>
<td>33</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Simulink model of completed project</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Output video of the completed system</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Free GPS app (Speed Box)</td>
<td>36</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Shadow effecting bounding box size</td>
<td>37</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3.1 Average angular error 8
Table 3.2 PNSR of various algorithms 8
Table 4.1 Detected speed of a car moving at 80km/h, and its percentage error 36
LIST OF ABBREVIATIONS

UAV  Unmanned Aerial Vehicle
SIFT  Scale-Invariant Feature Transform
SURF  Speeded Up Robust Features
3D  Three Dimensional
PNSR  Peak Signal to Noise Ratio
RGB  Red, Green, and Blue
2D  Two Dimensional
FPS  Frame Per Second
GPS  Global Positioning System
VHDL  VHSIC Hardware Description Language
HDL  Hardware Description Language
FPGA  Field-Programmable Gate Array
ASIC  Application Specific Integrated Circuit
IC  Integrated Circuit
CHAPTER I

INTRODUCTION

1.1 Introduction to Vehicle Tracking and Speed Estimation

In recent years, vehicle tracking has been applied in traffic surveillance with the intention of gaining traffic flow information, capturing traffic violations, and classifying vehicles. Vehicle tracking is an undertaking that can open up possibilities for countless other applications. For example, a traffic surveillance camera attached to a traffic light could detect and track vehicles and not be affected by pedestrians. However, we could also configure the image processing system to detect pedestrians and ignore vehicles. It’s this versatility that makes image processing a preferred alternative to other detecting methods.

That being said, one aim is to create a system that not only tracks vehicles but subsequently calculates its speed. Velocity has a very simple formula and applying it in image processing should not be that difficult. An object that is tracked will produce a middle point (centroid) which holds the value of the vehicle's midpoint coordinates. Using these coordinate we can determine the displacement, etc. Frame sampling rate, geometric and radiometric resolutions, and distortion amounts of the optical system of the camera affect the precision of the estimated speeds. There are several journals which recall the attempts to develop a speed estimation system. Each was different and unique in its own way. This means that vehicle tracking and speed estimation may have several ways to be determined.

The starting point of many works for traffic surveillance applications is based on the segmentation of the moving object, and for this purpose background subtraction methods are mostly used. For this purpose, each pixel of the successive frame images is subtracted, such that \( I(x, y, t) - I(x, y, t + \Delta t) \). The absolute value of this subtraction operation is used. However, since the aim of this project is to solely detect moving vehicles, Optical flow is
used since it specializes in tracking moving objects. In order to eliminate the object shadows, some other operations are often performed on the segmented images. In this paper, we choose to nullify the effects of shadows by running the system when there are no forecast shadows.

1.2 Problem Statement

Traffic surveillance is commonly used to detect and track moving vehicles. However, it does not compute the vehicle’s speed which is very important when it comes to road safety. The proposed project, if successful may aid traffic law enforcement to prevent drivers from exceeding the speed limits. Vehicle tracking is very versatile and can be reprogrammed to do additional tasks such as object counting, object distinguishing and more. However, low-tech speed sensors are still being used on highways and traffic lights. The proposed project also uses a more affordable and cheap method compared to high end vehicle tracking systems developed by surveillance companies.

1.3 Aim and Objectives

The development of vehicle tracking and speed estimation for traffic surveillance is the aim of this project. In order to achieve this aim, the objectives have been formulated as follows:

1. To develop a system to detect a moving vehicle.
2. To develop an algorithm that computes vehicle’s speed and display it on the output video.

1.4 Scope

1. The system is only tested during the day when there is no whether disruptions such as rain since it may be interpreted as noise by the system.
2. The Matlab/Simulink has been used as a simulation platform to provide easier analysis and alterations to the system.
3. Prererecorded video with 120x160 pixel resolution will be used to minimize the processing power required for analysis.
4. The camera used for the recording of traffic flow is at a fixed height and angle to overcome the depth perception limitation.
CHAPTER II

LITERATURE REVIEW

2.1 Case Study

2.1.1 Tracking an Unknown Moving Target from UAV

Target tracking is one of the most popular vision-based missions a UAV has to solve. Image processing algorithms for such mission can be categorized into two groups. The first group contains ones which operates using given model of the target. It enables direct estimation of relative position and attitude between UAV and target, such as SIFT (Scale-Invariant Feature Transform), SURF (Speeded Up Robust Features) and 3D-shape matching. The other group is composed of ones that run without any modeling such as Camshift, color-based morphology and optical flow. A UAV utilizes optical flow since it enables the detection of objects without prerequisites such as color or shape.
Figure 2.1: For a sequential image from a video of a moving car (top), vector field can be obtained by pyramid Lucas-Kanade optical flow (lower left) and the same information expressed using a color wheel (lower right).

The block diagram in Figure 2.1 shows the UAV utilizing Lucas-Kanada Algorithm to convert the image of the car into a vector field. The generated vector field provides the velocity information of each pixel. This information is displayed in colors by expressing the orientation and magnitude of the vector by varying hue and saturation, respectively.

2.1.2 Real-time Tracking of Round Object

This project uses the autothreshold block to convert an intensity image and convert it into a binary image using Otsu’s method. Thresholding is basically a filter that compares each pixel in an image with a value specified by the user. Should the input value be greater than the threshold value, the block outputs a 1; otherwise, it outputs 0. The resulting output would be a black and white binary image, where black are the pixels below the threshold, and white vice
versa. The ball tracking utilizes a Kalman filter which predicts the motion of an object using Bayesian estimation. Figure 2.2 shows the model of real-time ball tracking.

![Simulink Model of Real-Time Ball Tracking](image)

**Figure 2.2: Simulink Model of Real-Time Ball Tracking.**

### 2.1.3 Video Stabilization and Motion Detection using the Matlab Video Processing Toolbox

This project implements Matlab Video Processing Toolbox to overcome video turbulence and detect moving objects. The method used is similar to background subtraction, except that instead of a background without any objects, the current frame is compared to the previous frame. Frames which intensity has changed are indicating movement. Figure 2.3 shows the block diagram of the motion detection.
2.1.4 Comparison of Optical Flow Algorithms for Speed Determination of Moving Objects.

This paper compared two optical flow algorithms: Horn-Schunck and Lukas-Kanade. The optical flow is a velocity field in the image which transforms one image into the next image in a sequence.

Horn-Schunck is a global method that estimates everywhere. But Lucas-Kanade algorithm is a local window based method that cannot solve for optical flow everywhere. Lucas-Kanade is more robust to noise compared to Horn-Schunck. The Horn-Schunck method
is also very unstable in case of illumination artefacts in the image sequence. Table 1.1 shows the average angular error for both methods; Table 1.2 shows the PNSR of both algorithms.

**Table 1.1: Average angular error.**

<table>
<thead>
<tr>
<th>Angular error Method</th>
<th>Average angular error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucas &amp; Kanade</td>
<td>4.3</td>
</tr>
<tr>
<td>Horn &amp; Schunck</td>
<td>9.8</td>
</tr>
</tbody>
</table>

**Table 1.2: PNSR of various algorithms.**

<table>
<thead>
<tr>
<th>Method</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucas-Kanade algorithm</td>
<td>32.09</td>
</tr>
<tr>
<td>Horn-Schunck algorithm</td>
<td>30.71</td>
</tr>
</tbody>
</table>

2.2 Noise Filtering

In image processing, filters are applied to reduce noise and/or to prepare images for further processing such as segmentation. Some noise distributions are very annoying when are involved in intensity changes in video frames. They randomly and sparsely corrupt pixels. Therefore, there is a need for ways to implement smoothing techniques to remove different kinds of noise. Image filtering algorithms have several main objectives such as:
1) Suppression of noise
2) Preservation of edges
3) Removal of impulses

A class of filters that fulfil these requirements is the so called signal filter. Some commonly used signal filters are Gaussian Smoothing, Mean Filter, and Median Filter.

2.3 Image Segmentation

To detect an object, the image is normally segmented into blobs or regions using some common segmentation techniques such as background subtraction mean shift clustering and graph cuts. Segmented regions are then grouped together to represent an object based on some deterministic rules. In tracking algorithm the content of each frame is read and the background is estimated. The unwanted/interested objects are tagged by eliminating the background. Thresholding function is used to convert the grayscale image to binary so that the objects of interest can be highlighted by fixing a threshold limit.

2.4 Closing (Morphology)

Closing is defined simply as a dilation followed by an erosion using the same structuring element for both operations. It is an important operator of the field of mathematical morphology. Like its dual operator opening, it can be derived from the fundamental operations of erosion and dilation. Like those operators it is normally applied to binary images. Closing is similar in some ways to dilation in that it tends to enlarge the boundaries of bright regions in an image, but it is less destructive of the original boundary shape. As with other morphological operators, the exact operation is determined by a structuring element. The effect of the operator is to preserve background regions that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of background pixels.
CHAPTER III

METHODOLOGY

3.1 Flowchart of Overall Project

Figure 3.1: Flowchart of the overall project.
The proceeding flowchart shows the chronological order in which this project was completed. Each step is completed by utilizing blocks that are provided in the Image Processing Toolbox.

### 3.2 Computer Vision System Toolbox Software

Simulink is a graphical extension to MATLAB for modelling and simulation of systems. One of the main advantages of Simulink is the ability to model a nonlinear system, which a transfer function is unable to do. Another advantage of Simulink is the ability to take on initial conditions. When a transfer function is built, the initial conditions are assumed to be zero.

In Simulink, systems are drawn on screen as block diagrams. Many elements of block diagrams are available, such as transfer functions, summing junctions, etc., as well as virtual input and output devices such as function generators and oscilloscopes. Simulink is integrated with MATLAB and data can be easily transferred between the programs. Simulink is supported on Unix, Macintosh, and Windows environments; and is included in the student version of MATLAB for personal computers.

Simulink has a built in Computer Vision System Toolbox that supports a stream processing architecture through blocks. It allows the use of key stream/image processing techniques that are crucial for overcoming noise and differentiating an object from its background. Some of the tools provided in the Computer Vision System Toolbox are:

1) Analysis & Enhancement  
2) Conversions  
3) Filtering  
4) Geometric Transformation  
5) Morphological Operations  
6) Sinks  
7) Sources  
8) Statistics  
9) Text & Graphics  
10) Transforms
3.3 Converting video from RGB to intensity

Figure 3.2: Simulink model for RGB to intensity conversion.

The first step for this project is to convert the video we intend to simulate from RGB to intensity. RGB (red, green, blue) are the three colors that can be mixed to become any other colors in the color spectrum. In an RGB video, each pixel is represented by a combination of these colors. Though it provides a more accurate visual representation of the recorded object(s), having to detect 3 colors in every pixel is redundant. Hence, the simplest way is to convert the video from RGB to intensity. What converting the video to intensity does is represent each pixel in the video with a value ranging from 0 to 255. 0 being the color black; 255 being the color white. Any values in-between are shades of gray.

Figure 3.3: The range of intensity values from 0 (black) to 255 (white).
Figure 3.4: Output on the left displays the original video; Output on the right displays the intensity video (Grayscale).

This conversion is intended to simplify a large amount of the calculations in steps to come. Instead of having to place 3 separate thresholds for RGB (for image filtering), which uses more storage space, using intensity will only require the use of one threshold.

3.4 Optical flow computation

Figure 3.5: Simulink model for Optical Flow (Lucas-Kanade) computation.

To solve the optical flow constraint equation for \( u \) and \( v \), the Lucas-Kanade method divides the original image into smaller sections and assumes a constant velocity in each section. Then, it performs a weighted least-square fit of the optical flow constraint equation to a constant model for \( [u \ v]^T \) in each section, \( \Omega \), by minimizing the following equation:
Here, \( W \) is a window function that emphasizes the constraints at the center of each section. The solution to the minimization problem is given by the following equation:

\[
\sum_{x \in \Omega} W^2(x) u(x) + I_x \cdot v + I_t \cdot \hat{v}^2 \tag{3.1}
\]

When you choose the Lucas-Kanade method, \( I_t \) is computed using a difference filter or a derivative of a Gaussian filter.

\[
\begin{bmatrix}
\sum W^2_x I_x^2 & \sum W^2_x I_x I_y \\
\sum W^2_y I_x I_y & \sum W^2_y I_y^2
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix} = \begin{bmatrix}
\sum W^2_x I_x \\
\sum W^2_y I_y
\end{bmatrix}
\tag{3.2}
\]

Figure 3.6: Configuration of the optical flow block.
3.4 Noise filtration.

Filtering is commonly applied after an image or video has been thresholded into a binary image. Median filter is a nonlinear digital filtering technique, often used to remove noise. The reason Median filter is used so often is that it preserves edges while removing noise. The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the median of neighboring entries. It is particularly effective in removing speckle noise and salt and pepper noise (impulsive noise).

After the median filter, the signal goes through closed morphology, which is erosion and dilation back to back. Erosion is performed first to allow specks of noise that eluded the median filter to vanish. Then what remains of the signal is dilated.
3.6 Object detection and tracking

Once the input video has gone through all the stages of thresholding and filtering, it is put through the blob analysis block. Blob analysis has the functionality to produce many forms of statistics, which is crucial for detecting and tracking. For now, the bounding box option is checked. A bounding box is an M-by-4 matrix of [x y height width] bounding box coordinates, where M represents the number of blobs and [x y] represents the upper left corner of the bounding box. As the blob moves, the bounding box will follow.
Figure 3.11: Blob analysis block statistics.

Figure 3.12: Blob properties.
Figure 3.13 shows the configuration of the blob analysis block. Maximum number of blobs that can be detected is set to 80. Minimum blob area in pixels is set to 200, since setting it to a value smaller than that might make the block to mistake noise for an object. Maximum blob area is set to 3600. Finally, there is the option “Fill empty spaces in outputs”. It is imperative that this option be checked if there is to be any matrix multiplication to be done later on. What this option does is that it fixes the output of the blob analysis block to be completely filled by the number specified by the user. Hence, the bounding box matrix will always be 80-by-4. This allows for easier calculations, as seen further into the project.

![Function Block Parameters: Display bounding box for Detected Cars](image)

Figure 3.13: Draw shape block that uses the matrix of the bounding box to draw boxes around detected objects.
3.7 Calculation of object’s velocity.

Velocity is defined as the rate of change of position with respect to time.

\[ V = \frac{dx}{dt} \quad (3.3) \]

Where \( V \) is the velocity and \( x \) is the displacement vector. Displacement vector can be determined by calculating the distance between the objects current location and its previous location. Since the coordinates that are output from the blob analysis block are in 2-D \((x, y)\), the simplest way to determine the displacement of the two points is to use Pythagoras Theorem.
Displacement, $dx = \sqrt{(X - X_{-i})^2 + (Y - Y_{-i})^2}$ \hspace{1cm} (3.4)

Where $(X, Y)$ represents the centroid of the object in the current frame, and $(X_{-i}, Y_{-i})$ represents the centroid of the object in the previous frame. It’s important to note that when the displacement is obtained, the unit will be in pixels, not meter.

The time it takes for one frame to move to the next is dependent on the frame-per-second (FPS) of the video.

$$\text{Time for 1 frame, } dt = \frac{1}{FPS}$$ \hspace{1cm} (3.5)

These two basic equations will be the base for the next Simulink model. The variables that are needed to make it work are the centroid of the object in the current and previous frame. This information can be obtained by using the blob analysis block.
Figure 3.16: Blob analysis block for determining centroid.

This outputs an M-by-2 matrix where the row ‘M’ is the number of blobs detected (fixed to 80 in this case), and the 2 columns are X and Y coordinates of the blobs centroid. To obtain the previous frame’s centroid coordinates, a delay is added to the present centroid.

Figure 3.17: Simulink model for current frame centroid and previous frame centroid.
The video that is used to simulate the image processing system is 120x160 pixels in size and is 15 FPS. Should the length and width of the road covered by the camera be 30m and 10m respectively, several formulas can be derived:

\[
X \text{ distance per pixel} = \frac{10m}{120 \text{ pixel}} = 0.083 \text{m/pixel}
\]

\[
Y \text{ distance per pixel} = \frac{30m}{160 \text{ pixel}} = 0.1875 \text{m/pixel}
\]

Once the hypotenuse produces an output, it is divided by 1/15 which is the time for 1 frame. The result would be the velocity in unit m/s. To change the unit to km/h:

Converting m/s to km/h = \( \frac{m}{s} \times 60 \text{s} \times 60 \text{m} \times \frac{1}{1000} \)

\[
= \frac{m}{s} \times 3.6
\]

\[\text{Figure 3.18: Speed formula represented in Simulink blocks.}\]

Once the speed formula is completed, a scope is added to observe the moving object's speed.
Figure 3.19: Scope of detected object’s speed against time.

In Figure 3.2, there seem to be impossible speeds when a blob enters the camera's field of vision, and when a blob leaves the camera's field of vision.

(a)

Figure 3.20: (a) Both frames have the same number of blobs (b) both have different number of blobs.

The problem here is, whenever a new blob is detected, the current frame will have an additional row whereas the previous frame will not. Since the output of blob analysis is fixed to 80x2, there will be no matrix dimension error displayed. However, the system will continue inputting the data into equation 2, and the displacement of the blobs centroid will be wrong.

To solve this problem, there has to be a method to determine how many blobs are in the current frame and the previous frame. When the number of blobs is equal in both current and previous frame, the system proceeds with the calculation. However, when there is a change in
the number of blobs in either frames, current frame and previous frame values are made to be the same. This will make the speed be displayed as 0 for 1 frame.

![Function Block Parameters: Blob Analysis](image)

Figure 3.21: Check to make the output signal a variable.

As mentioned earlier, the centroid output is in a matrix form M-by-2 where M represents the number of blobs detected. By checking “Output blob statistics as a variable size signal”, the dimension of the centroid matrix will no longer be fixed at 80x2 (max number of blobs) but will change depending on the number of blobs detected. By perpetually checking the dimensions of this centroid output, it is possible to know when there is a change in the number of blobs in the previous and current frame.

To check the dimension of the centroids, connect a probe block that outputs the dimensions of the signal that it’s connected to. It will continuously check the dimension of the centroids. Then, apply a selector block to the dimension signal so that we only obtain the value for the row (M of centroid matrix). Once the current and previous centroid rows are known, both of them are subtracted with each other.
REFERENCES


A Feasibility Study”, International Journal on smart sensing and intelligent systems, VOL. 1, pp. no. 1.


