Virtual Lines Sensors for Moving Object and Vehicle Counters

Samuel Mahatmaputra Tedjojuwono *

Abstract— **Any method of counting objects and vehicles using computerized systems in a real live environment needs to have reliable results and flexibility of operation. The proposed work describes a method of counting moving objects and vehicles within a live or recorded video stream using multiple virtual line sensors that give flexibility to the users in analyzing any location for the number of vehicles within the given video, as existing traffic surveillance cameras may be sited in many positions/angles. The proposed research is an improved object counter. The results of this method have shown a promising accuracy of close to 100% detection in daylight and 91% at night.**

Index Terms— **video analytics, virtual sensors, object and vehicle counters, computer vision.**

I. Introduction

THE needs of traffic and pedestrian information systems are emerging. Automated precision methods for counting pedestrian, vehicles and other moving objects are needed that are simple enough to be implemented in real-life environments. Previous studies showed that the underlying needs that triggered such information systems are: to improve pedestrian volume modeling [1], to reduce labor costs associated with manual techniques and continuous data collection [2], simulations of pedestrian crossings and risk assessment [3][5], and parking management systems [4]. Other research described the use of traffic and pedestrian information for efficient road resource usage such as lighting [6]. The project made use of pedestrian behavior, and distance to certain objects/traffic to enhance safety and efficiency.

The approaches that were previously proposed to achieve such needs could be grouped into three categories: (1) Using manual labor (2) Using hardware support systems (3) Using video analytic software. Both the first and second categories will have some major drawbacks in a live traffic implementation. These drawbacks could be in the form of expensive labor costs to monitor traffic 24/7, human error, and are limited only to a certain size object counter functions. The hardware-supported system will involve a lot of sensors to be implemented, the need to safely modify the road infrastructure to install the hardware, and a rigid and lengthy deployment process. The objective of the proposed work is to be able to enhance the third category of using video surveillance system to extract traffic information.

The proposed software will use video sources, both live and recorded traffic videos as input and dynamically put virtual line(s) as sensors on the videos to count moving objects that pass or are captured by the sensors. The virtual sensors should be able to be placed at any position on the video and should be able to give results for each sensor accordingly. The proposed application can be used in conjunction with live traffic monitoring systems on both government and private institutions' systems. The contribution of the proposed systems will lie in two categories: (1) Counting the number of vehicles on a certain road, or counting the number of pedestrian or visitors, thus assisting in building a comprehensive traffic/visitor statistical data model, using adaptive and dynamically placed virtual line sensors. (2) The information provided by the proposed systems could also be used as a trigger to certain action(s) when a moving object is detected by the virtual sensors or if a certain threshold is exceeded. The actions could be to open road gates, and to trigger a frame-grabber function in capturing images of the pedestrians or moving vehicles to be further processed in sub-functions such as face recognition or license plate recognition systems.

II. Previous Work

Some earlier research on a specific pedestrian counting domain approached this through the scene-specific learning method [7]. The work distinguished the crowd-counting problem in two points: (1) line-of-interest counting (2) region-of-interest counting. Both used flow velocity field estimation with a combination of regression (tilt angle-learning), and with a ratio of 1:3 of training and testing data respectively. Additionally, another researcher used what they called the semi-supervised regression [8] method that minimizes the frame-labeling process, with the latest approach using privacy-preserved crowd-counting [9]. All of these methods claimed to have close to 99% accuracy; however, this comes with the effort of providing offline training data to the system.

More specifically in the vehicle-counting domain, a recent work [10] used adaptive bounding boxes combined with blob-size fitting to search for the boundary of the road and an adaptive background. However, special considerations need to be taken for minimum light or night-time conditions. It also was not tested on concave detected blobs. Thus, conditions like junctions, or if multiple roads are present in the video were not mentioned. With a strong similarity to the previous work, a virtual detector in the form of a rectangle can be used on videos of traffic flow [11]. In addition the average accuracy is 80% and the camera's view array should be placed rigidly in line with the road with no discussion of how to handle a diagonal to the video frame traffic scene. Moreover, object detection was used under specific layouts [12] and also installed on a DSP board [13]. However, both works have no discussion on the

*Manuscript received November 10, 2016; Samuel M T is with Bina Nusantara University, School of Information Systems, Jakarta, Indonesia (The Joseph Wibowo Center, Jl. Hang Lekir I No. 6, Senayan, Jakarta 12270) (e-mail: smahatmaputra@binus.edu).

applications' speed performance.

A very much closer approach to the proposed work used only a single virtual line [14], which could reach 85% accuracy in counting vehicles; although without further discussion on the time elapsing for the process. However, on a real live traffic flow video, more than one virtual line sensor is needed. This was implemented on the latest work [15] with what they called multiple time-spatial images. But the camera within this system should be placed rigidly inline with the road, and as sensors - virtual lines should be placed perpendicular to the road lanes' image array. It was also not tested for headlight problems in night traffic scenarios, and there was no further discussion on more complicated road scenarios such as cross junctions. Finally, both works did not test against other moving objects such as pedestrians, cyclists or motorcycles.

III. Proposed Methodology

The proposed traffic information system consists of four main modules: (1) Smoothing, (2) Detecting line placement from the user, (3) Registering all the points inside the lines, (4) Detecting movements of pixels' value in every registered point. These steps (figure 1) will be followed by the display of the detected moving objects or vehicles for each virtual line drawn by the user on the video screen. The output of the object counters could also be saved into text files or databases.

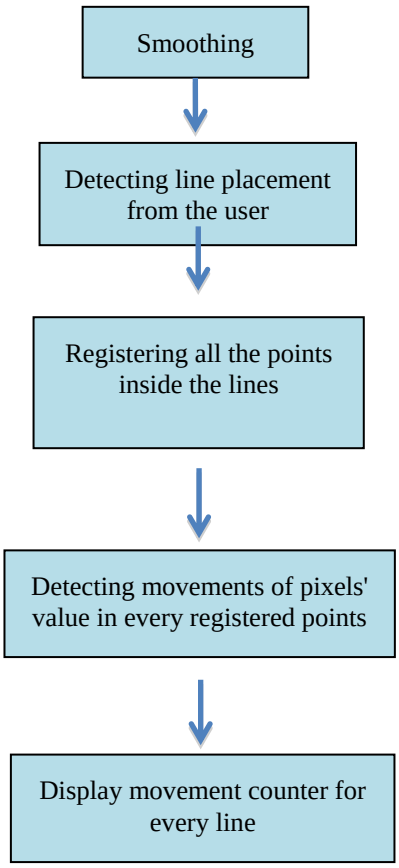

Fig. 1. The main modules of the proposed methodology start with the smoothing process, detecting line placement, registering points within a line vector, detecting pixel value delta on each line and displaying the results.

A. Smoothing or filtering

Smoothing (also called 'blurring') is a process of reducing the extreme frequencies in an image. There are three options in smoothing/filtering: mean, median and Gaussian smoothing [16]. This work proposes to use mean smoothing. Mean filtering works by averaging (mean) the pixel value of the current and the neighboring points and replacing the current pixel value with the mean value.

$$I'(i,j) = 1/M \sum_{k,l \epsilon N} I(k,l) \quad (1)$$

One way to implement it is by applying a matrix operator (ideally 3x3 matrix) to take the mean value of current and neighboring points.

$$Mean(i,j) = 1/9\varepsilon \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (2)$$

As an option to the above proposed approach, median filtering can be used instead. Median filtering basically has a similar matrix operator with mean filtering. However, it takes the middle value (after sorting) as the current pixel point instead of averaging the pixel's values.

B. Detecting line placement from the user

The aim of this work is to give the user the ability to place or draw the lines on any location on the video to count the number of objects passing the lines. In achieving this flexibility of virtual line sensor placement on a video screen, the application should be able to record the point where the user who triggered the event clicked for the first time, and continuously wait until the left button is released to record the second point's coordinates.

The algorithms of recording the line's two point coordinates could be described as follows:

```
// Output: Line(p1,p2)
// Detect line placement by the user on the screen
// and return the starting point and end point of the mouse
// movement.

Function Line detect_line_placement ()
{
   //Set listener for mouse events
   call videoScreenListener_MouseDown(MouseEvent e)
   call videoScreenListener_MouseMove(MouseEvent e)
   call videoScreenListener_MouseUp(MouseEvent e)
   Return Line(p1,p2)
}
```

Fig. 2. Function to detect line placement by the user on the video screen. The algorithms takes as input two parameters of point p1(x,y), and p2(x',y'). The function will then encapsulate and return the two points into a new data structure of Line(p1,p2)..

The detect line placement function shown in Figure 2 is returning a line comprising two points *p1* and *p2*. In this algorithm, the point p1 is the Euclidean coordinate of two integers (x,y) as the starting coordinate of the line, followed by the *p2* coordinate in a similar form of two integral

numbers (x,y) that mark the end point of the particular line being drawn by the user. This function needs to be called or executed every time the user triggers a mouse click. Details of the listeners are depicted in Figure 3.

```
// 1st listener
// Input mouse event.
// create new line at current mouse
// location as starting point.
// And set mouse down flag to true.

Function
videoScreenListener_MouseDown(MouseEvent e)
{
  line = new Line();
  line.point1 = e.Location;
  guidingLine = new Line();
  guidingLine.point1 = e.Location;
  mouseDown = true;
}

// 2nd listener
// Input mouse event.
// Validate mouse down flag and set a
// temporary line position.
// Will set down and move/ drag flat to true.
Function
videoScreenListener_MouseMove(MouseEvent e)
{
  if (mouseDown)
  {
    guidingLine.point2 = e.Location;
    mouseDownMove = true;
  }
}

// 3rd listener
// Input mouse event.
// Set the point for end of line.
// Generate members (points) of the newly
// created line.
// Register the line to a list of active
// line.
// Reset mouse down and drag to false.
Function
videoScreenListener_MouseUp(MouseEvent e)
{
  line.point2 = e.Location;
  line.generateMembers();
  lineList.Add(line);
  mouseDown = false;
  mouseDownMove = false;
}
```

Fig. 3. The first listener registers the mouse position on the screen upon the click as the starting point of a line about to be drawn. The second listener draws the temporary line as it dragged on the screen. The third listener sets the end point of the line, generates all the points within the line vector and adds the newly-created line into a list of active virtual lines.

The proposed method fosters flexibility in placing virtual lines to detect movement in any position within the video screen. To obtain this flexibility, three listener functions were implemented to capture mouse click-and-drag movements while drawing the line. The first function of the mouse down listener (left click) registers the mouse position on the screen upon the click as the starting point of a line about to be drawn. It starts by creating a new line *L* and assigns the starting point of *L.point1* taken from the user's mouse click action. To improve user experience a guiding line *gL* was created along with its starting point *gL.point1*.

The second listener draws the temporary line *gL* as the mouse pointer is dragged on the screen. While this action is being done, the end point of *gL* will be recorded at *gL.point2* and continuously rendered on the screen until the left mouse button is unclicked by the user.

The last listener sets the end point of the line *L*, and generates all the points within the line vector *L.vector[]* and adds the newly created line into a list of active virtual lines $L_{i..j}$. Each element of $L_{i..j}$ has to maintain its own points vector generated using the Bresenham[17] method of finding integral point coordinates within a line *L*. This method becomes important in order to adapt the natural coordinates of a point within a Euclidean 2D plane - which are represented by decimal points – into the computer screen point *(x,y)* coordinates which have to be integer numbers. Details on the process of generating all the points within a line vector will be described in the following section.

C. Registering all the points inside the lines

The image plane in a computer screen system is a two dimensional Euclidean plane where each point coordinate (x,y) within the plane should be represented using non fractional numbers. To represent an infinite number of points inside a line to this finite coordinate, a method to translate the real coordinate to integral number coordinates is needed. Thus, to register all the points inside each line and to well represent the line on the computer screen's coordinate system, a Bresenham's line drawing method was used [17].

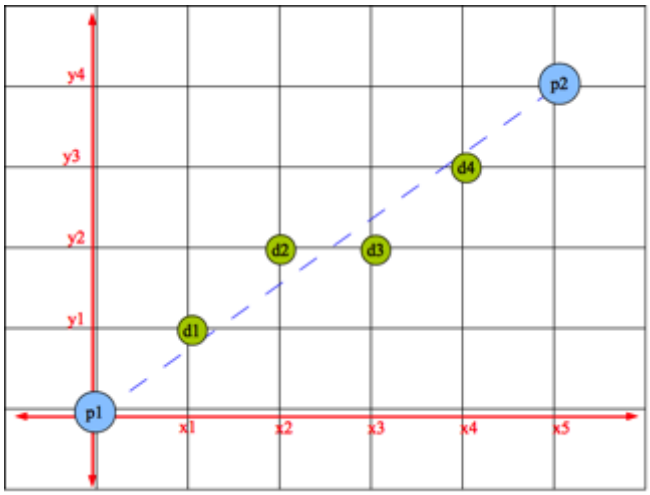

Fig. 4. Registering all the points (d1...d4) inside the lines *L* of *p1, p2*. An example from the first octant where the slope *m* of a line *L* that is represented using two points of *p1* and *p2* suffice the condition: $0 \leq m \leq 1$.

The method divides plotter movements into eight octants that can be used to determine every point inside a line that is expressed by two points *p1* and *p2*. The plotter movements are adapted to the proposed work as it incorporates only integral numbers within every point of next movement. Taking an example from the first octant where the slope *m* of a line *L* that is represented using two points of *p1* and *p2* suffices the following condition: $0 \leq m \leq 1$, every integer inside the points of *d1* until *d4* could be well calculated. This process is depicted in Figure 4.

The step by step choosing of every point inside the line of *p1* and *p2* are depicted in Figure 5 and Figure 6. In the first octant, to choose the next closest integer for both x and y-axis, a point could progress into two options of (x+1, y+1) or (x+1, y).

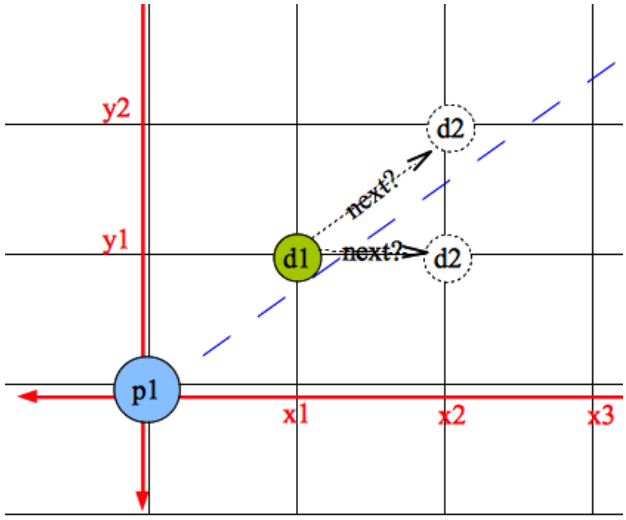

Fig. 5. The steps of choosing every point inside the line of p1 and p2. After determining d1 the algorithm determines which of the next steps will be a valid move for d2.

In choosing the options given by (x+1, y+1) or (x+, y), the error value of Ɛ is calculated using m as the slope. If the value of error value of Ɛ1 < Ɛ2 then (x+1, y) is taken for the next valid point. However, if Ɛ1 > Ɛ2 then (x+1, y+1) is taken instead. This comparison of error value Ɛ is done by comparing Ɛ1 with 0.5 of the epsilon.

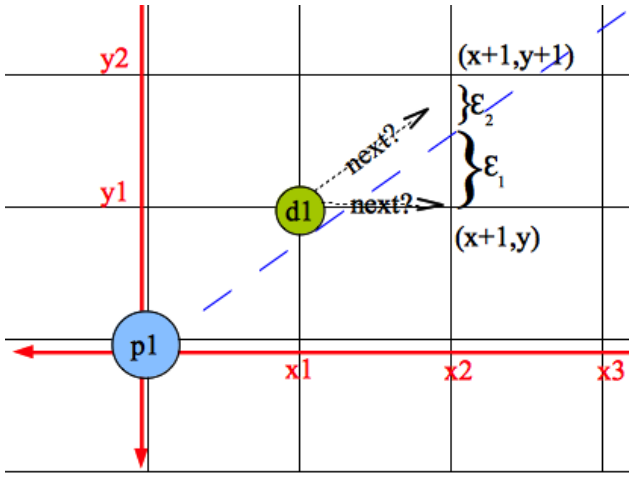

Fig. 6. The steps of choosing of every point inside the line of p1 and p2. If the value of error Ɛ1 < Ɛ2 then (x+1, y) is taken for the next valid point. However, if Ɛ1 > Ɛ2 then (x+1, y+1) is taken instead.

Within Figure 6, the next valid point that is closest to the real number of the line is (x+1,y+1) of d1 point as the error value Ɛ1 is larger than 0.5 of the computer screen epsilon. The work implementing the *Bresenham's* algorithms using the following condition is shown in Table 1.

TABLE I
THE EIGHT OCTANTS OF BRESENHAM'S ALGORITHMS

Octant	*Condition 1*	*Condition 2*
1	*x1<x2*	*0 ≤m≤1*
2	*x1<x2*	*0 >m≥-1*
3	*y2<y1*	*-1 >m > -∞*
4	*y2<y1*	*1 <m < ∞*
5	*x2<x1*	*0<m≤1*
6	*x2<x1*	*0 ≥m≥-1*
7	*y1<y2*	*-1 >m > -∞*
8	*y1<y2*	*1 <m < ∞*

D. Detecting movements in every registered points

The last step of the proposed system is to detect any object movement coming across every line. As all the points' coordinates inside every line array are already acquired by previous steps, this step is to detect any pixel value differences on each point in all drawn lines by comparing this value with all upcoming image frames at the same coordinates.

Establishing video stream S from input
(Live IP cam, recorded video, and live local cam)
Get latest frame F_i from stream S.
Construct back ground G_i from $F_{i..j}$
Detect moving object position by $pixel_\Delta$
Detect every line position $L_{0..N}$ on the screen.
Count $pixel_\Delta$ for every point p_k in $L_{0..N}$
Propose action trigger if $\Sigma\, pixel_{\Delta i..j} > T$

Fig. 7. Algorithms for detecting the movement of counting pixel difference pixelΔ. A certain threshold *T* for describing the significant level of movement is set. For every line *L*, if the $\Sigma\ pixel\Delta_{i..j} > T$ then there is significant movement on the line *L*. Action trigger will be executed.

To summarize our steps, Figure 7 describes the complete algorithms in detecting the movement before the counting of pixel difference pixelΔ. The algorithms started with the command of establishing video stream *S* from input (Live IP camera, recorded video, or live local camera). This video stream *S* will produce a series of still image frames called F_i. From a certain number of frames $F_{i..j}$ the system will construct the background G_i by averaging the pixel values of each point from the set of frames. Every next F_i pixels' values will compare G_i pixels' values to find out the difference that could be translated as a movement of object on top of G_i background. This difference of pixels' values *pixelΔ* is recorded inside a list for every line *L*.

The recording of *pixelΔ* for every line *L* is needed to count the magnitude of movement captured by each virtual line sensor. A certain threshold *T* for describing the significant level of movement is set as an input parameter. For every line *L*, if the $\Sigma\ pixel\Delta_{i..j} > T$ then it can be concluded that there is significant movement on the line *L*. This movement conclusion could be followed up by any kind of trigger to other functions such as counting number of movement – thus counting the number of objects crossing the line L. Some other follow-up functions – outside of the scope of this proposed work – such as triggering frame grabber function for automatic license plate recognition (ALPR) systems, could also be considered.

The pseudo code of the proposed algorithms can also be found in Figure 8. The pseudo code explains the process of detecting significant movements inside a line. This level of difference is marked by the identifier granularity. It is the threshold of how many pixels difference that will account for an object. After counting the number of differences inside the line, the system will include the findings as a movement that should be recorded and displayed on the screen.

```
Establish stream of frames
For every new frame {
     clone bitmap of frame
     Record all pixel as background

     draw user-defined line
     for every line {
            draw line ID
     }

     if (motion detect > granularity){
            For every point in a line {
                  record pixel value
            }

            Compare pixel value
            count Valid object
     }

     Reload background after T
}
```

Fig. 8. Pseudo code of the process of detecting significant movement inside a line. Movement conclusion above threshold *T* could be followed up by any kind of trigger such as counting the number of objects crossing the line. Followed by a reload of background construction function.

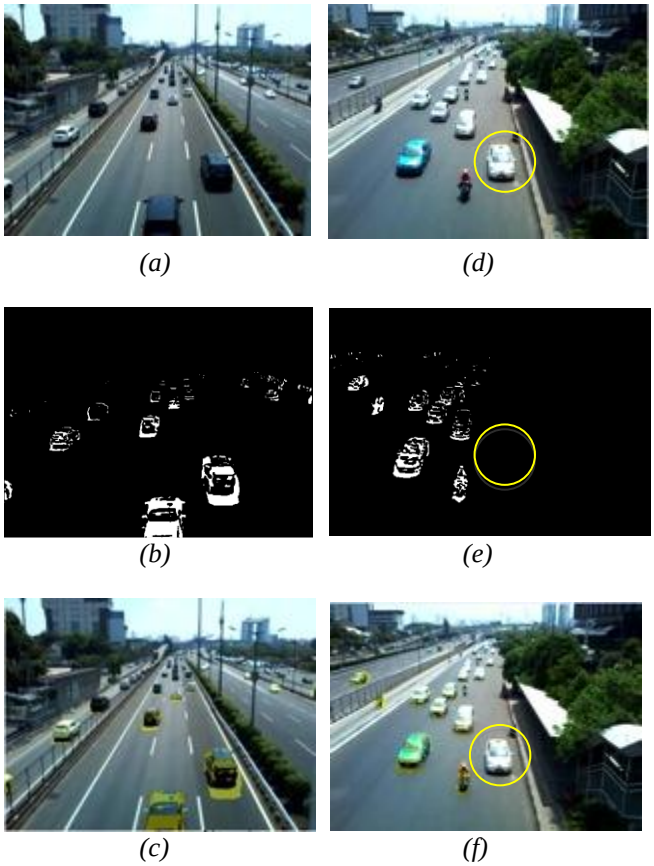

Fig. 9. Original video frames (image a, d). Result of background removal and foreground movement detection (image b, e) based on Σ $pixel\Delta_{i..j}$. Object movement detected overlaid on the original frames (image c, f).

The images resulting in the process of background removal are depicted in Figure 9 (a to f). These are two sets of images from two distinct videos a to c, and d to e. The original frame samples could be seen at pictures a and d, with the numbers of frame averages ranging from 14 to 20 fps.

Pictures b and e display the results of foreground movement detection based on Σ $pixel\Delta_{i..j}$. These are the frames that are being processed by virtual lines to check movement (marked with white area). The proposed work takes a sample of the background that has undergone a Gaussian smoothing, to minimize noise. The background construction will be recalled after every 10 milliseconds (T=0.01 second) to update background changes that are mostly caused by slight movements of the camera position, and the environment's shadows. The proposed work considered another novel work of *dynamic object detection, tracking and counting* [18], which implemented its background elimination process using the original image. The work implemented median filtering later at the post-processing stage in order to minimize the image's noise. This has resulted in what was called *non-zero pixel values* on the background removal process. Even though these are not the moving objects intended to be captured by the system - zero values in RGB color schema representing black, which is the removed background value. The proposed work would like to improve this method by implementing noise removal during a pre-processed stage in order to significantly improve the accuracy of the background construction process.

The system considered another original approach in the process of background construction that was proposed to be named as *double background filtering* [19]. It divided its process into four stages by firstly buffering the first five frames, accumulating optical flow information, eliminating overlapping optical flow, and lastly updating the background. This approach could significantly improve the process of minimizing the size of recorded videos by choosing only to record the movement's *delta* of each frame. However, because the nature of live traffic information promotes fast performance and accuracy, this method was not adopted due to its computationally expensive method that could potentially reduce the performance of a live object detection system.

In picture filtering, the proposed work considered the use of advance filtering technique proposed as two-stage PCA (Principal Component Analysis)-based filtering evaluated by a recent work [20]. The method takes as input noise images, applies pixel grouping, PCA filtering, and inverse PCA. It combines the two-stage PCA with non-local algorithms in order to produce noiseless reconstructed images. It has a significant positive result in reconstructing new clear images from heavily noised images. This is an extensive approach to reconstruct images with heavy computational process.

Furthermore, as a result of a simple Gaussian smoothing implemented by the system, in the set of images on Figure 9 *d* to *f* we could examine that noise comes from slight changes of the environment – or camera position - are minimum in comparison to the previous work [18]. All the white blobs (detected movements) are in fact the moving vehicles, which are valid input to the virtual line sensors later on. A result of a continuous background construction process could as well be observed from this set. A white taxi was stationed for a short while on the roadside in image *d* (circled) is immediately considered as the background by the background construction algorithm. This resulted in no white blob representation for the stationary taxi in image *e*, thus the area marked with black (0 values in RGB) was not highlighted in image *f* as well. Both images *c* and *f* are experimental results of overlaying the detected movements on top of the original video frames. All moving vehicles

could be highlighted successfully in the pictures, while continuously regenerating the most current background.

IV. The User Interface

One of the aims of this work is to provide ease of use for the user of the proposed system, such as flexibility in operating the surveillance application. Users are expected to easily use any video input and live video through IP camera – or CCTV, and place as many virtual lines as necessary at any position on the video screen using a mouse pointer.

Therefore, the expected output will be a computer application that serves as a traffic information system with an implementation on moving objects/vehicle counter. The application is designed to count both pedestrians and all types of vehicles. It also includes a graphical user interface

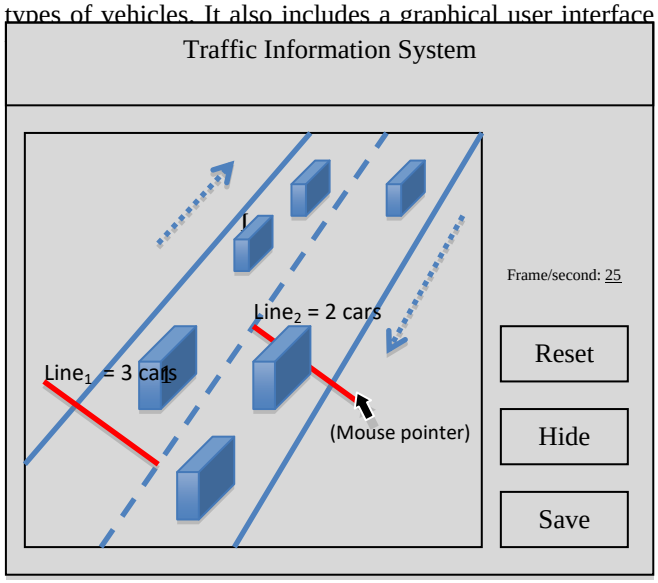

Fig. 10. The design of a graphical user interface to put virtual line sensors on the video screen. Each line will work independently as a sensor and will provide different counting and triggers. The virtual lines will be created using a user's mouse click and drag.

There are no restrictions on the position and the amount of these virtual lines placed on video screen, with each line working independently as an individual sensor and providing different counting result and trigger. The virtual lines are placed by the user with a simple mouse click and drag (depicted by L1…L3 in Figure 10). These lines can be set, reset, and hidden/shown using the menu. The application is designed to be a multi-window application. Thus, several video screens can be shown at one time and each of them work independently, having a distinct set of virtual lines and triggers. The result of vehicle/object counting could be saved in the format of a text file or connected to a database server. The application is also equipped with standard windows viewing facilities such as new windows, cascading, and tile views.

The result of implementing the system's user interface can be seen from Figure 11. This implementation of virtual line sensors has given every line the ability to count object movement independently. The lines could be placed anywhere in the video screen (using click and drag of the computer's mouse events), at any length, and without limitation of lines that could be drawn on the video screen.

Furthermore, to be able to monitor the performance (speed) of the systems, the information of the video's frame per second rate will be displayed as well. This information could serve as the indicator of how efficient the algorithm is working behind the frame rate and the speed of video being transferred from a remote camera – if one will be used in the future.

Fig. 11. Working application of moving object counters implementation on traffic flow. The shading overlaying every moving object (cars) on the screen is the marking created by the algorithms to show significant difference of the object pixels values and the background values. If the value of this pixel differences is larger than a certain value of granularity (threshold *T*) then the count for that line is increased (marked with message: "Count: [numbers]").

This could give users of the system more flexibility in placing the sensors as a mean of counters - that is mainly one of the goals of the proposed work. The shading overlaying every moving object (cars) on the screen is the marking created by the algorithms to show significant differences of the object pixels values and the background values in RGB (red/green/blue) channels. On every point within the virtual sensors line-1...line-n, the difference of pixel value (highlighted) will be counted. If the value of this count is larger than a certain value of granularity (threshold T) then the count for that line is increased and is marked with the message: "Count: [numbers]" that accompanies each virtual line. Users could direct this result to a database or simple text file as a result. This could, in a more detailed and flexible manner collect data from every position and corner of the video, and place much less restriction on the placement of CCTV camera that monitor the roads and parking areas. The camera result can be taken from any camera angle, as long as it clearly shows the moving objects and is not significantly blocked or shadowed.

V. Experimental Results

The system was tested in real traffic flow environments, at both daylight and night-time. Some of these traffic environments were well-managed using road lanes. However, some are not. In both cases the system could perform very well. A calibration is needed for conditions with minimum light such as during the night. As the image of the object is not obvious to the camera during night-time, the object movement is detected by using the headlight of cars or motorbikes. However, these vehicles also have rear lights that could mistakenly be marked as another object/vehicle. Thus, to avoid double counting, the granularity (or threshold *T*) parameter needs to be adjusted

such that the system will only counts a larger/wider array of light such as head lights of the vehicles, and neglect their rear lights. The experiment was carried out using the following system specification: 2.4 GHz Intel Core i5 with memory of 4GB 1333 MHz DDR3, along with a 2MP camera to record the video.

Fig. 12. Experimental result's screen capture of the object counting processes in daylight – no lane. The system could also record movement of motorcycles, cyclists and pedestrians that passed.

Fig. 13. Experimental result's screen capture of the object counting processes in day light – with lanes. Accuracy above 99%. The placement of virtual sensors (lines and their counters) could be freely placed on every position of the video.

Figure 12 until 13 depict the result's screen capture of the object counting processes in daylight. The condition of a no lane road can be found from Figure 12. The system could perform very well. However, there was double counting if an object or vehicle crossed two line sensors by moving sideways. The system could also record movement of motorcycles, cyclists and pedestrians that passed. Moreover, on a road with lanes the result was perfectly carried out. This can be seen from Figure 13 where the placement of virtual sensors (marked by lines and their counters) could be freely placed on every position of the video. Having said that, if the object's image is too small, thus it is smaller than the intended threshold, then the system will neglect its movement. The experimental result was carried out with the following parameters: granularity 0.02f.

The result of this work is that the camera angle could be placed anywhere, as long as it is looking toward the monitored area; whereas previously it had to conform to certain positioning conditions. This in contrast is an improvement to some of the earlier work [21][22] where the camera had to be positioned on top of the monitored area and was looking down perpendicularly to the objects.

The proposed research uses many lines in comparison to previous work [23][24] that was using only one virtual line sensor. By using an unlimited number of virtual line sensors the accuracy of every line to count the object passing through could be increased close to 100%. Users could freely place as many sensors as needed, anywhere on the image.

The angle of the line could range from 0 degree and goes back to 360 degree. There is no limitation on the virtual lines' angle. The proposed work has improved the earlier research that also used multiple lines [25], however with less flexibility, as the line could only be placed in a perfectly horizontal position (180 degree) within a rectangular ROI (region of interest) area.

(a)

(b)

(c)

Fig. 14. Experimental result tested on a minimum lighting condition at night. The result was on average lower than daylight conditions at 91%. This resulted from detecting the cars' headlight that were understood as a moving object by the system. In Figure (a) and (c), the light reflections on the road surface created by the cars' headlights (marked by the encircled areas) were understood as moving objects. Figure (b) camera orientation to record traffic from the back, resulting in the minimum quantity of reflection of a vehicle's headlight on the road surface.

The proposed work was also tested in minimum lighting conditions at night. The validity result was on average lower than the daylight result - at 91% as a consequence of detecting the cars' headlight reflections together with the moving cars as two different objects. As depicted in Figure 14(a) and Figure 14(c), the light reflections on the road surface created by the cars' headlights (marked by the encircled areas) were understood by the system as moving objects. These light reflections on the road surface give significant value variance on the pixel value delta calculation to be detected as object movements. Nevertheless, the problem came when there existed a gap between the light's reflection and the car. The gap could be understood as a gap between two distinct objects that resulted in double counting of moving cars at night. However, when there was no gap in between a car and its headlight's reflection on the road surface, then it was correctly assumed to be one moving car. The root of this setback is the position of the camera that was directly facing the incoming traffic where the headlight reflections on the road surface were significantly visible to the system. On the contrary, when the camera was oriented to record the traffic from the back, the problem was significantly reduced. This is confirmed and depicted in Figure 14(b), where the reflection of a vehicle's headlight on the road surface is minimal - thus not counted as a moving object.

Furthermore, none of the cyclists with no lights and pedestrians could be detected during minimum light conditions. None of the previous research that was mentioned above was tested in dark conditions.

Although it was tested under minimum light conditions, this work mainly focused on the accuracy and reasonable performance for the counting of object movement during daylight without significantly degrading the speed of performance carried out by the proposed system. To provide detailed results, the test was done on two videos that were taken on different types of roads and environments during daylight with the duration of two minutes each. The position of the camera was placed on top of the road angle at about 45 degrees to the road surface. Both videos have the same number of virtual lines applied (4 virtual lines) to count number of objects (vehicles and pedestrian) crossing any of these lines.

Fig. 15. Experimental result's screen capture of the object counting processes in daylight – four virtual lines on a road without lanes.

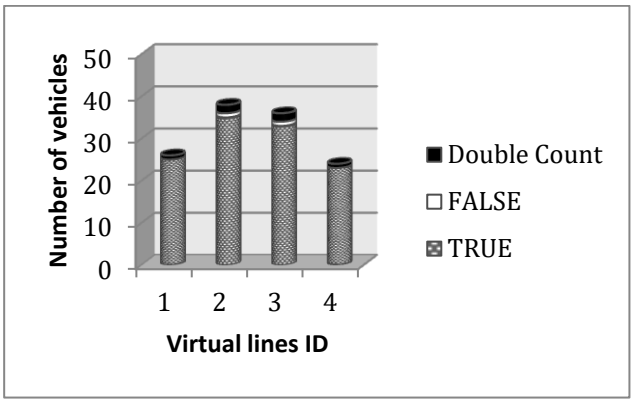

Fig. 16. Accuracy level at daylight using four virtual lines on road without lanes. Accuracy = 93.55%.

TABLE II
FOUR VIRTUAL LINES ON A ROAD WITHOUT LANES

LINE#	TRUE	FALSE	Double Count	TOTAL
1	25	0	1	26
2	35	1	2	38
3	33	1	2	36
4	23	0	1	24

Fig. 17. Experimental result's screen capture of the object counting processes in daylight – on a road with lanes.

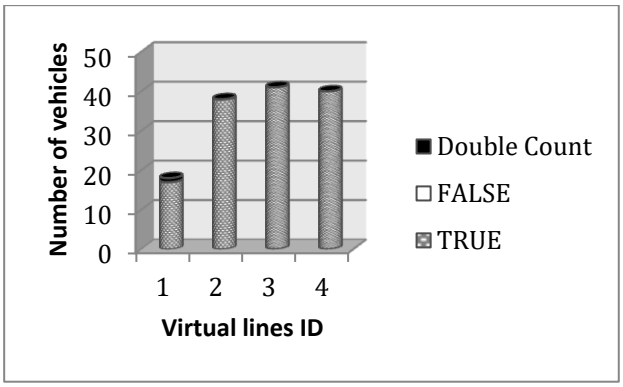

Fig. 18. Accuracy level in daylight using four virtual lines on a road with lanes. Accuracy = 99.27%.

TABLE III
FOUR VIRTUAL LINES ON ROAD WITH LANES

LINE#	TRUE	FALSE	Double Count	TOTAL
1	17	0	1	18
2	38	0	0	38
3	41	0	0	41
4	40	0	0	40

Details of the experimental results to compare the proposed method's result on road with lanes and without lanes are depicted in Figure 15 until Figure 18. They are supported by graphs and tables (Table II and Table III) representing experimental results, both using the same number of virtual lines (4 lines). Both cases used two minutes input videos of daylight traffic. In the first result of a road without lanes (Figure 15), the virtual lines were placed on the screen as such to cover the road width with length approximately fit to the width of a car. Within Figure 16 and Table II, the TRUE result representing correctly detected vehicles. The FALSE result happened when two vehicles crossed the same line at the same time resulting in only one of them recorded. The double count result is when one vehicle recorded by two neighboring virtual lines or one vehicle is recognized by the system as two vehicles. The later case could potentially happen for long separate body of vehicles such as: carts and containers.

For instance, line-1 in Figure 16 and Table II was crossed by 26 vehicles with one vehicle. It has one double count vehicle as a result of crossing both line-1 and line-2 which is its neighboring line. However, no false result was recorded by the line during the sample video. Moreover, on average roads without lanes resulted in further false and double count results. Line-2 and line-3 have two double count results and one falsely detected vehicle for each line. This result would be the consequence of the nature of traffic within a road without lanes, whereby vehicles could move diagonally or zigzag on the road vector. On the contrary, road with lanes had an improved result where only one double count vehicle was found on line-1 (Figure 18 and Table III) during the given two minutes sample video with the same daylight condition. This is majorly caused by the nature of ordered traffic within a well-managed road with lanes. The accuracy level could be enhanced at around 5.7% from 93.55% up to 99.27%.

The results of experimenting with the system at night is relatively lower in accuracy. The double count problem contributes the most to the error rate of the statistics. Firstly, this problem is caused by counting a car twice as a result of the gap between it and its headlight reflection. The gap separating them will be translated as two different objects. Secondly, two different virtual lines captured the same vehicle as the result of it driving between the two virtual lines. This problem commonly happens in the context of roads without lanes.

Fig. 19. Object counting processes at night using four virtual lines on road with lanes.

TABLE IV
FOUR VIRTUAL LINES ON A ROAD WITH LANES AT NIGHT

LINE#	TRUE	FALSE	Double Count	TOTAL
1	8	0	0	8
2	12	0	0	12
3	6	0	2	8
4	14	0	2	16

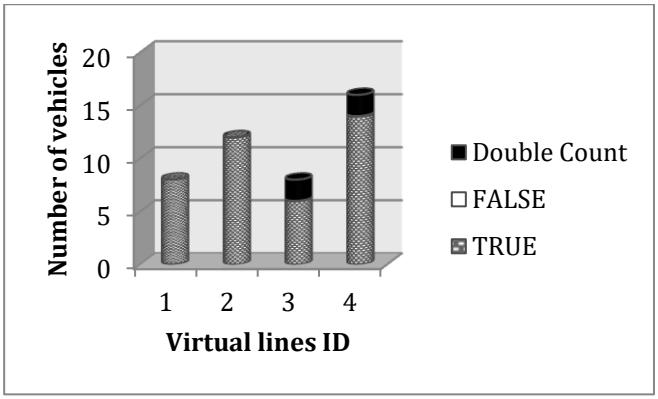

Fig. 20. Accuracy level at night using four virtual lines on road with lanes. Accuracy = 90.91%.

Figure 19 depicts the experimental result at night on the road with lanes. Two out of four virtual lines indicated more double counts than the rest. Table IV details the double count occurred at the last two lines which are placed directly facing the incoming traffic. The accuracy level could reach up to 91% as described in Figure 20.

Fig. 21. Object counting processes at night using four virtual lines on a road without lanes.

TABLE V
FOUR VIRTUAL LINES ON ROAD WITH LANES AT NIGHT

LINE#	TRUE	FALSE	Double Count	TOTAL
1	8	0	1	9
2	34	1	6	41
3	41	1	8	50
4	26	0	2	28

A better-positioned camera was also tested during night-time. The camera was oriented to capture the traffic from the back in order to avoid headlight exposure (Figure 21). However this was implemented on a road without lanes. The double count could well have occurred during the experiment as a result of vehicles driving in between two lines, or changing lanes that crossed two virtual lines (Table V).

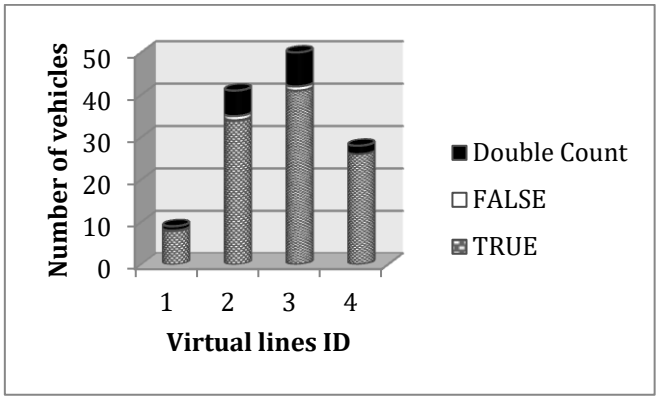

Fig. 22. Accuracy level at night using four virtual lines on a road without lanes. Accuracy = 85.16%.

The result of this was 85.16%, which is about 5% lower than the experiment on roads with lanes.

VI. Conclusion

In conclusion, the proposed system of using multiple virtual lines as sensors to count moving objects and live traffic flow proved to be effective, with a high accuracy within sufficient lighting conditions such as daylight. The accuracy of detecting the moving objects could be close to 100% as a result of having individual line monitoring specific areas on the screen. As the proposed system is able to accommodate multiple line sensors placement on the screen, the user could have unlimited locations to be monitored while still considering the size of the object being monitored kept above a certain threshold.

Moreover, as the placement of the virtual lines does not have to follow certain positioning restrictions or rules the proposed application works excellently for counting objects in various angles of traffic surveillance cameras. This could efficiently work for any existing surveillance system with many cameras already in place, without the need to adjust their positions.

Implementation of the proposed system could be prolonged for detecting the number of people in a certain area, such as counting visitors in grocery store isles and as a trigger to a frame-grabber for automatic license plate recognition.

Future development of the work could be extended to measure a number of points that have significant delta pixels value (movement) and classify the values into different groups such as trucks, cars, motorbikes/bicycle, or pedestrians. This would assist in a more precise counting system.

Acknowledgment

The author would like to extend his gratitude to the Research Department of JWC campus (Joseph Wibowo Center), Bina Nusantara University, Jakarta, Indonesia, in the support of this research.

References

[1] R. Greene, M. C. Diogenes, D. R. Ragland, and L. A. Lindau, "Effectiveness of a Commercially Available Automated Pedestrian Counting Device in Urban Environments: Comparison with Manual Counts", TRB Annual Meeting, UCB ITS TSC, 2008.

[2] B. Fanping, R. Greene, D. Ryan, C. Mara, and R. David, "Estimating Pedestrian Accident Exposure: Automated Pedestrian Counting Devices", Safe Transportation Research & Education Center, Institute of Transportation Studies, UCBerkeley, 2007.

[3] R. Hughes, N. RouphaiL, and C. Kosok, "Exploratory simulation of pedestrian crossings at roundabouts", Journal of Transportation Engineering American Society of Civil Engineers, 2003.

[4] M. W. John, and E. W. Robert, "Vehicle counting system for a vehicle parking lot", US Patent, number: US 5389921 A, 1995.

[5] P. T. Blythe, "Video-based vehicle and pedestrian tracking and motion modeling", Eleventh International Conference on Road Transport Information and Control, pp. 35 – 40, 2002.

[6] S. Nambisan, S. Pulugurtha, V. Vasudevan, M. Dangeti, and V. Virupaksha, "Effectiveness of Automatic Pedestrian Detection Device and Smart Lighting for Pedestrian Safety", Journal of the Transportation Research Board of the National Academies, ISSN 0361-1981, vol. 2140 / 2009, pp. 27-34, 2009.

[7] C. Yang, G. Haifeng, Z. S. Chun, and T. Yandong, "Flow Mosaicking: Real-time Pedestrian Counting without Scene-specific Learning", Computer Vision and Pattern Recognition, CVPR 2009, IEEE. 978-1-4244-3992-8, 2009.

[8] T. Ben, Z. Junping, and W. Liang, "Semi-supervised Elastic net for pedestrian counting", Pattern Recognition, vol. 44, no. 10–11, pp. 2297–2304, 2011.

[9] B. Antoni, Z. Chan, and L. John, "Privacy Preserving Crowd Monitoring: Counting People without People Models or Tracking", Computer Vision and Pattern Recognition, CVPR 2008, IEEE 978-1-4244-2243-2/08, 2008.

[10] B. Erhan, and T. Murat, "Automatic Vehicle Counting from Video for Traffic Flow Analysis", Intelligent Vehicles Symposium, 2007.

[11] L, Manchun, L. Damien, G. Pierre, and M. Kadder, "A Video-based Real-time Vehicle Counting System Using Adaptive Background Method", IEEE International Conference on Signal Image Technology and Internet Based Systems, 2008.

[12] C. Pornpanomchai, T. Liamsanguan, and V. Vannakosit. Vehicle, "Detection And Counting From A Video Frame", Wavelet Analysis and Pattern Recognition, ICWAPR, vol. 1, pp. 356 – 361, 2008.

[13] W. Kunfeng, L. Zhenjiang, Y. Qingming, and H. Wuling, "An Automated Vehicle Counting System For Traffic Surveillance", Vehicular Electronics and Safety ICVES, IEEE International Conference, 2007.

[14] Y. Yonghong, "A Traffic-Flow Parameters Evaluation Approach Based on Urban Road Video", International Journal of Intelligent Engineering & Systems, 2nd conference, 2009.

[15] C. M. Niluthpol, U. R. Nafi, and R. Mahbubur, "Detection and Classification of Vehicles From Video Using Multiple Time-Spatial Images", IEEE Transaction on Intelligent Transportation Systems, vol. 13, no. 3, pp. 1215, 2012.

[16] L. Jong-Sen, "Digital image smoothing and the sigma filter", Computer Vision, Graphics, and Image Processing, vol. 24, no. 2, pp. 255–269, 1983.

[17] J. E. Bresenham, "Algorithm for computer control of a digital plotter", IBM Systems Journal, vol. 4, no. 1, ISSN: 0018-8670, DOI: 10.1147/sj.41.0025, pp. 25 – 30, 1965.

[18] L. Vibha, H. Chetana, P. D. Shenoy, K. R. Venugopal, and L. M. Patnaik, "Dynamic Object Detection, Tracking and Counting in Video Streams for Multimedia Mining," IAENG International Journal of Computer Science, vol. 35, no. 3, pp 382-391, 2008.

[19] L. Nan, W. Jihong, Q.H. Wu, and Y. Li, "An Improved Motion Detection Method for Real-Time Surveillance," IAENG International Journal of Computer Science, vol. 35, no. 1, pp 119 -128, 2008.
[20] P. Andrey, T. Kirill, V. Vladimir, S. Evgeny, and M. Ivan, "Applications of Image Filtration Based on Principal Component Analysis and Nonlocal Image Processing," IAENG International Journal of Computer Science, vol. 40, no. 2, pp 62-80, 2013.
[21] K. Jae-Won, Kang-Sun Choi, Byeong-Doo Choi, and Sung-Jea Ko, "Real-time Vision-based People Counting System for the Security Door", Intelligent Systems Design and Applications, ISDA '08, 8th International Conference on, vol. 3, Print ISBN: 978-0-7695-3382-7, pp. 565 – 569, 2008.
[22] J. Barandiaran, B. Murguia and F. Boto, "Real-Time People Counting Using Multiple Lines", Image Analysis for Multimedia Interactive Services, 2008. WIAMIS '08. Ninth International , E-ISBN: 978-0-7695-3130-4, Print ISBN: 978-0-7695-3344-5, pp. 159 – 162, 2008.
[23] X. Liu, P. H. Tu, J. Rittscher, A. Perera, and N. Krahnstoever, "Detecting and Counting People in Surveillance Applications", IEEE Conference on: Advanced Video and Signal Based Surveillance, AVSS 2005, Print ISBN: 0-7803-9385-62005, pp. 306 – 311, 2005.
[24] Y. Kin-Yi, S. Wan-Chi, L. Ngai-Fong, and C. Chok-Ki, "Effective bi-directional people flow counting for real time surveillance system", Consumer Electronics (ICCE), IEEE International Conference, ISSN : 2158-3994, Print ISBN: 978-1-4244-8711-0, pp. 863 – 864, 2011.
[25] S. H. Kim, S. Junyuan, A. Alfarrarjeh, D. Xu, Y. Tan, and C. Shahabi, "Real-Time Traffic Video Analysis Using Intel Viewmont Coprocessor", Databases in Networked Information Systems, vol. 7813, Lecture Notes in Computer Science DNIS 2013, LNCS 7813, Springer-Verlag Berlin Heidelberg, pp. 150–160, 2013.

S. M. Tedjojuwono. Born in Surabaya, Indonesia 15th February 1978. He acquired his bachelor degree in Information Systems at Bina Nusantara University, Jakarta, Indonesia in 2000; and completed his Master's degree in Information Technology at Adelaide University, Adelaide, Australia in 2005.

He was Head of the Software Laboratory, Bina Nusantara University (Anggrek Campus) until 2003. After completing his Master's degree, he worked as Subject Content Coordinator (programming and artificial intelligence) for the same institution until 2012. He is currently the Head of Program (Information Systems – International program) for Bina Nusantara University, Indonesia. His latest paper in 2015, "Fast Performance Indonesian Automated License Plate Recognition Algorithm Using Interconnected Image Segmentation", was published at the International Conference on Soft Computing, Intelligent System and Information Technology - ICSIIT 2015, Springer and CCIS, indexed by Scopus, ISSN: 1865-0929, ISBN: 9783-662-46741-1, http://www.springer.com/gp/book/9783662467411 was awarded the best paper during the conference. His current research is in traffic information systems and automatic license plate recognition using video analytics to be implemented specifically for Indonesian vehicles' plate system.

Mr. Tedjojuwono was also the Vice President for the Post Graduate Student Association (AUPGSA) at Adelaide University Australia in 2004, was awarded with a teaching excellence award from Bina Nusantara University Indonesia in 2015, and received his IAENG membership in December 2015.