Feasibility Study on Development of a CMOS Vision Processor for Vehicle Tracking

Final Report

Prepared by:
Hua Tang

Department of Electrical and Computer Engineering
University of Minnesota Duluth

March 2007

Acknowledgements

The author is very grateful to Northland Advanced Transportation System Research Laboratory (NATSRL) for supporting this project. The author would like to thank Dr. Taek Kwon for his very valuable suggestions and discussions. The author also thanks to Dr. Eil Kwon for his discussion and review of the project.

Table of Contents

List of Figures

Executive Summary

Vehicle tracking is an important area of intelligent transportation systems (ITS) technology, which could be applied in a wide range of transportation applications. Tracking typically needs to monitor real-time vehicle movements, and thus real time tracking is highly desirable. However it is well known that vehicle tracking processes are computationally very intensive. In the past, regardless of different algorithms employed in vehicle tracking, they have been implemented using software approaches, e.g., FPGA (Field Programmable Gate Array), micro-controller or embedded micro-processor, and PCs. While software approaches have an advantage of flexibility in implementation and future modifications, its long computational time often prevents real-time vehicle tracking from high resolution spatial or temporal data. It is well known in the area of VLSI (Very Large Scale Integrated) circuit design that a customized and dedicated hardware implementation of any algorithm minimizes its computational time. This gives us the motivation for direct implementation of tracking algorithms in hardware (i.e., device level), whether it is a partial or full implementation, to enhance real-time operation.

The goal of this seed project is to investigate the feasibility and related issues in developing a tracking system with a new tracking algorithm based on vehicle motion detection, which is implemented in hardware whenever possible so that the computational time for tracking is minimized. The proposed overall tracking system consists of two parts. One part is the hardware, more specifically, a CMOS (Complementary Metal Oxide Semiconductor) hardware processor which is mainly responsible for vehicle motion detection. The other part is the software, for example an FPGA or micro-controller which is responsible for analyzing the data transmitted from the hardware and properly associating vehicles for tracking. The main computational time saving for the tracking process comes from the hardware part since the core of the new tracking algorithm, motion detection, is run on a dedicated hardware for that particular purpose.

The proposed tracking algorithm is simulated in MATLAB and tested on traffic images captured from an intersection. It is found that vehicle movements can be accurately identified in spite of some noisy motion. Also, in this project, we estimate the computational time for the tracking algorithm in hardware implementation and discuss high-level hardware designs for actual implementation of the tracking algorithm.

Chapter 1

Introduction and Previous Work

Real-time operation is essential for tracking vehicle movements. For example, [1] presents a method for tracking and counting pedestrians using a single camera. In order to accurately capture important movement information, the system must acquire pictures at a high-frame rate. However, due to a large computational delay required by software implementation of the tracking algorithm, frame rate is limited. As reported in [1], the peak frame rate is at most 30 frames/sec using low resolution gray scale images (256×256 pixels). As higher image resolution or colors are introduced for accuracy, the frame rate often drops to just a few per second, which may present significant problems in tracking accuracy [1].

There are two ways to improve real-time operations. One is to develop new tracking algorithms that are simpler and faster, thus requiring less computational time. While this is feasible and has been under research [4], a tradeoff exists between the tacking accuracy and algorithm complexity. In general, the more complex the tracking algorithm is, the more accurate the tracking is, so the longer the computational time is.

A more plausible solution is to make the computational of tacking algorithm faster by implementing the algorithm in hardware [12]. In order to get quantitative insight regarding the possible amount of improvements on the computational time for the tracking in the hardware approach as compared to the software approach, the tracking algorithm reported in [1] in software approach will be used as the reference work. The tracking algorithm in [1] is divided into three phases of operation: raw image processing, blobs graph optimization and Kalman filtering based estimation of object parameters. First, at the raw image level, Gaussian noise filtering, background subtraction and thresholding is performed to produce binary difference images in 256×256 pixels. The difference images are then supplied to the next phase of operation, where the changes with respect to two consecutive difference images are described in terms of motions of blobs. After extracting blobs from the difference images, blob motions with respect to two consecutive difference images is modeled as a graph optimization problem for which the overall goal is to find the blob motions that cause the smallest amount of size change with respect to two consecutive difference images. Note that in this process, additional constraints are enforced in order to make the graph manipulation computationally tractable [1]. But these constraints are based on the critical assumption that images are captured in a very high frame rate (which may contradict the actual situation of being not high frame rate). Also note that blob graph optimization is an iterative and enumerative process, which requires a lot of computational resources. In the final phase of the tracking algorithm, EKF (Extended Kalman Filtering) is used to estimate object parameters. Note that, the computational requirement of Kalman filtering is quite high and sometimes even beyond the system capability [5]. Without a proper timing control of Kalman filtering, the overall tracking performance may degrade significantly [5].

The overall tracking algorithm in [1] is implemented on a Datacube MaxVideo 20 video processor and a Max860 vector processor and is later ported to 400Mhz Pentium PC with C80 Matrox Genesis vision board. For the software implementation of tracking algorithm, peak frame rate is reported to be 30 frames/sec while in many cases it drops to 10~20 frames/sec depending on the number of objects and weather conditions [1]. An inherent conflict exists between the requirement of higher time and space resolution (or to have high frame rate) and the requirement of a longer computational time by the software implementation. Note that the same tracking algorithm presented in [1] has been applied to many other cases, such as [2] [3] and the same software approach is used.

Software implementation of the tracking algorithm, which already has a longer computational time, will experience a severe time-crunch problem when the algorithm is to improve the tracking performance by adding more processing steps. For example, considering the shape of objects being tracked adds a considerable amount of computational overhead [1]. Detection and classification of vehicles can only be done for two general classes (cars versus SUVs), while further classifying within the two classes significantly lengthen the computational time, preventing real-time classifications of vehicles [3]. To improve the tracking accuracy, some other operations, such as shadow removing, may be required, which again takes significant amount of computational time [1][6][7]. As a result, pure software approaches may not meet the more and more demanding requirement of real-time operations.

Motivated by the limitations of the software approaches for vehicle tracking, there has been recent research interests in hardware implementation of algorithms used in different areas of ITS applications. Yang proposes a 256×256-pixle CMOS image sensor used in line-based vision applications [8]. Nakamura implements a sensor with inherent signal processing functionalities [9]. Fish also implements a CMOS filter which can be used in lane detection applications [10]. Hsiao uses circuit design techniques from previous works [8][9][10] and proposes a hardware implementation of a mixed-signal CMOS processor with built-in image sensor for efficient lane detection for use in intelligent vehicles [11]. Hsiao does not compare the computational time of the processor to that of software implementation for the lane detection algorithm, but it is shown that a hardware approach can even improve the performance of simple lane detection [11].

Drawing from the current and future demands for real-time operations of vehicle tracking, the seed project intends study the feasibility of developing a new tracking system largely based on a CMOS vision processor. Note that our main initiative to implement a CMOS vision processor is to improve real-time operation of vehicle tracking, since a hardware implementation of tracking algorithm could significantly improve the processing time compared to the software approach. In addition, a few other advantages can be identified for hardware implementation of tracking algorithm. First, the performance of real-time vehicle tracking will be greatly improved. Many other tasks that previously could not be incorporated in the tracking process due to computational time constraint may now be enabled. Second, potential improvements on tracking accuracy may be expected. For instance, the performance of lane detection for a CMOS processor is improved compared to the software approach, as reported in [11]. This may not be surprising from the VLSI circuit design point of view, since a dedicated hardware has less undesired effects, such as accumulated digitization error [12]. Another benefit is that the hardware size could be much smaller compared to micro-controllers (software implementation), thus the overall system will be more portable. Moreover, the power consumption of dedicated hardware will potentially be hundreds of times less compared to software implementations [12]. This feature would be especially important in the future effort to save energy. Even more importantly, the cost of the tracking system could be reduced to a fraction of that of software implementations.

But we also identify some potential disadvantages of the hardware approach. First, the initial development cost may be huge and design cycle for a hardware processor is typically much longer than that of the software approach. This possible huge design effort could be only compensated for by extensive usage of the developed hardware in real ITS applications. Second, typically a hardware approach is not as flexible as a software approach. In the software approach, designers only need to change the software codes to adapt to new situations, whereas the hardware approach may not be easily adapted to new situations. Though reconfigurable design techniques can be applied during hardware design to achieve flexibility, this is at the expense of extra design effort and hardware components.

The rest of the report is organized as follows. Chapter 2 presents the proposed motion detection based tracking algorithm. Chapter 3 investigates the feasibility of hardware implementation for the tracking algorithm to achieve high frame rate real-time operation. Finally, Chapter 4 concludes the report.

Chapter 2

The Proposed Motion Detection Based Tracking System

The main initiative of hardware implementation of tracking algorithm is to minimize the computational time to improve real-time operation for vehicle tracking, but directly proceeding with hardware implementation of a tracking algorithm may make the overall design process impractical and un-rewarding considering other factors. The other factors that have to be considered for hardware design include the overall development cost, the overall hardware complexity (design cycle) and the required flexibility [12]. If the overall development cost is prohibitively high, then a software approach to implement tracking algorithm should be favored since micro-controllers are much cheaper. Similarly, if the design cycle is too long (time-to-market is thus seriously delayed), hardware implementation is also at a disadvantage. Considering all these factors, a strong argument for hardware implementation is that the hardware can be designed in a both time and cost effective way and the designed hardware will be re-useable in different tracking algorithms [12]. The later also relates well to the area of Hardware-Software-Codesign [12] since we may expect that some processing steps of the tracking algorithm are more suited to hardware implementation while the others are to software implementation. Considering all these factors, simply taking an existing tracking algorithm, such as the one in [1][2][3], and implementing it in hardware may not be a good solution. In fact, our study shows that implementing the tracking algorithm in [1][2][3] in pure hardware is extremely costly. But on the other hand, if we implement only some part of the tracking algorithm in hardware, the overall computational time saving is not promising. As a result, improving real-time operation of tracking is more than just pure hardware implementation, and instead it requires innovation in both tracking algorithm design, which has to be well suited to hardware design with reasonable design effort, and then hardware implementation of the tracking algorithm.

2.1 Overview of the proposed tracking algorithm

With the above guidelines, the seed project needs to consider the following two questions. First, what is the overall processing flow of the tracking algorithm? Second, for such a tracking algorithm, what processing operations can be implemented in hardware to minimize the computational time? These two questions are highly related to each other, and therefore can not be treated separately.

Our proposed solution to the above questions is a new tracking algorithm based on vehicle motion detection, with motion detection being well suited to hardware implementation. The motion detection is based on the popular block-matching algorithm used in image processing [14]. The motion vectors found from the block matching process contain information on how and where the blocks of the images move. For the proposed tracking algorithm, a system diagram is shown in Figure 1 below.

The processing flow of such a tracking system is explained as follows:
• Step 1: we assume that an image sensor or a digital camera captures images at a high frame rate, for example, 60-100 frames/sec. Note that this assumption has to be validated by relatively short computational time for the hardware approach.
• Step 2: each captured image frame is subject to Gaussian noise filtering to remove the thermal noise.
• Step 3: two consecutive image frames are first stored and then compared to identify object motion (in this case, the objects are vehicles). The vehicle motion detection is based on the block matching algorithm. The outputs of the block matching computation are the motion vectors of all blocks in the image.
• Step 4: the motion vectors that define whether each block moves and how it moves need to be digitized (if it is not digital signal) and transmitted to the next stage of processing.
• Step 5: in this final step, the data transmitted from the previous step is analyzed to identify moving vehicles and track how and where each vehicle moves.

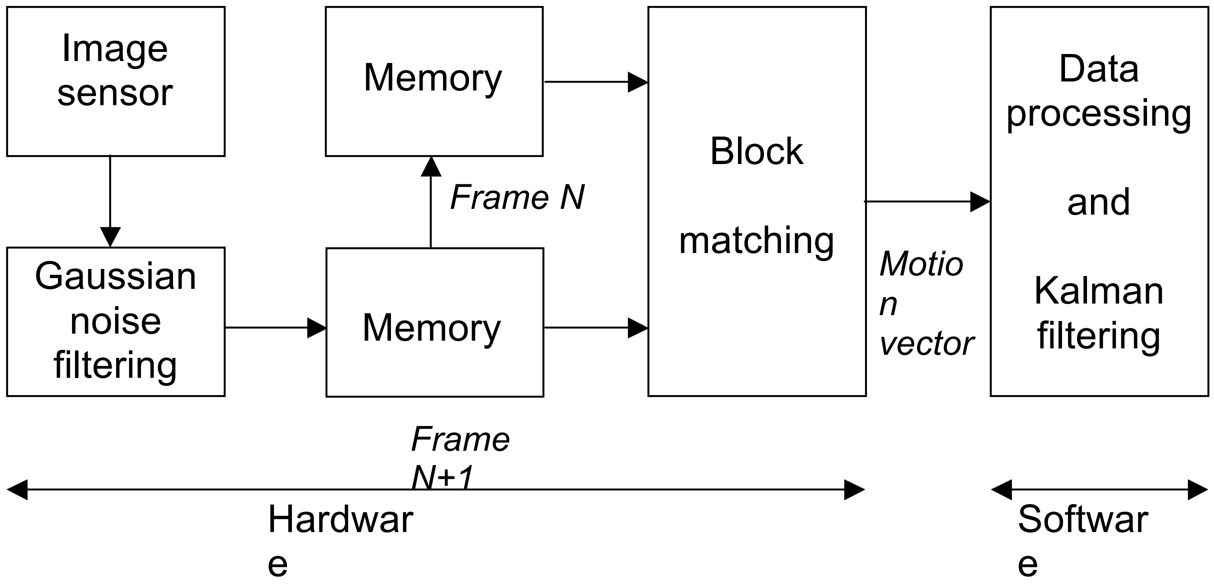


Figure 1. The system diagram of the overall tracking system

For the proposed tracking algorithm, step one to step four are well suited to hardware implementation, whereas the final step is suited to software implementation. As a result, an ideal plan to implement the tracking system is to design a CMOS hardware processor for vehicle motion detection and then a software system for vehicle association and tracking. The reason for such a decision is that we believe it is a good tradeoff between computational time and development cost. If the overall tracking system is implemented in hardware, the development cost and design complexity could be huge. Another reason is that a hardware processor for vehicle motion detection can be widely used in many image processing applications and ITS applications [14]. In that case, it would be worthy of the possible high development cost and design effort associated with hardware implementation of the tracking algorithm.

Step 1 and step 2 are typical operations of vision-based vehicle tracking system [1]. Their hardware implementation will be discussed in the next Chapter. Step 3 is the core of the proposed vehicle tracking algorithm. The algorithm aspect of this step, i.e. the block matching algorithm, is discussed next while the hardware implementation is discussed in next Chapter.

An illustration of block matching algorithm is shown in Figure 2 below. Suppose that the image size is M×S and the block size is n×n, then a total of (M×S)/(n×n) blocks are defined for each image (for illustration purpose, Figure 2 shows only 16 blocks for each frame). With respect to two consecutive images, shown as "current frame" (say frame number N) and "previous frame" (frame number N+1) in Figure 2, reference block A (the dark shaded region) in "current frame" can be considered as a moved version of block A' in "previous frame" and block A' is in the neighborhood area of the A. This neighborhood is defined by parameter p in all four directions (up, down, right and left) from the position of block A. The value of p is determined by the frame rate and object speed. If the frame rate is high, as we assumed in the proposed tracking system, then parameter p can be set small since we do not expect the block A has moved dramatically within a short period of time. In the neighborhood area of (2p+n)×(2p+n), there are a total of (2p+1)×(2p+1) possible candidate blocks and the block A' in frame N that gives the minimum matching value is the one that becomes A in frame N+1. The matching value, more specifically MAD (Mean Absolute Difference), is defined as follows [14]:

$$MAD(u,v) = \sum_{i=1}^{n} \sum_{j=1}^{n} | S(i+u, j+v) - R(i,j) |, \quad -p \le (u,v) \le p$$

where $R(i,j)$ is the reference block of size n×n, $S(i+u, j+v)$ is the candidate block within the search area and (u,v) represents the block motion vector. Though more accurate matching value computation can be obtained by using MSE (Mean Squared Error) criterion instead of MAD [14], MAD becomes a standard since the computational overhead of MSE may be significant due to square operation.

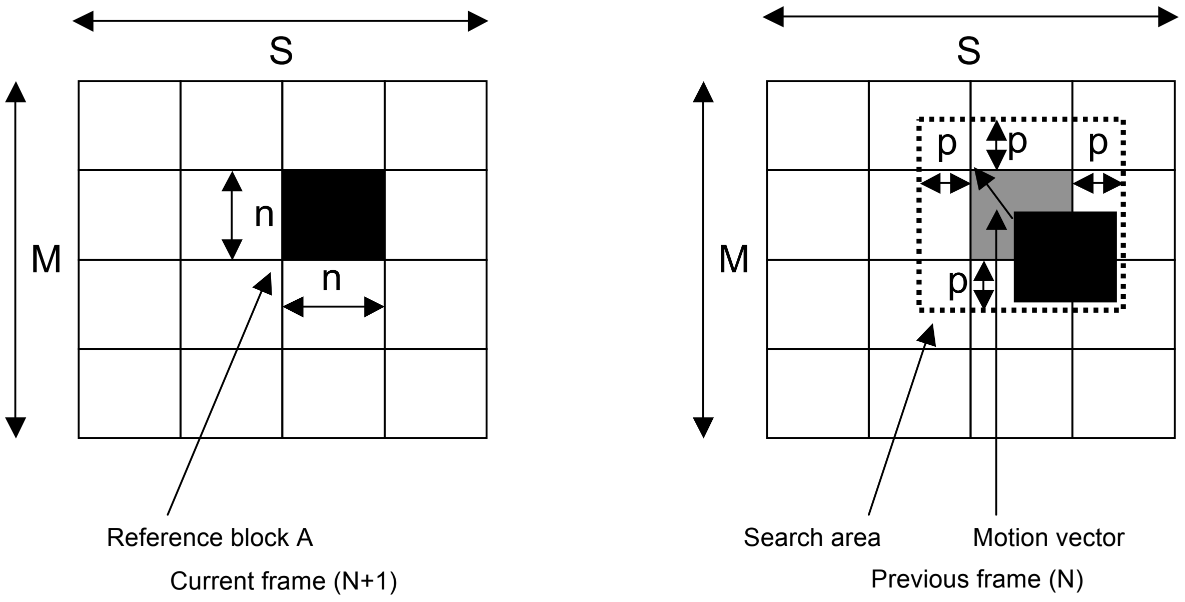


Figure 2. Motion estimation based on full search block matching

In the above computation of MAD, pixel values corresponding to light density can be used if the image is of gray scale. Block matching algorithm has been widely used in image compression to save memory space [14], but it involves heavy computations, i.e. a huge number of subtractions. Whether the tracking system is able to achieve high frame rate real-time operation critically depends on the computational time of the block matching algorithm. From algorithm point of view, one way to reduce computation time of block matching algorithm is to use partial search, instead of full search. That is, instead of trying out all possible combinations of motion vector (*u, v*), only a subset is explored [14]. The other effective way is to use small p so that the number of motion vectors to be explored is relatively small. This could significantly reduce computational time. But in order to warrant small p, the tracking system has to operate at high frame rate, which on the other hand demands small computational time. Another way to reduce computational time is to use hardware implementation of the block matching algorithm. If the algorithm is implemented in software, as will be shown below in the case of MATLAB simulation on a PC (Personal Computer), the matching computation of all blocks takes excessively long time. Hardware implementation employing parallel processing techniques is very promising to reduce the computational time. This is to be discussed in next Chapter.

The above block matching algorithm for motion estimation corresponds to step 3 in the proposed tracking algorithm. When block matching for all blocks is completed, the motion vectors are recorded and transmitted to the software part. This corresponds to step 4 of the tracking algorithm. As discussed in next Chapter, there are possible two techniques for hardware implementation of the tracking algorithm, analog and digital. If analog technique is used, a hardware unit for signal conversion in step 4 is necessary since step 5 is implemented in software which takes only digital signals as inputs.

Step 5 is the last step of the proposed tracking algorithm. It is also a typical operation employed in conventional tracking algorithms [1][2][3]. This step is kept for software implementation due to the possible huge design effort if it were to be implemented in hardware. Besides, the qualitative reasoning nature of this step may not be well suited to a dedicated hardware implementation.

2.2 Comparison with conventional tracking algorithm

In this section, we compare the proposed tracking algorithm to the conventional ones in [1][2][3]. Both algorithms adopt the video based tracking approach. That is, the inputs to the tracking algorithm are images at different time instants.

Regarding the difference, in general, motion detection in [1][2][3] is object-based, in contrast to block-based motion detection for the proposed tracking algorithm. [1][2][3] rely on background subtraction and then thresholding to identify objects (vehicles in the context). Using the terminology, these objects are called blobs [1]. The blobs can be extracted from the difference image using a contour following algorithm [1]. It does so for two consecutive image frames N and N+1, so that then these two difference images are compared to identify blob motion. To do that, it adopts a graph optimization approach. The blobs in each difference image is represented as a graph and the two consecutive graphs are optimized to find the a best matching of the blobs in terms of minimum total size change with respect to the two consecutive difference images. After this, blob motion data processing and Kalman filtering for prediction is used, similar to step 5 in the proposed tracking algorithm.

We identify several disadvantages of the conventional tracking algorithm and explain the detailed difference from the proposed tracking algorithm in amid of the discussion. First of all, the background subtraction involves a static background image, but the background may keep changing, which is not well addressed even with dynamic background updates [1]. Further, the image frame after background subtraction (the difference image) is then thresholded to obtain a binary image. The threshold value has to be obtained by background training and is set to the absolute maximum fluctuation of pixel values (range from 0 to 255, for instance). This involves manual setting and could introduce undesirable noise [1]. In contrast, our approach of block matching does not require a static background image and the background change has negligible effect for block matching provided that the frame rate is high and the background change is uniform which is usually true. This makes the proposed tracking algorithm more robust to whether and environment changes. Second, after thresholding, the image is a binary digital image and only objects (blobs) are remained in the image. Note that by now the pixel data information is lost and all we know is that whether each pixel corresponds to background or object. But in the proposed tracking algorithm, the rich pixel data information is kept and intentionally used for block matching computation. This makes the block matching algorithm potentially more accurate since it is based on pixel value difference instead of size difference in graph optimization of conventional tracking algorithm. Also, graph optimization is an iterative optimization process, which may take significant computational time in software implementation (in the order of milliseconds as reported in [1]) depending on complexity of the graphs. It also relies on the assumption of high frame rate to achieve the reported execution time. It the assumption is not satisfied, then the graph optimization may incur tracking error.

On the other hand, one possible disadvantage and the challenge of the project, is that the block matching algorithm in the proposed tracking system is even more complicated than graph optimization in conventional tracking algorithm from algorithm point of view and typically takes longer computational time. For example, block matching computation is in the order of minutes in MATLAB simulation. This means that the proposed tracking algorithm can not be applied in real applications if it is implemented using a software approach, such as MATLAB in general purpose PC and other embedded microprocessor. But on the other hand, block matching algorithm is well suited to hardware implementation. Therefore, we need to rely on efficient hardware implementation of the block matching algorithm so that the computational time for block matching would be minimized. In particular, in a first order estimation, the computational time of block matching in hardware implementation has to be lower compared to graph optimization in conventional tracking algorithm in order to have the proposed tracking algorithm outperform the conventional one. If this is true, we expect the conflict between the assumption of high frame rate and the actual computational time as in the conventional tracking algorithm would be solved or become less severe [1][2][3]. As a summary, the proposed new tracking system could improve real-time vehicle tracking, provided that the hardware implementation minimize the computational time of the block matching algorithm in step 3.

2.3 Simulation and verification of the tracking algorithm

In the context of the above described tracking system shown in Figure 1, our first step toward feasibility study of the proposed tracking algorithm is to simulate and verify the tracking algorithm in MATLAB [19]. All tested traffic images were obtained at the intersection before District I DoT building (intersection between Mesaba avenue and I-194) by a JVC Camcorder at 30 frames/sec. Note that these images were not captured optimally, in the sense that no special case was taken to mount the Camcorder to get a best view of the intersections. The recorded video was then converted to sequences of image frames using QuickTime Pro software [20]. By testing it on real traffic images, we found that vehicle movements could be accurately identified in spite of some noisy motion detection. In the following, we show some results obtained in MATLAB and illustrate some features of the proposed tracking system. The image size is 720×480 and the block size is 8×8. The parameter p that defines the search region is set to 8.

Figure 3 and 4 show two consecutive image frames with a number of vehicles (frame N and frame N+1). The vehicles enclosed by the red box in Figure 3 actually moved with respect to Figure 4. Specifically, vehicles A and E moved to the right, whereas vehicles B, C and D moved to the left. Other vehicles either did not move or too small in the Figure. Note also these two images were colored. When we tested the images on the proposed tracking algorithm, we converted them first to gray images with the same size. Thus, only light density information was used in blocking matching. Then, the two gray images were Gaussian-filtered to eliminate thermal noise and subsequently they were block matched to identify the motion vector for each block in the image. The block size was 8×8, therefore there were a total of 90×60 blocks defined in image frame. Here, there are tradeoffs in selecting the block size. If the block size is too large, then some blocks may not accurately represent a vehicle since each vehicle may be represented with multiple blocks. For example, a block may show only part of the vehicle and the rest of the block is just background. On the other hand, small block sizes may create multiple solutions in block matching. This is because the feature of a small block is less unique, and it is more likely that there exist many blocks nearby with similar features. This may result in wrong decisions for the motion vectors.

Next, in order to show the motion detection by the proposed tracking algorithm, we define the following definitions. Suppose that a block A has original coordinates as (x, y) in image frame N. To be uniform, (x, y) is taken as the coordinates of the upper-left corner of a rectangular block. Further suppose that the block A is found to be moved to (x', y') in image frame N+1, determined by the block matching process. Then we define motion of block A in the following five cases:

1 represents moving up if block A has (y'>y && |y'-y|>=|x'-x|)
2 represents moving down if block A has (y'<y && |y'-y|>=|x'-x|)
3 represents moving right if block A has (x'>x && |x'-x|>=|y'-y|)
4 represents moving left if block A has (x'<x && |x'-x|>=|y'-y|
5 represents no move if block A has (x'=x && y'=y)

Note that it is possible to define more refined movements, for example, moving both up and right. But we found that it is typically enough to identify the motion using the above five directions due to high frame rate and moderate resolution of the images. Also note that 5 does not mean necessarily that the block is a background block. In fact, a vehicle could also have 5 if it is not moving.

With the above definitions, we can now present the motion diagram showing the motion of each block computed from block matching. Since the block size is 8×8 pixels, all pixels in the block have the same moving direction. Therefore, it is enough to show how each block moves. This gave the following Figure 5 with dimension 90×24 and each element corresponded to the motion of that block. Note that only 24 out of 60 blocks in y-axis were shown in Figure 5, due to the upper part and lower part of the image frames anyway corresponding to background (background should not move). The red boxes in Figure 5 represent the same vehicles as in Figure 3. It can be seen that vehicle motion for those vehicles is accurately identified. For example, vehicles A and E had many values of 3 for the blocks corresponding to the vehicle positions in the image. Practically, the number of 3s is also a good idea of the size of the vehicle in the image. Here, the red box A actually represents two vehicles. Vehicles B, C and D had values of 4 for the blocks and the number of 4s can also be seen as good indication of the size of the vehicles. The left-most white vehicle and a few other vehicles beside B (refer to Figure 3) had no motion since they were waiting for the traffic signal. So, the motion diagram for these vehicles should be 5, which was the case as shown in Figure 5. But note first that there were some noisy motion detections, which typically happened to background blocks since the background was quite uniform and block matching computational might made wrong decisions. And this noisy motion also happened to vehicle E, especially at the right edges of E (some 1s and 2s there). This was because the vehicle E was moving out of the scene, and those blocks at the right edge could not find correct matching since they disappeared in the next image frame. This wrong decision could be corrected at the software part of the tracking algorithm, where information on current vehicle location, predicted future vehicle location (by Kalman filtering) and vehicle size change can be combined to fix it. This problem would not happen to vehicles staying inside the image for two consecutive image frames. Second, here in Figure 5, there is no way to differentiate vehicles if multiples vehicles are grouped and move in almost the same direction. To differentiate them, motion vectors need to be check up and Kalman filtering in the software part of the tracking system need to be used. Also, since we did not optimize the location of the Camcorder and view of the intersection, the traffic poles became a problem in block matching, which might be a source of error for noisy motion. In addition, there were no cars identified moving up or down.

Figure 5 gives the motion of each block, but could not tell where each block moves. To gain that information, we show the motion vectors in Figure 6 for y-direction (rotated 90 degrees for better view) and Figure 7 for x-direction. It can be seen that almost no blocks move in the y-direction except those block corresponding to vehicle E, which is almost the case in image frame N with respect to frame N+1. The vehicle that actually moved slightly along y-direction was vehicle E in Figure 3, and this movement was represented by 1 or 2 units. Note that, the values here should not be confused with the motion direction values as defined above and shown in Figure 5. Here, the values correspond to the movements in terms of (x, y) coordinates of the image, 0 to 720 along x-direction and 0 to 480 along y-direction. As a result, these values are called motion vectors since they show how much the vehicles move. In Figure 7, the block motion vectors in x-direction for all vehicles were shown. Right moving vehicles like A and E had positive values and left moving vehicles like B, C and D had negative values. For four of the five vehicles A, B, C and D, the motion vectors in x-direction were quite good and accurate. One problem here was that there were noisy motion detections for some blocks corresponding to the background. This could be corrected by defining smaller search regions (thus small parameter p) in block matching computation assuming high frame rate for the tracking system. Besides, there were also noisy motion detections around the edge of vehicle E due to vehicle E moving out of the scene. As mentioned in the above paragraph, this could be fixed in the software part of the tracking system. Practically, the values could be used to compute vehicle velocity. Of course, the effect of camera view on the vehicle velocity should also be taken into account in that case.

In the MATLAB verification of the tracking algorithm, the image size is 720×480 and the block size is 8×8. For computing the motion vectors of the 90×24 blocks in Figure 5, 6 and 7, the execution time was about 30 to 40 minutes in a Dell PC with 1Ghz XEON microprocessor. This time is simply NOT acceptable to have the proposed tracking algorithm to be useful in a software approach like the PC. Hardware implementations to cut down the execution time are discussed in next Chapter.

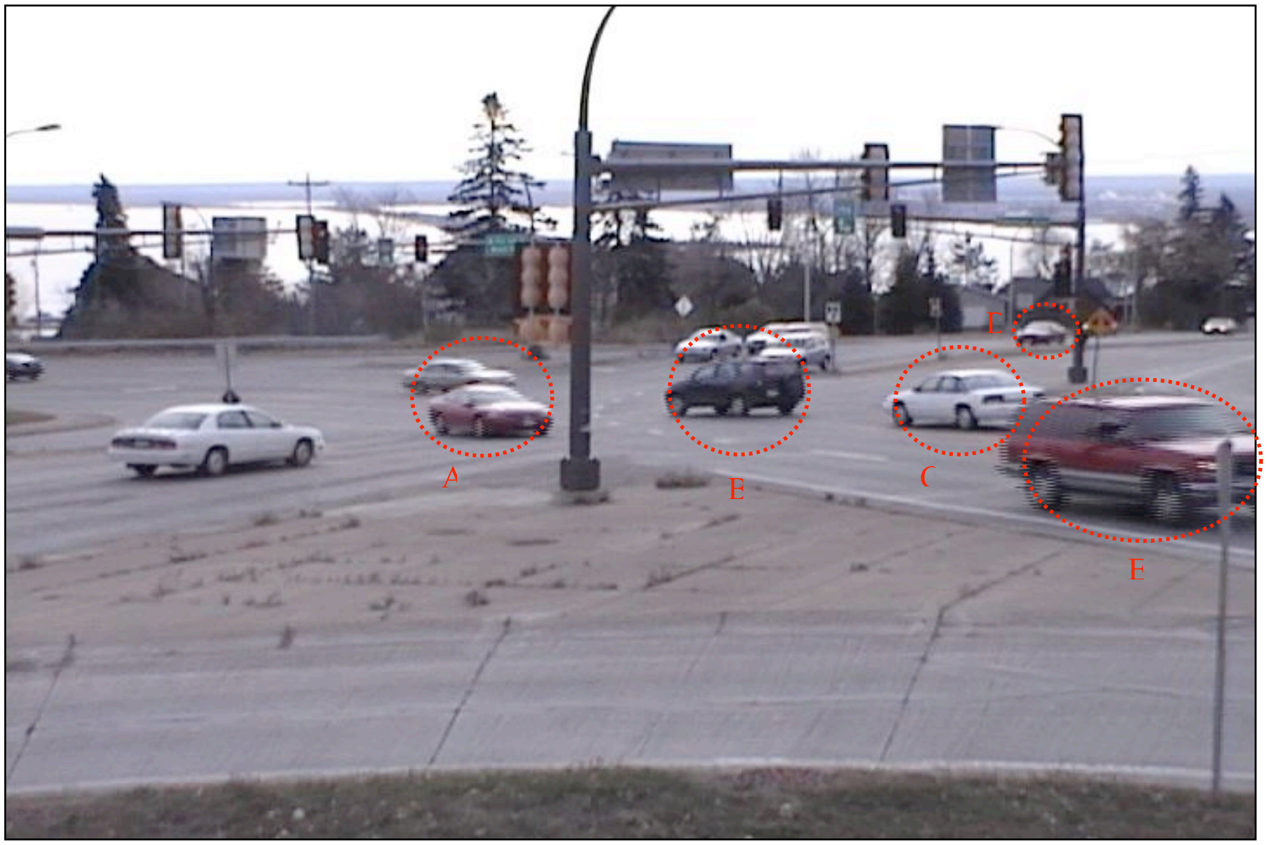


Figure 3. A traffic image captured at an intersection (frame N)

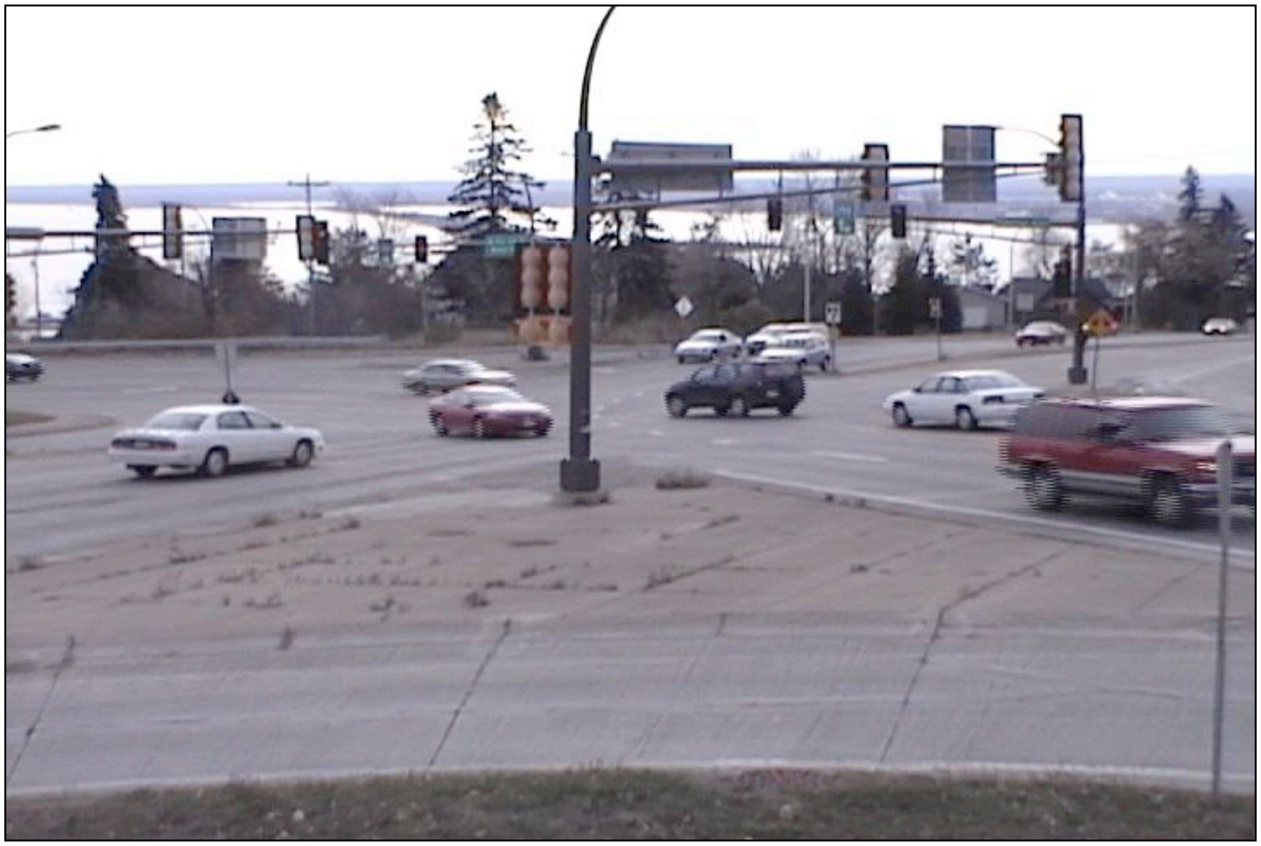
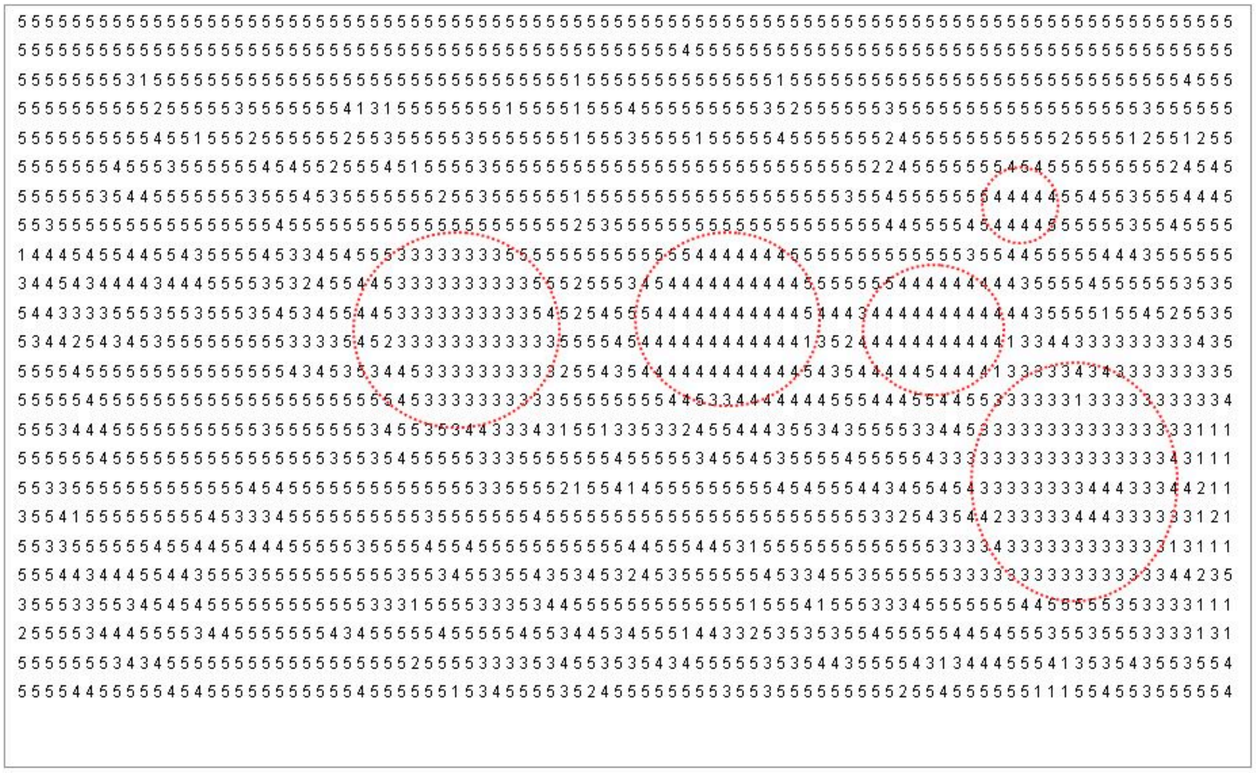


Figure 4. A traffic image captured at an intersection (frame N+1)



```
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 3 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5
5 5 5 5 5 5 5 5 5 5 2 5 5 5 5 5 3 5 5 5 5 5 5 5 4 1 3 1 5 5 5 5 5 5 5 5 1 5 5 5 5 1 5 5 5 4 5 5 5 5 5 5 5 5 5 3 5 2 5 5 5 5 5 5 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 4 5 5 1 5 5 5 2 5 5 5 5 5 5 2 5 5 3 5 5 5 5 5 3 5 5 5 5 5 5 5 1 5 5 5 3 5 5 5 5 1 5 5 5 5 5 4 5 5 5 5 5 5 5 2 4 5 5 5 5 5 5 5 5 5 5 5 2 5 5 5 5 1 2 5 5 1 2 5 5
5 5 5 5 5 5 5 4 5 5 5 3 5 5 5 5 5 5 4 5 4 5 5 2 5 5 5 4 5 1 5 5 5 5 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 2 4 5 5 5 5 5 5 5 4 5 4 5 5 5 5 5 5 5 5 5 2 4 5 4 5
5 5 5 5 5 5 3 5 4 4 5 5 5 5 5 5 5 3 5 5 5 4 5 3 5 5 5 5 5 5 5 2 5 5 3 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 5 5 4 5 5 5 5 5 5 5 4 4 4 4 4 5 5 4 5 5 3 5 5 5 4 4 4 5
5 5 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 5 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 5 5 5 5 4 5 4 4 4 4 5 5 5 5 5 5 3 5 5 4 5 5 5 5
1 4 4 4 5 4 5 5 4 4 5 5 4 3 5 5 5 5 4 5 3 3 4 5 4 5 5 5 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 3 5 5 4 4 5 5 5 5 5 4 4 4 3 5 5 5 5 5 5
3 4 4 5 4 3 4 4 4 4 3 4 4 4 5 5 5 5 3 5 3 2 4 5 5 4 4 5 3 3 3 3 3 3 3 3 3 3 5 5 5 2 5 5 5 3 4 5 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 3 5 5 5 5 4 5 5 5 5 5 5 3 5 3 5
5 4 4 3 3 3 3 5 5 5 3 5 5 3 5 5 5 3 5 4 5 3 4 5 5 4 4 5 3 3 3 3 3 3 3 3 3 3 5 4 5 2 5 4 5 5 5 4 4 4 4 4 4 4 4 4 4 4 5 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 4 3 5 5 5 5 1 5 5 4 5 2 5 5 3 5
5 3 4 4 2 5 4 3 4 5 3 5 5 5 5 5 5 5 5 5 3 3 3 3 5 4 5 2 3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 4 5 4 4 4 4 4 4 4 4 4 4 4 4 1 3 5 2 4 4 4 4 4 4 4 4 4 4 4 1 3 3 4 4 3 3 3 3 3 3 3 3 3 4 3 5
5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 3 4 5 3 5 3 4 4 5 3 3 3 3 3 3 3 3 3 3 2 5 5 4 3 5 4 4 4 4 4 4 4 4 4 4 4 4 5 4 3 5 4 4 4 4 4 5 4 4 4 4 1 3 3 3 3 3 4 3 4 3 3 3 3 3 3 3 3 5
5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 4 4 5 3 3 4 4 4 4 4 4 4 5 5 5 4 4 4 5 5 4 4 5 5 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 4
5 5 5 3 4 4 4 5 5 5 5 5 5 5 5 5 5 5 3 5 5 5 5 5 5 5 3 4 5 5 3 5 3 4 4 3 3 3 4 3 1 5 5 1 3 3 5 3 3 2 4 5 5 4 4 4 3 5 5 3 4 3 5 5 5 5 3 3 4 4 5 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1
5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 5 5 3 5 4 5 5 5 5 5 3 3 3 5 5 5 5 5 5 5 4 5 5 5 5 5 3 4 5 5 4 5 3 5 5 5 5 4 5 5 5 5 5 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 3 1 1 1
5 5 3 3 5 5 5 5 5 5 5 5 5 5 5 5 4 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3 5 5 5 5 2 1 5 5 4 1 4 5 5 5 5 5 5 5 5 5 4 5 4 5 5 5 4 4 3 4 5 5 4 5 4 3 3 3 3 3 3 3 3 4 4 4 3 3 3 4 4 2 1 1
3 5 5 4 1 5 5 5 5 5 5 5 5 5 4 5 3 3 3 4 5 5 5 5 5 5 5 5 5 5 3 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3 2 5 4 3 5 4 4 2 3 3 3 3 3 4 4 4 3 3 3 3 3 3 1 2 1
5 5 3 3 5 5 5 5 5 4 5 5 4 4 5 5 4 4 4 5 5 5 5 5 5 3 5 5 5 5 4 5 5 4 5 5 5 5 5 5 5 5 5 5 5 4 4 5 5 5 4 4 5 3 1 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1
5 5 5 4 4 3 4 4 4 5 5 4 4 3 5 5 5 3 5 5 5 5 5 5 5 5 5 5 3 5 5 3 4 5 5 3 5 5 4 3 5 3 4 5 3 2 4 5 3 5 5 5 5 5 5 4 5 3 3 4 5 5 3 5 5 5 5 5 5 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 2 3 5
3 5 5 5 3 3 5 5 3 4 5 4 5 4 5 5 5 5 5 5 5 5 5 5 5 5 3 3 3 1 5 5 5 5 3 3 3 5 3 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 1 5 5 5 4 1 5 5 5 3 3 3 4 5 5 5 5 5 5 5 4 4 5 5 5 5 5 3 5 3 3 3 3 1 1 1
2 5 5 5 5 3 4 4 4 5 5 5 5 3 4 4 5 5 5 5 5 5 5 4 3 4 5 5 5 5 5 4 5 5 5 5 5 4 5 5 3 4 4 5 3 4 5 5 5 1 4 4 3 3 2 5 3 5 3 5 3 5 5 4 5 5 5 5 5 4 4 5 4 5 5 5 3 5 5 3 5 5 5 3 3 3 3 1 3 1
5 5 5 5 5 5 5 3 4 3 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 5 5 5 5 3 3 3 3 5 3 4 5 5 3 5 3 5 4 3 4 5 5 5 5 5 3 5 3 5 4 4 3 5 5 5 5 4 3 1 3 4 4 4 5 5 5 4 1 3 5 3 5 4 3 5 5 3 5 5 4
5 5 5 5 4 4 5 5 5 5 5 4 5 4 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 1 5 3 4 5 5 5 5 3 5 2 4 5 5 5 5 5 5 5 5 3 5 5 3 5 5 5 5 5 5 5 5 5 2 5 5 4 5 5 5 5 5 5 1 1 1 5 5 4 5 5 3 5 5 5 5 5 4
```

Figure 5. The block motion of frame N with respect to frame N+1

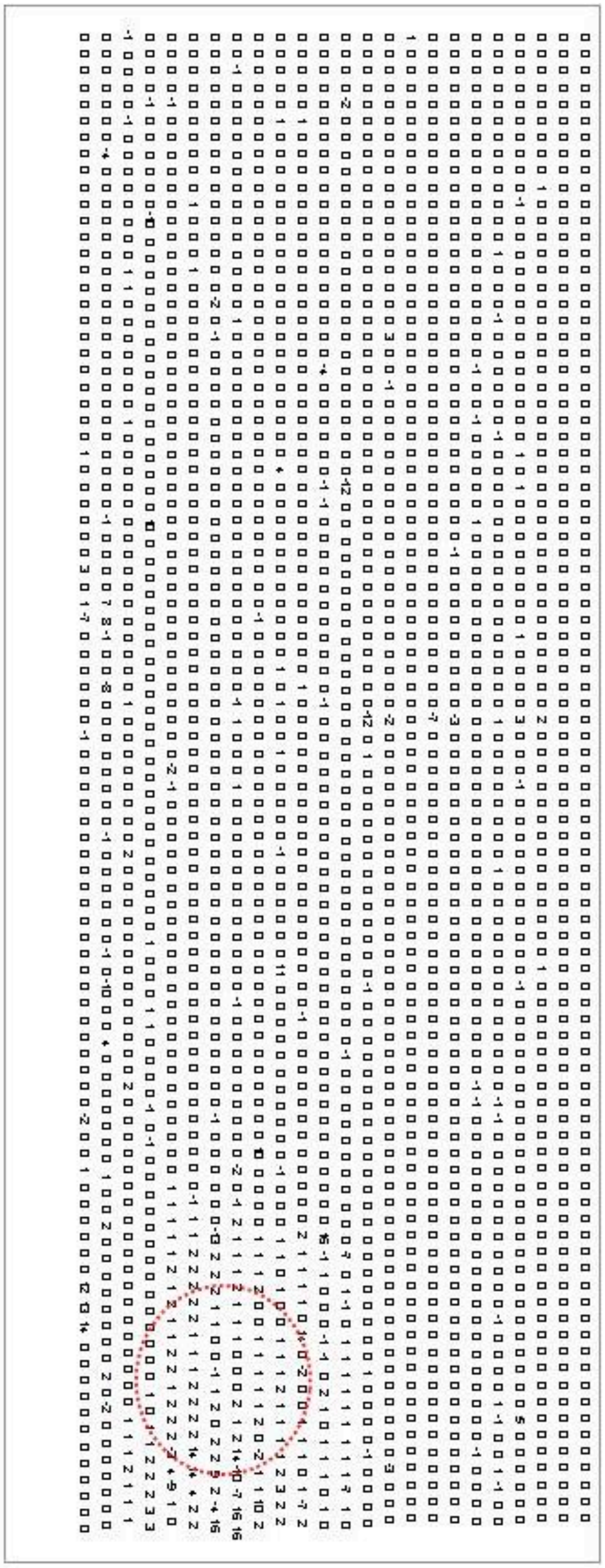


Figure 6. The block motion vector in y-direction

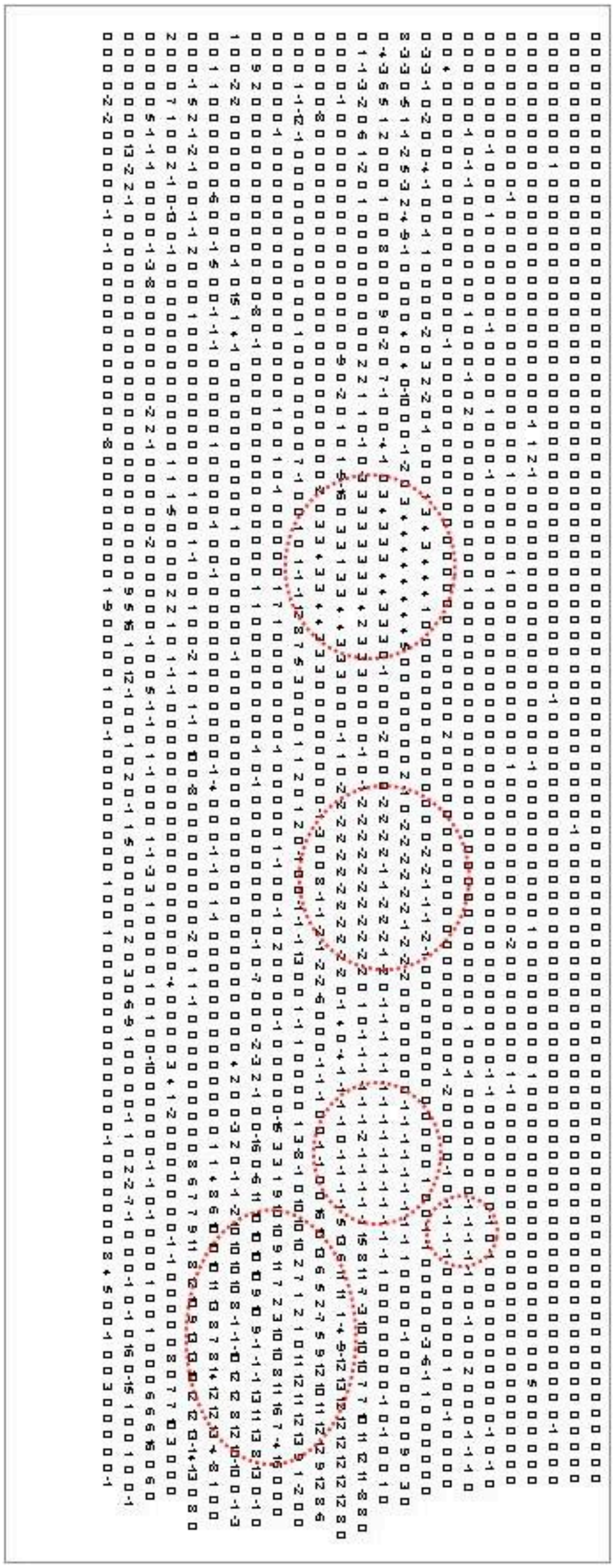


Figure 7. The block motion vector in x-direction

Chapter 3

Feasibility of Hardware Implementation for the Tracking Algorithm

In the previous Chapter, we describe the overall operation of the proposed tracking algorithm with block matching based motion detection. The algorithm is also simulated in MATLAB to verify its ability to track vehicles in spite of some noisy motion. The main motivation for such an algorithm is to enable hardware implementation of the tracking system since conventional tracking algorithm with graph optimization based motion detection may not be well suited to hardware implementation [1]. In spite of long execution time for the proposed tracking algorithm in MATLAB on a PC (the software approach), we hope that hardware implementation could significantly cut down the execution time so that the execution time for vehicle tracking would be smaller compared to conventional tracking algorithm implemented in software approach. Investigating the feasibility of design of a CMOS hardware processor for vehicle motion detection (corresponding to step one to step four of the proposed tracking algorithm) is the main topic of this Chapter. Next, we present first order timing analysis of the proposed tracking algorithm with hardware implementation to see whether it has the potential to achieve better real time operation than the traditional tracking algorithm with software implementation.

3.1 First order estimation of computational time for hardware implementation

Referring to the tracking system diagram in Figure 1, the first hardware part is the image sensor or digital camera. But it has to be high-throughput to minimize the latency when outputting data (the data is pixel values of the image). For example, a commercial image sensor, such as the MT9V111 sensor from Micron Technology [21], runs at a high frame rate of 60 frames/second with resolution of 352×288 pixels. The master clock is 27 MHz and the output is 8-bit parallel. Assuming 8-bit is used to represent each pixel, it takes 3.75 milliseconds to transport all the data to block matching in the proposed tracking algorithm. Note that each cycle of vehicle tracking has to finish within 1/60=16.67 milliseconds. Considering that the software part in Figure 1 (corresponding to step 5 of the proposed tracking algorithm) takes up to 10 milliseconds for vehicle association and Kalman filtering, this leaves only 16.67-10-3.75=2.9 milliseconds for motion detection based block matching. In conventional tracking algorithm [1], the motion detection based on graph optimization may not be able to meet the tight timing constraint. As a result, it is not possible to achieve high frame rate.

Therefore, we need to estimate whether the block matching computation in the proposed tracking algorithm can finish within such a timing frame. Note that the most important function in block matching algorithm is subtraction operation (refer to the definition of MAD criterion for block matching in last Chapter), which needs to be implemented by a high-speed hardware adder block (subtraction is implemented by the adder in hardware, the same as addition operation). For example, refer to Figure 2, if the block size is $n \times n$, then $n \times n$ additions are needed for one candidate motion vector (*u, v*). As mentioned in last Chapter, in the search region (defined by parameter p), there are a total of $(2p+1)\times(2p+1)$ candidate motion vectors to be matched (this is full search block matching), thus theoretically this means that there are $n\times n\times(2p+1)\times(2p+1)$ additions for one block matching. Then considering there are total of $(M\times S)/(n\times n)$ blocks in an image, this simply implies that the total number of additions in block matching is as follows:

$TC = n\times n\times(2p+1)\times(2p+1)\times(M\times S)/(n\times n)$

Then assuming that each addition in hardware takes time *td* to finish, a rough estimate of the total computational time would be

$TT = TC\times td$

So, then it comes down to find out *td*. To do that, we implement in Cadence [13] an adder circuit using TSMC 0.25μm technology [22]. This adder employs carry look-ahead technique [24], which typically has the smallest execution time compared to many other techniques [12]. When simulated in Cadence, the estimated execution time was about 2 nanoseconds considering also the time incurred when storing the output into a memory. Substitute this value into the above equation, we can find out that the total computational time for block matching is: (image size M=352, S=288, block size n=8, search region p=8):

$TT = 29297664\times2$ nanoseconds = 58.6 milliseconds

Although the adder execution time was pessimistic and could be made smaller, the above TT shows that using the simple hardware implementation was not enough since 58.6 milliseconds much larger than the timing constraint of 2.9 milliseconds for overall block matching computation.

Note that the previous method uses sequential computation of adder function. That is, before one candidate block matching is complete, the second one can not start. This is a waste of time, but save hardware. For example, the above sequential computation employs only one adder. Typically, to improve hardware throughput, pipelining technique can be used at the expense of extra hardware design. One possible pipelining technique in the context of block matching without incurring much extra hardware design is called the systolic array processing technique, which allows parallel processing of block matching [23]. In this approach, we would use $n\times n$ adder circuits for block matching computation (compared to only one adder for the previous case). It can be shown that using this parallel architecture, the number of additions for one block matching decrease to $(n+2p)\times(n+2p)$ from the previous $n\times n\times(2p+1)\times(2p+1)$ for sequential computation [23]. Then, the total execution time for block matching in this case would be (M=352, S=288, n=8, p=8):

$TTp = (n+2p)\times(n+2p)\times(M\times S)/(n\times n)\times td = 912384\times2$ nanoseconds = 1.8 milliseconds

Thus, it can be seen that now the execution time for block matching would meet the requirement of 2.9 milliseconds in order to achieve 60 frames/second vehicle tracking. In spite of only a first order estimation (we neglected many control overheads in hardware design), it shows that hardware implementation could potentially achieve real time operation of vehicle tracking. Also, if more advanced technology, such as 0.18µm, 0.13µm etc, is used, smaller execution time is expected.

Note that in the previous estimation, we assume that M=352, S=288, n=8 and p=8. Parameter n and p are quite reasonable under normal conditions. But it is arguable that M and S are enough for high resolution tracking. If both M and S are increased, the computational time is not satisfying. In that case, on solution is to further take advantage of massive parallel processing techniques at the expense of hardware cost and design complexity. For example, more adder circuits can be included in the hardware. Note that the number of adders in the above case of parallel processing is n×n. If we had *X* times that number of adders, we can partition the original image into X equal parts so that each group of adders (n×n) works on one part. This would help cut down the execution time for block matching to 1/*Xth* of TTp in the previous equation. In this way, block motion vectors can be very efficiently computed.

3.2 High-level view of hardware design for the tracking algorithm

In terms of hardware design, two possible techniques, digital and analog, can be used. Between them, digital technique for hardware design has been by far the most popular. In fact, there have been some digital VLSI implementations of the block matching algorithm for motion detection [16][17][23] for different applications such as low resolution video conferencing with 15 frames/second. On the other hand, analog technique for such type of hardware design is rarely used, not because of the performance issue, but mainly because of the design effort issue. Typically, analog technique for hardware design is much harder and more time-consuming. Next, we briefly discuss both techniques for hardware implementation of the proposed tracking algorithm.

For digital technique for hardware implementation, the signal processing flow is more straightforward and it can save Analog-to-Digital or Digital-to-Analog converters since the signal is always digital. Referring to Figure 1, the image sensor or digital camera can be commercial ones. Then in the second step of Gaussian filtering, standard adders and multipliers can be used. The detailed hardware design for block matching computation should take the systolic array architecture proposed in [23]. The output will be motion vectors in digital format, which are then transmitted to software part for vehicle association and Kalman filtering.

Analog technique for hardware design is an alternative. In fact, it typically saves more computational time than digital technique [18]. In this case, an image sensor may have to be designed instead of a commercial one. We favor the use of a built-in image sensor since an integrated image sensor could help shorten the image processing time. In the case of image sensor, output signal is analog current or voltage, the value of which is proportional to the pixel values (or the light intensity). The image can be a high-resolution one (for instance, 1024×1024) due to computational time saving compared to a low-resolution (256×256) image in traditional software approaches [1][2][3]. Then each image frame goes through Gaussian filtering to eliminate noise, which can also be efficiently performed using analog techniques (analog multipliers and analog adders) [18]. The resulting frame N can be stored temporarily in an analog memory. In this scenario, an effective analog memory could be arrays of capacitors to hold the voltage corresponding to pixel values of the image. Special techniques may be required to prevent voltage loss or leakage that affects accuracy of block matching computation [18]. When frame N+1 is available, the block matching module is used to detect vehicle motion for the two consecutive images. Again, parallel processing technique is probably necessary to minimize the execution time. Finally, the motion vectors obtained from block matching need to be converted to digital signal before they are transmitted to the software part. Thus, an Analog-to-Digital converter circuit is needed.

After receiving the data of block motion vectors from the hardware part, the software part performs vehicle association and tracking. Since these data are motion vectors of all blocks, any block with zero motion vector is either background or non-moving vehicles, and on the other hand any block with non-zero motion vector is a moving object or part of a moving object. Especially, we are interested in a number of neighboring blocks in image frame N that move to new positions but still form neighboring blocks in image frame N+1, which implies a moving vehicle. To check whether they still form neighbors can be done by simply looking into the corresponding motion vectors. The number of blocks involved for a certain vehicle depends on several factors, such as vehicle's size, vehicle's moving direction and camera's position. But in all cases, it is most likely that the vehicle in the image frame consists of more than one block in realistic images as those shown in Figure 3 to 7. Under ideal cases with no vehicle shadowing, blocking and noise, these motion vectors themselves track how and where each vehicle moves and Kalman filtering is not needed. But realistically, noisy or false motion, blocking and shadowing occurs frequently. So, Kalman filtering is still needed in the software part so that future positions of the blocks can be predicted. This is especially helpful in the case of vehicle blocking.

Chapter 4

Conclusion

This seed project studies the feasibility of developing a hardware based tracking system. We first survey the literature work on vehicle tracking algorithm and find conventional tracking algorithms are not well suited to hardware implementation. Developing a hardware based tacking system requires a revised tracking algorithm that could ease hardware implementation. Thus, we propose a new tracking algorithm with motion detection based on block matching. To validate such a tracking algorithm, we simulate it in MATLAB and test it on real traffic images obtained from an intersection. It is found that vehicle motion can be accurately detected in spite of some noisy motion for background and vehicles on the edge of the image. This error can typically be corrected at the software part.

With the proposed tracking algorithm validated, we study the feasibility of hardware implementation of the tracking algorithm. A first order analysis is applied to estimate the total computational time for block matching for a typical image size of 352×288 as specified by commercial high frame image sensors (60 frames/second). Assuming the block size is 8×8 and p is 8 for the search region, we find that the timing constraint to achieve 60 frames/second can be satisfied using parallel processing techniques in hardware design.

Overall, the proposed tracking system with hardware implementation shows promising features compared to conventional tracking system with software implementation. Future work along this direction includes actual hardware implementation of the tracking algorithm, detailed and accurate characterization of computational time, and development of the software part so that a prototype of the overall tracking system can be tested.

Two potential problems need further investigation. First, if the resolution of the image is further increased but high frame rate (e.g. 60 frames/second) needs to be maintained, hardware implementation with massive parallel processing techniques may be over-complex. Second, when the computational time of the tracking system is dominated by the software part, then implementing hardware for block matching computation alone may not be rewarding. In that case, moving (some) operations in the software part to hardware may be considered.

Bibliography

[1] O. Masoud, N. P. Papanikolopoulos, "A Novel Method for Tracking and Counting Pedestrians in Real-Time Using a Single Camera", *IEEE Transactions on Vehicular Technology*, Vol. 50, No. 5, Sep. 2001, pp. 1267-1278.

[2] O. Masoud, N. P. Papanikolopoulos, " The Use of Computer Vision in Monitoring Weaving Sections", *IEEE Transactions on Intelligent Transportation Systems,* Vol. 2, No. 1, Mar. 2001, pp. 18-25.

[3] S. Gupte, O. Masoud, R. F. K. Martin, N. P. Papanikolopoulos, "Detection and Classification of Vehicles", *IEEE Transactions on Intelligent Transportation Systems,*Vol. 3, No. 1, Mar. 2002, pp. 37-47.

[4] H. Veeraraghavan, O. Masoud, N. P. Papanikolopoulos, "Computer Vision Algorithms for Intersection Monitoring", *IEEE Transactions on Intelligent Transportation Systems,* Vol. 4, No. 2, Jun. 2003, pp. 78-89.

[5] S. S. Blackman, "Multiple-Target Tracking with Radar Applications", *Chapter 2, Artech House Inc,* 1986.

[6] Y. Wu, F. Lian, C. Huang, T. Chang, "Traffic Monitoring and Vehicle Tracking using Roadside Camera", *Proc. of 2006 IEEE International Conference on Systems, Man and Cybernetics.*

[7] Z. Jia, A. Balasuriya, "Vision Based Target Tracking for Autonomous Vehicles Navigation: a Brief Survey", *Proc. of 2006 IEEE International Conference on Systems, Man and Cybernetics.*

[8] N. Yang, G. Jianhong, " A 256×256 Pixel Smart CMOS Image Sensor for Line Based Stereo Vision Applications", *IEEE Journal of Solid State Circuits,* Vol. 50, No. 3, Aug. 2004, pp. 1055-1061.

[9] J. Nakamura, B. Pain, T. Nomoto, T. Nakamura, E. R. Fossum, "On-focal-plan Signal Processing for Current-mode Active Pixel Sensors", *IEEE Transactions on Electron Devices,* Vol. 44, No. 10, Oct. 1997, pp. 1747-1758.

[10] A. Fish, O. Yadid-Pecht, " CMOS Current/Voltages Mode Winner-Take-All Circuit with Spatial Filtering", *IEEE International Symposium on Circuits and Systems,* Vol. 3, May 2001, pp. 636-639.

[11] P. Hsiao, H. Cheng, S. Huang, L. Fu, "A Mixed-Signal CMOS Imager with Lane Detector for Use in Smart Vehicles", *Proc. of 2006 IEEE International Conference on Systems, Man and Cybernetics.*

[12] J. Rabaey, A. Chandrakasan, B. Nikolic, "Digital Integrated Circuits: A Design Perspective", *2nd Edition, Chapter 8, Prentice Hall, 2003.*

[13] "Cadence User Guide", *Cadence Design System Inc*, 2005.

[14] A. M. Tekalp, "Digital Video Processing", *Prentice Hall*, 1995.

[15] "SIMLINK and MATLAB Users Guides", *The Mathworks Inc*., 2006.

[16] Y. Huang, S. Chien, B. Hsien, L. Chen, "Global Elimination Algorithm and Architecture Design for Fast Block Matching Motion Estimation", *IEEE Transactions on Circuits and Systems for Video Technology,* Vol. 14, No. 6, Jun 2004.

[17] H. Yeo, Y. Hu, "A Modular High-Throughput Architecture for Logarithmic Search Block Matching Motion Estimation", *IEEE Transactions on Circuits and Systems for Video Technology,* Vol. 8, No. 3, Jun 1998.

[18] B. Razavi, "Design of Analog CMOS Integrated Circuits", *McGraw Hill*, 2001.

[19] "Using Matlab Version 7", *The Mathworks Inc*, 2006.

[20] "Quick Time Pro", *Apple Inc*, 20006.

[21] "MT9V111 Product Flyer", *Micron Inc*, 2006.

[22] "TSMC 0.25µm process user manual", *Taiwan Semiconductor Manufacturing Company*, 2002.

[23] C. H. Hsieh, T. P. Lin, "VLSI Architecture for Block-Matching Motion Estimation Algorithm", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 2, No. 2, Jun 1992, pp. 169-175.

[24] S. Rowan, P. Bushey, "Project report: Carry Look-ahead Adder Design in TSMC 0.25 micron technology", *ECE Department, University of Minnesota Duluth*, 2006.