Parallelizing an Algorithm for Object Recognition based on Edge Detection on Sun Fire

Mohd. Azam Osman, Maziani Sabudin, Muhammad Ismail Ibrahim
School of Computer Sciences
Universiti Sains Malaysia
11800 Penang, Malaysia

Abstract — Object Recognition is one of the essential parts for image processing. It has become the major areas for the technologies like Biometrics, image recognition, authentication and accessibility of major security system. There are lots of techniques such Neural Network that can be utilized for recognition, but each of technique comes with complication and drawbacks. Edge Detection technique has shown a better improvement and advantages for recognizing an image. The objective of this paper is to improve and implement the algorithm for object recognition based on edge detection. This paper will discuss on how we parallelizing the algorithm on SUN Fire computer (hybrid distributed memory architecture) and measure the processing time between sequential and parallel algorithms. The details of the design and framework used for parallelization of the algorithm called Canny algorithm such as three levels of architectures, pseudo code and method of parallelization which uses Java MPI will be discussing in this paper. Finally this paper will present the implementation and results.

Keywords: image processing and recognition, edge detection, biometrics.

1.0 Introduction

Object Recognition can be defined in several ways. Some interpret object recognition as *the visual perception of familiar objects* (**WordNet Dictionary**), and some says *detecting the presence and/or pose of known objects in an image* (**Wikipedia – The free Encyclopaedia**) while others just simply relate them to *Image Processing*. Because of object is a tangible and visible entity which can cast a shadow, Object Recognition is the way for us to identify and analyze this entity so that we can recognize it as a whole entity.

1.1 Detecting Edge

Edges may be seen as viewpoint dependent; these are edges that may change as the viewpoint changes, and typically reflect the geometry of the scene, objects occluding one another and so on, or may be viewpoint independent and these generally reflect properties of the viewed objects such as markings and surface shape. In two dimensions, and higher, the concept of a projection has to be considered.

A typical edge might be (for instance) the border between a block of red colour and a block of yellow, in contrast a line can be a small number of pixels of a different colour on an otherwise unchanging background. There will be one edge on each side of the line. Edges play a quite important role in all applications of image processing.

1.2 Type of Edge Detection

There are several algorithms that has been researched and developed to be used as the edge detector algorithm. While some are practically in research, many others have been successfully implemented and running on different type of platforms (UNIX, Mac, and Windows). Though most of these algorithms are still sequentially based algorithms, some have made room for improvement in parallel computation.

Below is the list of edge detection algorithms sorted in order of foundation year.

1st Order Generation
1. Roberts Cross (1965)
2. Prewitt (1970)
3. Sobel (1970)
4. Canny (1983)
5. Spacek (1986)

2nd Order Generation
1. Laplacian (1979 - 1987)
2. Marr-Hildreth (1980)

The most significant operator and commonly used are the Canny operator and then followed by Marr-Hildreth. Sobel is also one of the most commonly used Edge Algorithm operators.

Roberts Cross was the first algorithm technique used and is thus the fastest and simplest to implement. Sobel edge detection algorithm is similar to the Roberts Cross approach in that both use two kernels to convolve an image, where the second kernel is simply a 90 degree rotation of the first kernel [1]. Canny edge detection algorithm uses several steps to detect edges in an image. However, this algorithm processing time is slower due to its higher degree of complexity.

Detail of the few mention edge detection algorithms [2] are explain as below.

1. **Robert Cross algorithm**

The Roberts Cross algorithm performs a two dimensional spatial gradient density on the image. The main idea was to bring out the horizontal and vertical edges individually and then to put them together for the resulting edge detection.

2. **Sobel algorithm**

The Sobel edge detection algorithm technique is very similar to that of the Roberts Cross algorithm. Both of Robert Cross and Sobel ideas were the use of two kernels to determine edges running in different directions. Both of the algorithms are then also combine the results in a similar manner to obtain a full edge detection image. Once major difference between them is the kernels used to obtain these initial images. The Sobel kernels are more suitable to detect edges along the horizontal axis and vertical axis while the Roberts Cross kernels are designed to detect edges that run along the vertical axis of 45 degrees and the axis of 135 degrees.

3. **Canny algorithm**

The Canny edge detection algorithm is much denser than the others. Due to its complexity, it also takes notably longer to process the results. The first step is the image needed to be run through a Gaussian blurs to remove majority of the image noise. For the next step, an edge detection algorithm is applied such as the Roberts Cross or Sobel techniques. From there the angle and magnitude can be obtained and used to determine which portions of the edges should be concealed and which should be taken out. Finally, there are two threshold cut-off points. Any value in the image below the first threshold is dropped to zero. Any value above the second threshold is raised to one. For any pixels whose values fall within the two thresholds, their values are set to zero or one, based on their adjacent pixels and their angle [3].

1.3 Object Recognition

Currently, object recognition is one of the most popular developments at the moment. 3-D object recognition is still largely limited to blocks world scenes. They can only fully identify simple, largely polyhedral objects, while more complicated objects can only be tentatively recognized.

Roberts [3] was the founder of three dimensional model-based scene understanding. Using edge detection methods, he analyzed intensity images of blocks world scenes containing rectangular solids, wedges and prisms. Object scale and distance were resolved by assuming the object rested on a ground plane or on other objects. Recognition of one part of a configuration introduces new edges to help segment and recognize the rest of the configuration.

The TINA vision system, built by the Sheffield University Artificial Intelligence Vision Research Unit [5], was a working stereo-based three dimensional object recognition and location system. Scene data was acquired in three stages: (1) *subpixel* "Canny" detected edges were found for a binocular stereo image pair, (2) these were combined using *epipolar*, contrast gradient and disparity gradient constraints and (3) the three dimensional edge points were grouped to form straight lines and circular arcs. When a maximal matching was obtained, a reference frame was estimated, and then improved by exploiting object geometry constraints (e.g. that certain lines must be parallel or perpendicular).

1.4 Parallelization of Algorithm

Although there are many algorithms on edge detection existed, not all of them are consistent or have a high-speed computation. For example, the first edge detection algorithm, *Robert Cross algorithm*, though it is the fastest running algorithm but the result is not the best because it performs a two dimensional spatial gradient density on the image. Though canny algorithm is commonly used and performs better than the rest, the complexity and density of its algorithm causing it to run slower.

The method of either distribution or parallelism is required to enhance the workable and precise algorithm and while enabling them to be run on other machine architectures and also reducing the processing time by even greater.

1.5 Objective

The main objective of this research is to provide the improvement towards the edge detection algorithm by implementing parallel algorithm techniques into the canny edge algorithm and also running the enhanced algorithm on a different architecture that may not yet been tested by other which is using Sun Fire machines. For this, the development of parallelization of an algorithm would improve speeding up the processing time taken by sequential algorithm.

2.0 Related Work

The aims for doing the research are to find, enhance and implement the modification of object recognition using edge detection algorithm. By doing so, we need to identify which part of the algorithm needed to be adjusted, what were the previous researchers have done so far, what do we need to make improvements and what are the possibilities that could be added into it. Also, we will be stating the advantages and disadvantages of some of the algorithms.

2.1 Image recognition Approach

There are quite few approaches available when doing Image recognition. One of these approaches is getting the edge by using edge detection. It is by far one of the easiest and unambiguous methods to be created. Therefore, a lot of these algorithms have been researched and developed but only few of them have been enhance into the multiple computing solution.

2.2 Chosen Edge Detector Algorithm

The factors for choosing the algorithm are good results, useable and widely used while the main factor is providing the solution to the flaw existed in it. In this research, the Canny Edge Detection algorithm has been chosen. Canny algorithm currently produces one of the best processing results though it has one of the highest densities in the algorithm. Because of that, it has small flaw in sense that the processing time to produce the output is quite long.

The Canny edge detection algorithm is much complex than the others. Due to its complexity, it also takes notably longer to processes the results. As we have mention before, the step taken for producing the output takes lots of the computer resources and thus makes it slower than the others [6].

2.3 Parallelization of the chosen Algorithm

One of the main characteristic of a parallel algorithm is Data Dependencies. Other advantages on using the parallelism in an algorithm is when there are parts of the algorithm that have a code that are running in looping manner, meaning that there are looping condition (e.g.: for{} loops) in the algorithm as long as these loops does not have any dependencies data running within it.

Table 1 below shows some of the edge algorithm in different implementation between sequentially built algorithms and parallel ones.

Algorithms	Sequential	Parallel
1st Generation		
• *Robert Cross*	*v*	
• *Prewitt*	*v*	
• *Sobel*	*v*	
• *Canny*	*v*	*v*
• *Spacek*	*v*	
2nd Generation		
• *Laplacian*	*v*	
• *Marr-Hildreth*	*v*	

Table 1 Edge Detection Algorithm development

3.0 Proposed Design Methodology/Framework

In this section, we will be discussing the design and framework used for parallelization of the algorithm based on the type of image input, data flow based on the machine architecture and some possible pseudo code for the implementation.

We also will be including the original scheme of the canny algorithm as we will be testing and comparing both of the sequential and parallel methods and with that we will be assessing the computational time and the possible speed up.

3.1 Proposed solution

Below are the proposed steps for the Parallel Edge Detection application.

Client-side (initialization)

- *Initialization*
- *VNC link communication*

To start the program which is located on the Sun Fire machine, user will need to input certain command to be successfully connected to the server. The best way is to start a SSH session with the server using SSH Secure Shell Client and type the command to initialize the VNC (Virtual Network Computing) communication server. From there, users need to log to the Sun Fire using VNC Client Viewer for application displaying purpose.

Server-side (processing)

- *image information and edge extraction*
- *sequential and parallel edge processing*

User will start by activating the application using Java command for the GUI. Then, user will need to select the intended image (from any type of images such as JPEG, BMP, PNG or even GIF format). Image information extractions such as image properties are then displayed on the intended panel.

Then, the master server will start by running the sequential algorithm and then pass the parallel algorithm to the server nodes using MPI by pressing on the respective command button. There will always be a sequential algorithm in this project because not all of the code provided is available for parallelism technique. After each node completes the execution of the given task, each output assesses from each node will be regroup and revert to the master server.

After completing executing each available code, the server and its node will return the resulted edge image with other information required.

Client-side (results)

- *edge results*
- *time difference & speedup*

After receiving edge image from the server, the program will show the results and also the computation time needed for the edge algorithm to run. Other available functions in the client-side are computation time percentages and difference and speedup calculation.

3.2 Method of parallelization

For an algorithm to achieve parallelisation, we must invoke few steps so that each processor intended for running the algorithm in parallel forms are able to have an equal load or so.

Firstly, we must have a processor (in this case, the server itself) to control the load balance between the other processors and itself. Because the loaded image is sorted into an array which will be processed by a single processor, we are able to use the opportunity to alter the algorithm as shown as the steps below:

1. When the algorithm loads the image into array by getting the size, we will then divide them by groups.
```
picsize = width * height;
data = new int[picsize];

picsize1 = picsize / (edge_proc - 1);
picsize2 = -1;

for (int i = 1; i < edge_proc; i ++) {
  size[0] = picsize2 + 1;
  picsize2 = picsize1 * i;
  size[1] = picsize2;
}
```

2. Then, the server node will transmit a blocking send to all of other nodes as they (the child nodes) will be receiving the parts of the array pointer information.
```
   if (edge_rank == 0) {
     for (int iProc = 1; iProc <
edge_proc; iProc++) {
        MPI.COMM_WORLD.Send(size, 0, 4,
MPI.INT, iProc, 99);
     }
   } else {
     s = MPI.COMM_WORLD.Recv(size, 0, 4,
MPI.INT, 0, 99);
   }
```

3. When running parts of the algorithm which used the array, each processor will only process the array by its own pointer, leaving the other parts of the array untouched.
```
   for (int i = size[0]; i < size[1]; i
++) {
     if (data[i] > threshold)
       data[i] = 0xff000000;
     else
       data[i] = -1;
   }
```

4. Finally, when reaching the final stage of the edge algorithm process, all of the processors will send the processed array including their own pointer.
```
   if (edge_rank != 0) {
     MPI.COMM_WORLD.Send(data, 0, 4,
MPI.INT, 0, 99);
   } else {
     for (int iProc = 1; iProc <
edge_proc; iProc ++) {
       s = MPI.COMM_WORLD.Recv(dataA, 0,
4, MPI.INT, iProc, 99);
       for (int j = dataA[0]; j <
dataA[1]; j ++) {
         data[j] = dataA[2];
       }
   }
```

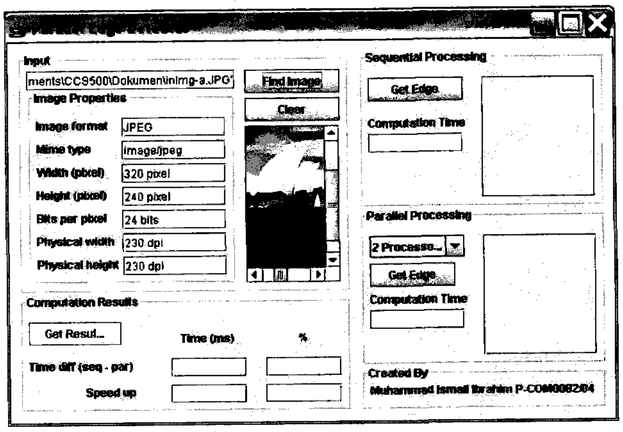
Implementation & Results

Here, we will be using a java application which will be used throughout the entire sequence of the program. Although the look is compact, this interface will accommodate all the input and output needed.

The input images snapshots that will be used to get the edges are all shown as below.

Figure 1 Image used with different image type (BMP, GIF, JPEG, and PNG)

Image with different mime formats (BMP, GIF, JPEG, PNG)



Input: ments\CC9500\Dokumen\in\img-a.JPG

Image Properties

Image format: JPEG
Mime type: image/jpeg
Width (pixel): 320 pixel
Height (pixel): 240 pixel
Bits per pixel: 24 bits
Physical width: 230 dpi
Physical height: 230 dpi

Sequential Processing
Computation Time

Parallel Processing: 2 Processo...
Computation Time

Computation Results
Time diff (seq - par)
Speed up

Created By
Muhammad Ismail Ibrahim P-COM0082/04

Figure 2 High Quality Image

In this section, we will show the steps that must be taken to be able to run the java application which is stored in the Sun Fire (Stealth) server and its nodes. The applications that we need to run on our own machine are:

- SSH Secure Shell Client, and
- RealVNC Client viewer.

Also required is the login account for the Stealth server so that we will be able to access and run the application which is stored within it.

Running the SSH:

1. Open the SSH Secure Shell Client.
2. Click on the Quick Connect.
3. On the Host Name field, type *stealth.cs.usm.my* (or *10.207.207.48* when using local area network, LAN). Also, key-in our login account in the User Name field.
4. When ask for password, type the password.
5. To start the VNC connection from the client, type the below command:
   - *vncpasswd*
     - ❖ This command is to set the password for the VNC server that will be used later on. This is for security reasons and it can be either set many times or once depends on the user.
     - ❖ When ask for password, insert the password that should contain at least 6 digits.

   vncserver.orig –geometry 1024x768 –depth 24
     - ❖ The geometry resolution can be comprised from either 640x480, 800x600, 1024x768 or other which depends on the client and the server maximum resolution).
     - ❖ The possible depths are between 8, 16 and 24 only.

   New 'X' desktop is stealth: 5
     - ❖ The server accessible port will be shown as above.
6. From here, we should now run the VNC client viewer. Only proceed to step 7 either when we want to disconnect from the stealth VNC server.
7. If we want to exit the server, type the below command:

   vncserver.orig – kill :#
     - ❖ # is the number of the server that we get from the previous VNC command (in this example is number 5).
8. It is compulsory for user to disconnect from SSH when they do not need to use them.

Running the VNC viewer:

1. On the Server field, insert the Stealth address including the server port open from the SSH command (e.g.: *stealth.cs.usm.my:5* or *10.207.207.48:5* when using LAN).
2. Before preceding the connection, these options must be check to ensure that the VNC is successfully loaded.

   Only use protocol 3.3 (this will ensure successful connection)

   Render cursor locally (this will help to move the mouse freely outside frame)
3. Because we used security measure, the password acquired from *step 5* in SSH command will be used.
4. Here, we will have the access control of the Stealth server, we can either load the program with either the command prompt or just click on the correct icon. When using the command prompt, type the below command on the correct folder:

   java edgeapp/EdgeJFrame
5. The Parallel Edge application will be loaded for the user, continue to use it until the program exit by user command.

Results

Figures below are some of the results gathered from the Sequential versus Parallel Edge algorithms.

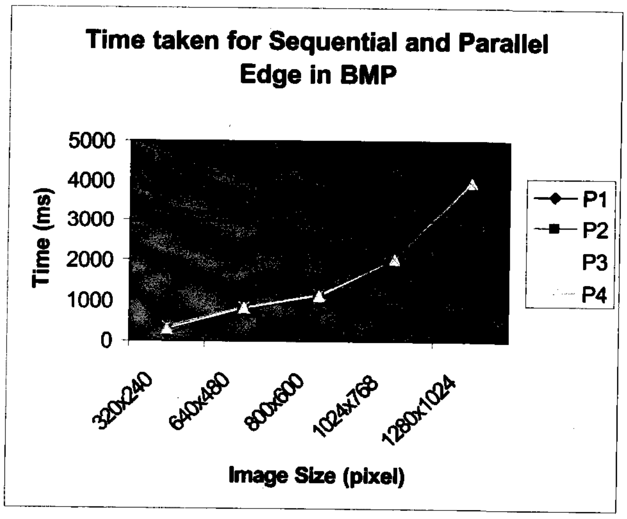


Figure 3 Processing time graph obtain from the same type of images (bmp)

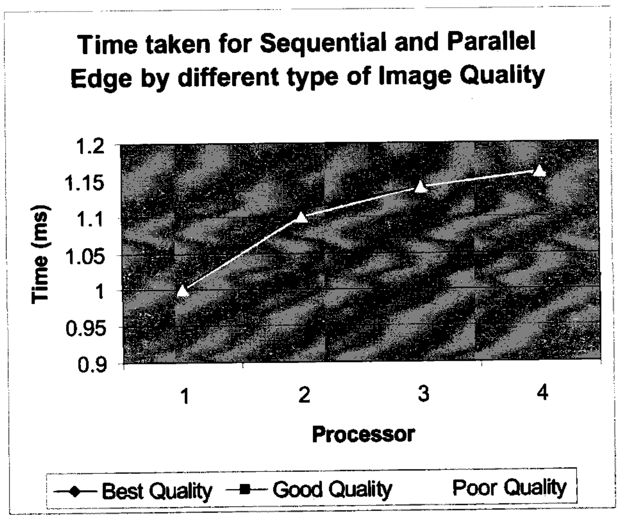


Figure 4 Speed up graph obtain from the different quality of images

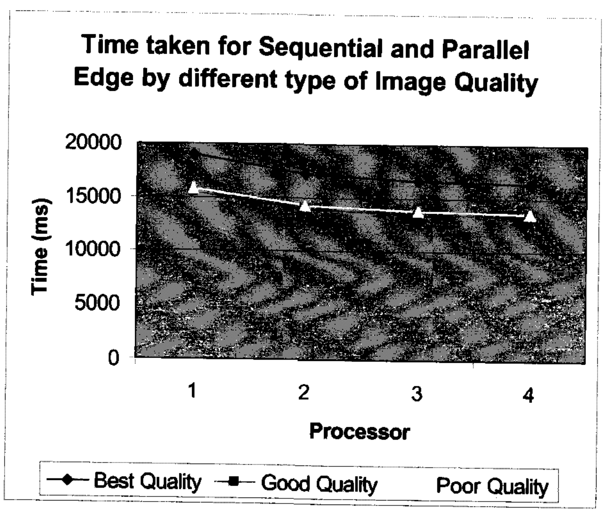


Figure 5 Processing time graph obtain from the different quality of images

Discussions

From the results, there are several factors that influence the processing time for each image. These factors are:

1. **Types of image.**
   From the results gathered, images with image format GIF (Graphics Interchange Format) has the fastest processing time between the other popular image formats (i.e.: JPEG, BMP and also PNG). This is because GIF image are ideal for images that have a lot of sharp edges and this is very essential when it comes to detecting edges in an image. Although JPEG and PNG images are equally slow because of the sharpness of images and the depth that they provide, but the edges that both of them provided are equally sharp and precise. BMP images are the rawest image available and do not have the sharpness and precision of the objects in the image thus their processing time are average.

2. **Qualities of image.**
   From the analysis that we have, a higher quality of image with high image sharpness will tend to have a slower processing time rather than an image with low quality and lots of image blurring. This is because the edge detection process requires the image to be blurred with a Gaussian Blurring before it is able to get the edges of the image. Even so, an image with normal attribute can be process with the fastest time.

3. **Computer idle state.**
   The most important factor when running the program is the state of the server and its nodes. When running an algorithm while the nodes are still not in idle mode will not only slow down the processing time, but may also cause the communication between the nodes and also the server to be delayed. Therefore, its better that we set an interval time between them to prevent any communication delay.

From the results and graphs that we have gathered, we can conclude that the factor of speed up makes from:

1. **Size of the picture.** (the bigger the better)
   The size image is one of the main factors in getting the speed up in parallel versus sequential processor. A smaller size of an image will not only waste the processor resource but does not help in increasing the time used by parallel machine. An image with larger pixel is suitable for this parallel task.

2. **Number of processors used.** (depends on size)
   Depending on the situation, a smaller number of processors are better used in getting the process done in faster time. But, naturally when it comes to a higher size of image or low quality image, number of processor used often help in reducing the time used and help increase the speed up.

But, there are some factors that slow down the processing time. These factors are:

1. **Images and Communication Overhead.**
   Using parallel processing on small images does not help in speeding up the process. This is because, when using parallel machine, there will be some or a lot of communications between servers and its node which makes up time. If an image is able to be processes by a single processor in much shorter time than when it is processes by a parallel machine, then it will be wasteful for us to used lots of processing power from all the nodes required. So, we should only use the parallel machine to process the large size image so that we can obtain the speed up rather than processing the smaller size image.

2. **Machine architecture.**
   Speed up may also be affected by the type of machine used in processing an algorithm. Not all machine architectures are suitable for parallelization and machine that used tight-coupled system is one of the examples of machine that may not produce required speed in processing the algorithm. For now, we have only tested in a loosely-coupled system and it is not confirm on how fast it is compared to tight-coupled system architecture.

5. Summary and Future Work

In this work, we presented an alternative approach in parallel Edge Detection algorithm using MPI technique in Sun cluster system with parallel architecture. In our project, we used both sequential and parallel algorithm technique to get the intended edges from same images. While processing the image, we also captured the time taken by both algorithm to compare and analyze the efficiency when using a single processor and also multiple parallel processor from the same system.

From the analysis that we had gathered, a small size image is better processed by a singular processor while a larger size image can be speed up by using multiple processors. A distorted or crisp image can also be processed by multiple processors but it must also in a larger size because the speed up obtains by using multiple processors are quite slow and will waste lots of machine resources.

The results of the parallel algorithm shows that the parallel processing could really help to speed up the sequence edge detection process. However, due to the downside of the communication overhead and increased number of message passes by the Sun Fire machines, its performance still shows that it's still far behind than expected.

We also provided other useful processes which are image information extraction and time calculation which could also help in comparing the results gathered from both sequential and parallel task.

Future Work

For the future, we can either re-enhance the parallel algorithm for a better speed up, or using this redeveloped edge detection algorithm to furthermore ongoing the real task in which getting the object recognition from the edges that we have obtained now. As from the results show, it seems that message passing has still not adequate to achieve the peak speed up that we had been estimated before. Therefore, another option for the future work is implementation of parallel socket programming using the current server which theoretically and practically more efficient from what the MPI programming could offer. Although there are lots of ongoing research and development in object recognition, we can speed up the current process of extraction the object time using this parallel edge detection algorithm.

REFERENCES

[1] Gonzalez, R. and Woods, R., "Digital Image Processing Second Addition", Prentice-Hall Inc., 2002, pg. 567-612.

[2] "Edge Detection Algorithm", http://www.ccs.neu.edu/home/mtrubs/html/EdgeDetection.html

[3] Roberts, L. G., "Machine Perception of Three-Dimensional Solids, Tippett, J. T. (ed.), Optical and Electro-Optical Information Processing", MIT Press, Ch. 9, Cambridge, Massachusetts, p159-197, 1965.

[4] Faugeras, O. D., Hebert, M., "A 3-D Recognition and Positioning Algorithm Using Geometric Matching between Primitive Surfaces", Proceedings 8th IJCAI, pp996-1002, 1983.

[5] Porrill, J., Pollard, J., Pridmore, T., Bowen, J. B., Mayhew, J. E. W., Frisby, J. P., "TINA: The Sheffield AIVRU Vision System", Proceedings 10th IJCAI, pp 1138-1145, 1987.

[6] Green, B., "Canny Edge Detection Tutorial", Drexel University: PRISM, http://www.pages.drexel.edu/~weg22/can_tut.html, 2003