Accelerating Occupancy Grid Map Computation for Real-Time Obstacle Detection

A Project Report

Submitted by

Sriramprasath
EDS14M012

in partial fulfillment for the award of thedegree of

Master of Design

in

Electronic Design

Indian Institute of Information Technology Design and Manufacturing, Kancheepuram, India

May 17, 2016

BONAFIDE CERTIFICATE

This is to certify that thethesis titled "**Accelerating Occupancy Grid Map Computation with GPU for Real-Time Obstacle Detection**"submitted by **Mr. Sriramprasath (EDS14M012)** to the Indian Institute of Information Technology Design and Manufacturing, Kancheepuram, for the award of **Master of Design in Electronic Design,** is a *bonafide* record of the project work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.Binsu J Kailath
Project Guide

Assistant Professor
Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127
India

Place: Chennai
Date:

I would like to dedicate this work to my parents and God, whose blessings and support has always been of immense support for progress in my life.

ACKNOWLEDGEMENTS

Firstly, I thank the Director of the Institute Prof. R. Gnanamoorthy for providing all infrastructural facilities required for my project as well as graduate study here.

I would like to express my sincere gratitude to my advisor Dr. Binsu J. Kailath for her continuous support during the course of my study and especially for her patience, motivation, and immense knowledge which helped me a lot for the completion of my project. Her guidance was very much helpful in preparing this report as well.

I also wish to express my sincere gratitude to my Co-guide Dr. Sudha Natrajan, Autonomous Vehicle Program - R&D, Tata Elxsi Limited, Chennai, for her support and guidance for this project.

I would like to thank research scholar Mr Dinesh G for his valuable suggestions regarding the usage of software tools. I also thank the technical staff in the Electronics labs for their timely help.

My sincere thanks also go to the stream coordinator Dr. P. Damodharan and all the members of faculty in Electronics Engineering for their valuable suggestions and comments during the project reviews.

Last but not the least, I thank all my friends for making my stay here a memorable one.

ABSTRACT

Obstacle detection and drivable area recognition is one of the most time-critical process in the motion planning of a self-driving vehicle. Computing Occupancy Grid Maps (OGM) from stereo image data is a promising solution because it is more cost-effective than active sensor based solutions. The drawback however is that it is computationally intensive and takes more processing time.

This work aims to accelerate the computation of OGM from stereo image streams by implementing parallel processing modules in the algorithm. This work also discusses the implementation of different disparity computation techniques in GPU and their impact on the performance of the overall algorithm. The objective is to improve the frame rate at which the stereo system is able to update the OGM while maintaining an acceptable level of accuracy.

A frame rate of 5.1 fps is achieved on a Nvidia Quadro K620 GPU for an image size of 1344x372.

Contents

Nomenclature

ADAS – Advanced Driver Assist Systems

GPU – Graphic Processing Unit

GPGPU – General Purpose GPU

FPGA – Field Programmable Gate Array

SoC – System On Chip

CAN – Control Area Network

OGM – Occupancy Grid Mapping

VGA – Video Graphics Adapter

SIMD – Single Instruction Multiple Data

CUDA – Compute Unified Device Architecture

NVCC – NVidia C Compiler

BM – Block Matching

BP – Belief Propagation

CSBP – Constant Space BP

Chapter 1 Introduction

The growing number of fatal road accidents has created awareness about the immediate need for intelligent vehicles. A 2013 GSMA report estimates that the global connected car market will hit \$2.7 trillion by 2018[1]. Government initiatives like Intelligent Transportation Systems (ITS) in India encourages development of connected vehicles and intelligent self driving cars by promising relevant infrastructure to accommodate such vehicles. Technologies such as Traction control system, Antilock Braking System (ABS), Electronic Stability Programme (ESP) can be seen as a precursor to the autonomous vehicle of the future. In this paper, we investigate a stereo camera based approach to create a virtual reconstruction of the environment namely an Occupancy Grid Map (OGM), which is essential for higher level algorithms trying to automate a vehicle.

1.1 Problem statement

The problem statement is to generate a dynamic Occupancy Grid Map from the stereo data. This process needs to be accelerated using GP-GPU to enable it to perform in real-time application.

OGM is a probabilistic model of the surroundings, which tries to record information like ground plane, obstacle position, object motion and ego motion (movement of the system itself) about a system and its surroundings. This OGM can be estimated by using sensor systems such as RADAR, LIDAR, Ultrasonic sensors which acquire the distance of an object from the vehicle. The stereo camera is a sensor system which uses two cameras to triangulate the 3D coordinates of an object within its field of view. The stereo camera system is a better and cost-effective solution to generate efficient OGM grids because of the visual nature of the system, which not only give us the position and dimensions of an object but also the same data can be used to identify the type of the object[2]. In this thesis, we investigate the GPU acceleration of various steps in estimating a dynamic OGM from a stereo camera sensor.

Chapter 2 Literature Review

2.1 Review of ADAS systems

In today's modern car market has become so safety aware that pretty much all car manufacturers are investing in active safety research. However a few noteworthy manufacturers lead this by implementing a precursor of these systems in all their cars. These manufacturers offer as an added package of these advanced safety systems, mentioned here are a few.

2.1.1 VOLVO 'Intellisafe'

Volvo, along with Mercedes- Benz is the one of the first manufacturer to identify the importance of Active safety systems by including the "Intellisafe" package which includes safety systems like blind spot detection system, lane departure warning system, pedestrian detection and active braking systems which are capable of avoiding accidents upto a speed of 50 kmph. The system uses Radar based obstacle identification. Volvo aims to achieve implementation of these safety systems in all standard Volvo models in offer by 2020.

2.1.2 Mercedes Benz Pre-crash safety systems

Mercedes Benz has one of the best in class pre-crash safety systems in cars as well as trucks. These systems which are named as distronic+, steering assist are capable of identifying impending crash scenarios and either brake in advance or even steer away from danger. These systems too, like the Volvo, use a millimetre radar system for obstacle detection.

2.2 Camera based Systems

Object identification and recognition is one of the key advantages of a camera based ADAS system. This in turn paves way for a future, more advanced and intelligent driver assist system or even a step towards the dream of a connected drive. There are two basic approaches to camera based sensor systems. The Monocular camera system which uses object recognition based on a trained program to identify objects it was created to recognize. The stereo camera system uses slightly different approach based on creating a depth map of the area of vision using 2 cameras which are placed in different positions with a distance ‘b’ (the baseline). Given both cameras have the same focal length ‘f’, and an object is recognized in both the cameras with a pixel disparity of ‘d’. Then the distance to the recognized object ‘Z’ is given by,

$$Z = f\frac{b}{d}$$

Once the disparity map is generated then the data can also be used in a way to identify the drivable area ahead of the vehicle. A combination of stereo and mono cameras is also a very promising area to focus[2].

2.3 ADAS development support

The rising need for safety systems has not only urged the automotive manufacturers into trying to develop active safety systems, but also encouraged the silicon manufacturers such as Texas to develop a chip optimised for ADAS development the following are the list of manufacturers offering products which help with prototyping an ADAS concept.

Table 2.1 List of ADAS products

Manufacturer	**Product**	**Target application**
Texas Instruments	TI TDAx chipset, and TDAxSoC Development board	Dedicated SoC optimized for vision processing in ADAS
NVIDIA	The Jetson TK1*, Drive PX	Embedded ADAS development
Freescale	Freescale ADAS development board	Basic ADAS applications
Renesas	Renesas ADAS starter Kit	Vision based ADAS development
Xilinx	Xilinx LogiADAK Zynq-7000 starter kit	Multicamera ADAS development in FPGA platform
Mobileye	Camera Development Kit	Vision based ADAS applications
Altera	Cyclone V SoC	ADAS development platform

The Jetson TK1 board is a viable option to developing a prototype for verification. This helps because the same code developed on any system with a NVIDIA CUDA compatible graphic card can be implemented in this board

Chapter 3   Occupancy Grid Map estimation

OGM estimation in a stereo system first involves obtaining a disparity map from a stereo image pair. This stereo disparity map represents the depth information collected by the camera. This disparity data can be then used to triangulate the depth of the objects and surfaces in view[3]. Then a U-V disparity map is used along with feature matching techniques to identify moving objects and stationary objects. The ground plane is also estimated using this technique; we will discuss this further in section 3.B. Then these results are used to arrive at a decisive result in the form of an Occupancy grid map which shows a map of the surroundings recognizing objects and roads/ traversable path. Fig. 1 shows the overall flowchart of generating a dynamic OGM from two sets of stereo image pairs.

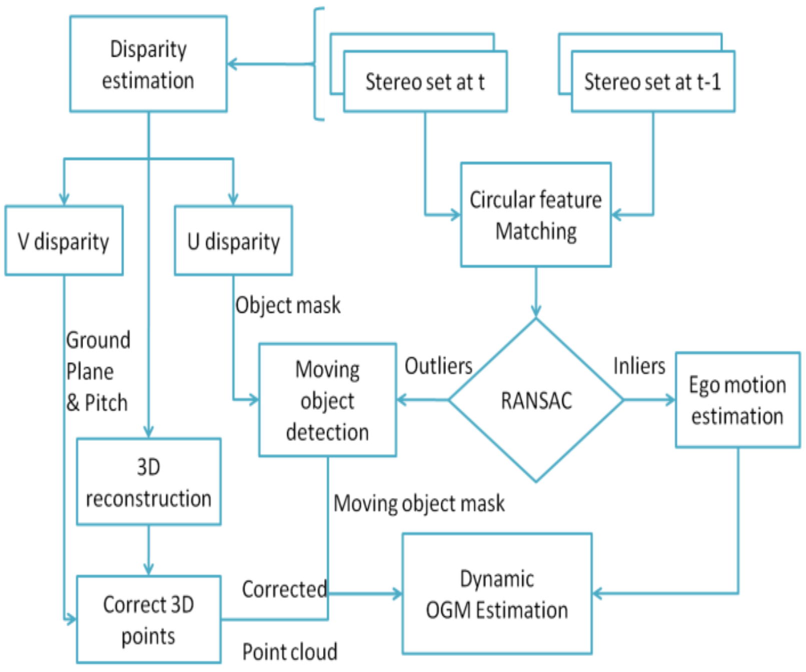

Fig 1. Process flow for Dynamic OGM estimation

3.1 System requirements

The following are the fundamental requirements placed on the system.

1. The system should identify drivable section ahead from the non-drivable sections. The system should be also able to identify moving and stationary objects.
2. This data needs to be updated in a dynamic Occupancy grid map.
3. The speed of the system needs to be enhanced either by the use of GP-GPU based parallelized processing techniques.

3.2 Obtaining the disparity map and U-V disparity

The disparity of an image refers to the perspective shift observed in the position of an object in a stereo image pair. Different techniques are available to obtain a stereo disparity map. They are examined in section 4.1.1. The disparity map is a representation of the depth information across the field of view of the camera. This information is used to obtain the actual depth of the object using the following equation.

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} (u_l - c_u)\cdot b/\Delta - b/2 \\ (v_l - c_v)\cdot b/\Delta \\ b\cdot f/\Delta \end{bmatrix} + \begin{bmatrix} 0 \\ h \\ 0 \end{bmatrix} \qquad (1)$$

Where X_w, Y_w, Z_w refer to the Cartesian coordinates of the pixel in the disparity image, θ refers to the pitch angle of the stereo camera setup, u_l, v_l refers to the pixel coordinates in the image axis, b refers to the baseline of the camera, f refers to the focal length and c_u, c_v refers to the location of the principal point. This process is referred to as 3D triangulation.

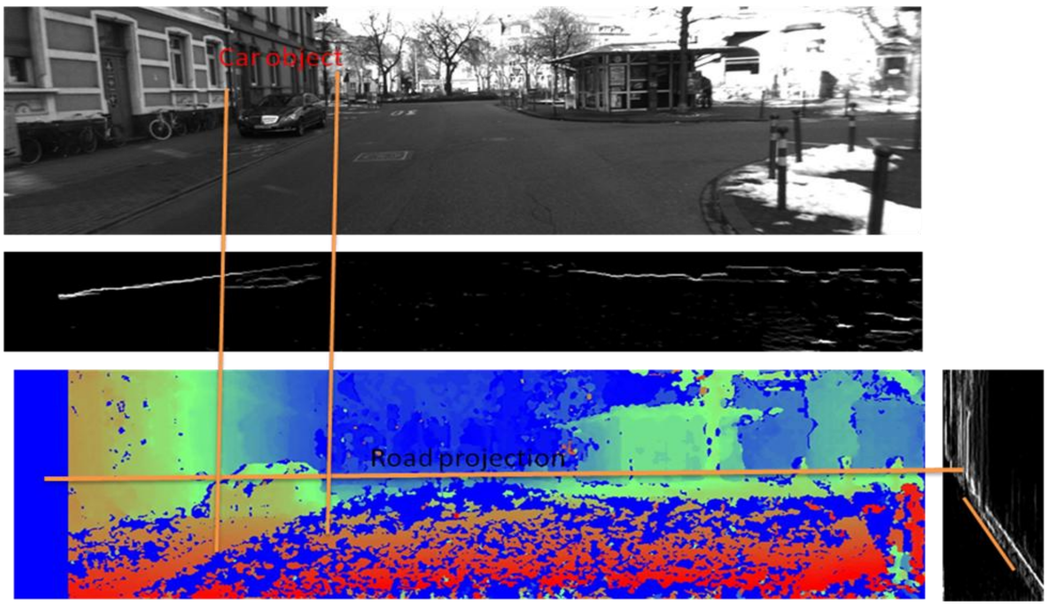

Fig 2. Top- Original image (greyscale); Middle- U disparity image; Bottom left- Disparity map; Bottom right- V disparity map

U-V disparity map is simply a row-wise and column-wise histogram plot of a disparity map. This representation of disparity map is largely sought after because of the many ways in which these maps can be used to deduce meaningful results. For example, in a row-wise histogram plot of a disparity image, also called as the V disparity image, the ground plane appears as a long bright line with a steady slope[4]. This can be used to identify the ground plane in any disparity image. In this work we also use the ground plane angle to identify the pitch of stereo camera to make corrections to the 3d points in the setup, as shown in the above equation.

U disparity map is very similar to the V disparity map, as it records a column-wise histogram plot of the disparity map. This data can be used to identify objects in an image as the columns in which the objects were found will have a bright horizontal line, representing an object of that width. This does not tell us however about the movement of the object, which we will discuss in the following section. Fig. 2 shows a U and V disparity map of an image.

3.3 Ego motion and independent moving object estimation

The first step is to identify features points in the image to track between the stereo pairs at time t and t-1. This can be done in a variety of ways using blob or corner feature detectors. In this system we are using the STAR detector from the OpenCV library. The star detector is a variation on the CenSurE feature detection algorithm[8].

Circular feature matching is then applied. It matches the features between four images as shown in Fig. 3. The four images represent the current stereo image set, recorded at time t and the preceding image set taken at time t-1. These features have a one-to-one relationship with the similar feature detected in their adjacent images. This circular feature matching is first calculated from the current left image to the current right image, then the current right image to the previous right image followed by the previous right to previous left and ending the loop by matching the previous left with the current left. This data is used to estimate the

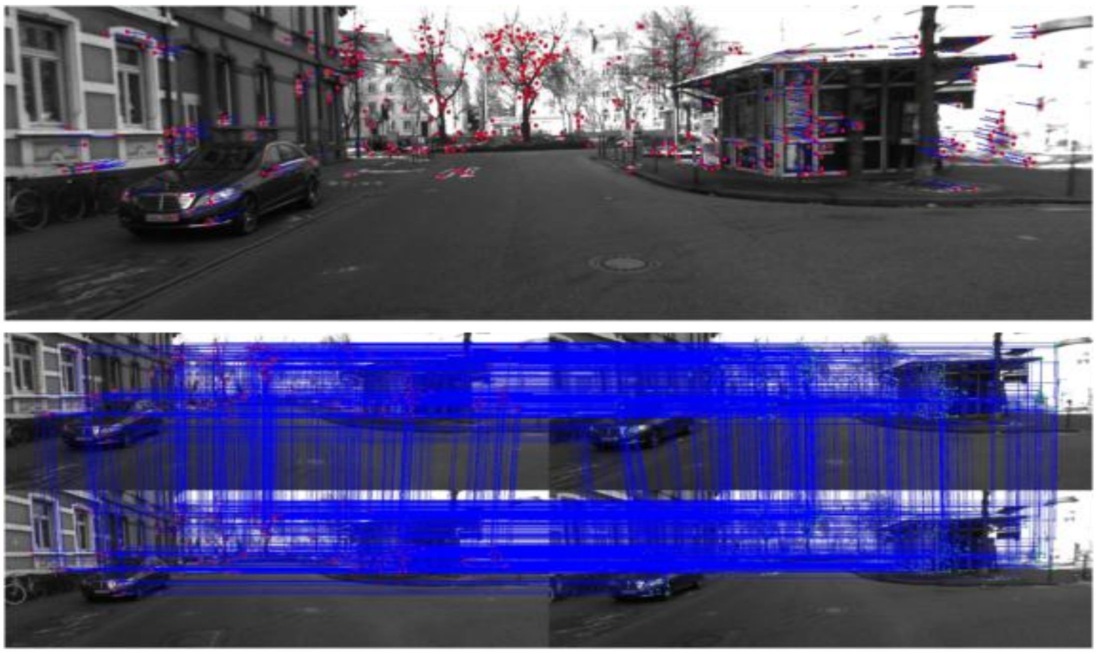
Fig 3. Image on top shows the detected features; bottom image is marked with the circular features connecting left and right images from time t and t-1

ego motion of the vehicle. For more details refer to section 5.1.3 of [5].

The circular feature matching detects flow of the optical features in the image. This however includes the movement of independent objects in the image as well as noises. So a method is needed to differentiate these ego motion feature points from the independent object motion points and the noise points. This is done via RANSAC, with a preset threshold for ego motion feature points. The ego motion feature points then appear as inliers while the independent object movements and noises appear as outliers. The outlier points further can be cross checked with the U-disparity map to separate out the moving objects, since objects appear as bright line in the U-disparity plot.

3.4 Dynamic OGM generation

The occupancy grid map is calculated from the 3D reconstruction calculated from the disparity image using (1). This 3D reconstruction model takes into account the pitch angle of the stereo camera system to be non zero. The pitch angle in this case is estimated from the V disparity map obtained from the earlier process. It can be shown that the line marked in the V-disparity figure in the image plane represents the ground plane in the3D world coordinate system as shown in [3]. Hence the line represented by the equation below corresponds to the ground plane in the world coordinate system, where α represents the slope, Δ represents the disparity and v_c represents the v intercept of the line. This line can be identified using Hough transform technique.

$$V = \alpha \cdot \Delta + v_c$$

Hence, given that we know the principal coordinates and focal length of the camera used, the value of the pitch angle can be identified by substituting these values into the following equation,

$$\theta = arctan\left(\frac{c_v - v_c}{f}\right) \quad (2)$$

After this process, the OGM can be estimated with the corrected 3D point cloud data. We first split the 3D points into cells $C(i,j)$ each assigned with points $n_{i,j}$, and the average height of points in a cell represented as $h_{i,j}$. The 3D points which are segmented as moving objects are to be counted separately as $n^m_{i,j}$. We use two parameters to find the likelihood of an object. $P(O|num)$, representing the odds of an object given a number of points are detected in a given cell. $P(O|height)$, representing the odds of an cell containing an object given an average height of the points in the cell. Since the image pixel data is non-linear in nature with respect to the distance from the camera, a sigmoid function $S(d_{i,j})$ based adjustment is applied.

$$n_{i,j}' = n_{i,j} \cdot S(d_{i,j})$$

Then an exponential function is used to model the fact that the more number of points we detect in a given cell, the chances of it being an object goes up exponentially. The logarithmic chance of the cell containing an object is opted for implementation.

$$P_{i,j}(O|num) = 1 - exp(-n'_{i,j} / \delta_n)$$

$$l_{i,j}(O|num) = log(\frac{P_{i,j}(O|num)}{1 - P_{i,j}(O|num)})$$

Similarly the logarithmic chances for a cell containing an object given an average height can be expressed as,

$$l_{i,j}(O|height) = log(\frac{1 - exp(-h_{i,j} / \delta_h)}{exp(-h_{i,j} / \delta_h)})$$

Finally the OGM is estimated as a weighted sum of the above two parameters as expressed below.

$$l_{i,j}(O) = w_n \cdot l_{i,j}(O \mid num) + w_h \cdot (l_{i,j}(O \mid height)$$

With $w_n + w_h = 1$ acting as weights. Then the detected objects are classified using the following condition.

$$O_{i,j} = \begin{cases} Notvisible & if\, n'_{i,j} < n_{tresh} \\ Object\ det\,ected & if\, l_{i,j}(O) \geq l_{tresh} \\ free & if\, l_{i,j}(O) < l_{tresh} \end{cases} \qquad (3)$$

$$M_{i,j} = \begin{cases} dynamic & if\, n^{d}_{i,j} > n^{s}_{i,j} \\ static & otherwise \end{cases} \qquad (4)$$

The $O_{i,j}, M_{i,j}$ represent the static and dynamic objects in the grid. n_{tresh}, l_{tresh} are both manually set values and $n^{s}_{i,j}$ represents the number of points in a static cell

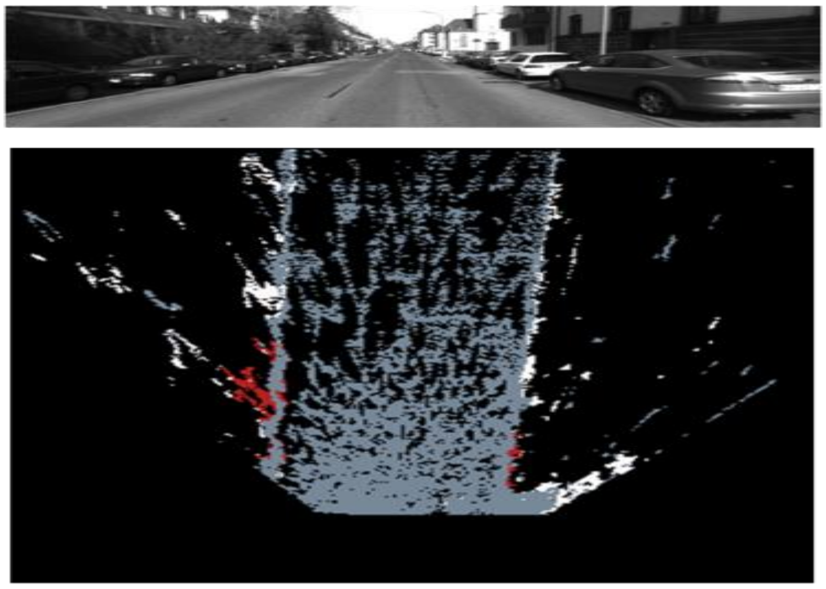
Fig 4.Image and estimated OGM (Gray: visible ground, White: Static object, Red: Moving object)

Chapter 4 GPU Implementation

A CUDA architecture focused parallelization technique was followed for GPU implementation of OGM. The GPU acceleration applies OpenCV CUDA function calls and custom CUDA kernels. The following section discusses how the hardware acceleration was realized.

4.1 Disparity Map acceleration

4.1.1 Calculating Disparity

The disparity map calculation is a time and resource consuming process. The parallelization was achieved using the OpenCV "cudastereo" library modules. The techniques investigated are Block Matching (BM), Belief Propagation (BP) [6] and Constant Space Belief propagation (CSBP) [7]. Block Matching is the common method used in basic disparity estimation techniques. This method involves using a block of pixels, usually a 9x9 window used to match between the left and right images, comparing edges/corners to estimate the disparity. This is a fast disparity calculation technique, but the detection accuracy is hugely affected.

The stereo belief propagation (BP) is a more accurate algorithm that uses a Markov random field based model to estimate the disparity map. The constant space BP is a faster version of the BP algorithm. For more information on the algorithm, refer [7], [6].

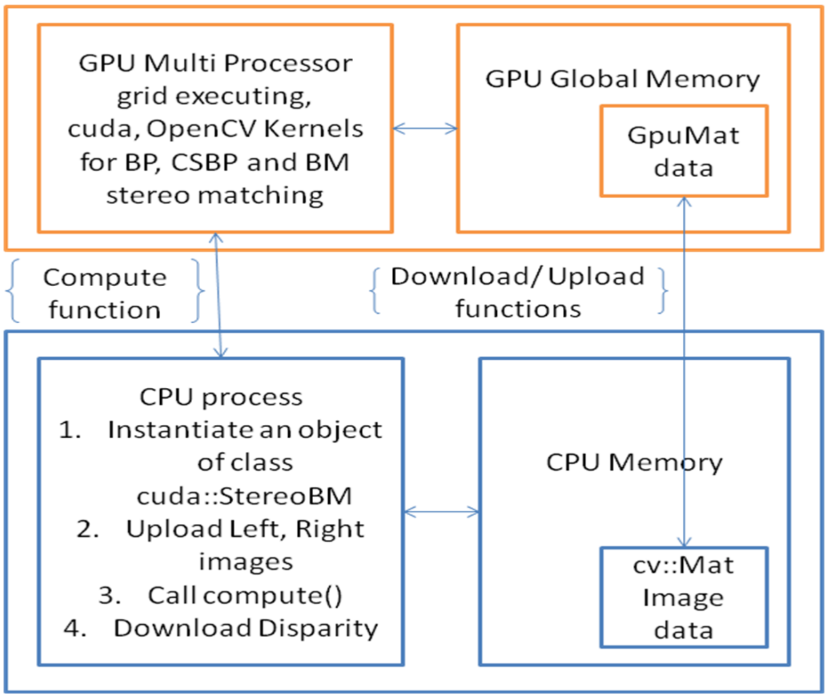
GPU Multi Processor grid executing, cuda, OpenCV Kernels for BP, CSBP and BM stereo matching
GPU Global Memory
GpuMat data
Compute function
Download/Upload functions
CPU process
1. Instantiate an object of class cuda::StereoBM
2. Upload Left, Right images
3. Call compute()
4. Download Disparity
CPU Memory
cv::Mat Image data

Fig 5. Block Diagram representing the process flow of disparity estimation in GPU using OpenCV

4.1.2 Calculating U-V disparity

UV disparity estimation was accelerated using a technique that is shown in Fig. 6. The image was split into predefined number of blocks. Each block runs a total number of threads which is given by,

$$(threads.x \quad threads.y) = \left(\frac{U}{blocks.x} \quad \frac{V}{blocks.y} \right)$$

where $threads.x, threads.y$ refer to the number of threads per block in 2D format, (U,V) refers to the size of the input image and $blocks.x, blocks.y$ refer to the number of blocks to be used. We first define the block size to be a constant and calculate the number of threads per block based on the image size.

Then each kernel can use the thread and block ID to identify the pixel it is allocated to process. The kernel algorithm is shown in the figure.

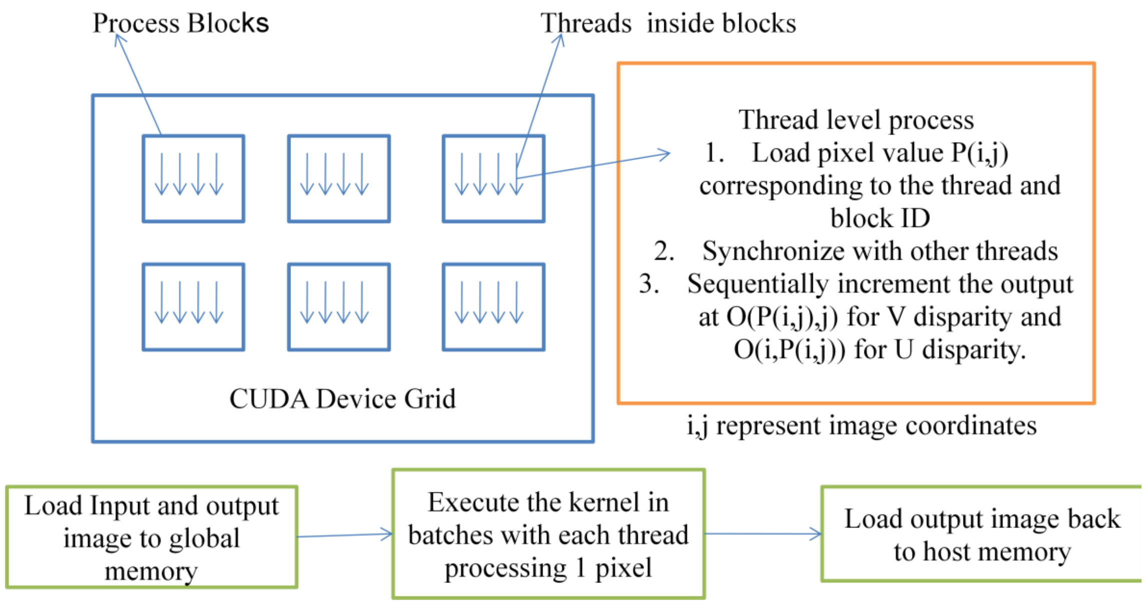
Process Blocks
Threads inside blocks
Thread level process
1. Load pixel value P(i,j) corresponding to the thread and block ID
2. Synchronize with other threads
3. Sequentially increment the output at O(P(i,j),j) for V disparity and O(i,P(i,j)) for U disparity.
CUDA Device Grid
i,j represent image coordinates
Load Input and output image to global memory
Execute the kernel in batches with each thread processing 1 pixel
Load output image back to host memory

Fig 6. Block diagram representing the generation of UV disparities in a CUDA device

As it can be seen in the block diagram, each thread loads a corresponding pixel from the disparity map and increments the corresponding bin from the corresponding row (column in case of U disparity map). As it can be seen from the kernel structure, the destination memory will be flooded with write requests and will result in conflicts or wrong results. This can be overcome by first syncing the threads before individual reads and writes, and using the atomicAdd() functionality to increment the bin values. The atomicAdd creates a queue of write back operations from each thread, hence resolving the problem.

4.2 Performance evaluation and comparison

In this section we shall discuss the performance of various disparity estimation algorithms implemented for the dynamic OGM estimation algorithm. Three algorithms were tested using aNvidia GeForce 740M with 4th generation i5 processor as host. This was compared with the performance of the same code in a system with NvidiaQuadro K620 GPU. The functionality was implemented using OpenCV CUDA libraries.

4.2.1 Evaluation of different Disparity estimation techniques

The first disparity estimation technique investigated was the block matching (BM). This is the fastest disparity estimation technique, however it can be seen in the disparity map that both objects and road lack detail along the edges. Figure 7 depicts the disparity map and OGM calculated using the BM technique.

Higher block sizes tends to give a patchy disparity map and a smaller block tends to be noisy. Hence the OGM estimated is also very less accurate both in terms of object and road detection accuracy. This implementation was done using theOpenCV,cuda::StereoBM class.

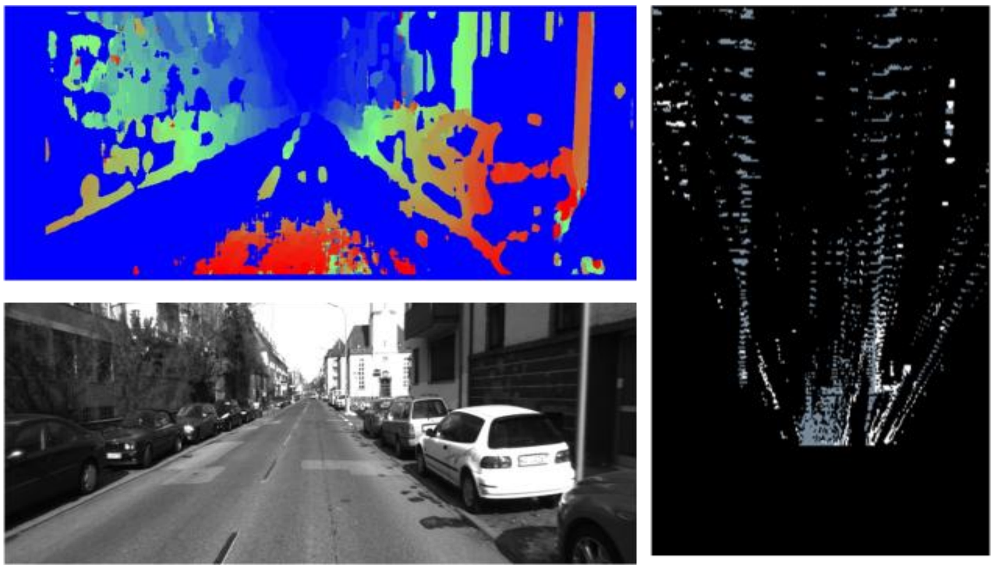

Fig 7. Block Matching (Left top: Disparity; Left bottom: Original image; Right: OGM result)

The second method uses a loopy belief propagation technique to estimate disparity. Under optimal parameter selection for number of iterations, number of levels of iterations and number of disparities, this method uses the most resource among all three algorithms and is also the slowest of the three. The image borders tend to have a lot of noise in this technique

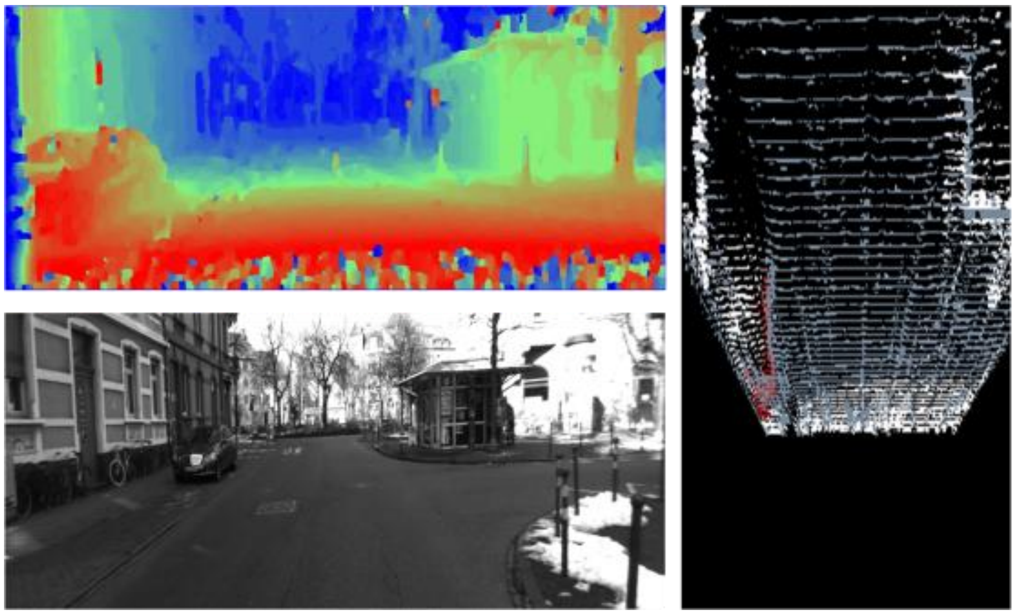

Fig 8.Belief Propagation (Left top: Disparity; Left bottom: Original image; Right OGM result)

resulting in frequent false detections, making this technique unfit for real time use as such. Figure 8 shows the results of BP method. This implementation was done using the cuda::StereoBeliefPropagation class from OpenCV.

The third method is a variation of the belief propagation technique. This technique offers a very smooth disparity output in a fairly moderate speed. This technique even though slower compared to the standard block matching technique, gives a better disparity quality while functioning at the same speed as a SGBM algorithm running in CPU. Figure 9 shows the results of this method. This was done using the cuda::StereoConstantSpaceBP class from OpenCV libraries.

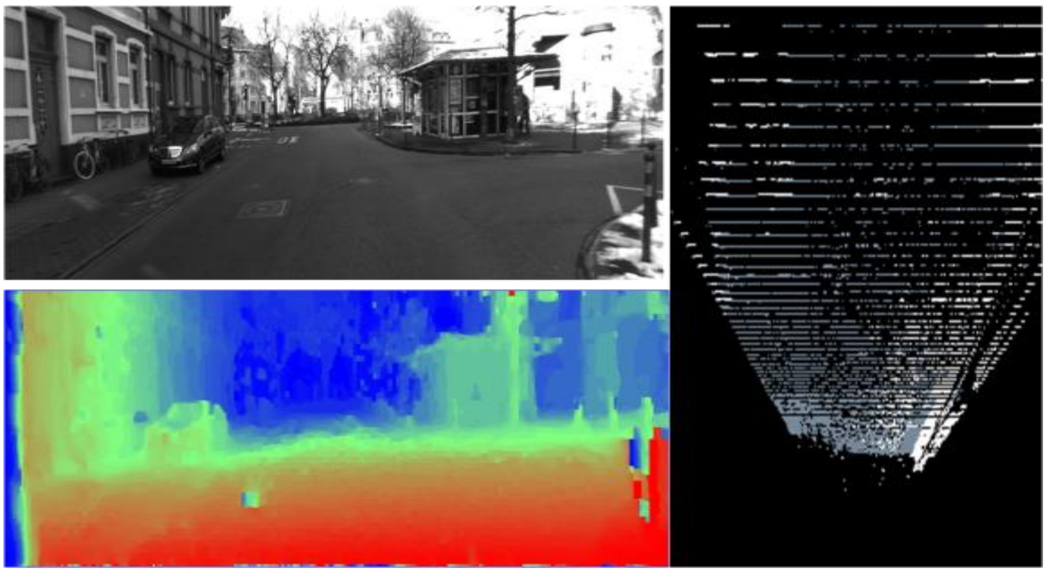

Fig 9. Constant Space Belief Propagation (Left top: Disparity; Left bottom: Original image; Right OGM result)

4.2.2 Comparison of GPU performance with CPU

The CPU trials were conducted on a 4th generation Intel i5 processor and the GPU implementation uses a Nvidia GeForce 740M graphics processor and also on a Nvidia Quadro K620. The experiment was conducted with the stereo images of KITTI data set available at [6]. The individual images are of dimension 1344x372 pixels.

Based on the table showing the performance characteristics of the disparity estimation techniques in a GPU, it is clear that as far as accelerating the OGM estimation from a stereo system is concerned, the disparity estimation technique used plays a critical role in real time usability of the system. Hence it is necessary to adapt a GPU friendly disparity estimation algorithm for real time OGM estimation. It can also be seen from the above investigation that the quality of the disparity map, in terms of smoothness and number of disparity levels has a direct impact on the resulting OGM grid.

Table 4.1 Execution Time in milliseconds for Disparity and OGM Estimations using different Processors:

Algorithm\ Processors	SGBM in CPU	BM in CPU	BM in GPU		BP in GPU		CSBP in GPU	
			Ge*	Quad*	Ge*	Quad*	Ge*	Quad*
Disparity Estimation	485	135	56		1936		520	
OGM estimation	738	395	272	190	2191	1750	803	535

The OGM estimation time and Disparity estimation time for GPU vs CPU is shown in the table above.

*The time consumed by the system running on GeForce 740M GPU is represented as Ge and Quad represents the time consumed by the same algorithm running on Quadro K620 GPU.

Chapter 5   Conclusion and Future Work

The primary aim of this project is to parallelize the occupancy grid map estimation algorithm to make the process better suited for real time use. A parallelization approach specific to Nvidia's CUDA architecture was followed.

The OGM computation has been accelerated by implementing GPU modules from OpenCV in place of CPU-only modules. Various methods of disparity estimation techniques and their effects on the OGM result have been evaluated. Block matching based approach has given the best performance but requires refinement in terms of accuracy.

For the CPU-GPU implementation of the approach, a frame rate of 5.1 fps is achieved on a Nvidia Quadro K620 graphics card. This contributes to 2x improvement in the processing speed over CPU-only implementation. This can be significantly improved with a high-end GPU like Titan.

Future work which focuses on modifying this algorithm to suit a SIMD architecture system will yield better results.

References

[1] GSMA, "Connected Car Forecast : Global Connected Car Market to Grow Threefold Within Five Years," no. February. 2013.

[2] S. Sivaraman and M. M. Trivedi, "Combining Monocular and Stereo-Vision for Real-Time Vehicle Ranging and Tracking on Multilane Highways," *14th Int. IEEE Conf. Intell. Transp. Syst.*, pp. 1249–1254, 2011.

[3] R. Labayrade, D. Aubert, and J. P. Tarel, "Real Time Obstacle Detection in Stereovision on Non Flat Road Geometry Through ' V-disparity' Represent at ion," in *Intelligent Vehicle Symposium*, 2002, pp. 646–651.

[4] C. Yu, V. Cherfaoui, and P. Bonnifait, "Evidential Occupancy Grid Mapping with Stereo-vision," in *Intelligent Vehicle Symposium*, 2015, no. Iv, pp. 712–717.

[5] Y. Li and Y. Ruichek, "Occupancy Grid Mapping in Urban Environments from a Moving On-Board Stereo-Vision System," *Sensors*, pp. 10454–10478, 2014.

[6] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient Belief Propagation for Early Vision," *Int. J. Comput. Vis.*, vol. 70, no. 1, pp. 41–54.

[7] Q. Yang, L. Wang, and N. Ahuja, "A Constant-Space Belief Propagation Algorithm for Stereo Matching ∗ ," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 1458–1465.

[8] KITTI Stereo Dataset available at, http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo

Appendix A Demo Setup

The setup and procedure for a demo is described in this appendix. The description assumes basic familiarity with the ubuntu 14.04 UI.

A.1 System Requirements

The following are the hardware and software requirements a demo.

1. PC with CUDA supported graphics card.
2. Ubuntu Version 14.04
3. Installation of CUDA 7.5 toolkit
4. OpenCV built with opencv-contrib modules and modules for CUDA support

A.2 CUDA toolkit Installation

The CUDA toolkit comes with a NVCC compiler which is necessary to compile the CUDA modules in the code. This toolkit can be installed as follows,

1. Open the link https://developer.nvidia.com/cuda-downloads
2. Select Linux-> x86_64 -> Ubuntu -> 14.04 -> deb (local)
3. Download the file named cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb
4. Open Terminal and,
    a. sudodpkg -i cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb
    b. sudo apt-get update
    c. sudo apt-get install cuda

Once this process is complete, you will be able to run CUDA C programs using the nvcc compiler.

A.3 Building OpenCV with extra modules and support for CUDA

In this step we need to build and install OpenCV with support for CUDA functionality and the extra modules in the opencv-contrib repository. The process can be simply executed with the following shell script.

1. Goto ~/Documents or anywhere you need to install the buid (note script was created for Documents directory).
2. Create a new file named installCV.sh and add the content of the script from below.
3. Open terminal and type
    a. cd ~/Documents
    b. sudochmod +x installCV.sh, type the password on prompt
    c. execute the scrip using ./installCV.sh
    d. The script will automatically install OpenCV with necessary modules and support for CUDA

installCV.sh content:

```
echo "Begin"

mkdirOpenCV

cdOpenCV

echo "Removing any pre-installed ffmpeg and x264"

sudo apt-get -qq remove ffmpeg x264 libx264-dev

echo "Installing Dependenices"

sudo apt-get -qq install libopencv-dev build-essential checkinstallcmakepkg-
configyasmlibjpeg-devlibjasper-devlibavcodec-devlibavformat-devlibswscale-dev libdc1394-
```

```
22-dev libxine-dev libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libv4l-dev
python-dev python-numpylibtbb-dev libqt4-dev libgtk2.0-dev libfaac-dev libmp3lame-dev
libopencore-amrnb-devlibopencore-amrwb-devlibtheora-devlibvorbis-devlibxvidcore-dev
x264 v4l-utils ffmpegcmake qt5-default checkinstall

echo "Downloading OpenCV source"

git clone https://github.com/Itseez/opencv.git

echo “Downloading extra modules (opencv-contrib)”

git clone https://github.com/Itseez/opencv_contrib.git

echo "Installing OpenCV"

cdopencv

mkdir build

cd build

cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local
-D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D
INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON -D
OPENCV_EXTRA_MODULES_PATH = ~/Documents/OpenCV/opencv_contrib/modules -
D ENABLE_FAST_MATH=1 -D CUDA_FAST_MATH=1 -D WITH_CUBLAS=1..

make –j8

sudo make install

sudosh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.conf'

sudoldconfig
```

```
echo "OpenCV is now ready to be used"

# EOF
```

The installation will take time depending on the system used and the speed of the internet connection.

A.4 Running the demo program

The demo program is a run-file compiled using CMake. The compile procedure is listed in the CMakeLists.txt file in the project parent directory. The project can be compiled and run by the following commands,

1. Configure the make file using “cmake .”
2. Build the runfile using “make”
3. Run the file using “./sample”