

References

- [1] A. Bevilacqua and S. Vaccari, "Real time detection of stopped vehicles in traffic scenes," in *Proceedings of the 2007 IEEE Conference on Advanced Video and Signal Based Surveillance*, (London), pp. 266–270, 2007.
- [2] J. Pan, Q. Fan, and S. Pankanti, "Robust abandoned object detection using region-level analysis," in *Image Processing (ICIP), 2011 18th IEEE International Conference*, (Brussels), pp. 3597–3600, 2011.
- [3] J. Davis and G. Bradski, "Real-time Motion Template Gradients using Intel CVLib," in *IEEE ICCV Workshop on Framerate Vision*, 1999.
- [4] A. Bruhn, J. Weickert, and C. Schnörr, "Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods," *International Journal of Computer Vision*, vol. 61, pp. 211–231, 2005.
- [5] G. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly Media, Inc., 2008.
- [6] F. Bartolini, A. Piva, and R. Piva, "Enhancement of the Horn and Schunck optic flow algorithm by means of median filters," in *Proceedings 13th International Conference on Digital Signal Processing DSP97*, (Santorini), pp. 503–506, 1997.
- [7] D. Sun, S. Roth, and M. Black, "Secrets of optical flow estimation and their principles," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, (San Francisco, CA), pp. 2432–2439, june 2010.
- [8] Q. Miao, G. Wang, C. Shi, X. Lin, and Z. Ruan, "A new framework for on-line object tracking based on SURF," *Pattern Recognition Letters*, vol. 32, pp. 1564–1571, oct 2011.
- [9] D. M. Chu and A. W. M. Smeulders, "Color invariant SURF in discriminative object tracking," in *Proceedings of the 11th European conference on Trends and Topics in Computer Vision - Volume Part II, ECCV'10*, (Berlin, Heidelberg), pp. 62–75, Springer-Verlag, 2012.

- [10] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-Up Robust Features SURF," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346 – 359, 2008.
- [11] R. Rios-Cabrera, T. Tuytelaars, and L. V. Gool, "Efficient multi-camera vehicle detection, tracking, and identification in a tunnel surveillance application," *Computer Vision and Image Understanding*, vol. 116, no. 6, pp. 742 – 753, 2012.
- [12] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I-511 – I-518 vol.1, 2001.
- [13] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proceedings of the Second European Conference on Computational Learning Theory, EuroCOLT '95*, (London, UK, UK), pp. 23–37, Springer-Verlag, 1995.
- [14] M. Dixon, N. Jacobs, and R. Pless, "An efficient system for vehicle tracking in multi-camera networks," in *Distributed Smart Cameras, 2009. ICDSC 2009. Third ACM/IEEE International Conference on*, (Como), pp. 1–8, 2009.
- [15] N. T. Pham, W. Huang, and S. H. Ong, "Probability hypothesis density approach for multi-camera multi-object tracking," in *Proceedings of the 8th Asian conference on Computer vision - Volume Part I, ACCV'07*, (Berlin, Heidelberg), pp. 875–884, Springer-Verlag, 2007.
- [16] O. Javed, K. Shafique, and M. Shah, "Appearance modeling for tracking in multiple non-overlapping cameras," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 26–33 vol. 2, June.
- [17] C. Arth, C. Leistner, and H. Bischof, "Object Reacquisition and Tracking in Large-Scale Smart Camera Networks," in *Distributed Smart Cameras, 2007. ICDSC '07. First ACM/IEEE International Conference on*, (Vienna), pp. 156–163, Sept.
- [18] C. Loy, T. Xiang, and S. Gong, "Multi-camera activity correlation analysis," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, (Miami, FL), pp. 1988–1995, 2009.

- [19] K. Denker and G. Umlauf, "Accurate Real-Time Multi-Camera Stereo-Matching on the GPU for 3D Reconstruction," *Journal of WSCG*, vol. 19, no. 1, pp. 9–16, 2011.
- [20] Nvidia, *Whitepaper NVIDIA's Next Generation CUDA Compute Architecture*. Nvidia, 2009.
- [21] N. Corporation, *NVIDIA CUDA C Programming Guide*. Nvidia, 2011.
- [22] D. Gabor, "Theory of communication," *Electrical Engineers, Journal of the Institution of*, vol. 93, no. 26, pp. 429–457, 1946.
- [23] J. G. Daugman, "Two-dimensional spectral analysis of cortical receptive field profiles," *Vision Research*, vol. 20, no. 10, pp. 847 – 856, 1980.
- [24] J. G. Daugman, "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters," *Journal of the Optical Society of America A*, vol. 2, pp. 1160–1169, Jul 1985.
- [25] T. Sanger, "Stereo disparity computation using Gabor filters," *Biological Cybernetics*, vol. 59, pp. 405–418, 1988.
- [26] E. H. Adelson and J. R. Bergen, "Spatio-temporal energy models for the Perception of Motion," *Journal of the Optical Society of America*, vol. 2, no. 2, pp. 284–299, 1985.
- [27] M. Ouali, D. Ziou, and C. Lurgeau, "Dense disparity estimation using Gabor filters and image derivatives," in *3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on*, (Ottawa, Ont.), pp. 483–489, 1999.
- [28] M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.
- [29] T. Lengyel, J. Gedarovich, A. Cusano, and T. Peters, "GPU Vision: Accelerating Computer Vision algorithms with Graphics Processing Units," 2011.
- [30] A. Abramov, T. Kulvicius, F. Wörgötter, and B. Dellen, "Facing the multicore-challenge," ch. Real-time image segmentation on a GPU, pp. 131–142, Berlin, Heidelberg: Springer-Verlag, 2010.

- [31] Z. Yang, Y. Zhu, and Y. Pu, "Parallel Image Processing Based on CUDA," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 3, (Wuhan, Hubei), pp. 198–201, dec. 2008.
- [32] C. Bruyns and B. Feldman, "Image Processing on the GPU: a Canonical Example," 2003.
- [33] K. Pulli, A. Baksheev, K. Korniyakov, and V. Eruhimov, "Real-time computer vision with openCV," *Communications of the ACM*, vol. 55, pp. 61–69, June 2012.
- [34] N. Cornelis and L. Van Gool, "Fast scale invariant feature detection and matching on programmable graphics hardware," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, (Anchorage, AK), pp. 1–8, june 2008.
- [35] X. Wang and B. Shi, "GPU implementation of fast Gabor filters," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, (Paris), pp. 373–376, 2010.
- [36] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *In VISAPP International Conference on Computer Vision Theory and Applications*, pp. 331–340, 2009.



Appendix A

Processing videos of multiple cameras with GPU

In the context of multi-camera vehicle tracking multiple video streams fetched using a single CPU. When processing those video streams using the GPU it is considered as a shared medium. The GPU is considered as mutually exclusive to the CPU threads which are created for fetching the frames. This is achieved by applying CPU lock for each code segment which is to be executed on GPU. One frame is fetched by a CPU thread and uploaded to the GPU. The Gabor Filter and all the other processing is applied on that fetched frame before it is removed from the GPU. Leaving a frame on the GPU memory and do all the processing and getting the results gives a advantage of reducing the memory transfer time between the GPU memory and the main memory of the computer. The code which is executed by each CPU thread is shown below.

```
gpuMutex.lock();
cutilSafeCall( cudaMalloc( (void **)&d_DataA_unpad, gabor.DATA_SIZE_ori_f));
cutilSafeCall( cudaMalloc( (void **)&d_DataA, gabor.DATA_SIZE_pad_f ));
cutilSafeCall( cudaMalloc( (void **)&d_padimgresult, gabor.DATA_SIZE_pad_c));
cutilSafeCall( cudaMalloc( (void **)&d_ResultGPU, gabor.DATA_SIZE_ori_c));
//Memory allocation on GPU for calculate the distance
cutilSafeCall( cudaMalloc( (void **)&d_data_energy, gabor.DATA_SIZE_ori_f));
cutilSafeCall( cudaMalloc( (void **)&d_resultPrevious, gabor.DATA_SIZE_ori_c));
//Memory allocation to find the ROI
cutilSafeCall( cudaMalloc( (void **)&d_minColLocation,sizeof(int)*resizedImageHeight
));
```

```
cutilSafeCall( cudaMalloc( (void **)&d_maxColLocation,sizeof(int)*resizedImageHeight
));
.
.
.
cutilSafeCall(cudaFree(d_imageDataChar));
cutilSafeCall( cudaFree(d_ResultGPU) );
cutilSafeCall( cudaFree(d_padingresult) );
cutilSafeCall( cudaFree(d_DataA) );
cutilSafeCall( cudaFree(d_DataA_unpad) );
gpuMutex.unlock();
```



Appendix B

Finding the moving segment closest to the camera

When the vehicles are tracked with multiple cameras the videos shows multiple vehicles on some occasions. On those situations it is important to separate those vehicles. As a solution when multiple vehicles appear on the capturing range of the video the vehicle which is closest to the camera is considered. According to the camera positioning each vehicle should appear closest to the camera. The algorithm checks for an existence of moving segments. The moving segments are searched by N number of CUDA threads where N is equal to the number of rows on the image. Each CUDA thread is assigned for each row and searched for the moving segments using a for loop. The first occurrence and the last occurrence of each row is recorded in the arrays of `d_minColLocation` and the `d_maxColLocation`.

A ROI is applied on the selected vehicle which is moving closest to the camera. Since the ROI is rectangular shape at least two of the four corners must be calculated. Those two corners of the ROI are calculated using the two arrays `d_minColLocation` and the `d_maxColLocation`.

```
__global__ void frontMovingSegmentKernal(float* d_data_energy,  
int* d_minColLocation,int* d_maxColLocation,int width,int height)  
{  
int row = blockIdx.x * blockDim.x + threadIdx.x;  
if(row < height)  
{  
d_minColLocation[row]=-1;  
d_maxColLocation[row]=-1;
```

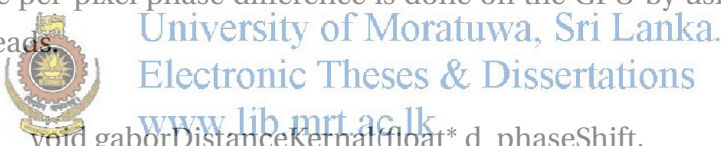
```
for(int j=0;j<width;j++)
{
if(d_data_energy[row*width+j]>1)
{
if(d_minColLocation[row]==-1)
{
d_minColLocation[row]=j;
}
d_maxColLocation[row]=j;
}
}
}
}
```



Appendix C

Calculation of Phase difference on GPU

The videos are fetched using the CPU threads and each frame is uploaded to the GPU frame by frame. The Gabor filter is applied on the adjacent frames of each video and a per-pixel phase angle is obtained. The per-pixel phase difference is calculated by the per-pixel phase angles of adjacent frames. This process of calculating the per-pixel phase difference is done on the GPU by using large number of GPU threads.



```
__global__ void gaborDistanceKernal(float* d_phaseShift,
Complexgf* d_resultPrevious,Complexgf* d_resultNow, int width,int height)
{
int row = blockIdx.y * blockDim.y + threadIdx.y;
int col = blockIdx.x * blockDim.x + threadIdx.x;
if (row < height && col < width)
{
float tanPrevious = d_resultPrevious[row * width + col].y/d_resultPrevious[row * width
+ col].x;
float tanNow = d_resultNow[row * width + col].y/d_resultNow[row * width + col].x;

d_phaseShift[row * width + col]=atan((tanPrevious-tanNow)/(1+tanPrevious*tanNow));
}
}
```

The above mentioned code is executed on GPU with assigning one GPU thread per each pixel. The $d_resultPrevious[row * width + col].y/d_resultPrevious[row *$

$\text{width} + \text{col}].x$ gives the tangent of the phase angle of that pixel of the previous frame and $\text{tanNow} = d_resultNow[\text{row} * \text{width} + \text{col}].y/d_resultNow[\text{row} * \text{width} + \text{col}].x$ gives the tangent of the phase angle of that pixel of the current frame. $\text{atan}((\text{tanPrevious} - \text{tanNow}) / (1 + \text{tanPrevious} * \text{tanNow}))$ gives the phase shift of current frame and the previous frame of a particular pixel. The phase shift represents the availability of the motion involved in the two frames.



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

Appendix D

Multi-Camera matching

The matcher constantly checks for the new images stored to the database. When it finds a list of new images it loads them one by one. The Speeded up robust features are extracted on those loaded images with the use of the GPU.

```
vector<int> matchIdZeroImageIDVector= dbConnect->loadImageIDWithMatchIdZero();
for(int i=0;i<matchIdZeroImageIDVector.size();i++)
{
IplImage* imageNoMatchId = dbConnect->loadImageCameraIPById
(currentCameraIp,secFrom70int,matchIdZeroImageIDVector[i]);
surf.setObject(imageNoMatchId);
surf.extractSURFGPU(0);
perviousCameraIp=getPerviousCameraIp(currentCameraIp);
if((strcmp(perviousCameraIp, "0") != 0))
{
vector<int> imageIDVector= dbConnect->loadImageIDByCameraDuration
(perviousCameraIp,(*secFrom70int-.75*3600),*secFrom70int);
vector<distanceID> distanceIDVector;
for(int j=0;j<imageIDVector.size();j++)
{
IplImage* imageTwo = dbConnect->loadImageById(imageIDVector[j]);
float matchError = surf.matchSURFGPU(imageTwo);
distanceID distanceIDVectorEliment;
distanceIDVectorEliment.x = matchError;
distanceIDVectorEliment.y = imageIDVector[j];
distanceIDVector.push_back( distanceIDVectorEliment);

```

```
}  
std::sort(distanceIDVector.begin(), distanceIDVector.end(), sortfunction);  
}  
}
```

The camera and the timestamp is also loaded when loading the newly added image from the database. Based on the camera information of the newly added image it is possible to find the camera information of the camera on which the vehicle might have appeared. With the use of that information the images which are saved in approximately 45 minutes earlier to the timestamp of the new image are loaded for matching. The best match is identified by sorting all the matching distances in the ascending order and finding the image which gives the minimum matching distance to the reference image. The best match is considered as previous occurrence that vehicle on the other camera.

