A Fast Parallel Algorithm for Comparing Real and Synthetic Video

Xiaodong Xu and D. Paul Benjamin

[1]Pace University, New York, NY 10038, USA
dbenjamin@pace.edu

Abstract. Robots are needed that can help people perform tasks in settings that are dangerous or repetitive and dull, in homes, in hospitals and in factories. This requires sophisticated human-robot interaction in which the robot can comprehend and predict human motions and plan responses that are cooperative and avoid harm. Our system uses a highly accurate physically realistic simulator coupled with stereo vision for 3D modeling and navigation that enables a robot to model and respond to human movements. The simulator is based on a 3D virtual world that is a virtual copy of the robot's environment. A central issue is that this virtual world must be updated in real-time for it to be useful. This paper describes a fast algorithm for detecting differences between the real world and the simulated world so that the robot can update the simulated world.

Keywords: Robotics, Virtual World

1 Introduction

1.1 A Subsection Sample

Computer vision has had a difficult time reproducing the human ability to understand visual scene information across a wide range of applications domains and environmental conditions. Recent evidence in cognitive psychology [27] and neuroscience [25] supports the proposition that simulation, the "re-enactment of perceptual, motor and introspective states" is a central cognitive mechanism that helps to provide context for planning. Shanahan [27] proposes a large-scale neurologically plausible architecture that allows for direct action (similar to a behavior-based approach) and also "higher-order" or "internally looped" actions that correspond to the rehearsal or simulation of action without overt motion. Barsalou [26] proposes that distributed structures in the brain's feature and association areas learn to recognize categories of experience. He proposes that these simulators can recreate small subsets of their content in what he refers to as "situated conceptualizations", which are embodiments of a simulation in a context: A situated conceptualization of a bicycle in a context for repair might be very different than in a context for riding and would include additional simulators to

complete the embodiment. Barsalou argues that by running the situated conceptualization as a simulation, the perceiver can anticipate future perception.

We are developing ADAPT (**A**daptive **D**ynamics and **A**ctive **P**erception for **T**hought), which is a robot cognitive architecture that integrates the structures designed by cognitive scientists with those developed by robotics researchers for real-time perception and control. Our long-range goal is to create a new kind of robot architecture capable of robust behavior in unstructured environments, exhibiting problem solving and planning skills, learning from experience, novel methods of perception, comprehension of natural language and speech generation.

A central structure of this architecture is its mental model of its environment, which consists of a 3D virtual world containing the relevant objects around the robot. This model is used to classify and predict the behaviors in the environment. For this model to be useful, it must be continually updated to maintain its accuracy with the environment. As new objects appear, they must be added. As objects disappear, they must be removed. As they change motion, their movement in the virtual world must be updated. This is a difficult task to achieve in real time.

The virtual and real worlds must be compared very quickly to detect and modify any differences. The component that handles this comparison is the MMD (match- mediated difference), which compares the input from a real camera to the input from a virtual camera at the same location in the virtual world. This paper describes a fast implementation of the MMD.

The MMD aligns the real and virtual images with an affine map, then finds a set of matched key points and places a normalized Gaussian at each of them. The normalized match quality is the inverse of the distance between matched points divided by the sum of all match errors. We use this as a coefficient of the Gaussians to create the MMD output, which is a list of the differences between the real and virtual worlds, ordered by size. Further details of the MMD algorithm are in [4]. Performing this calculation takes a half second or more. To make this approach efficient, we have developed a parallel implementation of the MMD that utilizes the GPU using CUDA.

In the next section, we describe how the MMD code was structured and modified to speed up its operation on each image. In the subsequent section, we show how to apply the MMD in parallel to the frames of video and present the resulting speedup.

2 Accelerating single images using MMD on the GPU

The initial MMD code from [4] runs on the CPU and takes slightly more than half a second to run. This is inadequate to handle the video stream from a typical robot camera in real time. The first step in improving the speed is to reorganize the code to run on the GPU.

There are three main parts of the MMD code. The first part is compareCam2Sim which finds the SIFT feature descriptor for the image. A SIFT feature [7] is a selected image region (also called keypoint) with an associated descriptor. Keypoints are extracted by the SIFT detector and their descriptors are computed by the SIFT descriptor.

It is also common to use independently the SIFT detector (i.e. computing the keypoints without descriptors) or the SIFT descriptor (i.e. computing descriptors of custom keypoints).

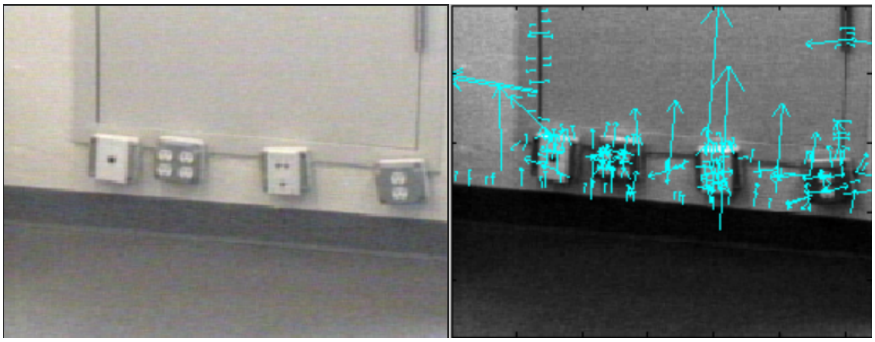

Fig. 1. An image (left) and its SIFT features (right). The sift features show the directions (arrow), size(length of arrow) and locations.

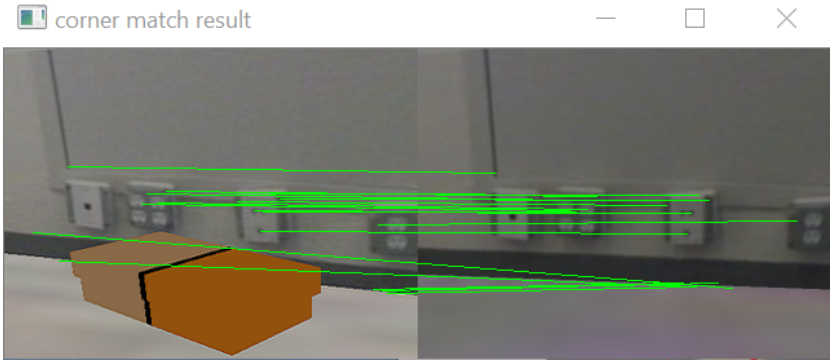

Fig. 2. Matching of SIFT features in the simulated image (left half of figure) and real image (right half). A box has been added to the simulated world.

Part two is the function AllignImageAffineM which aligns images to the same angle. Part three is the function cv2simDifference which computes the difference between two images.

The output of cv2simDifference is then cleaned up by removing noise introduced by the affinity transformation, normalizing the intensities by subtracting the mean, and filtering out very small differences between the images.

The three main parts of the code were timed on a workstation with 24 cores and 96GB memory. The results were:

compareCam2Sim 6.99 ms

AllignImageAffineM 60.57 ms

cv2simDifference 131.28 ms

As a result, cv2simDifference was chosen to be implemented on the GPU. Its code is:

```
for (j=0; j<height; j++)
    for (i=0; i<width; i++)
        for (k=0; k<channels; k++) {
            eSum=0.0;
            for (w=0; w<nMatches; w++) {
                id =(i-matches[w][2]); jd = (j-matches[w][3]);
                eSum += (mWeights[w]/mSum)*exp((double) -(id*id/(2.0*sx*sx) + jd*jd/(2.0*sy*sy)));
            }
            eSum *=eGain;

            offset = j*step+i*channels+k;
            newmask = (int)(eSum * 255.0);
            if (newmask>255) newmask=255; else if (newmask<0) newmask=0;

            if (newmask>0)
             newval =  (int)((double)img1ptr[offset]/(double)maskptr[offset]);
            else newval = img1ptr[offset];

            if (newval>255) newval=255; else if (newval<0) newval=0;

            img2ptr[offset] = newval; //min( 255, max(1,newval));
            maskptr[offset] = newmask;
```

Fig. 3. cv2simDifference code

The code has four nested loops that are suitable for parallelization on the GPU. The GPU code consists of four parts (timing in parentheses):

malloc allocates space on the GPU to receive the data from the CPU (55.90 ms);

memcpy copies the data from the CPU to the GPU (0.64 ms);

executing cv2simDifference using 1 block, 240 threads (148.77 ms);

returning the results to the CPU (16.22 ms).

This initial test consumes too much time in the kernel, so the GPU code was modified to use 240 blocks and 320 threads in each block. This led to a considerable improvement:

malloc allocates space on the GPU to receive the data from the CPU (56.73 ms);

memcpy copies the data from the CPU to the GPU (0.48 ms);

executing cv2simDifference using 240 blocks, 320 threads/block (12.26 ms);

returning the results to the CPU (16.22 ms).

The total time used for the first three parts of the MMD code is 85.69ms, which is a large improvement over the CPU code, but still too slow.

3 Using the MMD to Process Video on the GPU

The analysis in the previous section is based on processing one frame at a time. We now extend the MMD to handle a video stream.

The image and video processing is similar, because video is made of frames and each frame can be treated as single image. However, as video is a sequence of frames, we can store the video in a vector and use multithreading to accelerate the time of processing. The definition of vector is:

```
template < class T, class Alloc = allocator<T> > class vector;
```

where the descriptions of parameters are

- T − Type of the element contained.

  T may be substituted by any other data type including user-defined type.

- Alloc − Type of allocator object.

  By default, the allocator class template is used, which defines the simplest memory allocation model and is value-independent.

We define the parameter T as follows:

```
class CV2SimSmallFrame{
public:
   CV2SimSmallFrame(){}
   void makeImage2twoDImage(const Mat& bw, int index){
      vector<uchar*>& item = twoDImage[index];
      item.resize(bw.rows);
      for (int i = 0; i < bw.rows; ++i)
         item[i] = (uchar*)bw.ptr<uchar>(i);
   }
   void process(const Mat& image, int added = 0){
      …..
      GaussianBlur(bw_img, bw_img, cv::Size(5, 5), 0, 0,
                   cv::BORDER_REPLICATE);
      …….
      compareCam2Sim(&twoDImage[0][0], &twoDImage[1][0],
                  &twoDImage[2][0], &twoDImage[3][0],bw_img.cols,
                  bw_img.rows, &corner_list[0], &corner_list[1], &corner_list[2]);   }
```

```
CORNER_LIST* getCornerList(int index){
        …….
    }
private:
    …..
};
```

The Process method calls the GaussianBlur and compareCam2Sim functions which compare the corner features. This method is different from the method of processing a single image, as it processes every video frame in parallel using OpenMP.

OpenMP is a library that supports shared memory multiprocessing. The OpenMP programming model is SMP (symmetric multi-processors, or shared-memory processors): that means when programming with OpenMP all threads share memory and data.

OpenMP uses the fork-join model of parallel execution, in which programs begin as a single process (the master thread). The master thread executes sequentially until the first parallel region construct is encountered, and then creates a team of parallel threads that simultaneously execute statements in the parallel region.

The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads. Our OpenMP code is:

```
#pragma omp parallel for num_threads(local_list.size())
        for (int i = 0; i < local_list.size(); ++i){
            CV2SimSmallFrame& fA = cacheSSF[i * 2];
            CV2SimSmallFrame& fB = cacheSSF[i * 2 + 1];
            TimeStampFrame tsf = local_list[i];

            fA.process(tsf.frameA, -50);
            //timer.step("process frameA %.2f ms\n");
            fB.process(tsf.frameB);
            …….
        }
      ……
      }
```

OpenMP coding has two models, the static model and the dynamic model. Here we use the static model.

This final code, using OpenMP to run the MMD using 240 blocks and 320 threads per block, averages 25 ms processing time per frame. This is more than twenty times faster than the CPU version of the code and is sufficient to process the camera video in real time.

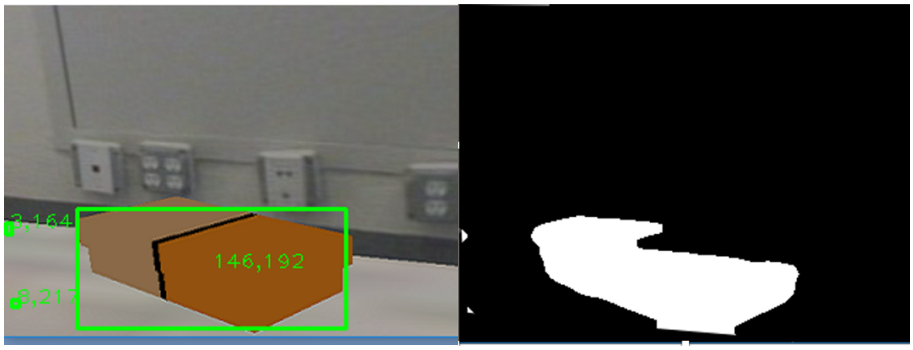

Fig. 4. The MMD output for the two images in Figure 2. The box at left indicates the region of difference, with a center at 146, 192. The MMD output mask at right shows the shape of the difference.

Conclusion

The Match-Mediated Difference (MMD) algorithm is useful in comparing video streams; however it is not capable of performing this in real time. In this paper we show how to achieve real-time performance by parallelizing the MMD using the GPU and OpenMP. The parallelized MMD code described in this paper is available at:

https://github.com/PaceRobotLab/ParallelMMD

References

1. Albrecht, S., J. Hertzberg, K. Lingemann, A. Nüchter, J. Sprickerhof, and S. Stiene. "Device Level Simulation of Kurt3D Rescue Robots," Proceedings of the 3rd International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (SRMED 2006), June (2006)
2. Benjamin, D.P., Lonsdale, D., and Lyons, D.M., "Embodying a Cognitive Model in a Mobile Robot," Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision, Boston, October (2006).
3. Benjamin, D.P., Achtemichuk, T.., and Lyons, D.M., "Obstacle Avoidance using Predictive Vision based on a Dynamic 3D World Model," Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision, Boston, October (2006).
4. Lyons, D.M., and Benjamin, D.P., Locating and Tracking Objects by Efficient Comparison of Real and Predicted Synthetic Video Imagery.‖ SPIE Conf. on Intelligent Robots and Computer Vision, San Jose CA, Jan. (2009).
5. OpenCV documentation, "Adding borders to your images", https://docs.opencv.org/3.4.0/dc/da3/tutorial_copyMakeBorder.html
6. OpenCV documentation, "Geometric Image Transformations", https://docs.opencv.org/3.1.0/da/d54/group__imgproc__transform.html
7. OpenCV documentation, "Introduction to SIFT (Scale-Invariant Feature Transform)",https://docs.opencv.org/3.3.0/da/df5/tutorial_py_sift_intro.html
8. OpenCV documentation, "Harris Corner Detection", https://docs.opencv.org/3.1.0/dc/d0d/tutorial_py_features_harris.html

9. OpenCV documentation, “Structural Analysis and Shape Descriptors”,https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis
10. Jason Sanders, Edward Kandrot, “CUDA by Example An Introduction to General-Purpose GPU Programming”, ISBN-13: 978-0-13-138768-3
11. Duane Storti, Mete Yurtoglu, “CUDA for engineers: An Introduction to High-Performance Parallel Computing”, ISBN-13: 978-0-13-417741-0
12. Xuan Mo ,“OpenCV Tutorial II: Video Processing,” iPAL Group Meeting.
13. Christiano Gava and Gabriele Bleser, “2D projective transformations (homographies)”.https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws11-12/3DCV_WS11-12_lec04.pdf
14. Sun Cheol Bae , In So Kweon , Choong Don Yoo, “COP: a new corner detector”, Pattern Recognition Letters 23 (2002) 1349–1360
15. Konstantinos G. Derpanis, “Overview of the RANSAC Algorithm”, May 13, 2010.
16. M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6):381–395, 1981.
17. R. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. University Press, Cambridge, 2001.
18. P. Torr and C. Davidson. IMPSAC: A synthesis of importance sampling and random sample consensus to effect multi-scale image matching for small and wide baselines. In European Conference on Computer Vision, pages 819–833, 2000.
19. P. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. Computer Vision and Image Understanding, 78(1):138–156, 2000.
20. Martin A. Fischler and Robert C. Bolles.Random Sample Consensus: A Paradigm for Model Fitting with Applications To Image Analysis and Automated Cartography June 1981 Communications of the ACM 24(6):381-395 DOI: 10.1145/358669.358692
21. John Lockman ,Introduction to Programming with OpenMP ,July 9, 2013
22. Chris McClanahan History and Evolution of GPU Architecture http://disi.unal.edu.co/~gjhernandezp/HeterParallComp/GPU/gpu-hist-paper.pdf
23. Luebke, David. GPU Architecture: Implications & Trends. SIGGRAPH 2008. http :// s 08. idav. ucdavis. edu/ luebke - nvidia - gpu - architecture. Pdf
24. Gao Xianming, Wang Baosheng, Zhang Xiaozhe, Wang Xu'an, "Software Data Plane and Flow Switching Plane Separation in Next-Generation Router Architecture", P2P Parallel Grid Cloud and Internet Computing (3PGCIC) 2015 10th International Conference on, pp. 194-199, 2015
25. Pezzulo, G., et al., “The mechanics of embodiment: a dialog on embodiment and computational modeling,” Frontiers in Psychology, **2,** A5, January (2011).
26. Barsalou, L.W., ―Simulation, situated conceptualization and prediction.‖ Phil. Tran. R. Soc. B(2009) 364, 1281—1289.
27. Shanahan, M.P., ―A Cognitive Architecture that Combines Internal Simulation with a Global Workspace.‖ Consciousness and Cognition, vol. 15, pages 433-449, (2006).