Computer Systems Department

Faculty of Computer & Information Sciences

Ain Shams University

HIGH PERFORMACE ACTIVITY MONITORING FOR SCENES INCLUDING HUMANS AND OBJECTS

a thesis submitted as a partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences.

By

Mohamed Hamdy Mahmoud
B.Sc. in Computer and Information Sciences,
Demonstrator at Computer Systems Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

Under Supervision of

Prof. Dr. Taymour Nazmy
Prof. of Computer Science,
Computer Science Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

Dr. Hossam Faheem
Associate Professor,
Computer Systems Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

Dr. Eman Shaaban
Lecturer,
Computer Systems Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

Cairo 2010

Acknowledgement

All praise and thanks to ALLAH, who provided me the ability to complete this work. I hope to accept this work from me.

I am grateful of *my parents* and *my family* who are always providing help and support throughout the whole years of study. I hope I can give that back to them.

I also offer my sincerest gratitude to my supervisors, *Prof. Dr. Taymour Nazmy, Dr. Hossam Faheem,* and *Dr.Eman Shaaban* who have supported me throughout my thesis with their patience, knowledge and experience.

Finally, I would thank my friends and all people who gave me support and encouragement.

Abstract

Action recognition is considered as an important research area for the last decade, presenting to the world many approaches and techniques to address different applications. Diversity of these approaches depends on utilizing different technologies. The thesis has the main focus of surveying the techniques behind and presenting two contributions that were achieved in the direction of high performance action recognition. The first presents “Muslim prayer action recognition system” as a novel application of action recognition which was implemented in real time and high accuracy. Regarding accuracy, that system was tested against a custom data set and an accuracy exceeding 93.5% was recorded. Besides, the overall system performs in 15 frames per second on 3.2GHZ a dual core processor, 2GB Memory machine which in turn achieves real time requirements. The second contribution presents a general framework based on *GPUs* to achieve significant *speed–up* in the performance of action recognition systems. Furthermore, the framework can be configured for many action recognition systems giving flexibility to the developer using the framework to integrate his custom components. This framework was tested against teamwork activity recognition system on NVidia 9500GT graphics card (32 500MHZ processors) compared to 3.2GHZ dual core processor 2GB RAM, and speedup of 20x was gained. The framework is considered a step towards a complete GPU-based suite for real-time action recognition applications.

.

Keywords: Computer vision, Action Recognition, Human, High performance

Table of Contents

List of Abbreviations

HPC:	High Performance Computing
CV:	Computer Vision
HMM:	Hidden Markov Model
SVM:	Support Vector Machine
GPU:	Graphics Processing Unit

List of Figures

List of Tables

1 Introduction

This chapter describes action recognition among diverse research areas and how high performance is related as a subsidiary science that helps avoid performance problems attached to proposed action recognition solutions. It introduces the action recognition system stages and how capturing devices and learning models as related significant issues. Moreover the chapter introduces high performance computing as crucial field of sceince concerned with solving performance problem related to action recognition, Further more it shows how GPU (Graphical processing unit) is one of the successful tools recently used in this computationally intensive domain.

1.1 Action recognition

Action recognition is based on utilizing machine intelligence preceded by transduction phase. Transduction here is the phase that converts the raw data into the form suitable for learning phase as illustrated in Fig. 1.1.

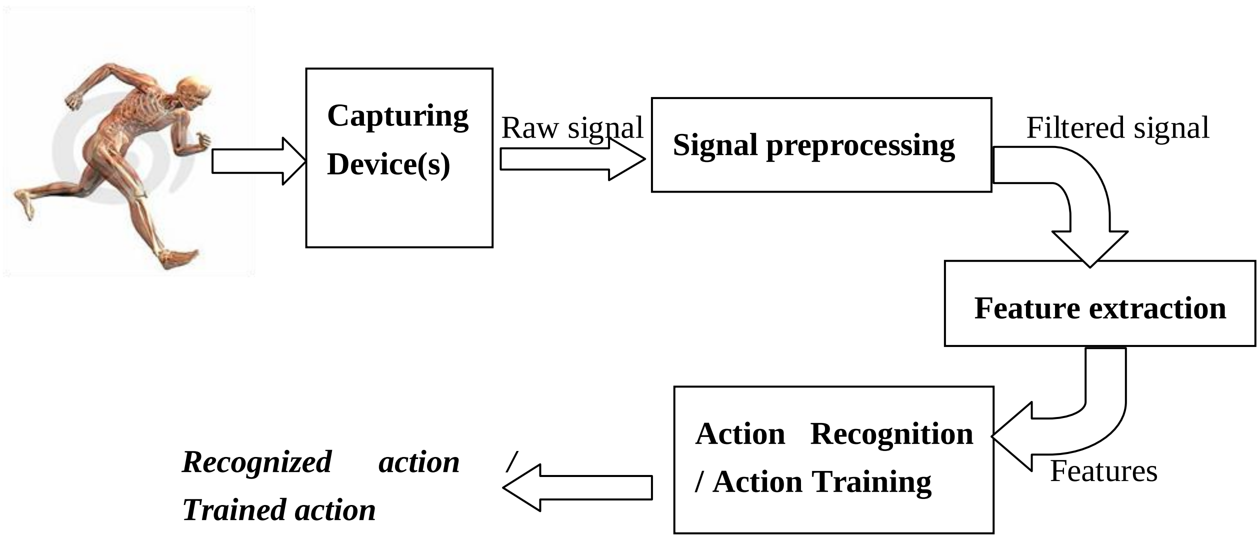


Figure 1-1 ***Action Recognition stages***

Capturing device(s) monitors the human actions converting them into digital form to be processed. This digital form serves as raw signal which processing is based on. Then the raw signals are filtered against noise or irrelevant information in *Signal Preprocessing* stage. Hence, *Feature extraction* phase can do its function which is to turn the filtered signal into features that can discriminate between actions to be classified. Features are the form of data that is the most appropriate for learning. After feature extraction is done, there are two choices depending on the mode of the system. If the generated features are for an action that needs to be classified, then features are the input of *action recognition* phase, otherwise action is in learning phase where these features are input of *training phase*. Following subsections shows different types of capturing devices and how preprocessing changes, besides the learning model common in the literature in action recognition phase.

1.1.1 Capturing Device(s)

As capturing device(s) changes, raw data that need to be processed indeed changes. One the main capturing devices used in the literature is *digital camera*, in which action recognition is considered as a *computer vision* problem.

Computer vision (Fig 1.2) is one of the research areas that are persisting among most challenging problems in the world of research. Computer vision has the mission of attaching the capability of computer eye (i.e. camera) to see the world as human eye can see, and accordingly computer vision involves processing images and sequences of images to extract information about the world that camera sees.

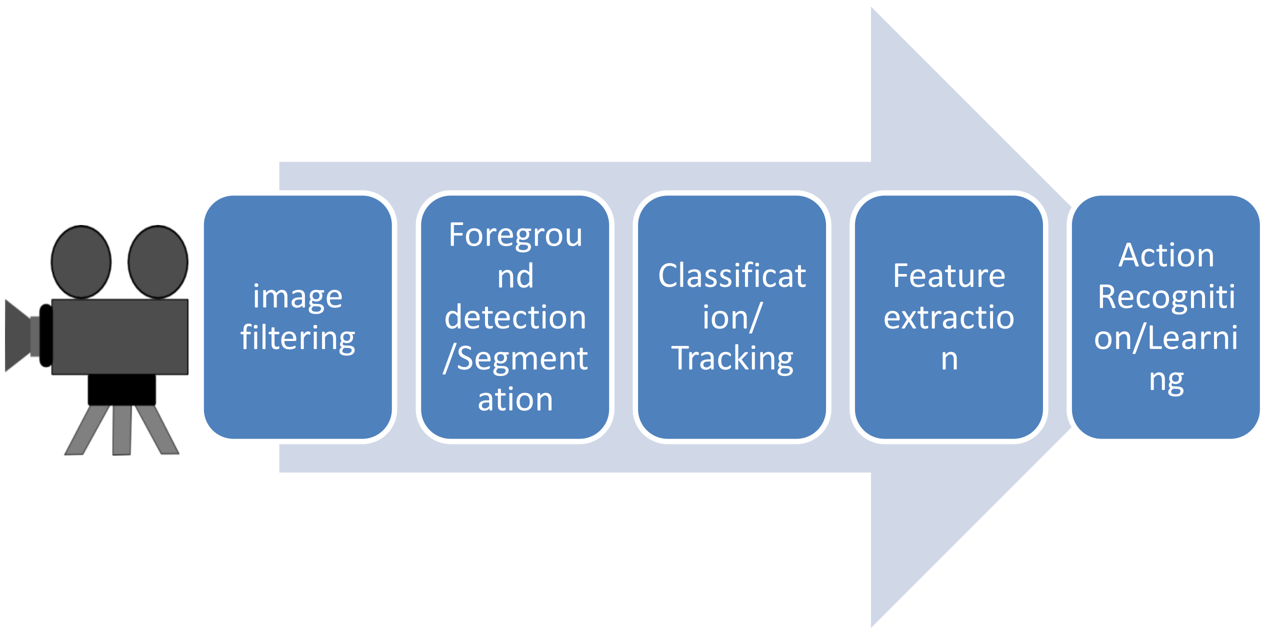


image filtering
Foreground detection /Segmentation
Classification/ Tracking
Feature extraction
Action Recognition/Learning

Figure 1-2 ***Computer vision for Action Recognition***

The process flow in Fig 1.2 illustrates the trip of frames captured from camera till intended information revealed which is either recognizing action or using the extracted features for training the action. Sometimes frames need enhancements after capturing them from the camera, this is the role of *image filtering*, then foreground is distinguished from background (*Foreground detection*), Hence binary image becomes the result which is segmented into objects (*Segmentation*).then If necessary objects are classified (e.g. human, ball, dog, cat,. .etc). Eventually human(s) are being tracked while feature is extracted and consequently these features are used for learning and/or recognition. Figure 1.2 illustrates the major way used in action recognition but it's not the only one. More approaches is described is Chapter 2.

Additionally, *Wireless sensor network* is used as a capturing device that emerges non-visual data that can be used to train learning model, and consequently that model can be used to monitor behavior. This approach is mainly used in situations where using camera is sensitive such as for patients in hospitals.
Signals from sensors serves as raw-data from which suitable features are extracted giving achieving at the end same goal as camera-based Action Recognition.

1.1.2 Learning Models

Of course machine learning is a crucial phase in action recognition especially when the application is designed to be generic. In other words adding new action is simply done by training the learning model. The main model used in this phase for *Action recognition* is HMM (Hidden Markov Model).recently combination of models that is based on the idea of blind source separation is used as well to identify actions. Details will be presented in Chapter 2.

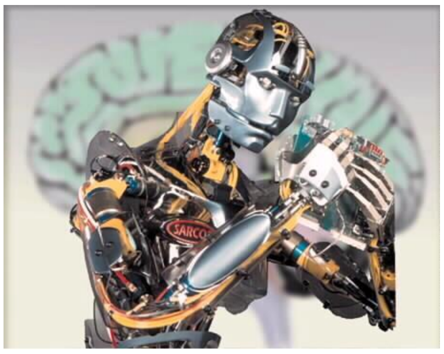


Figure 1-3 ***Machine Learning***

1.2 High performance computing

High performance computing (HPC) is one of the crucial topics that are live over this time. Problems facing humans need to be solved. So the first piece of thinking is to give a try to discover a solution regardless the performance or efficiency, however the need of these solutions to work in the real-life makes HPC very significance especially for computationally expensive solutions. This makes HPC used for many applications and becoming the major in the industry as presented Figure 1.4.

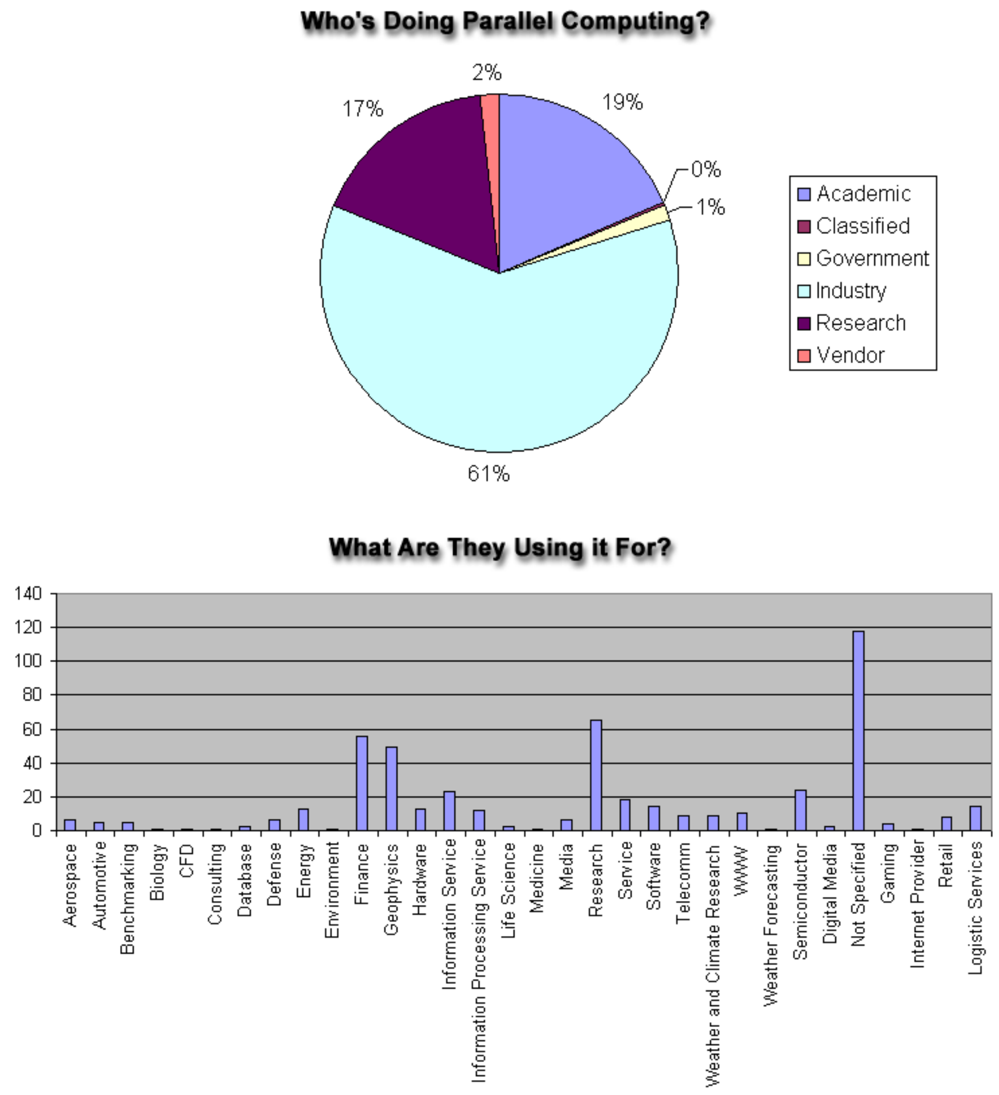


Figure 1-4 ***Who used parallel computing and what are its applications?*** **[1]**

HPC as a definition uses supercomputers and computing clusters to solve advanced computational problems [2]. HPC can be achieved via many ways including distributed computing, parallel computing as examples.

Both distributed computing and parallel computing exploits processing elements to achieve common goal, however they can be distinguished. In parallel computing processing elements have to access same shared memory, while in distributed computing each processing element have its own memory. Accordingly, systems can be classified as distributed, parallel or both. However such kind of computing can be established on different types of computer architectures classified by Flynn taxonomy.

1.2.1 Flynn Taxonomy of computer systems

Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction*** and ***Data***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple.***

Single Instruction, Single Data (SISD):

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and even today, the most common type of computer
- Examples: older generation mainframes, minicomputers and workstations; most modern day PCs.

Single Instruction, Multiple Data (SIMD):

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
    - Processor Arrays: Connection Machine CM-2, MasPar MP-1 & MP-2, ILLIAC IV
    - Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

Multiple Instruction, Single Data (MISD):

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
    - multiple frequency filters operating on a single signal stream
    - multiple cryptography algorithms attempting to crack a single coded message.

Multiple Instruction, Multiple Data (MIMD):

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- Note: many MIMD architectures also include SIMD execution sub-components

1.3 Graphics Processing Unit (GPU)

In literature *GPU* (Graphical Processing Unit) is a specialized processor used to offload 3D rendering from microprocessor [3]. Its increasing processing power make it released for general purpose (Fig. 1.5), Hence it was the birth of *GPGPU* (General purpose GPU). This means that function of GPU is not limited for 3D rendering, it is now available to do optimize any piece of processing. As a result, GPU has been the state-of-the-art infrastructure that attracts a lot of researches to use its power to achieve HPC.

Figure 1-5 ***NVidia TESLA card for HPC on GPU.***

1.3.1 **CUDA GPU Architecture**

The CUDA architecture [4] is built around a scalable array of multithreaded Streaming Multiprocessors (SMs). When a CUDA program on the host CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to multiprocessors with available execution capacity. The threads of a thread block execute concurrently on one multiprocessor. As thread blocks terminate, new blocks are launched on the vacated multiprocessors.

A multiprocessor consists of eight Scalar Processor (SP) cores, two special function units for transcendentals, a multithreaded instruction unit, and on-chip shared memory. The multiprocessor creates, manages, and executes concurrent threads in hardware with zero scheduling overhead. It implements the __syncthreads barrier synchronization intrinsic with a single instruction. Fast barrier synchronization together with lightweight thread creation and zero-overhead thread scheduling efficiently support very fine-grained parallelism, allowing, for example, a low granularity decomposition of problems by assigning one thread to each data element (such as a pixel in an image, a voxel in a volume, a cell in a grid-based computation).

To manage hundreds of threads running several different programs, the multiprocessor employs a new architecture called SIMT (single-instruction, multiple-thread) by NVidia. The multiprocessor maps each thread to one scalar processor core, and each scalar thread executes independently with its own instruction address and register state. The multiprocessor SIMT unit creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps. (This term originates from weaving, the first parallel thread technology. A half-warp is either the first or second half of a warp.) Individual threads composing a SIMT warp start together at the same program address but are otherwise free to branch and execute independently.

SIMT architecture is akin to SIMD (Single Instruction, Multiple Data) vector organizations in that a single instruction controls multiple processing elements. A key difference is that SIMD vector organizations expose the SIMD width to the software, whereas SIMT instructions specify the execution and branching behavior of a single thread. In contrast with SIMD vector machines, SIMT enables programmers to write thread-level parallel code for independent, scalar threads, as well as data-parallel code for coordinated threads.

As illustrated by Figure 1.5, each multiprocessor has on-chip memory of the four following types:

1. One set of local 32-bit registers per processor,
2. A parallel data cache or shared memory that is shared by all scalar processor cores and is where the shared memory space resides,
3. A read-only constant cache that is shared by all scalar processor cores and speeds up reads from the constant memory space, which is a read-only region of device memory,
4. A read-only texture cache that is shared by all scalar processor cores and speeds up reads from the texture memory space, which is a read-only region of

device memory; each multiprocessor accesses the texture cache via a texture unit that implements the various addressing modes and data filtering

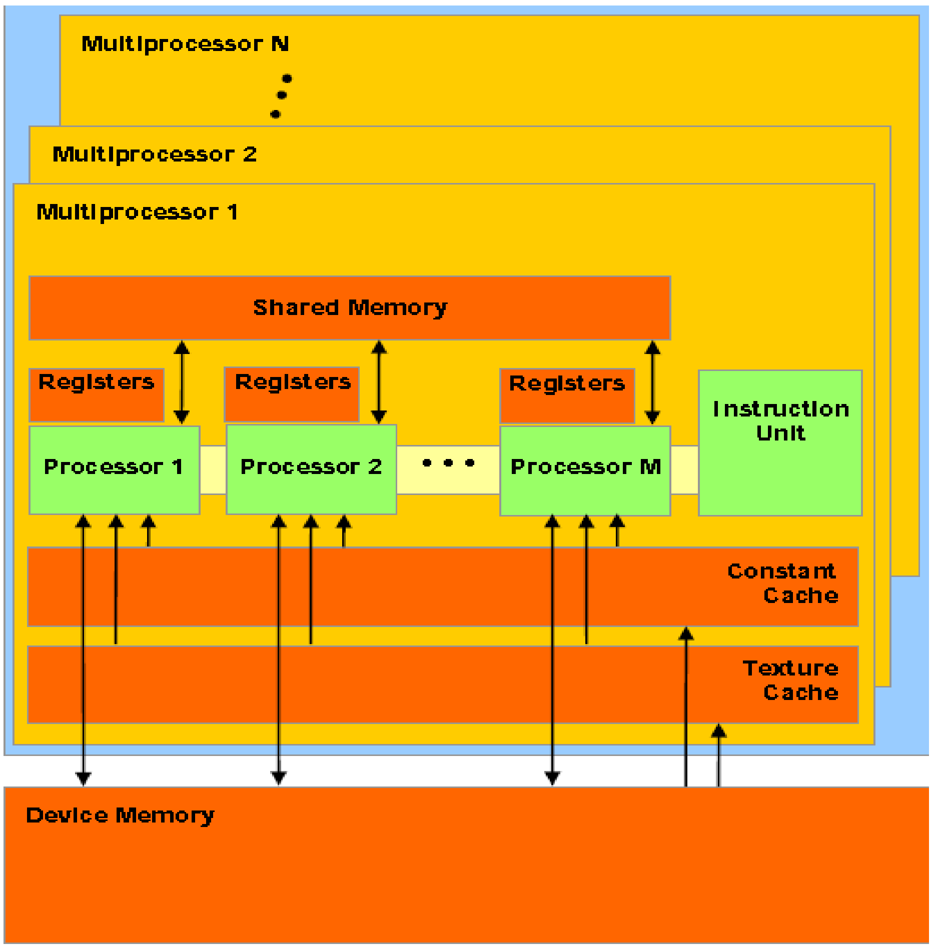


Figure 1-6 ***Hardware Model of NVidia GPU*** ***(*****A set of multiprocessors with on-chip shared memory.*****).***

1.3.2 GPU Computational Interfaces (SDKs)

GPU development is supported by many computational interfaces including OpenCL (Open Computing Language), DirectX Compute, Third party wrappers are also available for Python, Fortran, Java and MATLAB.

The core architecture used here in our thesis is NVidia CUDA architecture which in addition supports a special CUDA SDK to deal with GPU programmatically. Besides It recently gives a great support for OpenCL computation interface.

OpenCL is analogous to the open industry standards OpenGL and OpenAL, for 3D graphics and computer audio, respectively. OpenCL extends the power of the GPU beyond graphics (GPGPU). OpenCL is managed by the non-profit technology consortium Khronos Group. OpenCL is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors. AMD, RapidMind, NVidia and IBM companies have joined OpenCL community showing their adoption the OpenCL specs announced till the moment.

1.4 Motivation

Due to significant applications of action recognition especially for surveillance systems, a lot of researchers have been oriented to model solutions for recognition of actions in high accuracy along the last decade. Major of research work has been directed towards building reliable action recognition systems, as a result enhancing the accuracy has been the main goal; however another crucial issue that may lead to inapplicability of proposed reliable systems due to high performance problems. That made us highly considering accuracy and performance enhancement as essential factors in system applicability of real life systems.

1.5 Objectives

The objectives of this research are to investigate the state-of-the-art techniques in action recognition and high performance solutions related to the stages of action recognition, build *real-time* action recognition system with *high accuracy*, and to build a general framework based on *GPUs* to achieve dramatic *speed–up* in the performance of action recognition systems. The framework should be implemented with capability of customization to work with any application (e.g. teamwork activity, single person activity).

1.6 Thesis Outline

Chapter 2 *(Literature Review)* takes the reader to a deeper look inside the shell. It shows previous work related to the thesis showing challenging problems, diverse techniques and solutions to solve confronting problems. Chapter 3, 4 presents the research contributions achieved during the work in this thesis. Chapter 3 presents a novel application in action recognition which is recognition of Muslim prayer actions in real time and high accuracy. This system achieves real time processing in terms of optimizing the recognition technique. Chapter 4 presents proposed framework based on GPUs) to speed–up the performance of activity recognition of teamwork including single agent and multi-agent (team) actions. The framework can be customized for any action recognition application. Chapter 5 augments some concluding remarks and directions for future work.

2 Technical background and previous work

This chapter shows different techniques used in the literature for action recognition and related high performance computing work in a historical order. So this chapter presents detection and tracking of objects which is a pre-requisite step for any camera-based action recognition system. Afterwards, various approaches of action recognition is presented either camera-based or sensor-based. Finally GPU work in the direction of computer vision generally is presented.

2.1 Object Detection , Classification and Tracking

This section serves as the base stone on which major action recognition work is built. Actually till the moment object detection and tracking is an open research area as will be shown here. Fig 2.1 clarifies the detailed process flow to action recognition passing detection, classification and tracking. Object Detection has differentiates significant objects from the background. Classification function is to categorize the object according to monitored features into classes to know to deal with. Task of Tracking is to establish correspondence mapping between objects across frames, such that each object position, velocity, etc can be maintained[5].

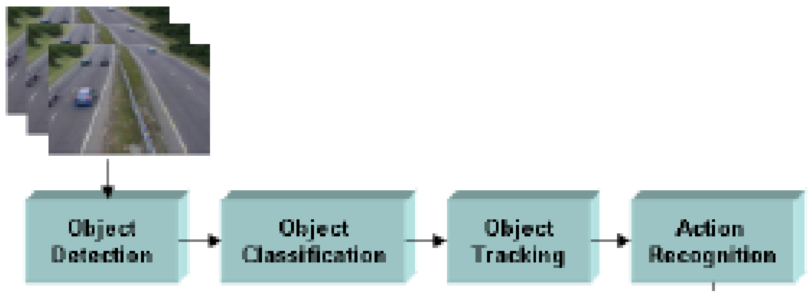


Figure 2-1 ***Process flow of Object Detection, Classification, Tracking to Action Recognition***

2.1.1 Object Detection and segmentation

A lot of research work has been done to achieve high accuracy of object detection till the moment. The challenge has been to maintain the balance between precise detection and real time processing. Detection has different jobs to do depending on the application. This section presents the outer shell of object detection as a major area in computer vision research.

Color based Object Detection(CBOD) , an approach of object detection that is applied where color of object is predefined or labeled with specific color. In these static environments detection is done by thresholding around the defined color. This approach is accurate and can be easily achieved in real time, however; it is dummy against unknown environment where colors of the objects are not predefined. In the same way, skin detection was investigated as an important application under *CBOD* [6] [7] . It has been used as a preprocessing to detect skin colored objects such as hand, face (Fig. 2.2).

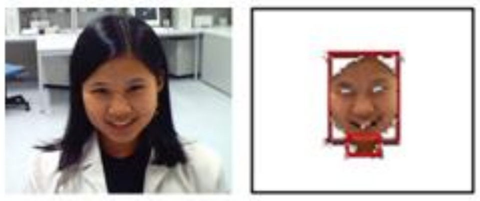


Figure 2-2 Skin Detection

Another approach that depends on motion is *Motion based Object Detection* (MBOD). MBOD depends on building models to characterize areas that belong to the background and accordingly foreground objects is the complement (Fig.2.3). This technique is the most widely used in surveillance systems.

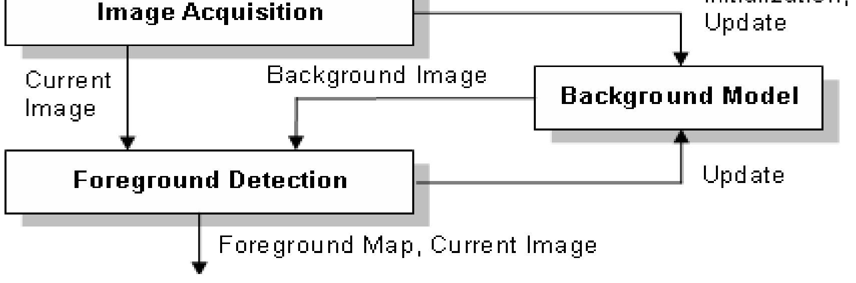
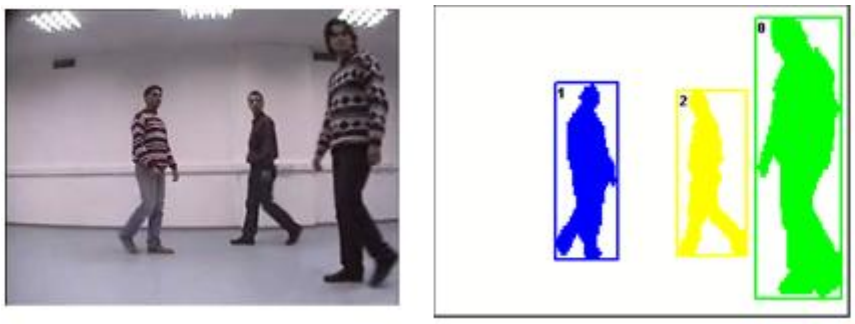


Figure 2-3 Motion Based Segmentation

Massive research efforts have been done seeking robust background model of MBOD for different environments. *Thanarat Horprasert et al* presented an algorithm that is able to detect moving objects with static background but they handled shadow removal as a challenging problem because shadow is also moving but it is not an object [8]. *Rui Tan,et al* built a model specialized for extract vehicles as the main function of traffic segmentation[9]. *Rui Tan,et al* used Adaptive Gaussian mixture model in which the dimension of the parameter space at each pixel can adaptively reflects the complexity of pattern at the pixel. *Sheng-Yan Yang, et al* combined spatial and color coherency with the pixel-wise GMM to determine the background model [10] that leads to an accurate object contours even in dynamic scenes.

Actually presented work above assumes that camera is static, so some of the work has been directed to achieve discrimination of objects even when camera is in motion. *Yan Zhang et al* presented robust moving object detection algorithm [11] where camera is fixed on a moving vehicle, Experimentations on numerous real world driving videos have shown the effectiveness of the proposed technique. Moving objects such as pedestrians and vehicles up to 40 meters away from the camera have been reliably detected at 10 frames per second on a 1.8GHz PC.

After each pixel of the frame is classified as background or foreground pixel, the foreground regions are segmented to define regions of objects (Fig.2.4) using Connected Component Labeling Algorithm [12].



(a) **(b)**

Figure 2-4 (a) Frame , (b) Labeled Detected objects

2.1.2 **Object Classification**

Object classification has the task of categorizing the object according to its type (e.g. ball, car, human, etc). The raw input is the silhouette of the object to be classified. This type of classification is named shape based classification and this is the commonly used type for surveillance systems generally or action recognition specially.

Dedeoğlu, Yiğithan et al presented an approach in [5] that is able to classify objects as human, human group and vehicle (Fig. 2.5) based on silhouette template database. Distance function is measured between the query silhouette to be classified and the database. The query silhouette will be categorized as class C if the shortest distance is found between that silhouette and a silhouette belonging to class C within specified tolerance considering probability of silhouettes of untrained objects.

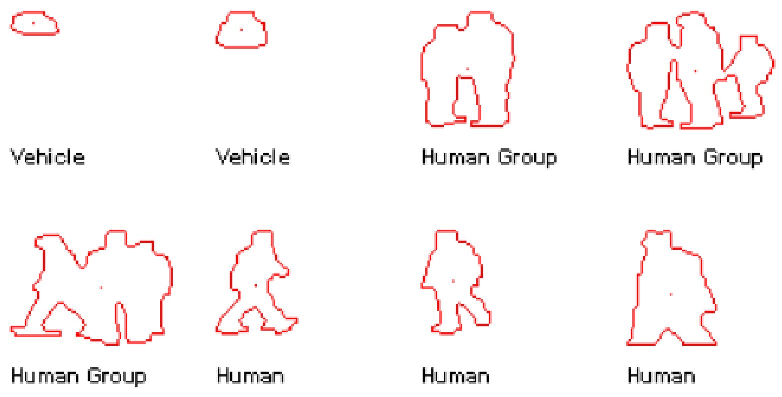


Figure 2-5 Object classification [5]

Jianpeng Zhou et al presented human classification algorithm based on codebook learning named DSCL (distortion sensitive competitive learning) [13] as a part of human tracking system.

The concept of object classification is also used to categorize and classify postures of the same object.[14] Where posture of human is classified using Support Vector Machine (SVM) that have as input affine invariant Fourier descriptor of the human contour corresponding the posture. The same idea can be used to determine posture of hand or sign language alphabetic (Fig. 2.6). [15], [16] are instances of using classification for the aim of sign language translation. (It is worthy to mention that in sign language application skin color is the approach of object detection, however in human postures classification is preprocessed by *Motion Based Object Detection*).

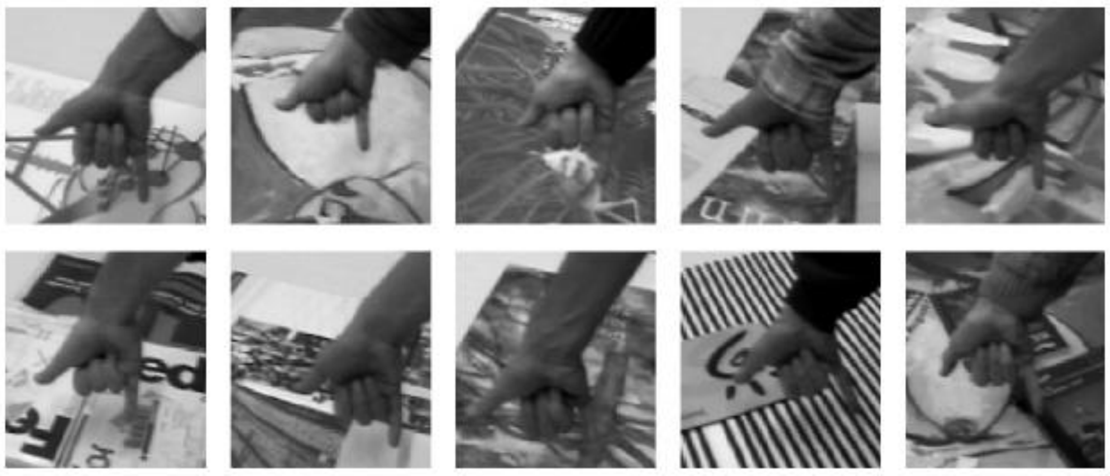


Figure 2-6 Hand postures for sign language alphabetic.

2.1.3 **Object Tracking**

Object tracking has the function of mapping corresponding objects across frames captured from image acquisition system either by camera or stored video for the aim of maintaining their location (Fig. 2.7). Object tracking simply do a tangible job for which object detection, classification are performed.

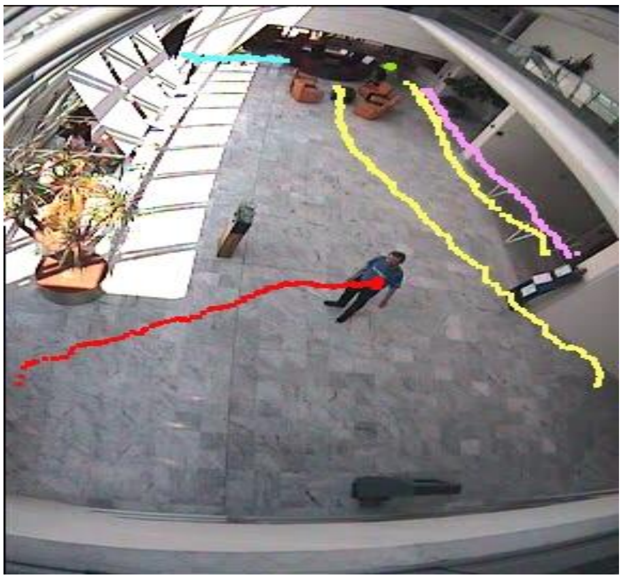
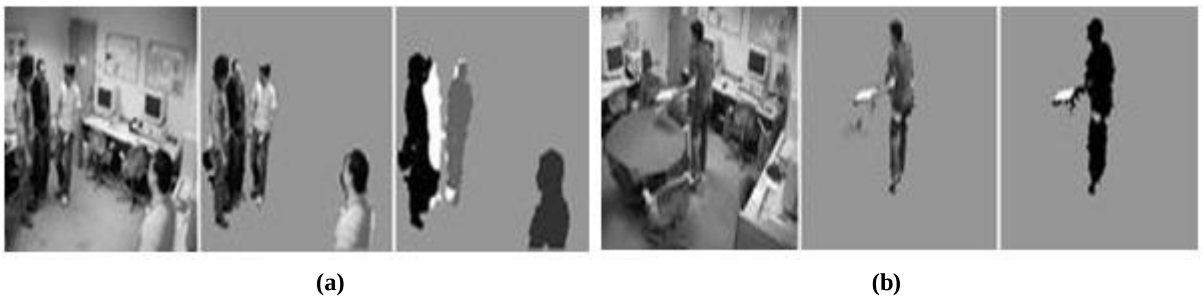


Figure 2-7 Object Tracking

Dedeoğlu et al [5] exploited the object features such as size, center of mass, bounding box and color histogram which are extracted in previous steps to establish a matching between objects in consecutive frames. Also their approach handles occlusion in a heuristic way by defining the state of merging the objects and the state of object split. A combination of correlogram and histogram information is used to model object and human color distributions by *Mart´ı Balcells Capellades, et al* [17] to be able to detect when people merge into groups and segment them during occlusion. Identities are preserved during the sequence, even if a person enters and leaves the scene. The system is also able to detect when a person deposits or removes an object from the scene (Fig 2.8).



(a) **(b)**

Figure 2-8 (a) tracking on occlusion (b) Tracking objects interacting with human [17]

Yao-Te Tsai et al presented better accuracy system which is 100%,90.3 %,80% for 2,3,4 occlding objects ,respectively [18], while no numerical accuracy reported in [17].Eventually, *W.W. Mayol et al* presented an approach that allows a natural handling of occluding objects, and permits dynamic changes of the scale and size of the sampled region [19].

To special purpose tracking, Hand tracking is what to mention as the prerequisite step for areas related to gesture recognition and sign language (Fig. 2.9). *Nguyen Dang Binh, et al* developed a method based on Kalman filter which is robust and reliable on hand tracking in unconstrained environments and then the hand region extraction fast and accurately [20].

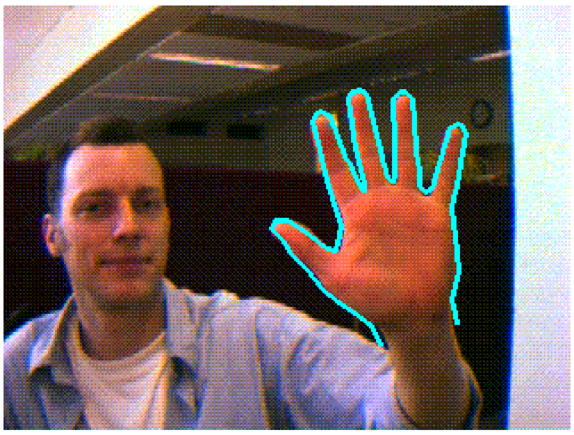


Figure 2-9 Hand Tracking

2.2 Action Recognition

This section presents the literature of action recognition as two categories. The first one is based on computer vision denoted as Vision Based Action Recognition (VBAR) presented and is the second is based on sensors denoted as Sensor Based Action Recognition (SBAR). Due to many contributions efforts done in VBAR relative to SBAR. The chapter categorizes VBAR into recognition single agent actions and teamwork actions.

2.2.1 Single agent action Recognition

The success of Hidden Markov Model (HMM) as a Machine learning model stimulates early researchers of action recognition area to apply it as a startup trails [21]. *Mathew Brand et al* presented an algorithm for coupling and training HMMs showing its ability to model interactive process [22]. They tested it against 52 sequences for total of 3 human actions (i.e. roughly 17 sequences for each action) and accuracy of 94% was recorded (Fig 2.10).

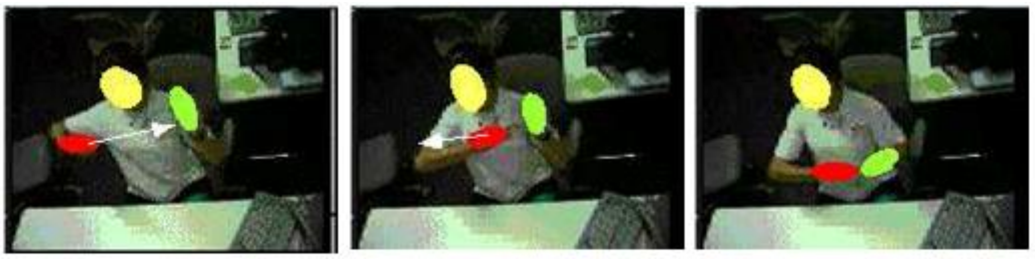


Figure 2-10 single whip action used by Mathew Brand et al

Xiaolin Feng Perona, et al presented an algorithm for the recognition of human actions in image sequences. The algorithm [23] consists of 3 stages: background subtraction, body poses classification, and action recognition. A pose is represented in space-time called `movelet'. A movelet is a collection of the shape, motion and occlusion of image patches corresponding to the main parts of the body. The (infinite) set of all possible movelets is quantized into codewords obtained by vector quantization. For every pair of frames each codeword is assigned a probability. Finally, HMM based Recognition is performed by simultaneously estimating the most likely sequence of codewords and the action that took place in a sequence. Data set has been prepared in which each body parts can be identified. They made special cloths which has distinguishable color for each body part (Fig.2.11). Experiments are done on 11 human actions which are higher than data set used in [22] , only 3 actions. 80% is recorded as overall accuracy; however 98% is the accuracy rate for half of the actions.

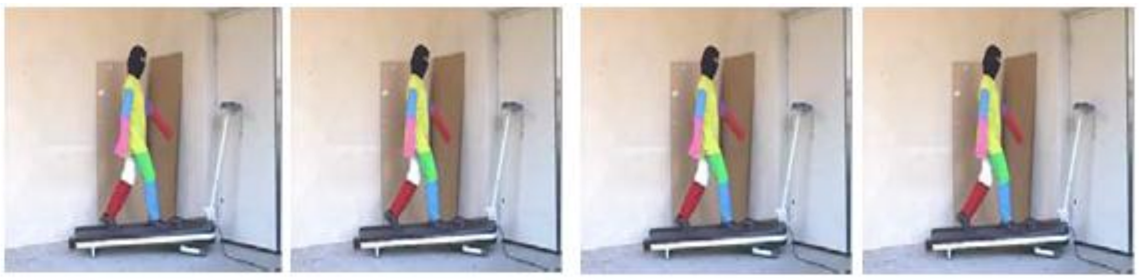


Figure 2-11 Example of Training Images and Movelet. Left two: two consecutive images in the training sequence. Right two: estimated rectangular shapes for both frames to form a movelet

The idea behind which this works well is as follows. If K is the count of human actions that need to be recognized, then H_k, k = 1...K where H_k is the HMM assigned to action k, each movelet has been assigned to one codeword. Therefore, for each training action k, codeword sequence is obtained to represent its movelet sequence. Its HMM state (codeword) transition matrix A_k can be easily trained from this codeword sequence.

To recognize which action the observed sequence (X_1, X_2…, X_T) represents and what is its best representative codeword sequence, the following HMM solutions are applied. First, for each action hypothesis k ∈ (1 … K), Viberbi algorithm [21] is applied to search for the single best state (codeword) sequence which maximizes the following probability:

$$q^*_{k1}, q^*_{k2}, \ldots, q^*_{kT} = \max_{q_{k1}, q_{k2}, \ldots, q_{kT}} P(q_{k1}, q_{k2}, \ldots, q_{kT} | \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T, A_k) \quad \text{(Eq 2.1)}$$

Where $q_{kt} \in$ (C1 … CN), The human action H_{k*} that best represents the observed sequence is determined by:

$$k^* = \arg\max_k P(q^*_{k1}, q^*_{k2}, \ldots, q^*_{kT}, \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T | A_k) \quad \text{(Eq 2.2)}$$

Correspondingly the sequence $q^*_{k*1}, q^*_{k*2}, \ldots, q^*_{k*t}$, is the recognized most likely codeword sequence to represent the observed sequence (X_1,X_2, … ,X_T).

This idea illustrated in the context above is the base upon which most of HMM-based contributions in action recognition are built

Smart class room is one of the main applications where action recognition can play an important role to move the intelligence educational process big step forward. *Haibing Ren et al* presented a system for realizing action in smart class room [24]. First, the system proposes a hybrid human model and employs 2-order B-spline function to detect the two shoulder joints in the silhouette image to obtain the basic motion features including the elbow angles, motion parameters of the face and two hands. Then, Primitive-based Coupled Hidden Markov Model (PCHMM) is presented for natural context-dependent action recognition.

To another transition that happened in 2004, where *Christian Sculdt et al* presented the first intensive action recognition data set (Fig 2.12) widely named as **KTH** data set. Besides they applied Support Vector Machine (SVM [25]) approach to recognize actions [26]. The main idea is to build local-space time features to represent the motion in 2D image structures which in turn serves as the input of SVM as the classification model which was widely used as margin classifier in visual pattern recognition.

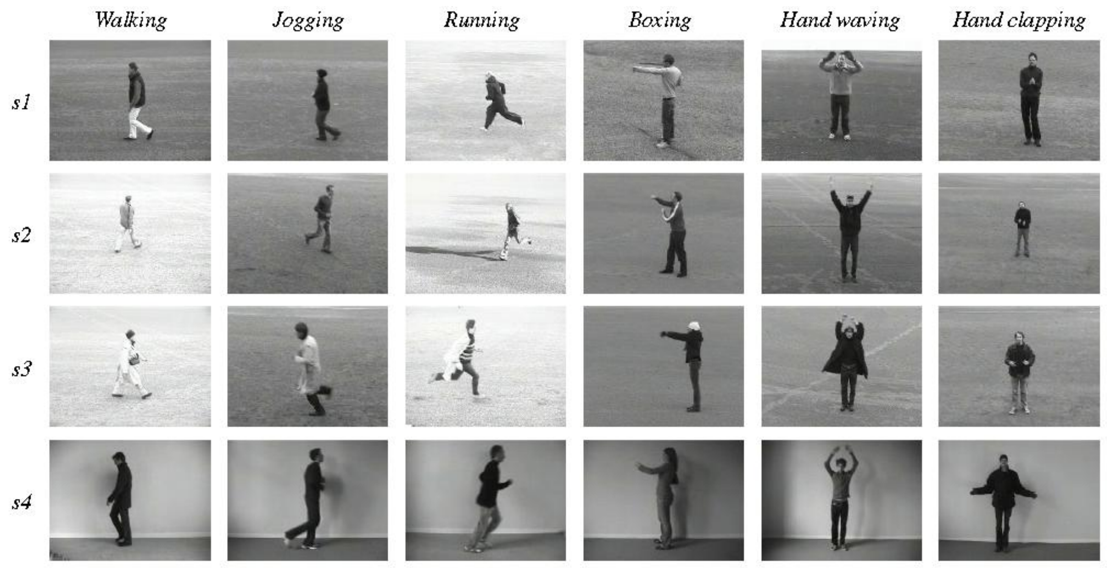


Figure 2-12 KTH Action Recognition Dataset containing 2391 sequences of 6 human actions performed by 25 people

To the use of DHMM (Discrete Hidden Markov model) in action recognition [27], *Vili Kellokumpu et al* presented a system illustrated in (Fig 2.13).

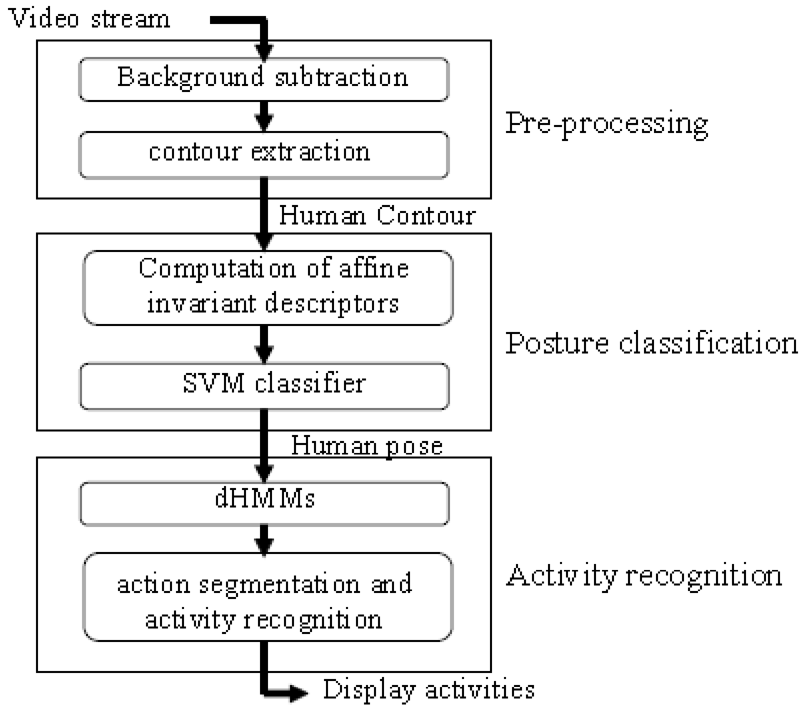


Figure 2-13 processing stages of Vili Kellokumpu et al 's system [27]

DHMM is HMM with discrete inputs. The discrete inputs in the system are human postures. This can be clarified as follows. Firstly human contours are extracted from the video to be classified to one of the trained postures using SVM. This will ensure that the output of classification method will be discrete in this classification phase affine invariant descriptors are extracted from contours to ensure that classification is independent of affine transformations (e.g. scale, rotation...Etc). After posture classification is done, HMM should be trained with sequence of human postures that describe the specified action and in turn HMM can recognize actions using sequence of postures of the query action. The system was tested on dataset of 20 activities rather than only six by *Christian Sculdt et al* and 90% is recorded as recognition accuracy [26].

What if activity is fed into the system from another view port different from the trained one? For this purpose M Ahmad et al presented a new method for human action recognition from any arbitrary view image sequence that uses the Cartesian component of optical flow velocity and human body silhouette feature vector information. Principal component analysis (PCA) was used to reduce the higher dimensional silhouette feature space into lower dimensional feature space. The action region in an image frame represents Q-dimensional optical flow feature vector and R-dimensional silhouette feature vector. Each action was presented using a set of HMMs and they model each action for any viewing direction (Fig. 2.14) by using the combined (Q + R)-dimensional features at any instant of time. Experiments were performed on the proposed method by using KU gesture database and manually captured data.

Experimental results of different actions from any viewing direction are correctly classified by their method, which indicate the robustness of their view-independent method [28].

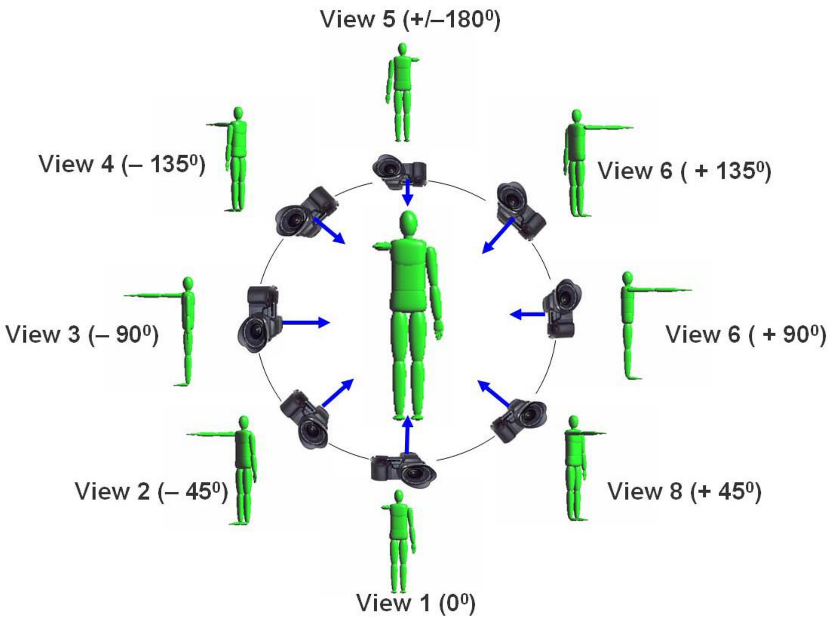


Figure 2-14 Multiple views for training sequence

Paul Scovanner et al introduced a 3-dimensional (3D) SIFT (Scale Invariant Feature Transform [29]) descriptor for video or 3D imagery such as MRI data. Their experiments show how this new descriptor is able to better represent the 3D nature of video data in the application of action recognition. A bag of words approach was used to represent videos, and present a method to discover relationships between spatio-temporal words in order to better describe the video data [30]. Besides, the experiment show how 3D SIFT is able to outperform previously used description methods in an elegant and efficient manner (Figure 2.15) . 3D SIFT was tested on ***BLANK ET AL*** data set of 92 videos prepared in [31].

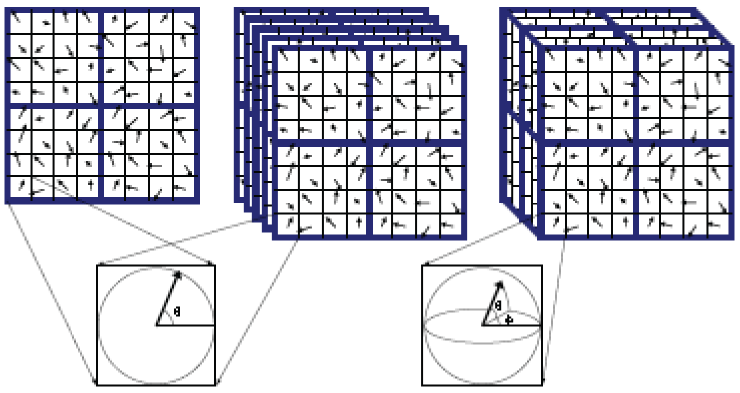


Figure 2-15 The left image shows the familiar 2D SIFT descriptor. The center shows how multiple 2D SIFT descriptors could be used on a video without modification to the original method. The right shows the 3D SIFT descriptor with its 3D sub-volumes, each sub-volume is accumulated into its own sub-histogram. These histograms are what make up the final descriptor.

Action recognition can be used vitally in Robotics applications. *Odest Chadwicke et al* presented a method for kinematic pose estimation and action recognition from monocular robot vision through the use of dynamical human motion vocabularies [32]. Their motion vocabulary is comprised of learned primitives that structure the action space for decision making and describe human movement dynamics. Given image observations over time, each primitive infers on pose independently using its prediction density on movement dynamics in the context of a particle filter. Furthermore, *Matthew Loper* presented three-component Action recognition pipeline [33] in environment illustrated in Figure 2.16. First, a new off-the-shelf active sensor is employed to obtain observations. Then, he utilized his observables to generate a set of pose-inspired features. Eventually, he used those features to obtain a history over actions, with the use of a HMM and a Gaussian likelihood function.

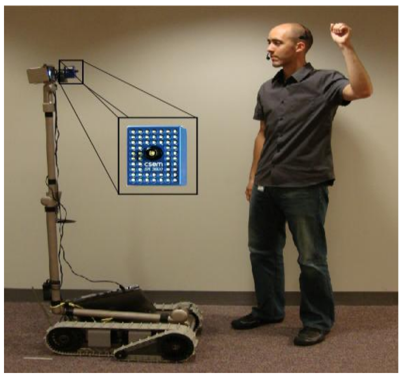


Figure 2-16 Loper's environment: iRobot PackBot EOD mobile robot, SwissRanger camera, and Bluetooth headset [33].

KTH data set has been recently reused in 2009 as the experimental data set by *M. Angeles Mendoza et al* in action recognition using PoHMM (product of HMM), Figure 2.17 [34].

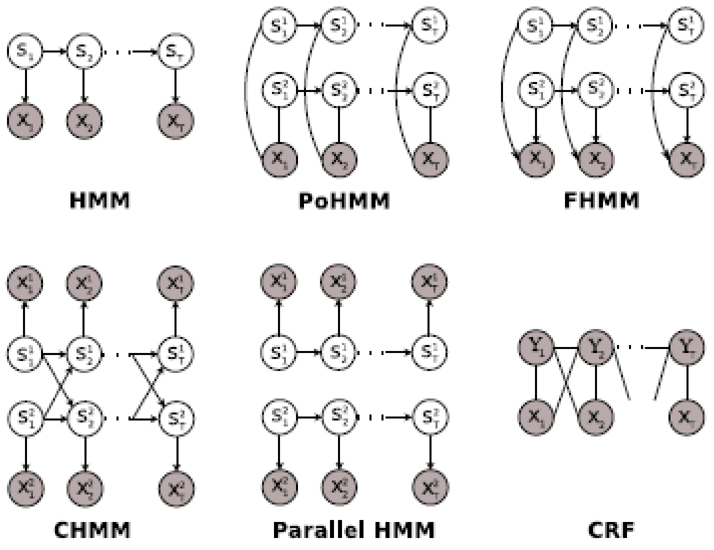


Figure 2-17 Diagram of the state-space models. Empty circles represent the hidden states (S) and shaded circles observed data (X). In CRF, Y represents the observed label. Arrows represent causal dependence, edges represent non-causal dependence. It can be observed that FHMM and PoHMM both represent a large state-space through simultaneous HMMs with causal and non-causal dependence of the observations respectively.

A local feature called temporal-state shape context (TSSC) that can capture local space-time shape characteristics effectively by *Pei-Chi Hsiao et al* [35] on BLANK ET AL data set. The time warping effects are compensated by segmenting the video sequence into several temporal states defined by soft fuzzy intervals. Encouraging results are obtained in the experiments, even with deformations in real world videos. Spatial consistency information between two actions can also be provided by their method, which is useful for action analysis applications.

Zhu Li et al presented a luminance field manifold trajectory analysis based solution for human activity recognition on KTH dataset, without explicit object level information extraction and understanding. This approach is computationally efficient and can operate in real time. The recognition performance is also comparable with the state of art in comparable set ups.

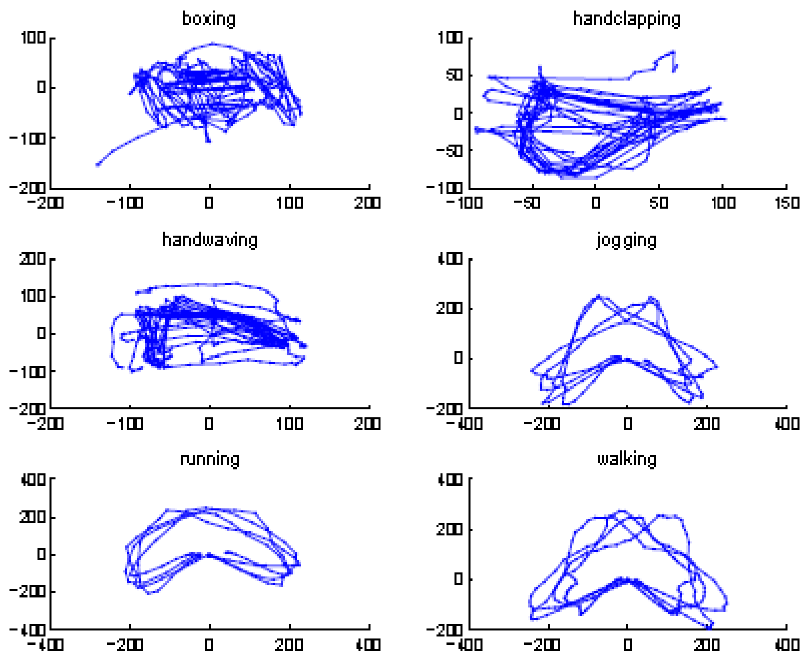


Figure 2-18 Human Action Luminance Field Trajectory Examples [36]

Besides, *Lamberto Ballan et al* used the same data set to categorize actions using an effective combination of a new 3D gradient descriptor with an optic flow descriptor, to represent Spatio-temporal interest points [37].

Now to the use of different action representation, *Gu Junxia et al* presented multiview action recognition system but with different action representation [36]. The human action is represented as a sequence of joints in the 4D spatio-temporal space (Fig. 2.18), and modeled by two HMMs, a conventional HMM for global movement feature and an exemplar-based HMM for configuration feature. Firstly, an adaptive particle filter is adopted to track the marker-less actor's 3D joints. Then, the combined features are extracted from the full body tracking results. Finally, the actions are classified by fusing two HMMs. The system was tested against an action dataset from IXMAS dataset [38]. They chose 7 actions, performed by 12 actors, each 3 times (Figure 2.19) . The volume of interest is divided into 64×64× 64 voxels (i.e. *voxel* is a volume element. This is analogous to a pixel, which represents 2D image data)

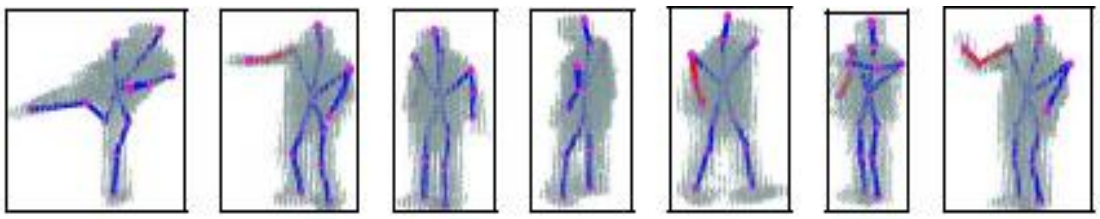


Figure 2-19 3D actors and their joints (from left to right their actions are "kick", "point", "stand", "turn around", "walk in a circle", "check watch" and "wave".)

2.2.2 Team work Activity Recognition

As another territory of action recognition research area, Team work activity recognition has been an interesting point which can be applied in fields including nursing, sports and military.

Regarding Nursing home environment (Figure 2.20), *Datong Chen et al* presented multi-modal system for detecting human activity and interaction patterns [39]. They defined a four-level context hierarchy for representing daily activities of patients, staff, and visitors. From bottom to top, the four levels are conceptual element (CE), individual person activity event (IE), group activity feature (GF) and group event (GE).

Figure 2-20 Social activity in nursing environment

Extending these applications, *Florent Fusier et al* presents a real-time video understanding system which is able to recognize activities in surveillance system in Airports (i.e. open door environment) [40].

Gita Sukthankar et al presented a system for recognizing team actions generally [41]. To represent team behaviors in the domain of MOUT (Military Operations in UrbanTerrain) planning, they employed two classes of spatial models: 1) team templates that encode static relationships between team members and external landmarks; and 2) spatially-invariant Hidden Markov Models (HMMs) to represent evolving agent team configurations over time. These two classes of models can be combined to improve recognition accuracy, particularly for behaviors that appear similar in static snapshots. One step further, *Linus J. Luotsinen et al* presented two contributions which together can significantly increase the robustness of teamwork activity recognition [42]. First they introduced a technique to reduce high dimensional continuous input data to a set of discrete features, which capture the essential components of the team actions. Second, they prefixed the actual team action recognition with a role recognition module, which in turn allows them to present the recognizer with arbitrarily shuffled input, and still obtain high recognition rates.

2.2.3 **Sensor Network based Activity Recognition**

Sensor Network based activity recognition mainly depends on distribution of wireless sensors and then sensor data is retrieved from which features are extracted and consequently action can be recognized. In this way, *Jie Yin et al* presented a novel two-phase approach for detecting abnormal activities based on wireless sensors attached to a human body [43]. Recognizing abnormal activities is a particular important task in security monitoring and healthcare applications of sensor networks, among many others. Their approach first employs a one-class support vector machine (SVM) that is trained on commonly available normal activities, which filters out the activities that have a very high probability of being normal. They then derived abnormal activity models from a general normal model via a kernel nonlinear regression (KNLR) to reduce false positive rate in an unsupervised manner. Later on, *Eric Guenterberg et al* presented a technique inspired by continuous speech recognition that combines segmentation and classification using Hidden Markov Models [44]. This technique is distributed and only involves limited data sharing between sensor nodes, as illustrated in Figure 2.21.

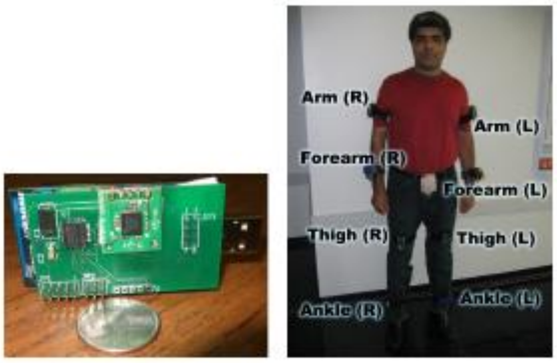


Figure 2-21 Sensor node and placement

2.3 GPU computing on computer vision

GPU provides a streaming, data-parallel arithmetic architecture. This type of architecture carries out a similar set of calculations on an array of image data. The single instruction, multiple-data (SIMD) capability of the GPU makes it suitable for running computer vision tasks, which often involve similar calculations operating on an entire image.

Special-purpose computer vision hardware is rarely found in typical mass-produced personal computers. Instead, the CPU is usually used for computer vision tasks. Optimized computer vision libraries for the CPU often consume many of the CPU cycles to achieve real-time performance, leaving little time for other, non-vision tasks.

GPUs, on the other hand, are found on most personal computers and often exceed the capabilities of the CPU. Thus, GPU can be used to accelerate computer vision computation and free up the CPU for other tasks. Furthermore, multiple GPUs can be used on the same machine, creating architecture capable of running multiple computer vision functions in parallel [45]

Computer vision tasks are computationally intensive and repetitive, and they often exceed the real-time capabilities of the CPU, leaving little time for higher-level tasks. However, many computer vision operations map efficiently onto the modern GPU.

Applications of specialized GPU design for computer vision application include Hyper-spectral image processing [46] where Javier Setoain et al presented an investigation of multi-level parallel implementations.

2.3.1 **Object detection and tracking on GPU**

In the direction of object detection and tracking, *Qian Yu et al* [47] achieved significant acceleration of motion detection of moving framework based on GPU which leads to building background model and detecting motion regions at 18fps 320x240 videos. *Peihua Li et al* presented parallel implementation of mean shift tracking [48] on GPU as 6 kernel functions. They make use of K-means clustering to partition color space into distribution of small pins; consequently all key components were mapped onto GPU with great speedup.

2.3.2 **Machine learning on GPU**

A lot of research has been done towards enhancing performance of detection, tracking and segmentation phases. Furthermore, noticeable effort has been exerted to achieve significant improvement in implementation of machine learning algorithms as a result of its great contribution in various computer vision applications.

A) GPU HMM (Hidden Markov Model)

Chuan Lui presented GPU implementations of HMM [49]. He analyzed the parallelism in the three algorithms for HMM training and classification, i.e. forward algorithm, Viterbi algorithm, and Baum-Welch algorithm, for graphical processing units (GPU). Based on the analysis, a prototype program for HMM training and classification was implemented on the NVIDIA CUDA platform. The evaluation shows for forward algorithm, His CUDA implementation achieves performance of 23.3 GFLOP/s and has up to 800× speedup over the implementation based on single core CPU; and for Baum-Welch algorithm, the performance is 4.3 GFLOP/s and there is also 200× speedup over CPU implementation.

B) GPU SVM (Support Vector Machine)

Austin Carpenter presented GPU implementation of SVM by Austin Carpenter [50]. Due to its impressive generalization performance on a wide range of statistical prediction problems, the Support Vector Machine (*Cortes and Vapnik*, 1995) has been, for several years now, the most widely used kernel learning algorithm; however, the computationally expensive nature of training SVMs and, to a lesser extent, using them to predict new data points has limited the amount of training data that researchers have been able to employ. Moreover, the common practice of using cross-validation to select the SVM nuisance parameters (C, λ in classification and C, λ, ϵ in regression) most suitable to a particular prediction problem compounds the SVM's computational cost and puts further constraints on the feasible size of training sets. cuSVM brings the massively parallel processing power of GPUs to bear upon this problem, and as a result, its training times on a set of benchmarks were 12-73x faster than those of arguably the most popular CPU solver, LIBSVM (Chang and Lin, 2001). Its performance gains in prediction, at 22-172x, were even greater. In these tests, cuSVM was run on a NVIDIA GTX 260 GPU.

2.3.3 Computer vision libraries on GPU

This subsection presents computer vision libraries that were developed to achieve relative speedup to the CPU sequential implementation. This was done in terms of general library contain various computer vision functions on GPU. Hence computer vision system developer can use this functions as successive calls, however this libraries cannot ensure optimal speedup at least in terms of communication overhead of copy data between CPU and GPU for each call. The advantage here is the speedup in relatively no-time against special purpose GPU design.

A) OpenVidia

James Fung et al presented OpenVidia [51]. OpenVidia is a library and API aimed at providing a graphics hardware accelerated processing framework for image processing and computer vision. OpenVidia explores the creation of a parallel computer architecture consisting of multiple Graphics Processing Units (GPUs) built entirely from commodity hardware (Figure 2.22). OpenVidia uses multiple Graphics Processing Units in parallel to operate as general purpose parallel computer architecture. It provides a simple API which implements some common computer vision algorithms. Many components can be used immediately and because the project is Open Source, the code is intended to serve as templates and examples for how similar algorithms are mapped onto graphics hardware. Instances of implemented functionality includes image processing techniques (Canny edge detection, filtering), image feature handling (identifying and matching features) and image registration

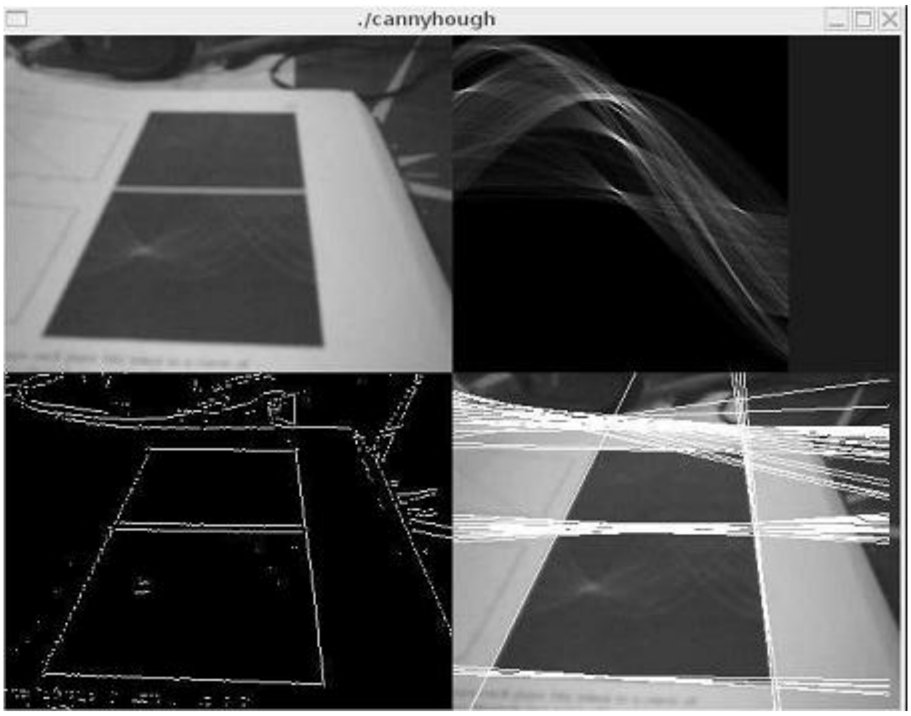


Figure 2-22 Screen grab of a GPU Accelerated Radon/Hough transform that uses the vertex (geometry) processing, rasterizer (line/primitive drawing) and fragment processing (per-pixel operations) accelerated by the GPU pipeline.

B) GPUCV

GPUCV is an extension of Intel's Open CV [52] library that is fairly popular for developing interactive computer vision applications. It is meant to support GPU acceleration for applications written with Open CV without modifying the application source code [53].

Jean-Philippe Farrugia et al developed GPUCV, an open library meant for easy development of GPU accelerated image processing and computer vision applications. First, it allows easy shaders handling. A "Filter Manager" class provides for shaders creation and application on images. Since OpenGL and the GLSL shading language are used for shaders, any system supporting OpenGL 2.0 will support the interface. It supports not only the fragment shader kernel scheme, but also using vertex shaders, which are rarely used for image processing and vision algorithms on GPU (unlike the fragment shaders, they allow to modify the coordinates transmitted through the graphics pipeline). Shaders can be dynamically generated as meta-shaders which allow flexibility in setting up possibly many parameters. The programming interface proposes to use any geometry for shader computing to allow geometric methods. Other libraries generally impose a screen aligned quad mapped with the processed image. All these characteristics are unique to GPUCV and cannot be exploited with OpenVidia. Besides, GPUCV is an extension of

Intel’s Open CV library that is fairly popular for developing interactive computer vision applications. It is meant to support GPU acceleration for applications written with Open CV without modifying the application source code.

3 Real time Muslim prayer action recognition system

This chapter presents a system capable of recognizing major postures and actions performed by a Muslim normally during the prayer, an unexplored area in the Action Recognition field. The chapter presents the details of the system implementation phases and the proposed techniques used for detection. The system was tested against a custom data set and an accuracy exceeding 93.5% was recorded.

This piece of work achieves the objective as a novel accurate action recognition system, but also as a real time system solving confronting performance problems. Besides, this system opens a new direction of research in action recognition by focusing on modeling Muslim prayer actions. The presented work can serve as a bootstrap for embodying this territory in action recognition interests.

Applications of this system include teaching children how to pray. It also can be used by the elder to determine whether something is missing in their prayer. Besides, this work can help the future to have an embedded product that can be used anywhere for the same purpose. In summary, this system serves as an interactive way of teaching children and recent Muslims.

3.1 Prayer Actions Recognition System

Most video-processing based systems encountered in practice contain a basic module known as the front end. Our prayer actions recognition system is not an exception. The action recognition phase is implemented as a specialized module using a custom model developed specifically to fit this purpose.

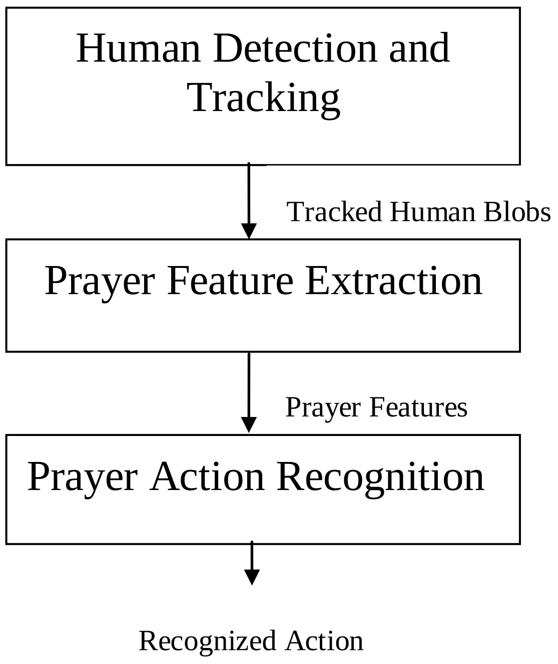

Human Detection and Tracking
Tracked Human Blobs
Prayer Feature Extraction
Prayer Features
Prayer Action Recognition
Recognized Action

Figure 3-1 System Stages

Fig. 3.1 shows the main stages of our system. The first phase detects and tracks the human by producing a sequence of blobs (section 3.1.1). The second phase (section 3.1.3, 3.1.4) extracts a set of discriminated prayer features assuming a constrained environment (section 3.1.2). These features are the keys for recognizing prayer actions. Finally, the prayer recognition phase tracks the features to classify the motion (section 3.1.5).

3.1.1 **Human Detection and Tracking**

Object detection and tracking is one of the common phases for any video-processing based system. Different techniques may be used to satisfy the same goal. The process flow of the technique used in the *front end* processing phase is shown in Fig. 3.2 After the image is acquired, a background model is used to define the background and hence the foreground can be easily detected. After the foreground image has been detected, the Connected Components algorithm is used to generate blobs for the objects in the foreground. Such blobs are checked against human properties. If it is confirmed as a human blob, it is passed to the tracking phase responsible for correspondence mapping through a sequence of frames [5].

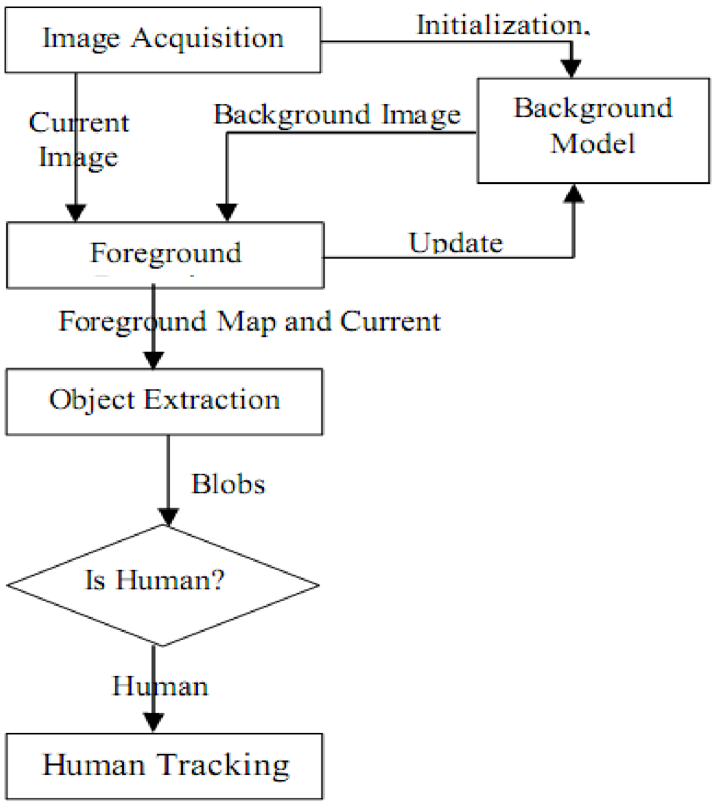


Figure 3-2 Human Detection and Tracking **[5]**

3.1.2 **Prayer Recognition Environment Setup**

The recognition phase assumes that the camera is capturing a side view of the prayer, whether to the left or to the right. Also the prayer must be totally covered in the scene, as illustrated in Fig. 3.3.

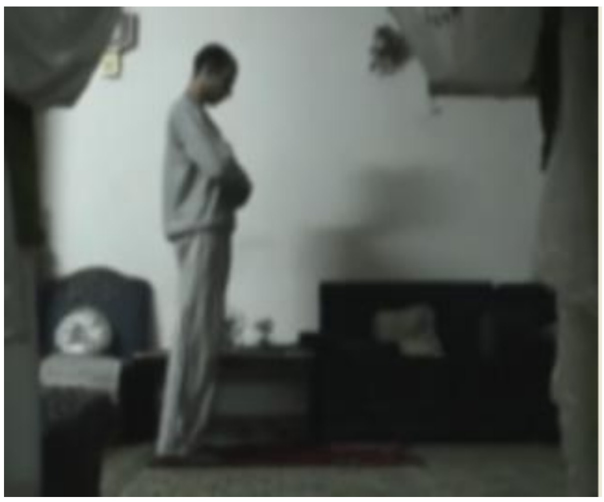


Figure 3-3 Environment Setup

3.1.3 **Prayer Feature Extraction**

During this phase, a set of key features are extracted for discriminating different postures of the prayer. The features depend on some geometric parameters of the prayer's body; including the bounding rectangle, the center of mass, and the backbone axis angle.

Backbone Axis Angle: Backbone axis angle θ_b is the angle between the y-axis and the back-axis of the detected body as illustrated in Fig. 3.4 This feature was used as an indicator to the degree of inclination.

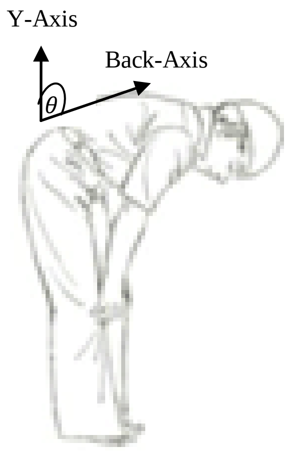
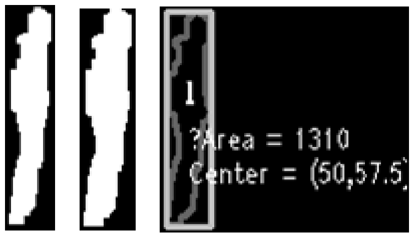


Figure 3-4 Backbone Axis Angle

The approach for the detection of this angle depends on the extracted contour around the body that is analyzed to determine the key points used directly to detect the backbone axis angle θ_b as detailed in the following sections.

A) Contour Extraction

Contour extraction was implemented using dilation on the blob, followed by subtracting the dilated blob from the one before dilation, as shown in Fig. 3.5.



(a) (b) (c)

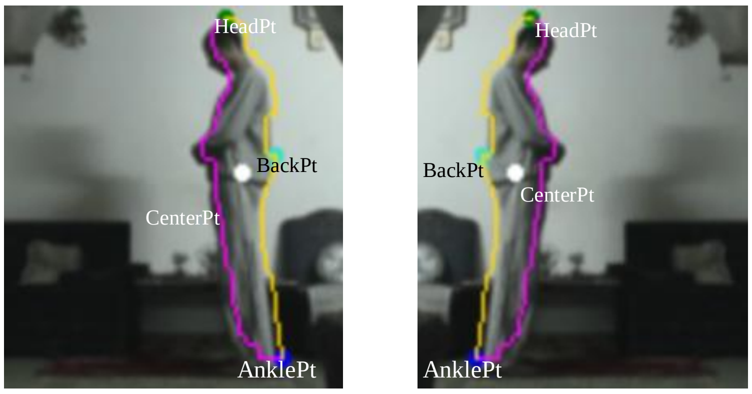
Figure 3-5 (a) Blob [I1] (b) Dilated Blob [I2] (c) Contour [I2–I1]

B)Key Points Extraction:

The backbone axis angle θ_b is to be used in action recognition. For this goal, a set of key points is detected as shown in Fig. 3.6. The key points are Center Point (CenterPt), Ankle Point (AnklePt), and Head Point (HeadPt) Such key points are extracted after analyzing the extracted contour. The computational logic used for the detection of each key point follows.

1) CenterPt

CenterPt is estimated by only getting the mean of points that belong to the contour.



(a)Left view **(b) Right View**

Figure 3-6 Key Points (Center Point, Bottom Point, Back Point and Head Point)

2) AnklePt

AnklePt is the point on the contour with y equals/tends to the minimum and farthest/nearest x depending on the orientation of the prayer (left or right) as shown in Fig. 3.6.

$$AnklePt = arg\ min_{Pti}\left(\begin{matrix} Pt_i.Y - Y_{\min}: \\ (Pt_i.Y - Y_{\min}) < 0.03 * prayerHeight \end{matrix}\right) \quad \text{(Eq 3.1)}$$

.

3) HeadPt

HeadPt is the point of the head on the contour, modeled as the point with maximum Euclidean Distance from *AnklePt*.

$$HeadPt = \arg\ max_{Pti}\left(dist(Pt_i, AnklePt)\right) \quad \text{(Eq 3.2)}$$

3) BackBone

Points on the back of the prayer are either points from HeadPt to AnklePt or from AnklePt to HeadPt. Fig. 3.7 shows the directed line segment AnklePt → HeadPt. Arrows in fuchsia represent the directional projections of the 1st set of points (from HeadPt to AnklePt) on the line segment.

Arrows in yellow represent the corresponding projections for the 2nd set (from AnklePt to HeadPt). To determine which set of points is the right set, the mean of projection is calculated for each set. Points on the back are determined as points of maximum mean of projection.

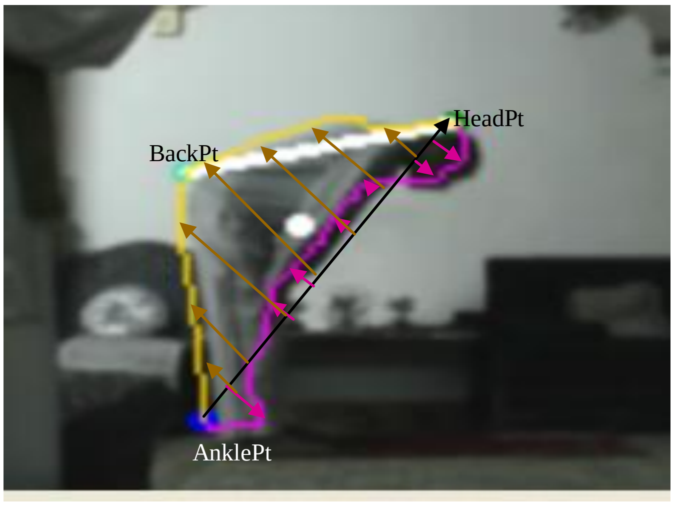


Figure 3-7 Projections of the Two Sets: From AnklePt to HeadPt, From HeadPt to AnklePt

Backbone axis angle is the angle between the Y-Axis and the vector along the back (***BackVector***) in the direction of *HeadPt* (Fig 3.8)

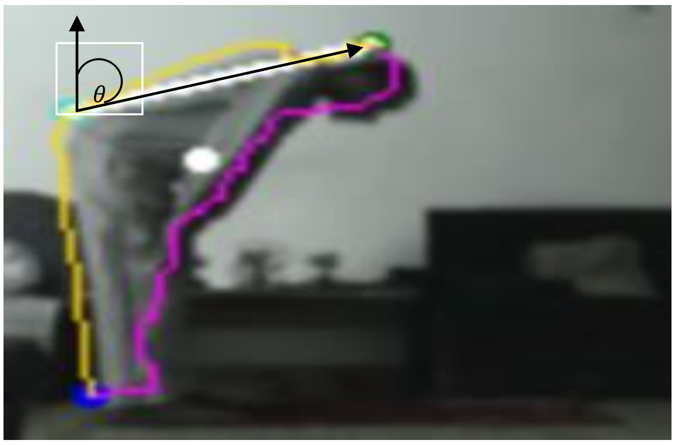


Figure 3-8 Points of interest: Bottom Point, Back Point and Head Point

$$\theta_b = acos\left(\frac{(\mathbf{BackVector}) \cdot (\mathbf{YAxis})}{|\mathbf{BackVector}| * |\mathbf{YAxis}|}\right) \qquad \text{(Eq 3.3)}$$

Regarding the detection of the *BackVector,* There were two methods that were achieved through the work in this part of thesis. That process was a key issue of performance enhancement.

3.1.4 **Back Vector Extraction**

This section presents the two methods that were implemented to detect BackVector from which backbone axis angle is extracted using Equation 3.3. The section shows the evaluation of the two approaches in terms of performance.

Method1

BackVector was modeled by the following steps. The first to solve is to find the line segment that presents the back of the prayer. Then *BackVector* is calculated as vector parallel to the calculated line segment in the direction of the *HeadPt*. The Back line segment is modeled as a segment within specific range of pixels (depends on the height of the prayer in pixels). This line segments should minimize the average projection length and at the same time as possible reaching the maximum number of points in the range. The steps are detailed as follows.

A) Linear correlation minimization

As shown in Fig 3.9, To find the line equation (i.e. to find a,b,c which formulates a x + b y + c = 0) which minimizes projection distance from points to the line.

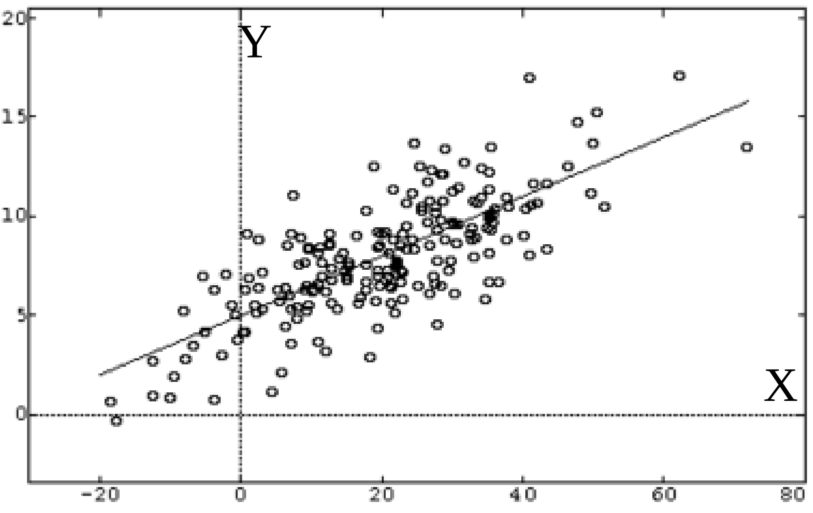


Figure 3-9 Linear correlation

The problem is solved as follows by building the covariance matrix which is as follows

$$\text{CovarianceMatrix} = \begin{bmatrix} \sum_{i=1}^{N} Xi^2 & \sum_{i=1}^{N} Xi * Yi & \sum_{i=1}^{N} Xi \\ \sum_{i=1}^{N} Xi * Yi & \sum_{i=1}^{N} Yi^2 & \sum_{i=1}^{N} Yi \\ \sum_{i=1}^{N} Xi & \sum_{i=1}^{N} Yi & N \end{bmatrix} \quad \text{(Eq 3.4)}$$

Then performing Eigen decomposition of this matrix from which a, b, c are extracted as the Eigen vector (which is of length 3) with the minimum Eigen value (if the all points belongs to the same line the smallest Eigen value equals zero).

Then inverse correlation coefficient (denoted as γ) in our case is presented as follows (which is interpreted as the average projection of the points on the line)

$$\gamma = \frac{\sum_{i=0}^{N} Abs\left(\frac{a * Xi + b * Yi + c}{a^2 + b^2}\right)}{N} \quad \text{(Eq 3.5)}$$

B) Back Vector extraction

To Extract the Back Vector, Firstly back line is extracted as detailed in the pseudo code in Fig 3.10.

```
input: List of points on human contour from head point to the ankle point but on the
back direction (i.e. points decided on the back as shown in section 3.2.3
output: a, b, c representing the back line equation ( a x + b y +c = 0)


Min $\gamma$ = $\infty$
for L= 0.25*H to 0.5*H
comment: where H is the prayer Height
begin
        for I = 0 to N-L comment: where N is the number of points along the back
        begin
                Calculate $\gamma(i, L)$
                comment: $\gamma(i, L)$ is the correlation coefficient of points starting from I
                with Length L taking O(N) as clear from Equation 3.11
                if ($\gamma(i, L)$ < Min $\gamma$) then
                begin
                        Min $\gamma$ = $\gamma(i, L)$
                        Save points of Min $\gamma$ .
                        Save a,b,c
                        comment: where a, b, c are coefficients of the line equation ( a x +
                        b y +c =0)
                end if
        end for
end for
```

Figure 3-10 Backline extraction

Secondly, the *BackVector* is extracted by Eq 3.6

$$BackVector = (HeadPrPt - BackPrPt) \quad \text{(Eq 3.6)}$$

Where *HeadPrPt* is the *HeadPt* projection on the line, *BackPtPrPt* is the *BackPt* projection on the line (i.e. *BackPt* here a point belonging to points of maximum correlation that is the furthest from the *HeadPt*). Projection of a point (x0,y0) on a line (a x + b y +c = 0) is calculated as in Equation 3.7.

$$PrPt.X = x0 + p * a,\ PrPt.Y = y0 + p * b\ ,\ p = \frac{-(a*x0 + b*y0 + c)}{a^2+b^2} \quad \text{(Eq 3.7)}$$

The complexity of the algorithm is $O(H * N^2)$. the algorithm was tested on a 3.2GHZ dual core processor, 2GB Memory machine and the overall system performance was recorded as 2 frames per second *which is* away from real time requirements.

Method2

The BackVector is modeled in two steps. Firstly the BackPt is extracted, then using BackVector is extracted using BackPt and HeadPt.

A) BackPt Extraction

BackPt is a point on the back points of the prayer with the angle *HeadPt-Pt*$_i$*-AnklePt* minimized such that distance to *HeadPt* is near half of the prayer height (note that *Pt*$_i$ is an arbitrary point on the contour).

$$\theta_i = acos\left(\frac{(HeadPt - Pt_i)\cdot(AnklePt - Pt_i)}{dist(Pt_i, HeadPt) * dist(Pt_i, AnklePt)}\right)$$

$$M_i = dist(Pt_i, HeadPt) - dist(Pt_i, AnklePt)$$

$$A_i = dist(Pt_i, HeadPt) + dist(Pt_i, AnklePt)$$

$$BackPt = arg\ min_{\ Pti}(\theta_i : |M_i| < \gamma * A_i) \quad \text{(Eq 3.8)}$$

Where · denotes the dot product between $HeadPt - Pt_i$ and $AnklePt - Pt_i$ vectors. The best estimate value of γ is 0.1.

B) Back Vector Extraction

The BackVector the vector pointing from BackPt to the HeadPt as shown in Eq 3.9.

$$BackVector\ = HeadPt - BackPt \quad \text{(Eq 3.9)}$$

The Time complexity of the implemented $algorithm$ was $O(N)$. The algorithm was tested on 3.2GHZ a dual core processor, 2GB Memory machine and the overall system performance was recorded as 15 frames per second , which in turn make it serve real time requirements needed for any application based on prayer action recognition.

3.1.5 Prayer Actions Recognition

Prayer actions recognition represented here mainly depends on features of the backbone axis angle, the width and the height of the prayer in a side view, and the center of mass. In this section the mathematical logic of the Qiam, Ruku, Jalsa and Sujud postures are presented.

A) Qiam Posture

Qiam is the initial posture of the prayer, as illustrated in Fig. 3.11. It is recognized when the backbone axis angle θ_b is equal to 0 with a tolerance of ε°, and CenterPt.Y is equal to half of the human height with another tolerance α (Equation 3.9).

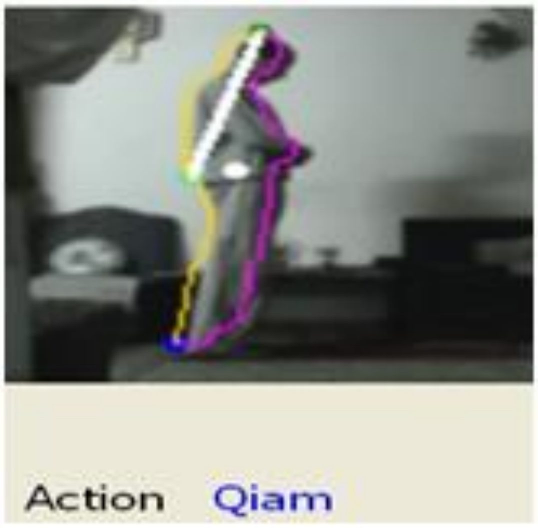


Figure 3-11 Qiam Posture

$$Qiam:$$

$$\theta_b \leq \xi,$$

$$CenterPt.Y = (0.5 \pm \alpha) * prayerHeight \qquad \text{Eq(3.9)}$$

The best estimate for ε and α were found to be 23° and 0.1, respectively.

B) Ruku Posture

Ruku is the posture of putting the hands on the knees in the manner illustrated in Fig. 3.12.

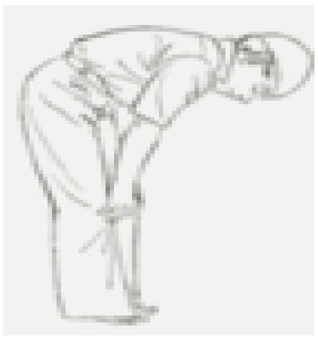
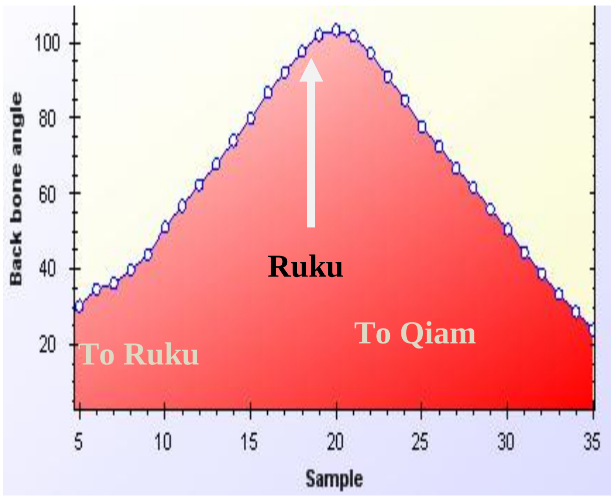
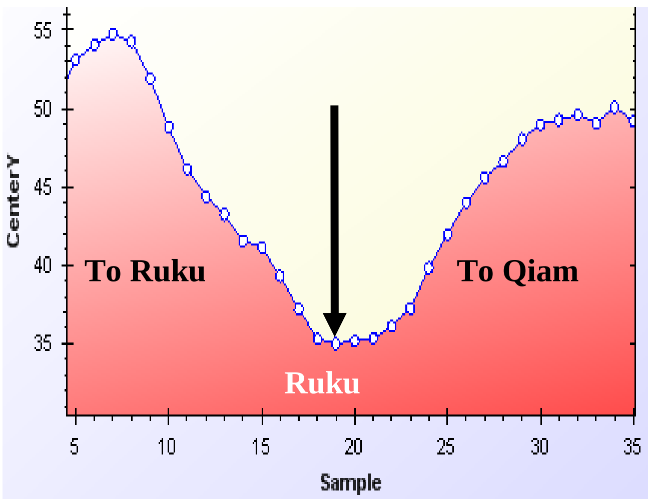


Figure 3-12 Ruku

In order to recognize the *Ruku* posture, the backbone axis angle θ_b and *CenterPt.Y* were recorded while the prayer was moving from *Qiam* to *Ruku*, then back from *Ruku* to *Qiam*. Fig. 3.13 shows how θ_b and *CenterPt.Y* are changing across the frames of the *Ruku* posture.



a) Backbone axis angle θ_b against sample/time



b): CenterPt.Y against sample/time

Figure 3-13 θ_b and CenterPt.Y across frames

As illustrated in Figure 3.13 (a), θ_b increases from Qiam to reach around 90° on Ruku then decreases from Ruku to Qiam to be near 0 .Simultaneously, Y-coordinate of *CentertPt* decreases from Qiam to Ruku, then increases back to Qiam (Figure 3.13 b).

Having analyzed that observation in Figure 3.13, recognition of the *Ruku* posture goes as follows:

1) From Qiam to Ruku Action: Transition from Qiam to Ruku is detected when the backbone axis angle θ_b is increasing while CenterPt.Y is decreasing (Fig. 3.14).

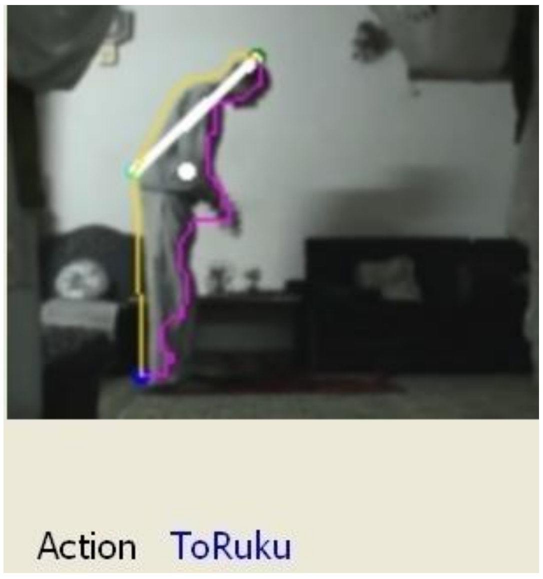


Figure 3-14 To Ruku Action

$$ToRuku:$$

$$\theta_b \ is \ increasing: Th1 < \theta_b < Th2,$$

$$CenterPt.Y \ is \ decreasing: CenterPt.Y > Th3 \quad \text{Eq(3.10)}$$

The best estimate for *Th1*, *Th2* and *Th3* were found to be 23°, 77° and 0.28 * *prayer-Height*, respectively.

2) Ruku Posture: Ruku posture is identified by having the backbone axis angle within 90° and CenterPt.Y within a suitable range to distinguish it from the Sujud posture (Fig. 3.15).

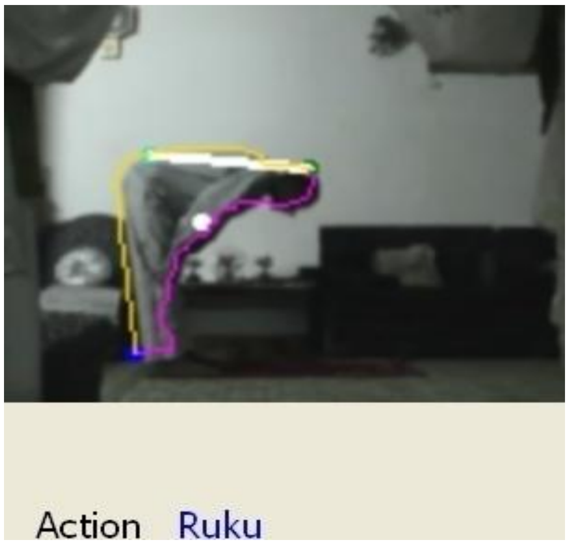


Figure 3-15 Ruku Posture

$$Ruku:$$

$$Th1 < \theta_b < Th2,$$

$$:CenterPt.Y > Th3 \qquad \mathrm{Eq}(3.11)$$

The best estimate for *Th1*, *Th2* and *Th3* were found to be 77°, 115° and 0.22 * *prayer-Height*, respectively.

3) From Ruku to Qiam Action: Transition from Ruku to Qiam is detected as the backbone axis angle θ_b is back decreasing while CenterPt.Y is increasing (Fig. 3.16).

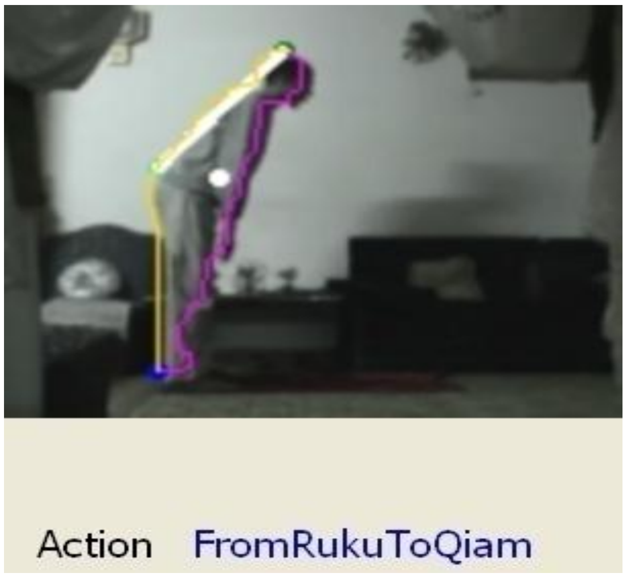


Figure 3-16 From Ruku to Qiam Action

$$FromRukuToQiam:$$

$$\theta_b \text{ is decreasing: } Th1 < \theta_b < Th2,$$

$$CenterPt.Y \text{ is increasing: } CenterPt.Y > Th3 \qquad \text{Eq(3.12)}$$

The best estimate for Th1, Th2 and Th3 were found to be 23°, 77° and 0.28 * prayerHeight, respectively.

C) Sujud and Jalsa Postures

Change of the backbone axis angle θ_b from Sujud and Jalsa have the same behavior as transitions from Ruku and Qiam transitions (i.e. increasing then decreasing). The only change is in the range. CenterPt.Y change behavior is the opposite (i.e. decreasing then increasing).

1) Sujud Posture: The detection of the Sujud posture (Fig. 3.17) is similar to the detection of the Ruku posture. Backbone axis angle θ_b is checked to be greater than a certain threshold while the y-coordinate of the center of mass achieves its minimum.

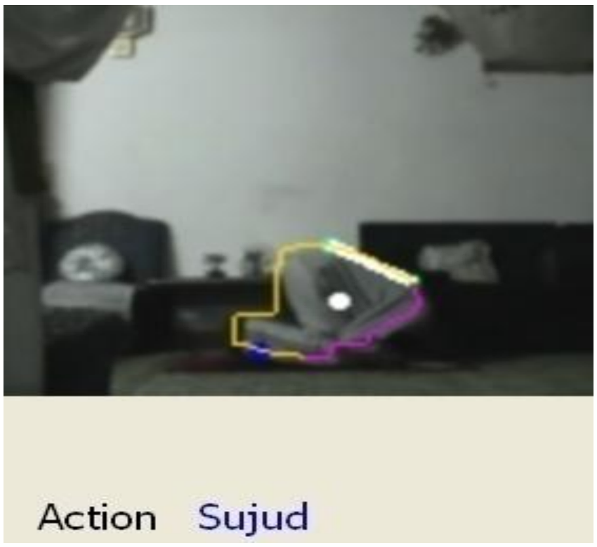


Figure 3-17 Sujud Posture

$$Sujud:$$

$$\theta_b > Th1,$$

$$CenterPt.Y < Th2 \quad Eq(3.13)$$

The best estimate for Th1 and Th2 were found to be 115° and 0.2 * prayerHeight, respectively.

2) Jalsa Posture: The detection of Jalsa posture is similar to the detection of the Qiam posture. The backbone axis angle θ_b is somewhat small as Qiam, the y-coordinate of the center of mass is within a range greater than the degree of Sujud and less than a certain threshold (Fig. 3.18).

$$Jalsa:$$

$$0 < \theta_b < Th1,$$

$$Th2 \le CenterPt.Y < Th3 \quad \text{(Eq 3.14)}$$

The best estimate for Th1, Th2 and Th3 were found to be 20°, 0.2 * prayerHeight and 0.28 * prayerHeight, respectively.

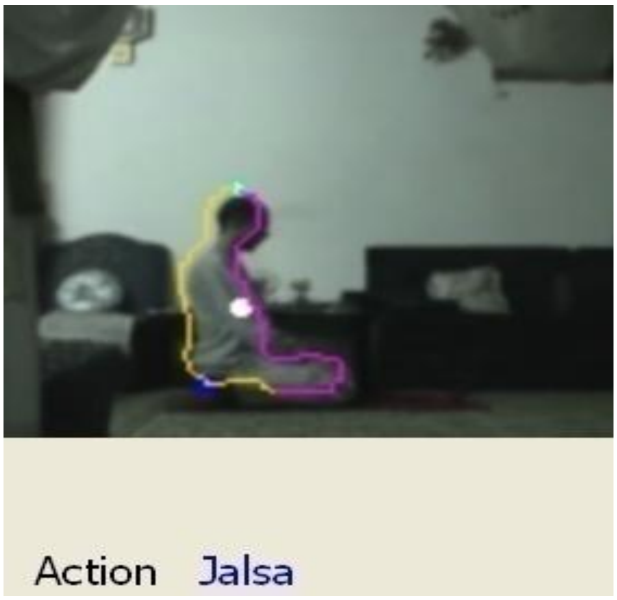


Figure 3-18 Jalsa Posture

Discussion about estimations in action recognition phase is presented in Appedix A.

3.2 Experimental Results

The first problem that faced us to evaluate this approach was the absence of a data set for this type of actions. So the proposed model has been tested against a custom dataset of about 26 videos with varying prayer parameters. Parameters include physical properties such as fat, thin, short, tall, young and old prayers; also the model has been tested against different environments, such as at home and in the mosque.

We measure how accurate the system is by determining the percentage of correct recognition of a particular action against the total number of occurrences of that action.

The results of the system tests follow from

$$\frac{\sum_{i=1}^{nV} \frac{nC_i}{nA_i}}{nV} \qquad \text{(Eq 3.15)}$$

where nC_i is the number of frames where a posture is recognized correctly (i is the video index), nA is the total number of frames of that particular posture in video i, and nV is the number of videos (in our case, 26). This equation is calculated for every posture. The results are listed in Table 4.1.

Posture	Accuracy
Qiam	98.7%
ToRuku	93.7%
Ruku	94.71%
FromRukuToQiam	97.8%
Sujud	99.51%
Jalsa	99.83%

Table 3-1 System Accuracy

Regarding performance, real time processing was achieved by Method 2 for the extraction of *BackVector* .The system was tested on 3.2GHZ a dual core processor, 2GB Memory machine and the overall system performance was recorded as 15 frames per second.

The achieved results have been published in Second International Conference on Computer and Electrical Engineering [54].

4 GPU framework for action recognition

This chapter presents proposed framework based on GPUs) to achieve speed–up in the performance of activity recognition including single agent and multi-agent (team) actions. The framework can be customized for any action recognition application. Through the rest of chapter, we will refer to the target users of the framework as developers. This chapter is organized as follows. Section 5.1 presents the proposed framework. Section 5.2 shows high performance approaches in the framework. Section 5.3 shows how the framework could be configured for any system. Section 5.4 presents experimental results that report the achieved speedup.

4.1 GPU Teamwork Activity Recognition Framework

We investigated the major techniques used in action recognition and computer vision. The goal was to determine the components of general framework on which many applications can be built. Figure 4.1 presents the workflow of the proposed GPU framework. The input is raw images captured of moving camera, and the output is the recognized activity. The colored modules are the ones with general GPU implementation to be configured according to application requirements. The non-colored modules (Agent oriented feature extraction, Team oriented feature extraction) are implemented by the developer according to the potential application running under the framework. However the non-colored modules are implemented by the developer, the framework provides the developer with GPUCV functions which gives him support to achieve speedup with minimal effort. Furthermore, the framework accepts the developer implementation either on GPU or CPU.

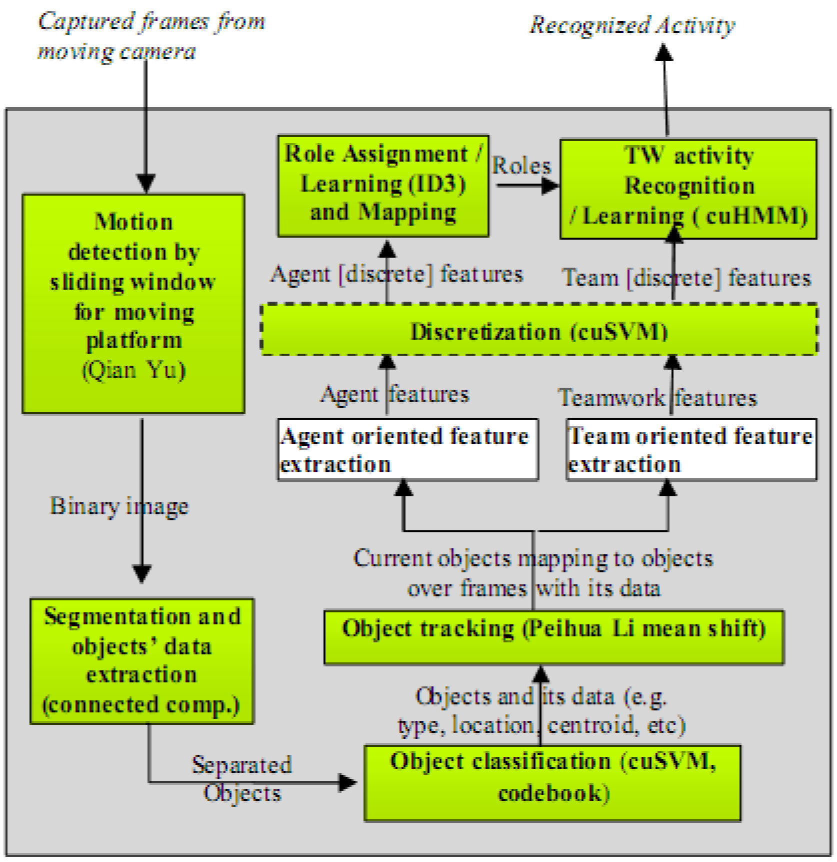

Captured frames from moving camera
Recognized Activity
Motion detection by sliding window for moving platform (Qian Yu)
Role Assignment / Learning (ID3) and Mapping
Roles
TW activity Recognition / Learning (cuHMM)
Agent [discrete] features
Team [discrete] features
Discretization (cuSVM)
Agent features
Teamwork features
Agent oriented feature extraction
Team oriented feature extraction
Binary image
Current objects mapping to objects over frames with its data
Segmentation and objects' data extraction (connected comp.)
Object tracking (Peihua Li mean shift)
Objects and its data (e.g. type, location, centroid, etc)
Separated Objects
Object classification (cuSVM, codebook)

Figure 4-1 Process flow of the GPU framework

As illustrated in Figure 4.1, the framework spans all stages required for any action recognition system with specs of each component defined on the arrows. Besides, the framework has been built such that functionality of GPUCV (i.e. GPU version of Intel OpenCV functions) [53] is supported and integrated as an infrastructure that can be used by the developer.

Next sections present some details implementation of each component in the framework that integrated together to comprise the framework.

4.1.2 Motion Detection by sliding window for Moving platform

Qian Yu et al [55] achieved significant acceleration of motion detection of moving platform based on GPU which leads to building background model and detecting motion regions at 18fps 320x240 videos. The process of detection on GPU was implemented as two steps, warping images and computing the background model. The main objective was to minimize memory transferring between n GPU and CPU.

In the GPGPU framework, the fully programmable vertex and fragment processors provides powerful computational tools for general purpose computations. In order to implement an algorithm on the GPU, different computational steps are often mapped to different vertex and fragment shaders. Vertex shaders allow us to manipulate the data that describes a vertex. Fragment shaders serve to manipulate a pixel. The GPU implementation of the sliding window based method is separated into two steps, warping images and computing the background model. Also, Minimize memory transferring between GPU and CPU is an objective, therefore the inputs for the implementation are the sequential image data and homography transformation, and the output is the motion mask (namely the difference image) for each frame. The overview structure of the implementation is shown in Figure 4.2.

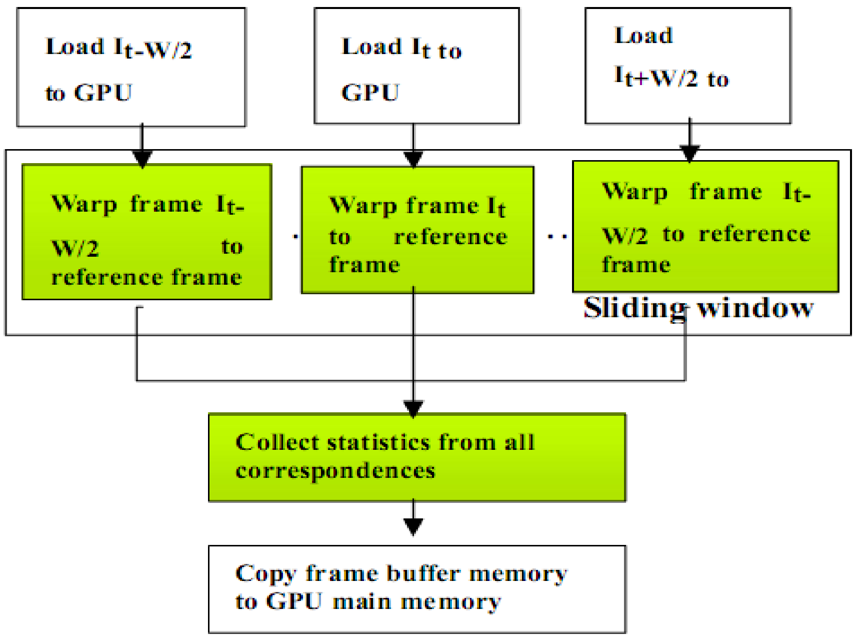

Load $I_{t-W/2}$ to GPU
Load I_t to GPU
Load $I_{t+W/2}$ to
Warp frame $I_{t-W/2}$ to reference frame
Warp frame I_t to reference frame
Warp frame $I_{t-W/2}$ to reference frame
Sliding window
Collect statistics from all correspondences
Copy frame buffer memory to GPU main memory

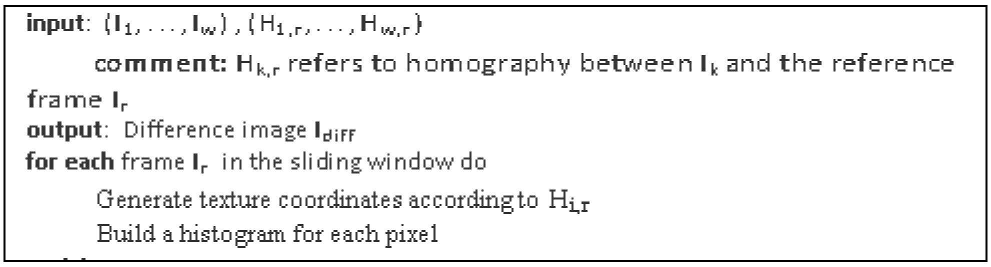
Figure 4-2 Quan Yu et al Motion detection on GPU. [55]

The implementation needs to store all frames in the sliding window. The sequential frames are stored as two- dimension textures in the GPU memory. The most recent frame is loaded as a texture and the oldest frame is moved out of the texture pool. The warping involves changing the texture coordinates and is implemented in the vertex pro- file. The vertex profile takes the 3x3 homograph matrix as input and outputs the warped texture coordinates by simple matrix multiplication operations. The transformed texture coordinates are applied in the fragment profile.

To compute the background model, different statistic functions can be applied in Eq.4.1 (i.e. f is the statistic function), such as the mode, median and the mean of the samples in the sliding window. For motion detection from a moving platform, the mode is usually preferred to the other two. This is because samples are very limited, i.e., the size of the sliding window is quite small. The mean and the median, which do not differentiate the foreground and background pixels, will lead to a biased background model. The mean and the median are only proper when enough samples are available. The computation of the mode requires to build a histogram and then to find the bin with the largest number of samples (there is another way to establish a dynamic histogram by constructing a binary search tree, which involves too many branching operations and dynamic data structures, thus it is not proper for GPU implementa- tion).

$$I_{bg} = f\left(\mathrm{H}_{\mathrm{r}-\mathrm{w}/2,\mathrm{r}}\mathrm{I}_{\mathrm{r}},\ldots,\mathrm{I}_{\mathrm{r}},\ldots,\mathrm{H}_{\mathrm{r}+\mathrm{w}/2,\mathrm{r}}\mathrm{I}_{\mathrm{r}}\right)\ (Equation\ 4.1)$$

One histogram is built for each location in the reference frame and each bin records the number of hits over the sliding window. This is different from the normal concept of a normal histogram that is built on the whole image. The overview of the GPU implementation is shown in Algorithm 1. A fixed number n of bins is used to construct a histogram.



```
input: $(I_1,\ldots,I_w)$, $(H_{1,r},\ldots,H_{w,r})$
        comment: $H_{k,r}$ refers to homography between $I_k$ and the reference
frame $I_r$
output: Difference image $I_{diff}$
for each frame $I_r$ in the sliding window do
        Generate texture coordinates according to $H_{i,r}$
        Build a histogram for each pixel
```

Figure 4-3 Qian Qu Algorithm [55]

The approach that uses the mode as the background is independent of the result at the previous time, thus it is able avoid the spread of registration errors and outliers. However, the computation of the mode is still complicated. When the registration quality is good enough, an alternative is to compute the background model. It is obvious that computing the mean is much easier for GPU implementation. Firstly, the background result at the previous time is warped to the current reference frame as an estimated background model, and the mean of the samples that are close enough to the estimated model is used as the new background model. If the number of the samples that are close enough to the estimated background is too small, we use the average of all samples instead.

4.1.3 Segmentation and objects' data extraction.

In this stage GPU *connected component* is applied to segment objects in the scene (Figure 4.4). The speedup is gained by dividing image into N blocks then merging the computations [56]. But experiments show that the cost of this algorithm depends on the size and complexity of the labeled objects, so this technique may inherently be not faster than the CPU classical 2-passes sequential algorithm implemented using Intel OpenCV version

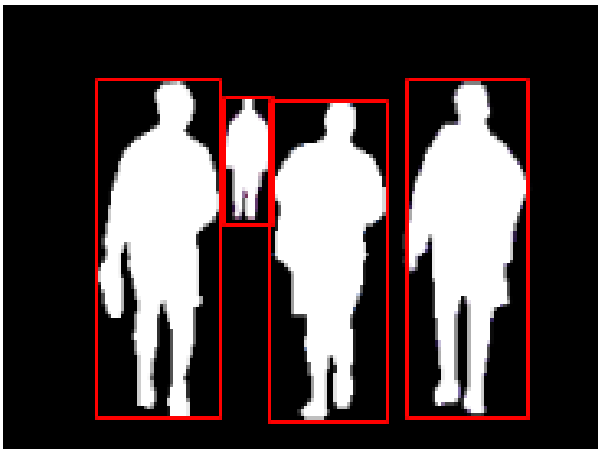


Figure 4-4 Segmentation and object extraction

Connected component on GPU of several discrete processing stages. First, each pixel in the input image is assigned unique ID containing (x, y) information for their particular 2D location in the image. Next, a process similar to run-length encoding is performed to create series of connected pixels down each column of the image. Each of these vertical runs is then merged with adjacent ones across the image, from right to left, to form the final equivalence classes, which produce a set of connected components with unique IDs. Finally the results are downloaded from the GPU to the CPU. These steps are listed below.

- Compute vertical runs
- Column processing
- Post column processing
- Compute bounding boxes
- Retrieve results

The OpenGL API and GLSL shading language are utilized in this work and all processing is performed via the processing of vertices and fragment rasterization. No general purpose libraries are used, such as NVIDIA's CUDA, ATI's Stream SDK, or OpenCL.

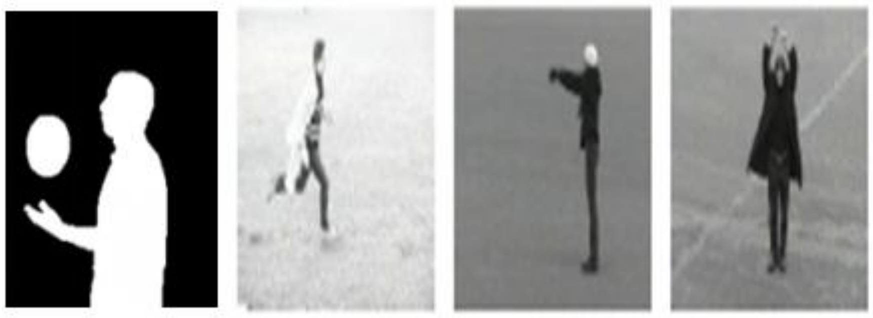
Shaders: OpenGL vertex and fragment shaders are used throughout this implementation. Various stages of this work's algorithm operate primarily on vertices in the vertex shader, while other stages perform all processing in the fragment shader. A variety of different shaders are used for each distinct stage, as well as different shaders within sub step of each stage.

Textures: two texture formats are used in that work for storing pixel ID values as well as bounding box information. For the pixel IDs the GL_LUMINANCE32F format was used. Only the red channel is utilized for storing pixel IDs. This makes limitation on utilizing only the 23-bit mantissa portion of each floating point number since this work stores (x, y) information in each value and also relies on blending operations. It is worth noting here that due to this limited number of usable bits, this work is only able to process image resolutions up to 2047x2047. The GL_ALPHA32F format would have been preferable for storing pixel

IDs since this format consumes 25% of the memory, but testing showed the NVIDIA 8 and 9 series GPUs to be inefficient at rendering to single channel alpha texture targets. Also utilize the GL_RGBA16F texture format is utilized for storing bounding boxes, where each (x1, y1, x2, y2) bounding box coordinate is stored in the 4 RGBA channels.

4.1.4 **Object classification (cuSVM , codebook)**

This stage of processing is responsible for categorizing detected objects into classes. Speedup of this stage can be achieved by multilevel implementation. For instance, Jianpeng Zhou et al presented codebook algorithm to classify humans [57], while Vili Kellokumpu et al used SVM in [14] to classify human postures (Figure 4.5). So codebook can be used to ensure filtering out non- human objects, then human blobs is presented to SVM to classify human postures.



(a) **(b)**

Figure 4-5(a) Classification of object as a ball and human (b) Classification of different human postures.

So for that purpose GPU implementation of SVM is integrated. The root of cuSVM's training performance advantage over CPU-based solvers such as LIBSVM is the parallelism that can be exposed in the most computationally intensive part of the modified SMO (Sequential Minimal Optimization) training algorithm, the calculation of the two rows of the kernel matrix corresponding to the indices in the working set. The row corresponding to ihigh is used by the second order working set selection heuristic and both are necessary for the update of the KKT conditions (3.6). As is noted in [58], $-\lambda * |x-y|^2$ from the Gaussian kernel (3.2) can be expressed in terms of matrix-vector multiplication as $-\lambda * (2 * x.y - x.x - y.y)$. Before initiating the modified SMO decomposition [50], cuSVM computes on the CPU $x_i \cdot x_i$, i = i, ..., m, and transfers the result to GPU memory. Then, during training iterations, cuSVM uses NVIDIA's CUDA basic linear algebra library, CUBLAS, to compute the necessary $x \cdot l$ and a GPU helper function to subtract $(x \cdot x + l \cdot l)$, multiply by λ , and exponentiated.

4.1.5 Object tracking

In this component correspondence mapping of blobs across frames are constructed. Peihua Li et al’s work was based on the idea of partitioning the color space of objects which leads to a quite small number of histogram bins representing color distribution [59]. That histogram was the base component of *mean shift tracking* algorithm.

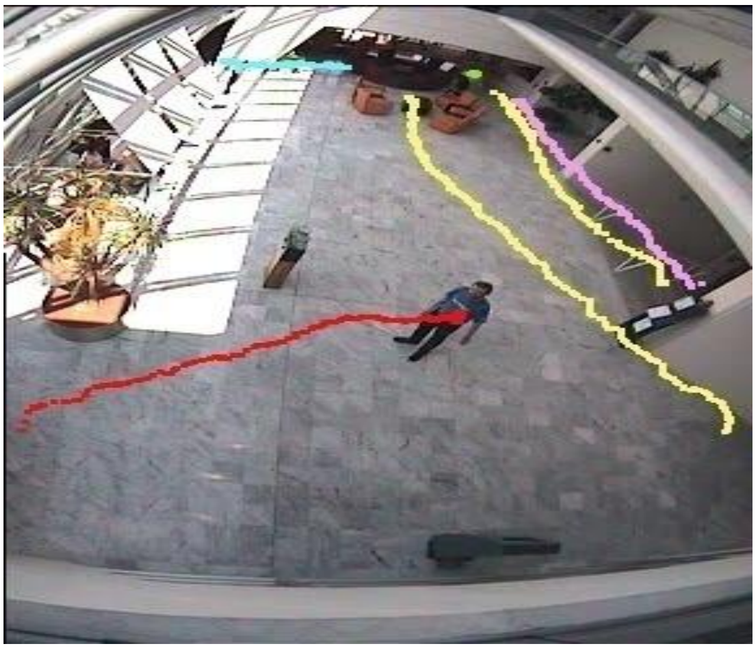


Figure 4-6 Object tracking

Traditional Mean Shift algorithm uses a large number of color histogram, say typically 16x16x16, which makes parallel implementation infeasible. K-Means clustering is employed to partition the object color space that enables us to represent color distribution with a quite small number of bins. Based on this compact histogram, all key components of the Mean Shift algorithm are mapped onto the GPU. The resultant parallel algorithm consists of six kernel functions, which involves primarily the parallel computation of the candidate histogram and calculation of the Mean Shift vector [59] .

4.1.6 Disscretization (cuSVM)

After the detection and classification of objects, the next is to handle features into HMM for recognition of actions in discrete form. That is because discrete HMM is more accurate than continuous as a rule in the literature, This phase encapsulate discretization of agent features (e.g. postures into a set of predefined ones, velocity as fast or slow,.., etc), and team oriented features (e.g. cohesion as separated or merged) as illustrated in Figure 4.7.

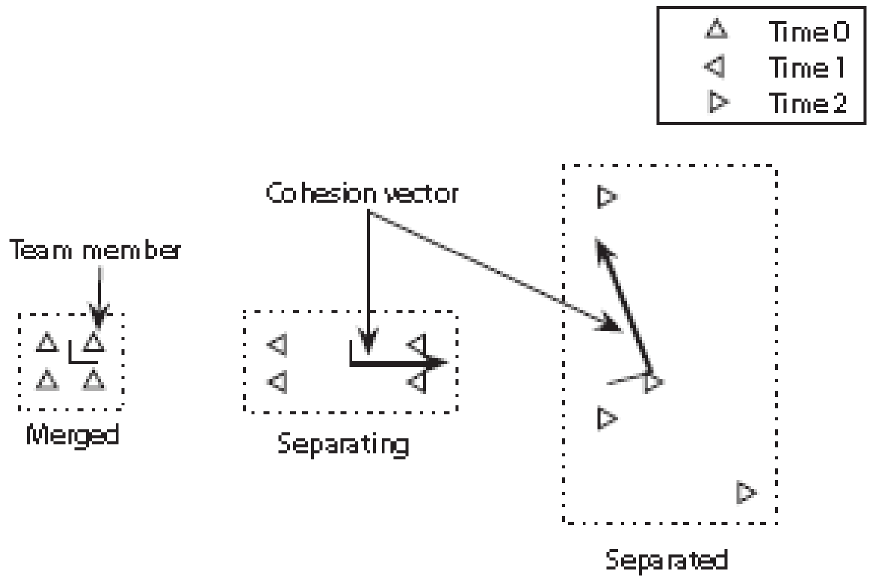


Figure 4-7 Discrete Team features

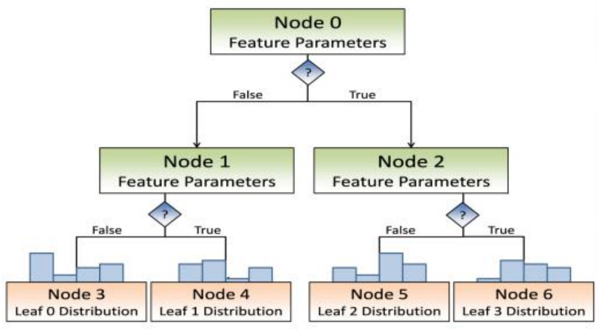
In this phase, the framework provides the developer with GPU implementation of SVM to discretize features (i.e. previously explained in section 4.1.3). Moreover, the developer can integrate his own discretizer into the framework.

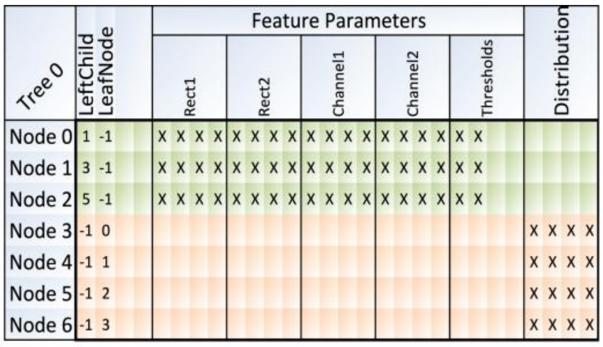
4.1.7 **Role Assignment / Learning (C5.0/ID3) and Mapping**

In this stage each agent in the team is assigned to a role such that learning phase is done in a consistent and accurate way. Role assignment and mapping is done using C5.0 which is based on ID3 *decision tree learning* algorithm [60]. The classification involves mapping the data structure that describes a decision forest to a 2D texture array. Navigation through the forest for each point of the input data is done in parallel. Regarding training, the responses of the training data are computed into a set of candidate features, then, these responses are scattered into a suitable histogram using a vertex shader.

The strategy for the evaluation of a decision forest on the GPU is to transform the forest's data structure from a list of binary trees to a 2D texture (Figure 4.8). The data associated with a tree are laid out in a four-component float texture, with each node's data on a separate row in breadth-first order.

In the first horizontal position of each row, the texture coordinate of the corresponding node's left child is stored. Note that it is needed to store the right child's position as it always occupies the row after the left child. Also all the feature parameters necessary to evaluate the Boolean test are stored for the node. For each leaf node, a unique index is stored for the leaf and the required output a distribution over class labels learned during training.





Tree 0	LeftChild	LeafNode	Feature Parameters: Rect1	Rect2	Channel1	Channel2	Thresholds	Distribution
Node 0	1	-1	X X X X	X X X X	X X X X	X X X X	X X	
Node 1	3	-1	X X X X	X X X X	X X X X	X X X X	X X	
Node 2	5	-1	X X X X	X X X X	X X X X	X X X X	X X	
Node 3	-1	0						X X X X
Node 4	-1	1						X X X X
Node 5	-1	2						X X X X
Node 6	-1	3						X X X X

Figure 4-8 A decision tree structure containing parameters used in a Boolean test at each parent node, and output data at each leaf node. Right: A 7 × 5 forest texture built from the tree. Empty spaces denote unused values.

4.1.8 TW activity Recognition / Learning (cuHMM)

Activity Recognition is the last phase in the framework that has the function of either learning or recognition according to the mode of the framework. The main Learning model that has been used in action recognition with a great success is HMM. Three kernel GPU implementation functions are provided: *Forward*, *Viterbi* and *BalmWelch* algorithms [49]. HMM supported in this phase is not restricted only to teamwork actions but can be generally used in action recognition phase of any application.

The corner stone behind gaining the speedup of HMM algorithms was parallel reduction of computations. Given, M sequences of observations produced from the same HMM of N states. Further, each sequence is of the same length L. To compute the forward probabilities in parallel for all sequences, a three dimensional trellis was formed as a cuboid shown in Figure 4.9. Each slice in the cuboids parallel to the top 3yellow) face is a trellis for one of the M sequences. Each slice parallel to the front 3green) face corresponds to one column of trellises for all M sequences. Each element in the slice corresponds to a state in the trellis for some sequence of observations.

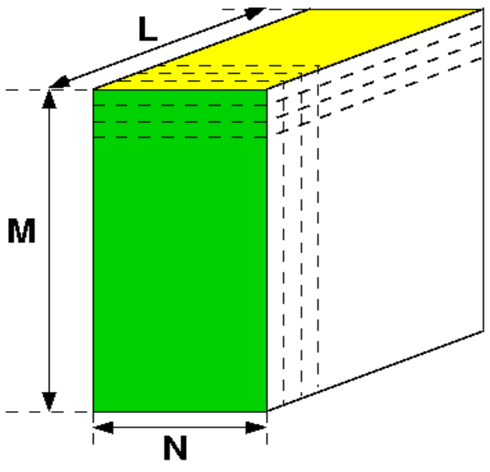


Figure 4-9 The 3d trellis as a cuboid

In the serial implementation, the cuboid is computed from top to bottom. Slices parallel to the yellow face are computed in sequence. For each slice, the computation goes from front column to back column in sequence. In the parallel implementation, the cuboid trellis is computed from front to back. At each individual step, An entire slice parallel to the green face is computed. For each element of the slice, the task is the same. In the CUDA implementation, each kernel thread is responsible for one state in the slice.

The computation of one slice of the cuboid resembles matrix multiplication as shown in Figure 4.10. The $M \times N$ matrix C is the previous slice in the cuboid; the $N \times N$ square matrix A is the state transition probabilities; the $N \times |V|$ matrix B is observation emission probability matrix; and the matrix D is the result. For an element Dig, i.e. The j^th element on the i^th row in matrix D, is computed as

$$D_{ij} = B_{O_i} .* C_i \times A_j \quad \text{(Equation 4.2)}$$

Where O_i is the observation for ith sequence in current slice and $.*$ denotes element-by-element multiplication. The $C \times A$ part is the same as matrix multiplication. For a row in the C matrix, the output is fixed. What is needed is to load the corresponding row in the emission probability matrix and to store the element-by-element product of the two rows and use the resulting product row to multiply the corresponding row in A.

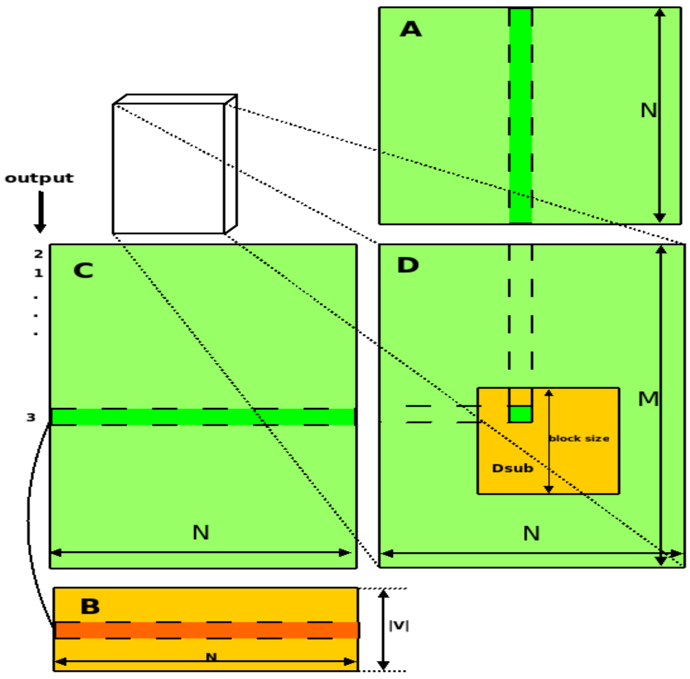


Figure 4-10 The 3d trellis as a cuboid

In the CUDA programming model, threads are known as kernels running on parallel devices. The kernel threads are organized into grids of blocks. For the partition of kernels into blocks and grids, the prototype program uses the same schema as in matrix multiplication example the NVIDIA CUDA Programming Guide [4].

- Each thread block is responsible for computing one square sub-matrix D_{sub} of D.
- Each thread within the block is responsible for computing one element of D_{sub}.

The block size here is chosen to be 16. The number of threads per block is a multiple of warp size and remains below the maximum number of threads per block [4].

The Viterbi algorithm can use the same design. In each kernel thread, instead of computing the summation of the probabilities, The maximum one is computed and the backtrace pointer is stored.

In Baum-Welch algorithm, the computation of one step backward pass follows the same pattern of forward algorithm. As the only change is index, The same kernel/block/grid organization is used as in forward algorithm. However due to the change in matrix index, the computation cannot be represented through matrix multiplication.

4.2 High performance approaches in the framework

Parallelism of processing for applications under the presented framework can be achieved using the following approaches:

1. Component wise parallelism on GPU in which the processing logic of each colored component in Figure1 is distributed over GPU processing elements.
2. Sequential phases running on the framework can be pipelined (e.g. while detected objects of a frame are segmented in segmentation phase, the motion process is performed in the next frame).
3. Independent task parallelism which can be exploited in object classification and action recognition phases. Object classification is a pure parallel task where multiple objects can be classified simultaneously. In action recognition HMMs are built for each action which can run in parallel.

Methods above show how the parallelism is done. Hence a question appears. Is there another degree of freedom for configuring the hardware to gain more performance improvement? Our investigations led to two types of variations on the hardware platform for accelerating framework task:

1. Multiple GPU tasks can be done by attaching more cards to the same PC if applicable (High speed bus will be critical issue here).
2. The framework can be installed over PCs connected by high speed LAN with each PC attached with one or more GPUs as known as GPUs-Grid.

4.3 Configuring Systems Running under the Framework

There are two types of configurations required for full utilization of the GPU framework: one is software and the other is hardware.

4.3.1 Components ‘configuration (software)

In this stage, the developer makes selections of stages that will be utilized during the processing of his target system, then adjusts the parameters of each component that fit into system requirements. The configuration parameters for each phase are listed as follows.

- Motion Detection by sliding window for Moving platform: length of sliding windows with 91 as the default length, and number of histogram pins.
- Segmentation and objects’ data extraction: the number of N partitions.
- Object classification (SVM) : feature vector length, and number of classes
- Mean Shift tracking: the number of clusters K as the tracking algorithm is based on K-means clustering [59].
- Role Assignment and mapping (ID3 algorithm): mode (TreeLeaf, ForestLeaves and etc), number of trees and number of classes.
- TW activity Recognition (HMM): number of hidden states, feature vector size, blocks size and number of actions. The default value of the block size is 16.

4.3.2 Environment setup (hardware configuration)

After components’ configuration, the next is to install the hardware components in hand. The minimum requirement is to have one node with one GPU card on which all configured GPU components would run, the first approach in section 4.2.

For more complex situation where environment contains grid of nodes with each node containing at least a GPU card, all types of parallelism approaches in section 4.2 are applicable. Pipelining and Independent task parallelism are achieved either within same machine with multiple GPUs or across high speed LAN. The two methods are detailed as follows.

1. Multi-GPUs on a single machine: This is most recommended for tasks that require processing relatively large data to overcome communication cost over intranet. This will be more suitable for pipelining the stages of the configured systems (e.g. detection, tracking, segmentation, etc).
2. GPU Grid distributed over different machines: This is recommended if a lot of learned data is stored over network and hence features would have relatively small size to broadcast over network then gain recognition data. This would be suitable for classification and recognition that use large data for learning (e.g. object classification, action recognition phases).

The key of successful configuration of the framework is how to organize available resources to fit processing requirements of the target system with optimal GPU utilization The optimum goal here is to configure the framework to minimize the processing time (denoted as T) required for recognizing actions given by Eq 1:

$$\mathrm{T} = \sum \mathrm{PTcpu(i)} + \sum \mathrm{PTgpu\,(j)} + \sum \mathrm{CTgpu\_cpu\,(k)} + \sum \mathrm{CTnode\_node\,(l)} \qquad \text{(Equation 5.1)}$$

Where PTcpu(i) is the processing time of task i on a CPU. PTgpu(j) is the processing time of task j on a GPU. CTgpu_cpu(k) is the communication time for task k between a GPU and a CPU. CTnode_node(l) is the communication cost for task l between two nodes belonging to the GPUs-Grid if applicable.

4.4 Experimental Results

This section provides speedup results for each of the GPU components comprising the framework. Besides, it provides overall speedup of the framework in terms of a selected application.

4.4.1 Motion Detection by sliding window

GPU version using the adaptive *mean* as the background model with 91 frame- sliding window can run at around 18 fps on 320x240 resolution videos. The *mode* approach can run at 10 fps with the same setting. The GPU version of the *mean* approach achieves around 12x speedup over its standard CPU counterpart. The GPU version of the *mode* approach achieves around 15x speedup.

4.4.2 Segmentation and objects' data extraction

This part is not recommended to use due to its GPU version is slower than CPU version. In case of image with 2048x2048 as dimensions on 9800GT, the speedup is 0.769231x which means slowdown, however it could result in speedup if GPU with higher capability is used. So it is subject to use depending on HW and complexity of input objects.

4.4.3 Object classification and Discretization (cuSVM)

Achieved speedup in cuSVM reached about 35x for training SVM, 84x for corresponding prediction. This speedup was recorded against MNIST data set (LeCun et al., 1998) on NVIDIA GTX 260 GPU.

4.4.4 Object tracking

The experiments show a speedup in blob tracking reaching 3.36x using the famous CAVIAR data set on GeForce 8800 GTS GPU.

4.4.5 **Role Assignment / Learning (C5.0/ID3) and Mapping**

129x speedup in argmax mode using object recognition as an application on GPU nVidia GeForce GTX 280 (240 stream processors)

4.4.6 **TW activity Recognition / Learning (cuHMM)**

880x speedup of forward algorithm GPU implementation while 180x is the speedup of Baumwelch algorithm GPU. This test was done on 512x512 (number of states x number of sequences) on NVIDIA G92 with 512MB RAM.

4.4.7 **Evaluation of framework speedup**

To test the applicability of the framework, we selected the system in [42] which is "role based activity recognition in observations of embodied agent actions".

The researchers in [42] used labeled data set, so motion detection, segmentation and tracking are filtered out in components' selection step of the configuration. The components' configuration of that system is presented as follows:

- Discretization: in this component, the developer uses custom thresholds to transfer features from continuous to discrete.
- Role Assignment and mapping (ID3 algorithm): TreeLeaf mode, single tree and 4 classes (4 different roles).
- TW activity Recognition (HMM): number of hidden states is five, feature vector size is 6, and number of actions is six. .

This system was tested using UCF dataset on NVidia 9500GT graphics card (32 500MHZ processors) against 3.2GHZ dual core processor, and speedup of 20x was gained. Gained speedup increases dramatically as capabilities increases.

5 Conclusion and future work

This thesis investigated the state-of-art techniques in action recognition and high performance solutions related to the stages of action recognition, then *real-time* action recognition system with *high accuracy* was built. Moreover, a general framework based on *GPUs* was built to achieve dramatic *speed–up* in the performance of action recognition systems. The framework was implemented with capability of customization to work with any application (e.g. teamwork activity, single person activity).

Regarding recognition of Muslim prayer actions, we presented a novel action to model in the context of action recognition research. The model manifested an accuracy surpassing 93% and was implemented to run in real time. Some challenges were resolved in this work such as determining the prayer's orientation and building a custom dataset with lots of variations for testing.

Our technique is customized for recognizing prayer actions, so an advantage here is that the model doesn't need to be trained. The proposed equations just transform our human perceptions into a computational model that a computer can use.

The work presented here can classify some postures (e.g. Ruku, Sujud and Qiam). Other challenges confronting this work include:

- More Prayer Actions
    - Detection of Takbeerah action
    - Detection of Ema'a (pronounced E-ma-ah) action
    - Detection of foot postures in Qiam, Ruku and Sujud

- Multi-Camera Prayer Action Recognition
    - Detection and Validation of Front Postures
    - Detection and Validation of Back Postures

  as opposed to using the side view only.

- Group Praying Actions
    - Does a member exceed the Imam?

Regarding building GPU Framework, We presented a GPU framework for action recognition that can be configured to many action recognition systems (e.g. single agent, teamwork). Furthermore key approaches for good utilization of the framework were presented. Experiments shows how the framework is configured in teamwork action recognition system with speedup of 20x achieved on NVidia 9500GT graphics card.

Future work of the framework includes enrichment of each phase on the framework with more GPU implementations of relevant algorithms. Examples include GPU implementations of following components.

- Particle filters for object tracking.
- Correleogram based segmentation algorithm to resolve the problem of overlapping objects.
- Codebook based algorithm in object classification phase.

Additionally, HW configurations of the framework can be done automatically for given execution environment (i.e. the GPU grid nodes and number of GPU cards per each node)

A
Discussion about Estimations in Action Recognition Phase

The basic features upon which action recognition is based are the Y coordinate of center and the backbone angle θ_b. The reason why X-cordinate is used is to make the recognition approach do not depend on whether the prayer is fat or thin, or wearing something like robe.

Due to the direct logical relation between θ_b and $CenterPt.Y$ to the postures to be recognized, th initial model was clear to mode. For instance in Qiam posture θ_b should be small and $CenterPt.Y$ should be around the center of the prayer. According to that logical relation, remaining of the action are modeled the same way, as follows.

A) Qiam Posture

$$\theta_b \leq 23°\,,$$

$$CenterPt.Y = (0.5 \pm 0.1) * prayerHeight$$

B) ToRuku Posture

$$\theta_b\ is\ increasing: 23° < \theta_b < 77°,$$

$$CenterPt.Y\ is\ decreasing: CenterPt.Y > 0.28\ *\ prayerHeight$$

C) Ruku Posture

$$77° < \theta_b < 115°,$$

$$: CenterPt.Y > Th3$$

D) From Ruku to Qiam

$$\theta_b \text{ is decreasing: } 23° < \theta_b < 77°,$$

$$CenterPt.Y \text{ is increasing: } CenterPt.Y > 0.28 * \text{prayerHeight}$$

E) Sujud Posture

$$\theta_{\text{b}} > 115°,$$

$$CenterPt.Y < 0.2 * prayerHeight$$

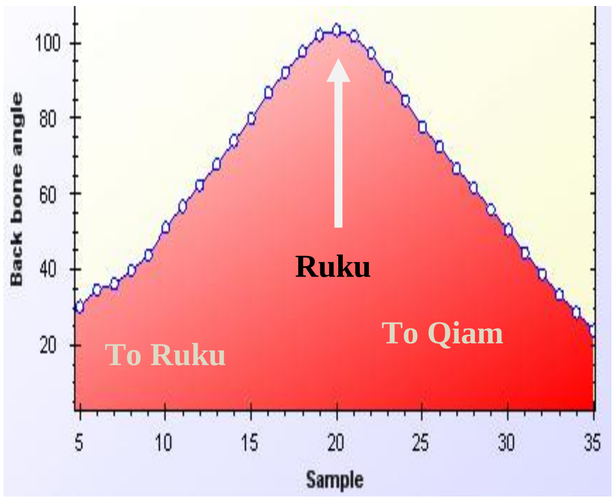
F) Jalsa Posture

$$0 < \theta_b < 20°,$$

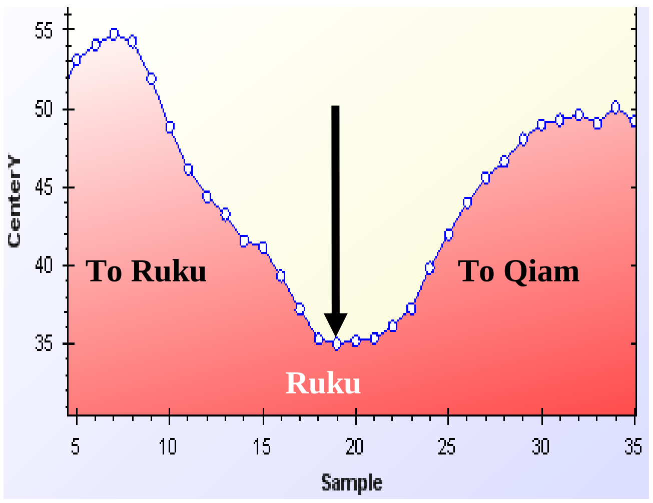
$$0.2 * prayerHeight \leq CenterPt.Y < 0.28 * prayerHeight$$

Choosing the Best Estimate

The best estimate was determined according by trial and error operations with the help of analysis of the figures as below for all videos. The thresholds were chosen to optimize the accuracy in all videos according to the equation $\frac{\sum_{i=1}^{nV} \frac{nC_i}{nA_i}}{nV}$ presented in experimental results section in Chapter 3.



a) Backbone axis angle θ_b against



b): CenterPt.Y against sample/time

Note: There were 25 other figures similar to the presented above for each video. The figure above presents a sample to illustrate the main idea.

References

[1] Top 500. [Online]. http://top500.org/

[2] (2008) Wikipedia. [Online]. http://en.wikipedia.org/wiki/High-performance_computing

[3] (2009, Sep.) Wikipedia. [Online]. http://en.wikipedia.org/wiki/Graphics_processing_unit

[4] NVidia, *NVidia CUDA programming guide*. 2008.

[5] Y. Dedeoğlu, *MOVING OBJECT DETECTION,TRACKING AND CLASSIFICATION FOR SMART VIDEO SURVEILLANCE (MSc Thesis)*. the Institute of Engineering and Science,bilkent university, 2004.

[6] S. L. Phung, A. Bouzerdoum, and D. Chai, "SKIN SEGMENTATION USING COLOR AND EDGE INFORMATION," in , Paris, 2003.

[7] F. Gasparin and R. Schettini, "Skin segmentation using multiple thresholding," in , San Jose, CA, USA, 2006.

[8] T. Horprasert, D. Harwood, and L. S. Davis, "A Statistical Approach for Real-time Robust Background Subtraction and Shadow Detection," in , 1999.

[9] R. Tan, H. Huo, J. Qian, and T. Fang, "Traffic video segmentation using adaptive-K gaussian mixture model," in , 2006.

[10] S.-Y. Yang and C.-T. Hsu, "BACKGROUND MODELING FROM GMM LIKELIHOOD COMBINED WITH SPATIAL AND COLOR COHERENCY," in , 2006.

[11] Y. Zhang, S. J. Kiselewich, W. A. Bauson, and R. Hammoud, "Robust Moving Object Detection at Distance in the Visible Spectrum and Beyond Using A Moving Camera," in , 2006.

[12] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Connected_Component_Labeling

[13] J. H. Jianpeng Zhou, "Real Time Robust Human Detection and Tracking System," in , San Diego, CA, USA, 2005.

[14] V. Kellokumpu, M. Pietikäinen, and J. Heikkilä, "Human Activity Recognition Using Sequences of Postures," in , 2005.

[15] M. Zahedi, "Robust Appearance-based Sign Language Recognition," in , Aachen University, 2007.

[16] R. Akmeliawatil, M. P.-L. Ooi, and Y. Chow Ku, "Real-Time Malaysian Sign Language Translation using Colour Segmentation and Neural Network," in , 2007.

[17] M. B. Capellades, D. DeMenthon, and D. Doermann, "AN APPEARANCE BASED APPROACH FOR HUMAN AND OBJECT TRACKING," in , 2003.

[18] Y.-T. Tsai, H.-C. Shih, and C.-L. Huang, "Multiple Human Objects Tracking in Crowded Scenes," in , 2006.

[19] W. W. Mayol and D. W. Murray, "Tracking with General Regression," vol. 19, no. 1, 2008.

[20] N. D. Binh, E. Shuichi, and T. Ejima, "Real-Time Hand Tracking and Gesture Recognition System," in , 2005.

[21] "A step-by-step tutorial on HMMs". [Online]. http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html

[22] B. M., N. Oliver, and A. Pentland, "Coupled hidden Markov models for complex action recognition," in , San Juan, Puerto Rico, 1997.

[23] X. Feng and P. Perona, "Human Action Recognition By Sequence of Movelet Codewords," in , 2002.

[24] H. Ren and G. Xu, "Human action recognition in smart classroom," in , 2002.

[25] L. Wang, "Support Vector Machines : Theory and Applications," in , 2005.

[26] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing Human Actions: A Local SVM Approach," in , 2004.

[27] C. Schuldt, I. Laptev, and B. Caputo, "Human Activity Recognition Using Sequences of Postures," in , 2005.

[28] M. Ahmad and S.-W. Lee, "HMM-based Human Action Recognition Using Multiview Image Sequences," in , Hong Kong, 2006.

[29] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," in , 2004.

[30] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional sift descriptor and its application to action recognition," in , Augsburg, Germany, 2007.

[31] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as Space-Time Shapes," in , 2005.

[32] O. Jenkins, G. Gonzalez, and M. Loper, "Tracking Human Motion and Actions for Interactive Robots," in , 2007.

[33] M. Loper, "Action Recognition on a Mobile Robotic Platform," in , 2008.

[34] M. A. Mendoza, N. Perez de la Blanca, and M. J. Marin-Jimenez, "POHMM-BASED HUMAN ACTION RECOGNITION," in , 2008.

[35] P.-C. Hsiao, C.-S. Chen, and L.-W. Chang, "Human Action Recognition using Temporal-State Shape Contexts," in , 2008.

[36] G. Junxia, D. Xiaoqing, W. Shengjin, and W. Yousho, "Full Body Tracking-Based Human Action Recognition," in , 2008.

[37] L. Ballan, M. Bertini, and A. Del Bimbo, "RECOGNIZING HUMAN ACTIONS BY FUSING SPATIO-TEMPORAL APPEARANCE AND MOTION DESCRIPTORS," in , 2009.

[38] The IXMAS database. [Online]. https://charibdis.inrialpes.fr

[39] D. Chen, R. Malkin, and J. Yang, "Multimodal Detection of Human Interaction Events in a Nursing Home Environment," in , 2004.

[40] F. Fusier, et al., "Video Understanding for Complex Activity Recognition," in , 2004.

[41] G. Sukthankar and K. Sycara, "Robust Recognition of Physical Team Behaviors Using SpatioTemporal," in , 2006.

[42] L. J. Luotsinen and L. Bölöni, "Role-Based Teamwork Activity Recognition in Observations of Embodied Agent Actions," in *7th international joint conference on Autonomous agents and multiagent systems*, 2008.

[43] J. Yin and Q. Yang, "Sensor-based Abnormal Human-Activity Detection," in , 2007-2008.

[44] E. Guenterberg, H. Ghasemzadeh, V. Loseu, and R. Jafari, "Distributed Continuous Action Recognition Using a Hidden Markov Model in Body Sensor Networks," in , 2009.

[45] J. Fung, *Chapter 40: Computer vision on the gpu. In GPU Gems 2: Programming Techniques for High-Performance Graphics and General Purpose Computation*. Addison-Wesley, 2005.

[46] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado, "GPU for Parallel On-Board Hyperspectral Image Processing," in , CA,USA, 2008.

[47] Q. Yu and G. Medioni, "A GPU-based implementation of Motion Detection from a Moving Platform," in , 2008.

[48] P. Li and L. Xiao, "Mean Shift Parallel Tracking on GPU," in , Berlin, Heidelberg, 2009.

[49] C. Liu, "a CUDA Implementation of Hidden Markov Model Training and Classification," in , 2009.

[50] A. Carpenter, "A CUDA IMPLEMENTATION OF SUPPORT VECTOR CLASSIFICATION AND REGRESSION," in , 2009.

[51] J. Fung, S. Mann, and C. Aimone, "Openvidia: Parallel gpu computer vision," in , 2005.

[52] G. Bradski, A. Kaehler, and V. Pisarevsky, "Learning based computer vision with intel's open source com puter vision library," 2005.

[53] J. P. Farrugia, P. Horain, and E. Gueh, "GPUCV: A FRAMEWORK FOR IMAGE PROCESSING ACCELERATION WITH GRAPHICS PROCESSORS," 2006.

[54] M. H. Elhoseiny and E. Shaaban, *Muslim Prayer Actions Recognition*. Dubai: IEEE, 2009.

[55] a. G. M. Qian Yu, "A GPU-based implementation of Motion Detection from a Moving Platform," in , 2008.

[56] S. M. O'Connell, "A GPU Implementation of Connected Component Labeling," in , 2009.

[57] J. Zhou and J. Hoang, "Real Time Robust Human Detection and Tracking System," in , San Diego, USA, 2005.

[58] B. Catanzaro, N. Sundaram, and K. Keut, "Fast support vector," in *25th international*

conference on Machine learning (ICML), New york, 2008, pp. 104-111.

[59] P. Li and L. Xiao, "Mean Shift Parallel Tracking on GPU," in , Berlin, Heidelberg, 2009.

[60] T. Sharp, "Implementing Decision Trees and Forests on a GPU," in , 2008.