Vision System Development Through Separation of Management and Processing

Amir Afrah, Gregor Miller and Sidney Fels
Electrical and Computer Engineering
University of British Columbia
2332 Main Mall, Vancouver, B.C. Canada V6T 1Z4
Email: {aafrah, gregor, ssfels}@ece.ubc.ca

***Abstract*—We are addressing two aspects of vision-based system development that are not fully exploited in current frameworks: abstraction over low-level details and high-level module reusability. Through an evaluation of existing frameworks, we relate these shortcomings to the lack of systematic classification of sub-tasks in vision-based system development. In this paper we present our work-in-progress which addresses these two issues by classifying vision into decoupled sub-tasks, hence defining a clear scope for a vision-based system development framework and its sub-components. Firstly, we decompose the task of vision system development into data management and processing. We then proceed to further decompose data management into three components: data access, conversion and transportation. We present the Vision Utility (VU) framework which provides abstraction over the vision system data management and verify this approach through an example vision system.**

I. Introduction

This paper presents a new approach to vision system development that is based on our novel classification of the field of computer vision. We define the *vision problem* such that it encompasses data capture, transport, conversion, manipulation and analysis, and provide a classification based on this which extends into a framework for vision system development. Each category encapsulates specific functionality to improve re-use and provide an appropriate abstraction layer to hide low-level details from the user.

The activity in the field of vision processing has accelerated rapidly in the past 15 years which has led to an increasing demand for development of end-to-end systems for deployment and prototyping of new vision processing algorithms and methods. In addition to general system development issues, vision has specific requirements that introduce particular challenges: vision data sources vary physically, in communication protocol and in data formats; and the data used is often large in nature which in turn requires high-bandwidth and powerful machines.

The majority of research within the computer vision field focuses on creating new algorithms and methods for performing the analysis and manipulation of the image data and system development issues are treated as secondary, however this lack of focus on system development issues has forced system developers to explicitly deal with low-level data management details which leads to insufficient reusability of the developed modules. Previous research has shown that lack of scope definition and overlap across frameworks leads to a breakdown in component reusability[1].

We present an overview of current approaches for vision system developemnt in Section II and proceed to present our approach in Section III. In Sections IV and V we present the Vision Utility framework and its architecture which provides a proof of concept for this new approach to vision system development and we discuss the results in Section VI.

II. Related work

There are a number of frameworks that attempt to address the subtasks of vision system development, such as OpenCV, VXL and Gandalf [2], [3], [4]. This section presents an overview of these frameworks as well as an evaluation of their approaches.

The Open Computer Vision library (OpenCV)[2] is a comprehensive and widely used image processing framework. The overall design of OpenCV relies on declaring data type definitions for image and vision entities and providing functions for operating on and extracting data from them. OpenCV also provides limited system development support for developers to easily create vision systems. OpenCV provides a framework for accessing data from cameras installed on the system that utilizes OS specific frameworks (such as V4L). OpenCV also provides a function based approach for image format conversion and resizing. These functions access images of different formats from disk and convert the data into OpenCV's native image class. OpenCV also provides routines that allow the programmer to resize images using a number of different interpolation methods, extract sub-regions of images to sub-pixel accuracy and extract specific channels from a multi-channel image.

VXL is a vision development framework that consists of a collection of C++ libraries for computer vision based development[3]. VXL contains four core libraries for numeric processing (VNL) that provides methods for matrices, vectors, decompositions, optimizers, etc, image access and manipulation (VIL), geometry definition (VGL), and other platform-independent vision related functionality (VSL, VBL, VUL). In addition to the core libraries, VXL contains a number of other libraries that cover a variety of vision problems. VXL provides a modular framework where each package could may be used as a lightweight unit independent of others.

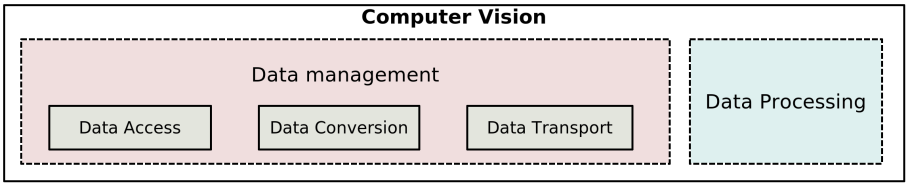


Fig. 1. Graphical representation of our classification of the vision problem.

Gandalf is a function based computer vision and numerical analysis library that defines a set of data types relating to mathematical and computer vision operations [4]. Gandalf consists of four packages: a Common package that defines simple structures and data types used by other packages; a Linear algebra package with a large number of routines for matrix and vector manipulations; an Image package that declares the image structure and provides low-level image manipulation routines; and a Vision package that provides a number of standard image processing, computer vision and geometrical routines.

The approach offered by these frameworks is to expose image access, manipulation and processing routines through a function based API. There are however a number of significant shortcomings that are inherent to the design and approach utilized by these frameworks: Firstly these frameworks do not provide a clear definition of vision components. The functionality for accessing, manipulating and processing routines is offered to the users of the framework at the same level, using the same API. This lack of definition leads to an inconsistent level of details being exposed to the user as they are forced to deal with low-level operations such as pixel manipulation and high level operations such as searching for features in the whole image. Secondly these frameworks do not provide a comprehensive solution for vision system development, although OpenCV provides a camera access mechanism, the solution offered is intermediate and incomplete, also there is no support for transportation of vision data in these frameworks. Thirdly the API offered by these frameworks provides poor support for modularization. These frameworks essentially provide a set of functions and do not address the need for developing code that is easily reused or can be scaled to larger systems.

III. Classification of the Vision Problem

The definition of the vision problem in our approach encompesses all aspects of the vision system development task which includes retrieval of vision data from sources, transportation of data amongst the components of the system and processing of the data. Given this definition of the domain, we proceed to classify the vision problem into a number of decoupled sub-components.

A. Management and Processing

The main objective of vision systems is to process image data in order to either extract high-level descriptions from images such as location of persons or objects etc. or manipulate image data (e.g. applying a smoothing filter or correct radial distortion). The fundamental difference between these two types of processing is the type of output they produce. The first task produces high-level meta-data, whereas the second task produces image data. We refer to the task of processing images which includes both analysis and manipulation of image data as *data processing*.

To perform the task of data processing in vision systems the developer must address the following issues: retrieve the data from sources (e.g. cameras, video files, range scanners); deliver the data from sources to modules in charge of performing the actual processing; modify the format to match that expected by the processing modules; and deliver the output from processing modules to the module in charge of storing, displaying or using the output for any other purpose. We refer to these tasks collectively as *data management* with respect to the vision problem. As described, data management is composed of a number of different non-trivial sub-tasks.

Our classification of the computer vision is shown in Figure 1, which indicates a strong separation between data management and processing. It can also be seen that the data management component can be further decomposed into three decoupled sub-components.

B. Data Management Sub-components

Data management is the task of accessing, pre-processing and delivery of input and output to and from processing. We decomposed data management into three decoupled sub-components: data access, data transportation and data conversion. Although these three components are stand-alone and completely decoupled from one another, they can collectively be utilized to address the data management requirement of vision systems in an abstracted way.

Data Access: There are a wide variety of devices and other media (e.g. files) that could be used as sources of image data for the processing module. The data access module addresses the task of obtaining data from these devices. More specifically, the data access module deals with configuration of the source and retrieval of image data in a standard format.

Data Transportation: In many vision systems, especially large scale systems, the components of the system such as source and processor modules are often distributed over a network or physically connected to several machines via a communication medium such as a bus. The data transport module addresses the need for inter-communication of data and control amongst the different modules of the vision

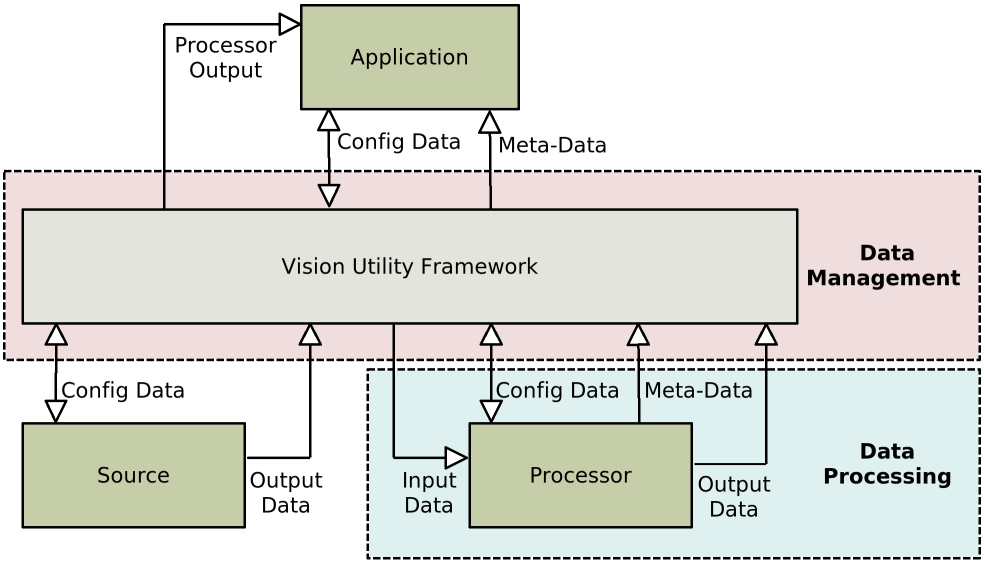


Fig. 2. System development using the VU framework. The scope of the framework is clearly marked.

system.

Data Conversion: Different sources employ a large variety of data formats and compression schemes to represent the image data. In order for modules to communicate data effectively, they need to agree on the communicated data types. The data manipulation module addresses the need for conversion of data formats in order to allow devices with different native representations of image data to communicate.

IV. Vision Utility framework

Developing frameworks based on our classification is advantageous because it clearly defines the functionality which should be supported by the framework and the necessary components to be supplied by users. The VU framework is directly based on the classification of the vision problem presented above and provides the data management sub-component of vision via an API that provides abstraction over the low-level details. The main objective of the VU framework is meet the following three requirements: de-couple access to source data from processing, hide image data format details from users, and provide abstraction over inter-component communication.

A. System Development Model

VU is a component based framework which allows programmers to create vision systems by developing *applications* that configure and connect *sources* and *processors*. In this model, sources are modules that produce image data, processors are modules that perform processing on data, and the application is the control centre of the whole system.

VU provides a system development environment that encapsulates the data management functionality while providing an API that implicitly preserves the natural decoupling between the data management task and data processing task. Figure 2 shows the scope of the VU framework and its relationship to the application, source and processing blocks.

Image data sources within VU are generalized as black boxes with an interface that includes the configuration parameters and the output. In the VU framework all sources are abstracted as configurable virtual cameras. The configuration interface of the source block exposes the internal configuration of the source (such as resolution, exposure, white balance, etc.)

We classify the processing task into two categories: extraction of meta-data, and manipulation of data. Within VU, the first type is called the *image analysis processor* and the second type is called the *image manipulation processor*. Figures 4(c) and (d) show the difference between the two processors. While they are both presented as black boxes with an input and configuration data, the image analysis processor produces meta-data whereas the image based processor produces image data. The input is image data and the configuration parameters allow customization of the algorithm to accommodate the nature of the input data. Meta-data is a high-level description extracted by the analysis process and the output data is an image. In practice it is possible for a processor device to be a mixture of the two models and provide both meta-data and output image data.

The application uses VU to access, configure and connect sources to processors and retrieve meta-data. Figure 4(c) shows the representation of the application block within the framework. The application can receive the output of the source and processor as well as the configuration parameters and meta-data.

B. Communication Model

VU provides two methods for communicating configuration and data amongst devices: remote function calls and peer to peer data transfer. Sources only support the first method whereas processors support both.

Remote function calls allow the application to read and write data to a device. The data can be configuration pa-

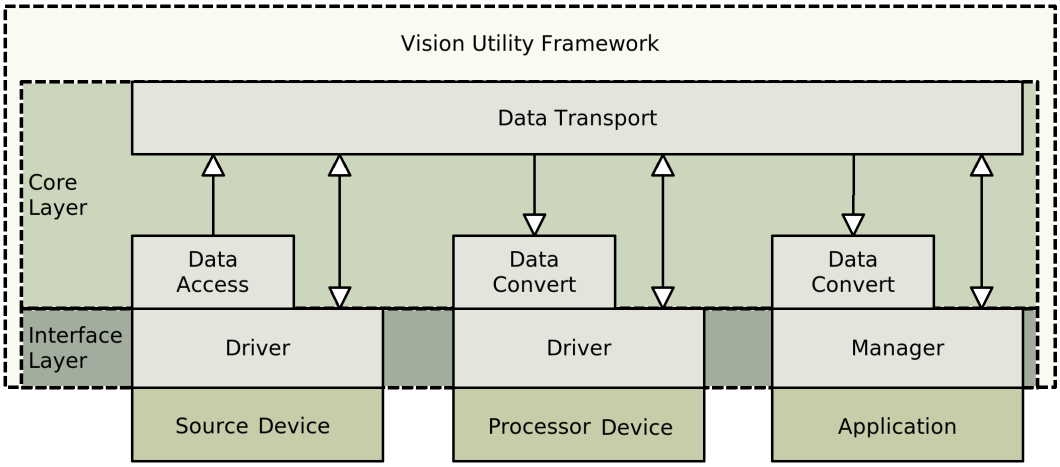


Fig. 3. The VU architecture.

rameters or meta-data produced by the device. For source devices the application can call the remote functions at any time; however, for processor devices there is a synchronous callback which is invoked upon processing a frame of data. This callback blocks and allows the application to exchange data with the device. Through this callback the application is allowed to control, modify and verify resulting data of the processing module and possibly make decisions whether to move on to the next frame or reprocess (further process) the current frame.

Peer to peer data transfer allows devices to communicate their outputs to each other and to the application. The output here refers strictly to image data as other outputs are transferred via the remote function call mechanism described above. Limiting the output of the devices to image data is done in order to provide the functionality of image manipulation framework and allow for possible extension of the framework to multi-processing devices via cascading of the outputs.

V. VU Architecture

The VU framework provides the data management required for vision based application development through an interface that abstracts the details of the data management operations. The architecture of the VU framework therefore contains two layers: The Interface layer and the Core layer. Figure 3 shows the overall architecture of VU.

A. Interface layer

The Interface layer contains two components; the *Application Interface* which provides the programmer access to the functionality provided by the Core layer, and the *Driver Interface* which provides a contact point for device developers allowing them to create drivers for existing sources and processors. Since communication in VU is based on asynchronous callbacks, in both instances the interface routines allow the user to register a number of callbacks to perform actions based on arriving events.

The Application Interface layer provides routines to expose the functionality of the VU framework to the vision application programmer. It provides access and abstraction over the functionality of the Core layer (device query, device communication, device interconnection and callback handling) to simplify the task of the developer.

The Driver Interface offers a standard contact point for the communication between devices and VU while providing abstraction over the implementation details of devices. Devices can be virtually anything, ranging from software routines to hardware accelerated implementations via GPUs, as long as they provide a 'driver' that adheres to the defined interface. The Driver Interface provides the mechanism to allow developers to register their services in order for them to be VU compatible. These services must include handlers for configuration providing output and the main processing of the device.

B. Core layer

The Core layer of the VU framework performs the data management services that include data access, data conversion and data transport. Because of the decoupled nature of data management sub-tasks, each of the internal sub-components of the Core layer are developed as independent entities that could be utilized collectively as demonstrated in the Core layer and also as standalone packages. As shown in Figure 3 the data access component is associated with the source device, data conversion is available for processors and the application. The data transport component mediates communication amongst the modules. Following is an overview of each sub-component:

a) Data Access: The large diversity in image data formats has created a need for a general framework that provides uniform access to image sources regardless of access protocols, physical instantiation and native data type. The aim of this framework would be to retrieve image data in native format from any image and video source using uniform access methods.

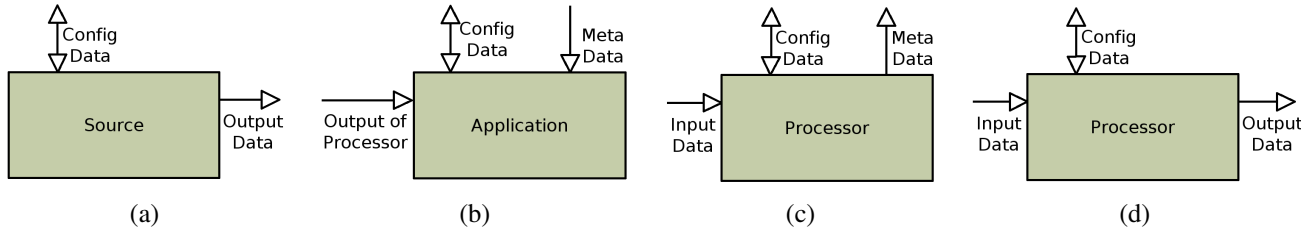


Fig. 4. Representation of the a) source, b) application, c) analysing processor and d) image based processor modules in the VU framework.

To unify access to image data, our model presents image data sources as virtual cameras with a unified interface. We term this concept the *Unified Camera Framework* (UCF). UCF defines and implements two descriptions: the *image specification* and the *device communication protocol*.

The image specification within UCF is the definition of the meta-data format that provides a complete description of its associated image data. This specification identifies the properties and the format of the image data allowing the consumer of the data to be able to decode and use the image data. Typical image properties include resolution and compression type. However, there are many other properties that need to be included in order to provide a comprehensive description of images. The device communication protocol within UCF specifies the method for communication of data to and from devices. This protocol would be implemented on top of the existing device driver in order to allow that device to be used as a UCF compatible device.

b) Data Conversion: Although there exist a number of different existing conversion packages (such as ImageMagick[5], DevIL[6], and Netpbm[7]), they require the users to deal with explicit details of conversions. We propose an image manipulation framework that allows for seamless transformation between data with different properties. This framework would rely on the image specification provided by the data access framework explained in the previous section. The reliance on the standard image specification makes this framework conceptually simple as it can be represented as a black box. The image specification would provide a uniform method for identification between the input and output of the framework and based on the output requirements the framework would apply the appropriate conversions and transformations.

c) Data Transport: Transportation of data between different source and processing modules is an important aspect of vision system development. The vision problem is inherently a distributed task due to its employment of multiple sensors. Communication between these sensors and processing modules is not a trivial task. To address data transport requirements we have utilized the freely available Hive framework that provides a peer to peer network for inter-module data communication as well as centralized control[8], [9].

VI. Results and Evaluation

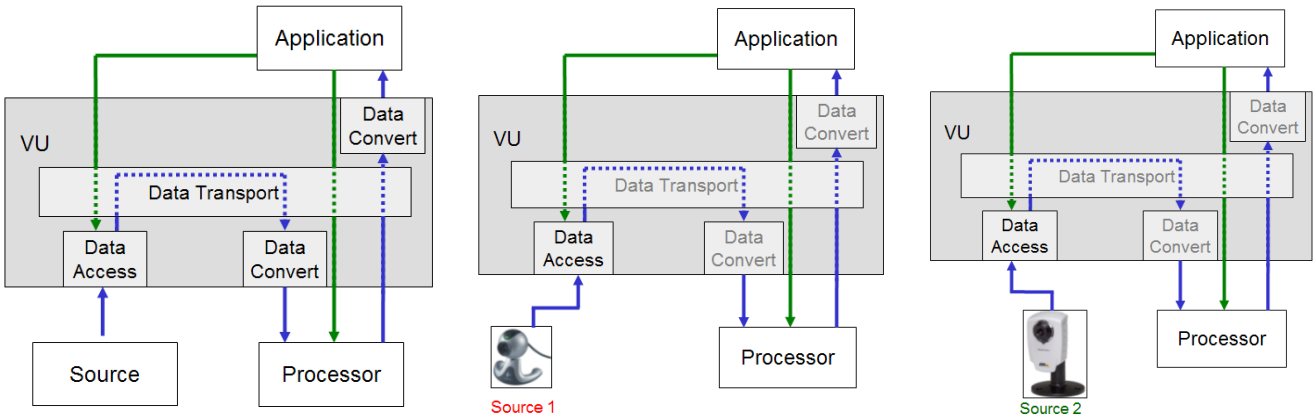
We have designed several tests to verify our approach towards vision based system development; more specifically, to demonstrate that VU meets our three requirements: abstraction of source access details, image format details and inter-module communication. These tests form part of our prototype to demonstrate that such a framework is feasible, but does not represent a mature system. VU is still a work-in-progress and we intend to fully flesh out the individual components as future work.

In order to carry out these tests we have developed two sources and a processor. The sources are an Axis 207 network camera that produces JPEG images and a Logitech USB webcam that produces raw RGB frames, both of which can be configured to provide images at different resolution. The processor performs image resizing and foreground object detection. Figure5a show the flow diagram of the system that we have developed using VU. The application configures and connects the source to the processor as well as receiving and displaying the output of the processor to the screen.

To demonstrate VU's ability to provide abstraction over the details of source access and image format details we evaluate the effort required for changing sources (with different native data formats) in this system. Figure5b and 5c show the flow diagrams of the system utilising two different sources. The only change necessary to swap sources using VU is a single call inside the application that assigns the source of the system. Although the two sources employ completely different data exchange protocols and image formats these differences are hidden by VU allowing the processor and the application to remain unchanged.

Using the application we can change the resolution of the new source to 320×240 from VGA in order to demonstrate VU's ability to provide abstraction over actual data properties. The processor is unaware of this change and continues to receive VGA images. Figure 6 shows the application's output for the three different configurations using the two cameras.

To demonstrate VU's ability to provide abstraction over the inter-module communication details we evaluated the overhead that is required to distribute the modules that make up the above system over the network. The result of this experiment is that the code complexity is identical to the case where the modules are co-located as VU provides abstraction over the inter-module communication details. The only difference is that the application must know the location of the modules (such as the IP address of the machine hosting the module). We plan to implement an auto-discovery mechanism in the future.



(a) (b) (c)

Fig. 5. (a)The flow diagram of the VU test application, (b)the flow diagrams of application using source 1, (c)the flow diagram of the application using source 2

(a) (b) (c)

Fig. 6. The result of the vision system with different cameras and resolution settings: a) USB camera 640×480 b) Axis camera 320×240 c) Axis camera 320x240 - processor output 200×100

VII. Conclusion

In this paper we presented our work-in-progress solution to the current problems in vision system development. We have proposed that the fundamental limitation with the current approach offered through various frameworks is the lack of conceptual high-level classification of the vision problem into smaller sub-components.

In order to address the lack of sub-task classification in computer vision we proposed a decomposition of the vision problem into the following decoupled components: data access, which addresses retrieval of image data from sources; data transformation, which addresses format conversion of image data; data transportation, which addresses the communication of data between modules in a vision system; and data processing, which addresses the analyzing and manipulation of image data.

Based on the above classification we presented a framework that provides the functionality of the data access, data transformation and data transportation components through an API that abstracts the details of each component from users. We described the programming model of this framework and demonstrated the ability to provide higher level abstraction over vision data management using this new development paradigm.

Our proof-of-concept system implementing the VU framework will be made available for research use.

References

[1] A. Makarenko, A. Brooks, , and T. Kaupp, "On the benefits of making robotic software frameworks thin," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[2] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, 1st ed. O'Reilly Media, Inc., October 2008.

[3] VXL, "http://vxl.sourceforge.net/."

[4] Gandalf, "http://gandalf-library.sourceforge.net/."

[5] ImageMagick, "http://www.imagemagick.org/."

[6] DevIL, "http://openil.sourceforge.net/."

[7] Netpbm, "http://netpbm.sourceforge.net/."

[8] A. Afrah, G. Miller, D. Parks, M. Finke, and S. Fels, "Hive: A distributed system for vision processing," in *Proc. of the Int. Conf. on Distributed Smart Cameras*, September 2008.

[9] G. Miller, A. Afrah, and S. Fels, "Rapid vision application development using hive," in *Proc. International Conference on Computer Vision Theory and Applications*, February 2009.