Stereo Vision System for an Autonomous Robotic Platform

Lars Cederholm & Niklas Pettersson
Mälardalen University, IDT
Supervisor: Professor Lars Asplund

June 1, 2011

Abstract

hej

Contents

List of Figures

Chapter 1

Introduction

Currently being developed by researchers and faculty at Mälardalen University (MDH) is an FPGA stereo camera system designed to be used for a vision system in robotics. To allow for testing algorithms without implementing them on the FPGA system, a communication protocol allowing the camera to send images and data over Ethernet or USB to a PC system was designed and an implementation was created that interfaces with the Open Source Computer Vision[4](OpenCv) library. Due to safety requirements in a robotic implementation the Ada programming language was used to implement the communication protocol. To allow a seamless connection between OpenCv and the communication, Ada bindings for the C version of OpenCv was designed and implemented as well. The reason for creating Ada bindings against the C version of OpenCv instead of the newer C++ version is due to the limitations of interfacing Ada with C++[1].
The introduction contains four different parts, first is the introduction to the OpenCv library, the second part is an introduction to OpenCvAda, third is a guide to interfacing the Ada programming language with C or C++ and the last part is the communication protocol.
After the introduction to the major areas of the report comes the three sections of OpenCvAda, Design, Benchmark and User guide followed by the two sections about interfacing Ada with C and C++, then comes the requirements and design of the communication protocol. The last parts of the report is the conclusions and summary followed by the future work section. In the appendix we can find complete code examples to the OpenCvAda user guide and a cheat sheet for OpenCvAda.

1.1 OpenCv

OpenCV is an open source computer vision library written in optimized C and C++ and runs under Linux, Windows and Mac OS X. OpenCV is designed to be an infrastructure for computer vision that is easy to use while maintaining computational efficiency and a focus on real-time applications.
The library contains over 500 functions that cover many areas related to vision, such as factory product inspection, medical imaging, security, user interface,

[1]See more in the C++ to Ada section.

camera calibration, stereo vision and robotics. OpenCV has been used in many applications since its first release in 1999. Applications where it has been used include image stitching of satellite images, noise reduction in medical imaging and many others.

1.2 OpenCvAda

Due of the potential usefulness of OpenCV in robotics and other safety critical systems and the reliability of Ada in mission-critical applications. By creating Ada bindings for OpenCV it is possible to combine the usefulness of OpenCV with the reliability of Ada. The challenges of creating these bindings are to keep the look and feel of the original C library and maintaining the performance.

1.2.1 Design

When designing OpenCvAda, the main design challenge was to keep it as similar as possible to OpenCv in C while making it behave as much as an Ada library as possible. OpenCv is built from a series of modules while OpenCvAda uses packages that split most modules into one package with types and another with operations related to those types.
To match Ada naming conventions and avoid name conflicts, OpenCvAda uses a different naming convention then in C. The names of the functions and types will be the same as in C but due to Ada being case-insensitive and Ada not allowing types and functions having the same name, words are separated with underscore rather then uppercase letters.
OpenCv uses several void pointer types to "fake" generic functions in C. There are several ways of mapping these to Ada from overloading functions to unchecked_-conversions, for different types different methods is preferred.

1.2.2 Benchmark

When designing and implementing OpenCvAda it was important to keep the functionality and performance as close to the C precursor as possible. To show this we use execution time comparisons between Ada, C and Python versions of the same application and also comparing the results of the C/C++ samples implemented together with OpenCvAda.

1.2.3 User guide

To ease the development when working with OpenCvAda a user guide has been written to explain the commonly used types and functions. The user guide explains the functions with both text and code examples. The full application examples are added as an appendix.

1.3 Interfacing Ada with C and C++

The majority of work in this thesis is related to interfacing Ada with existing C and C++ code which OpenCV is written in originally. Different problems arise when attempting to match parameters and return values from C/C++ with

corresponding types in Ada, this section describes and discusses the problems associated with the interfacing.
Both Ada and C have well defined standards, due to this it possible for Ada to provide extensive interfacing support with C. Although the interfacing is trivial for the most part, due to the types in both languages are compatible with each other, there still exists some special cases where Adas strong typing does not interact well with C.
Unlike its predecessor C, C++ does not have a well defined standard, which makes interfacing C++ and Ada problematic. It is possible to some extent to create a binding between Ada and C++, but this binding will have serious restrictions in its compatibility between different compilers and even different versions of the same compiler. The reason for this problem is the lack of a standard for C++ compilers, without a standard to follow the compilers can take certain liberties during compilation which results in different name mangling schemes and different ways to structure the object files. Other problems related to interfacing Ada and C++ are caused by features in the C++ language, such as vectors and to some extent templates.
This section discusses the problems and solutions to different interfacing problems that arise between Ada and C/C++.

1.4 GIMME Communication Protocol

The communication protocol contains a series of requirements detailing what the protocol should be able to handle, with the main requirement being flexibility and extendability. The protocol should handle parallel transactions from one or several devices, regardless of the type of transaction. The medium should be raw Ethernet with a possibility of using USB, the devices are identified using their MAC address.
Designed as a layered header system built on top of raw Ethernet frames. One header on each frame in a transaction to distinguish each frame and make it possible to use several concurrent transactions one device and one header for each transaction to describe the transaction. In that section all the headers are explained in detail.
The PC side of the communication is explained first by the architecture of the system and then a closer look at the individual subsystems available. The PC implementation of the protocol interfaces through a shared API with OpenCvAda allowing for example direct transfer between camera image to an Ipl_Image used in OpenCvAda.

Chapter 2

Design of OpenCvAda

This part details the choices made in the design of OpenCvAda.

2.1 Naming conventions

The original naming convention in OpenCv uses uppercase letters to denote new words in names, which is not very readable in Ada, due to Ada being non-case sensitive. So in order to increase the readability and usability of OpenCvAda, it conforms to the Ada 95 Quality and Style guide[1] to the greatest extent possible while keeping the structure and naming convention similar to the original C library.
A special case exists to match the naming convention in OpenCv the main image type is called IplImage and those is called Ipl_Image in OpenCvAda.

2.1.1 Types

Types in OpenCvAda will use the OpenCv names but each word will be separated with an underscore instead of only an uppercase letter. OpenCvAda still keeps the Cv prefix as well but adds an underscore after it as well.

```
typedef struct CvMat {
...
}

type Cv_Mat is record
...
end record;
```

2.1.2 Functions

Functions in OpenCvAda will use the same naming convention as detailed in types, except for the functions that have the same name as a type then we will add Create_ after Cv_. This is needed since Ada does not allow a function or procedure to have the same name as a type. Ada allows overloaded functions though so if there is already a Cv_Create_ function it will not cause problems unless they have the exact same parameters.

```
CvSize cvGetSize( ... );

function Cv_Get_Size ( ... ) return Cv_Size;
```

```
CvMat cvMat( ... );

Cv_Create_Mat ( ... ) return Cv_Mat;
```

2.1.3 Macros and Constants

Macros and constants in C are generally defined in uppercase and with an underscore between each word, with a few exceptions. Constant value macros are defined as constants at package level in OpenCvAda. For example, a constant declared with the #define pragma in C would in Ada be declared in an appropriate package in the following way.

```
#define CV_NO_DEPTH_CHECK 1

Cv_No_Depth_Check : constant := 1;
```

Function macros follow the same naming convention as constant values, however the functions are wrapped and imported as described in Macros from C in section 5.3.1.

2.1.4 Pointers

The majority of functions in OpenCv takes pointers as parameters and in many cases pointers to pointers. As a result of the latter case, each type in OpenCvAda has an access type associated with it because Ada does not support an access to an anonymous access type. The access types share the same name as the type they are an access to with _ptr appended to the name (for example an access type to Cv_Size is named Cv_Size_Ptr).

```
type Cv_Mat is record
        ...
end record;

type Cv_Mat_Ptr is access all Cv_Mat;
```

Some special cases requires the use of generic Interfaces.C.Pointers, for more information see Pointers in section 2.2. Just like the access types, many of the types in OpenCvAda has a Interfaces.C.Pointers.Pointer associated with the type. These pointers also share the same name as the type they are pointing to with the suffix _pointer added.

```
type Cv_8u_Array is array (Integer range <>)
        of aliased Integer;
```

```
package Cv_8u_Pointer_Pkg is
        new Interfaces.C.Pointers(Integer ,
                                  Unsigned_8 ,
                                  Cv_8u_Array ,  0);
subtype Cv_8u_Pointer is Cv_8u_Pointer_Pkg.Pointer;
```

In many cases Ada requires that anonymous access types are not used, so each type in OpenCvAda will have an access type associated with it named _Ptr.

2.1.5 Packages

OpenCv is divided into modules, the package structure of OpenCvAda is based on these modules. The packages are named according to the modules they represent, in the case of large modules sub-packages are created to separate data types and operations. For example, in OpenCv there is a module called Core. The Core module as its name implies supplies the core functionality of OpenCv. Because the Core module defines many data types and contains numerous operations it has been divided into two packages in OpenCvAda. The top package is called Core contains all the defined data types and the second package is a sub-package called Core.Operations.
Smaller modules such as Highgui are represented by a single package in order to minimize the number of packages in OpenCvAda.

2.1.6 Generic packages

Matrices in OpenCv behave differently from the other data types, it is possible to declare a matrix which contains any data type. Because of the generic nature of the matrices they have been placed in generic packages to ease the access of matrix data. Each type of matrix has its own generic package.

2.1.7 Summary

Language	**Type**	**Constructor Function**	**Function**
C	CvMat	cvMat	cvGetSize
Ada	Cv_Mat	Cv_Create_Mat	Cv_Get_Size

Language	**Macro/Constant**	**Pointer**	
C	CV_RGB	CvMat *	
Ada	Cv_Rgb	Cv_Mat_Ptr	

2.2 Pointers

Types that requires the use of pointers (all of them) in OpenCv have almost all of them three different pointers associated with them.

1. Access type with the suffix_Ptr.
2. Interfaces.C.Pointers.Pointer with the suffix _Pointer.
3. Array with the suffix _Array

This does not include the two dimensional arrays that is available for certain types.
The more complex types in OpenCv have release functions associated with them, these functions require access to _Ptr types to function, this requires that the type is created as a or contained in a _Ptr since Ada does not allow access access parameters. One issue with this is that the it might not always be clear when to use _Ptr and when to use the normal type. Possible solutions to this, is wrapper functions removing one access or simply suggesting that if a type is used as access then it should be created as an _Ptr type. To avoid using more memory we have decided on the second option, so if a type is passed to Ada as a pointer and Ada copies the variable data instead of the address there will be two copies of the variable in memory and the one that is created in C will not be release able since the address is lost.
The _Pointer type is used in cases where arrays do not produce the correct results or when conversion between types are needed see Type conversions for more information. It is possible to keep using the normal types associated with a _Pointer since it can be converted very easy when needed by passing access to a variable or access to the first element in an array instead. So while the _Pointer types might add complexity they are often needed to produce the correct results and due to type conversions(section 2.4 hiding the _Pointer type from the user might not be an advisable move, since this would create unnecessary amounts of generic functions[1] (basically every function would need to be generic) or hamper the usability by only allowing certain types in each function. A version of the second way is actually used with Cv_Arr overloading but still keeps the Cv_-Arr type as a possibility and Cv_Arr have a more limited scoop on what it can contain then other types.

2.2.1 Arrays

Except for the types that are known to need arrays (where the arrays are used as parameters to functions or in records) basic types and some often used types have array types in OpenCvAda. Except for the _Array types _Pointer also behaves like an array in all cases, but is not as intuitive to use in Ada. The _Array type can be used as a _Pointer so even when OpenCvAda requires the _Pointer the user should still be able to use the _Array. Even the two dimensional array allows the user to use the more standard Ada arrays and passing them as a _Pointer_Array when needed.
Whenever possible OpenCvAda will use the _Array and the _Array_Ptr instead of _Pointer. The _Array_Ptr types are used almost exclusively in records since Ada and C arrays behaviour are not equal in records see Pointers in records(section 5.6.1 for more information.

2.2.2 Cv_Void

Cv_Void is used as a substitute for generic functions in OpenCv, the issue with Cv_Void is that there is not known set of types that are mapped as a Cv_Void, instead what types a Cv_Void can be depends on the function they are sent to.

[1]If generics was used at package level instead this would require the user to instantiate each package for every type that will be used. This assumes that in each package only one type exists that would benefit from being generic.

This causes the preferred method of having overloaded functions for each possible type to not be a realistic method. Cv_Void in OpenCvAda instead relies on Unchecked_Conversions that allows the user to convert them into any other C compatible pointer, a few of the often used types have Unchecked_Conversions already created in OpenCvAda in other cases it is up to the user to create them. Examples of this can be found in the OpenCvAda user guide(section 4).

2.2.3 Cv_Arr

The Cv_Arr type is used as a Cv_Void replacement for matrices, images and occasionally other arrays. This allows for the possibility to use a better way of mapping the functions in Ada, overloading the functions with a Cv_Mat and an Ipl_Image version instead of using type conversions.

Overloading

Overloading functions is the easiest way to map functions with void parameters in Ada but doing this is nearly impossible in most cases in OpenCv see Cv_-Void(section 2.2.2 for the reasons why. Since Cv_Arr only has a finite set of possible types mainly Cv_Mat_Ptr and Ipl_Image_Ptr , using overloaded functions will make the api easier to use.

2.2.4 Strings

To make interfacing with other Ada applications easier and overall easier to use, we have decided on using the standard Ada String type instead of the Interfaces.C.Strings package. One issue this choice presents is that it requires the user to add the nul character to each string to match its C equivalent, this is solved with wrapper functions adding the nul character to all strings passed as parameters.
In a few cases the Cv_String_Pointer type is needed, the type is comparable to an array of strings and can be either used as is or “converted” from the Cv_-String_Array type. More information about the _Pointer types can be found in Pointers(section 2.2) and Arrays(section 2.2.1).

2.3 Records

In OpenCvAda there is three major types of records, Cv_Arr based records, sequences and dynamic types and null records.

2.3.1 Cv_Arr records

The records used as Cv_Arr is most of the time Ipl_Image and Cv_Mat. Ipl_-Image is a simple record with an pointer containing an array with the data, see type conversions for information how to get the desired data out of it. Cv_-Mat on the other hand is available in two versions one simple record similar to Ipl_image and one generic package that allows for more seamless access to the data inside the matrix.

2.3.2 Sequences and dynamic types

Sequences in OpenCv uses the Cv_Seq type or types derived from Cv_Seq. Cv_Seq is a classic linked list with a generic payload access to the payload can be done from a reader/writer system or directly. All the Cv_Seq macros and support functions are available in OpenCvAda, but Cv_Seq causes problems in OpenCvAda due to Ada not allowing the kind of on the fly type conversions that C allows. In OpenCvAda to access the data from a Cv_Seq reader/writer requires a pointer conversion on the payload of the sequence, most of types used in OpenCv will require a Unchecked_Conversion. A very limited set of types have conversions added to to OpenCvAda, an example using Unchecked_-Conversions can be found in the OpenCvAda user guide.
Due to the method of accessing the data in a Cv_Seq (direct access to a pointer in the record) other methods such as generic functions are not as intuative to work with that is why unchecked_conversions are used. Another reason for this model is that in one small application it is sometimes possible to access a Cv_Seq and expecting several different types of data.

2.3.3 Null records

Certain types in OpenCV are opaque, meaning that their implementation is hidden to the user. In OpenCV these types are mainly found in the C part of the library and are used as pointers to C++ classes to avoid the need for separate implementations in C and C++. These opaque data types can be mapped directly to Ada as null records and an access to a null record behaves exactly like a C opaque pointer.
Because OpenCVAda provides a binding to the C library of OpenCV null records are used instead of importing the actual implementation of the C++ classes that the opaque types in C reference. The drawback of opaque data types is that the data is not easy to access without knowledge of how the data is stored or by having functions that extract the data. This drawback is not of any concern in OpenCVAda since the opaque types are only used for types that are internal to the library (for example handles to capturing devices).

2.4 Type conversions

Because C/C++ is loosely typed it is possible to convert one type to another with a simple type cast, this is a feature often used in OpenCV. Ada however, is a strongly types language and such conversions are not always possible unless the types being converted are sub-types. In order to convert other types it is necessary to use unchecked conversions that are available through the generic package Ada.Unchecked_Conversion. In comparison with C/C++ the unchecked conversions still adds a layer of protection in the form of checking if the sizes of the converting types match, if they do not the compiler will generate a warning.
The functions in OpenCV usually support multiple types of data, such as images, arrays and matrices. Because of this, most functions that operate on data in OpenCV take a void pointer as a data parameter and then cast it to the correct type in the function with the help of type information built into the data types.

Chapter 3

OpenCvAda Benchmark

3.1 What we want to test:

3.1.1 Why test?

Two of the major design goals in OpenCVAda are that the performance should be similar to OpenCV and operations on identical data should give the same result. To assert that these two goals are met several tests have been created to compare the execution time and output of different programs written identically in C, Python and Ada.
Running benchmarks and comparing the results of the samples provided with OpenCv compared to OpenCvAda versions of those samples, allows us to assume that the import to Ada does not change the performance or the results.

3.1.2 Comparing OpenCvAda with OpenCv

To make sure that OpenCvAda is comparable in speed and memory usage to OpenCv written with Python and C/C++, all the tests will have three versions one for each language.
C/C++ is compiled as C++ using the compiler supplied with GNAT GPL 2010 (20100603). Ada is compiled with the Ada compiler in GNAT GPL 2010 (20100603). Python tests are executed using version 2.7.
In Windows the OpenCv library is compiled with Visual Studio 2010 with Python and Qt[1] support added, third party libraries are flagged to be rebuilt. In Linux compilation is done with the standard compiler supplied with the distribution.

3.1.3 Scaling of OpenCvAda

OpenCvAda should be capable of handling applications with a larger scale then normal (example: large images) so three tests have been designed to measure performance on different cases where larger scales could be applied. This is to make sure that when importing OpenCv to Ada no issues arise on fringe cases due to memory leaks or related things. (example: OpenCvAda allocates data twice)

[1]Qt[5] 4.7.1, LGPL version

3.1.4 Does the speed of OpenCvAda change in another OS

This is to make sure that OpenCvAda is capable of running equal in both Windows and Unix-based systems in comparison with Python and C/C++.

3.1.5 Samples

To make sure that OpenCvAda does not change the results of functions compared to OpenCv, a set of samples from OpenCv was implemented in Ada and the results are compared with the results of the OpenCv samples. This will catch any major problems with OpenCvAda compared to the OpenCv samples, or in a few cases find problems with the original samples.

3.2 What we tested with:

3.2.1 The Benchmarker application

The Benchmarker is an Ada application that uses Ada.Real_Time to time applications. To test any overhead with this application, we have the null tests that run empty applications in all three languages to measure the time needed to initialize.

3.2.2 The Memory usage application

The Memory usage application measures the memory allocated by different applications over time. The program was developed to compare the memory usage of applications written in different programming languages (C, Ada and Python) to assert that they behave in a similar way and also to detect potential memory leaks.
Data is collected once per second and the lowest, highest and average memory usage is calculated and stored to later be saved together with each point of data gathered during execution.
The data gathered is written to a comma-separated-value file to make it easier to import into a spreadsheet for a better overview and generate diagrams of the memory used over time.

3.2.3 Platforms

These operating systems will be tested:

- Windows 7[?]
- Fedora Linux 14[?]

The versions of OpenCv used is SVN revision 4784. The reason for using the SVN version is that certain elements (such as the use of web-cameras and videos) in OpenCV 2.2 (the latest release version) does not work in Windows 7. Using our own compiled version also allows the use of the improved QT version of Highgui in addition to fixing the broken things in version 2.2.

3.2.4 Hardware

The hardware used in the tests will be these computers:

- Lenovo X201 with built in camera.
  - Windows 7 tests.
  - Fedora Linux 14 tests.
- HP Pavilion dv3550eo with built in camera.
  - Memory tests.

3.2.5 Image and Video files used

- heic0602a.jpg
  - Dimensions: 15852x12392
  - Size: 62.1 mb
  - From NASA image gallery [3].
- FlickAnmation.avi
  - Dimensions: 300x250
  - Frames: 59
  - Size: 1,52 MB
  - Distributed with Windows 7.

3.3 The actual tests:

Several programs have been created to test the performance of OpenCVAda. This section describes each test and shows the results of the tests.

3.3.1 Window creation

This test was devised to check the overhead of the built-in window handler and general performance of loading and showing images in a window. The actual test consists of creating 100 windows through the Highgui package and showing an image in each window. After all the windows have been created they are all destroyed and the test finishes. The current implementation does not use the available function that destroys all created windows but instead iterates over each window and destroys them one by one. The reason for this is because at the time of implementation the Cv_Destroy_All_Windows function in OpenCV was not working as intended.
Figure 3.1 shows the slowest, highest and average execution times for 100 executions of the program. As can be seen in the diagram, Ada performs slightly faster than C in all cases, whereas Python has the most stable albeit slowest execution times.

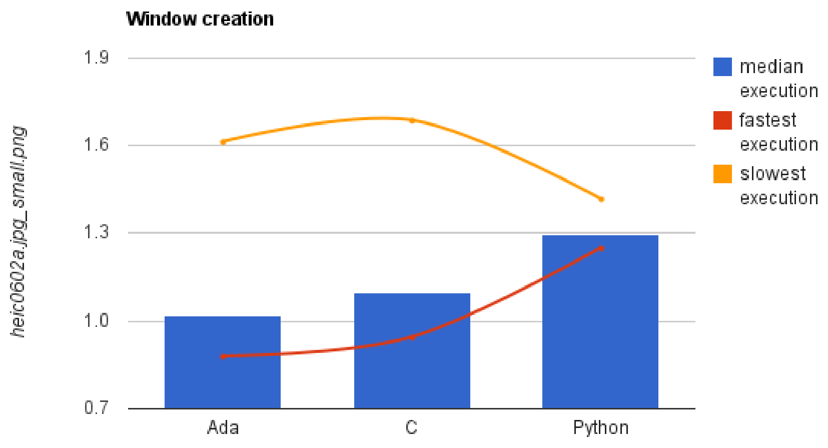


Figure 3.1: Benchmark of window creation and image loading.

3.3.2 Large image I/O

This is a test of large file I/O and image operations, the program loads a large image file, re-sizes it and saves the re-sized image to the hard drive. The image used in this test has a resolution of 15852x12392 pixels and is re-sized to 640x480 pixels.
The test application was run 100 times and the slowest, highest and average execution times are presented in Figure 3.2. As with the window creation test, Ada maintains the lowest average execution time compared to C and Python, although the difference in the average case between Ada and C is negligible.

3.3.3 Algorithms

This tests the performance of algorithms available in OpenCV on each frame in a video. The algorithms tested are a Canny edge detector and Hough transform to detect circles. This test can also be a good indicator of potential memory leaks due to the large number of images that are created and released.
The test was performed on a short video clip consisting of 59 frames at a 300x250 pixel resolution.
The result of the algorithm test can be seen in Figure 3.3. Like the previous tests, the application was run 100 times and the slowest, highest, and average execution times are presented in the diagram. In contrast with the previous tests, C performs faster than Ada, although the the difference between the languages remains marginal.

3.3.4 Memory usage

This test is designed to compare the memory usage of applications. The applications used in this test are a subset of the samples supplied with OpenCV

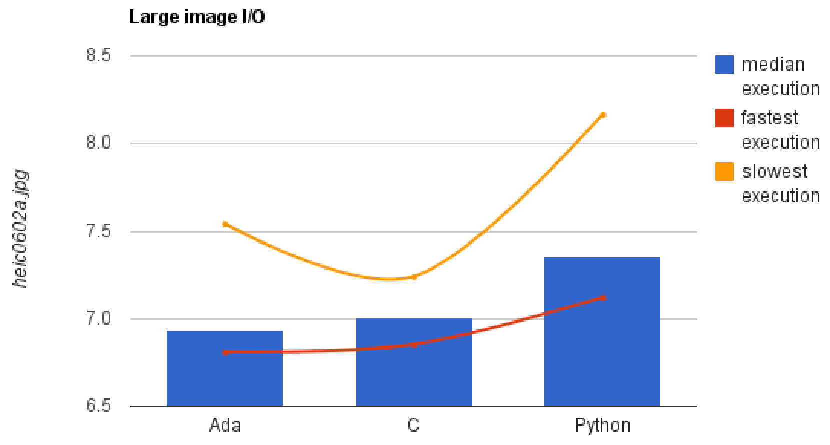


Figure 3.2: Benchmark of operations on large images.

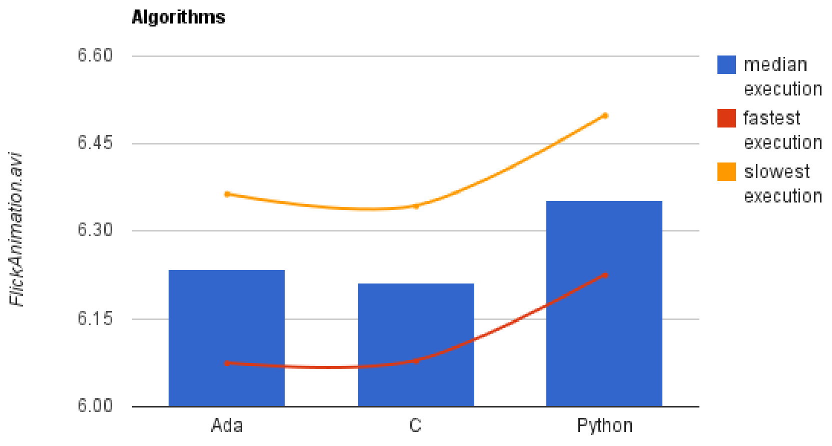


Figure 3.3: Benchmark of image processing algorithms.

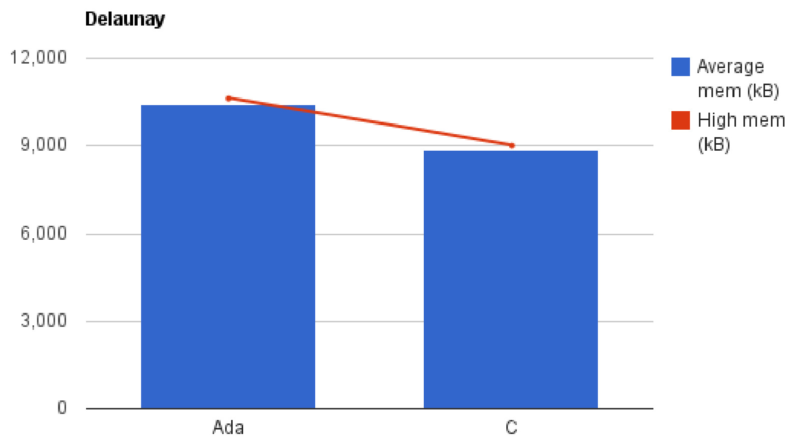


Figure 3.4: Overall memory usage of Delaunay triangulation.

in C. The memory test can also be used to see if there are any memory leaks, which can be seen by an increasing use of memory over time in an application. A simple application is used as a reference for the memory usage difference between Ada and C, more information about this can be found in Executable reference(section 3.3.6).
A subset of 4 sample applications were used to test how much memory they utilize over a 3 minute period, the results for each application in presented in the diagrams below. Each application has two diagrams associated with it, one diagram shows the highest, lowest and average memory usage and the second diagram shows the memory usage for each second of the execution. The subset of applications consist of the following:

- Delaunay triangulation - Generates random Delaunay triangulations.
- Farneback optical flow - Displays the optical flow from a live camera feed.
- Motion - Tracks objects motion from a live camera feed.
- Polar transform - Does a polar transform on images from a live camera feed and then reconstructs the image.

The memory usage of the Delaunay triangulation in OpenCVAda is on average 17 percent higher than its OpenCV counterpart as can be seen in Figure 3.4. Figure 3.5 shows that the behaviour of memory allocation is similar in both OpenCV and OpenCVAda.
Figure 3.6 shows again that the memory usage in OpenCVAda is higher than OpenCV, 11 percent more memory is used on average. Figure 3.7 shows the similarity of the memory allocation behaviour throughout the execution. The reason why the curves look different is because the application constantly allocate and release images and as a result the memory information from OpenCVAda was

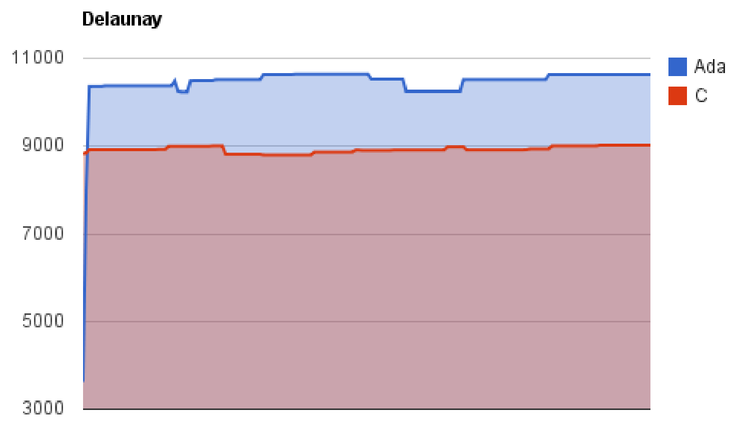


Figure 3.5: Memory usage per second of Delaunay triangulation.

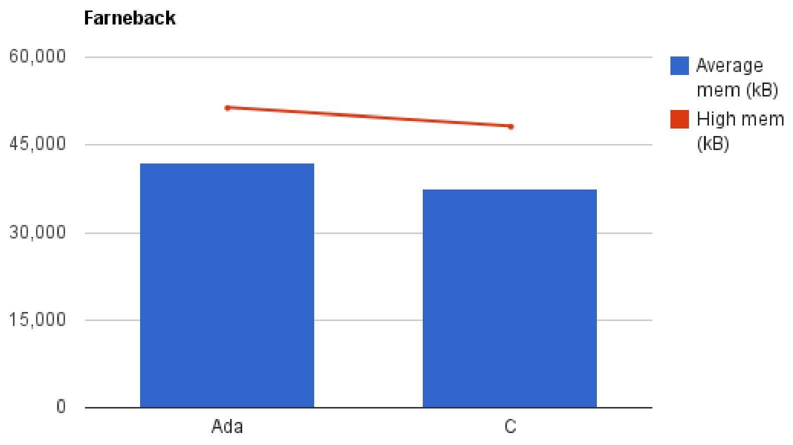


Figure 3.6: Overall memory usage of Farneback optical flow.

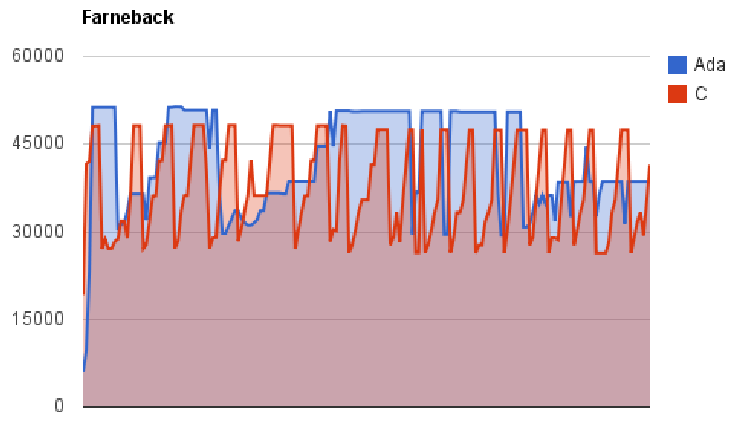


Figure 3.7: Memory usage per second of Farneback optical flow.

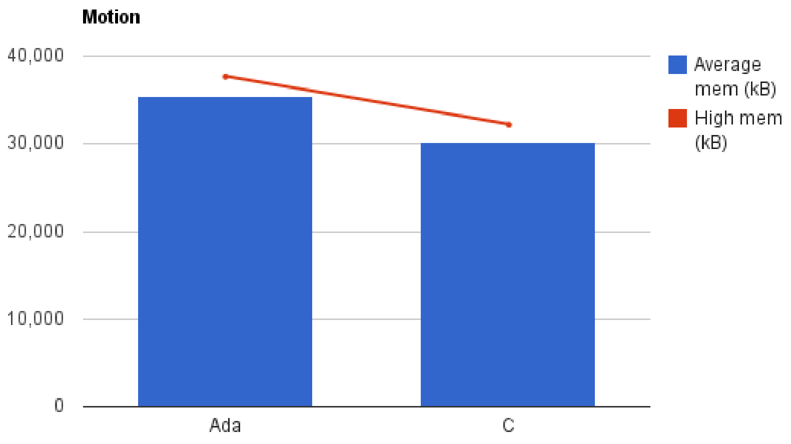


Figure 3.8: Overall memory usage of motion tracker.

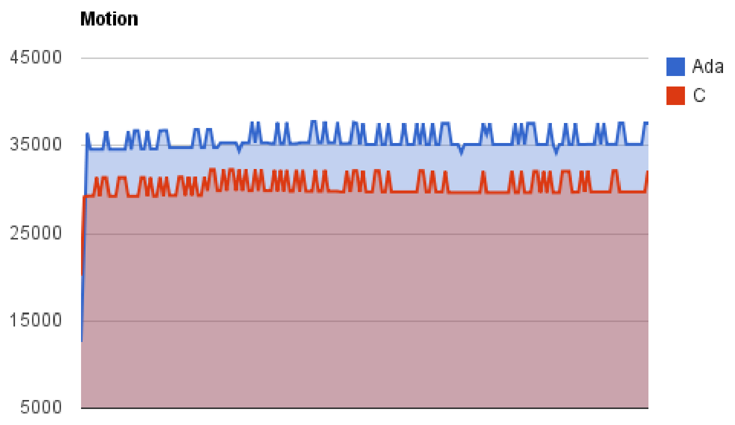


Figure 3.9: Memory usage per second of motion tracker.

polled at peaks more often than OpenCV.
As with the previous applications, Figure 3.8 show that OpenCVAda use more memory than OpenCV. Approximately 10 percent more memory is used when the application is execute through OpenCVAda. Figure 3.9 shows that the behaviour of the applications memory is very similar.
In the final test OpenCVAda utilize more memory yet again, as can be seen in Figure 3.10 approximately 17 percent more memory is used in OpenCVAda compared to OpenCV. The memory allocation is behaving in a similar manner in both OpenCVAda and OpenCV which is shown in Figure 3.11.

3.3.5 Linux

Comparing the execution time of the Window creation and Large image I/O between Linux and Windows, this allows for a base comparison between the operating systems. The results can be seen in Figure 3.12 and 3.13.
Without a deeper analysis of Fedora Linux execution times it is hard to take anything away fro these diagrams except that compared to Windows the executions in Linux are much closer between languages then in Windows, however the same trends can be seen with Ada being the fastest close to C and Python coming in last with a margin. If we look at Executable reference (section 3.3.6) we will see that the execution time is not related to Linux applications being slower and it is more likely that the culprit lies in the compiler and or settings used to compile the OpenCv libraries. Other variables could range from GTK/QT differences to file systems and without more data it is impossible to give a more exact picture. The important part of the data in the scoop of this report is that in both Linux and Windows have OpenCvAda execution times very close to OpenCv C.

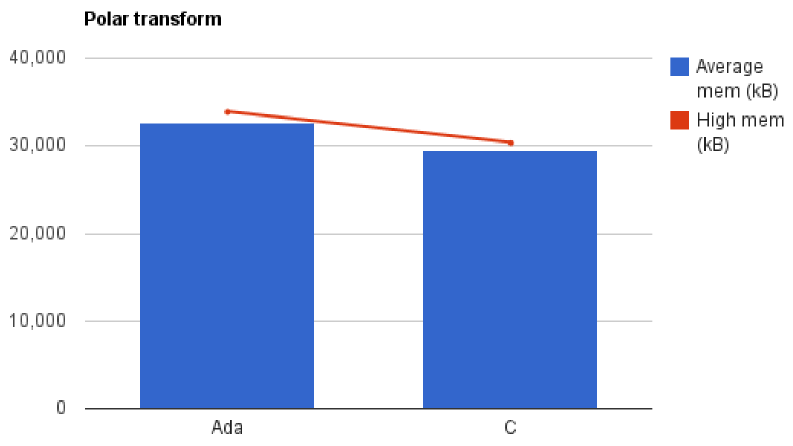


Figure 3.10: Overall memory usage of polar transformation.

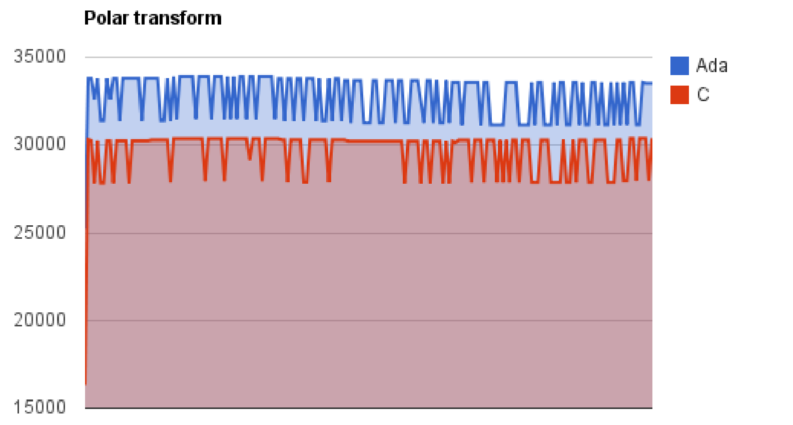


Figure 3.11: Memory usage per second of polar transformation.

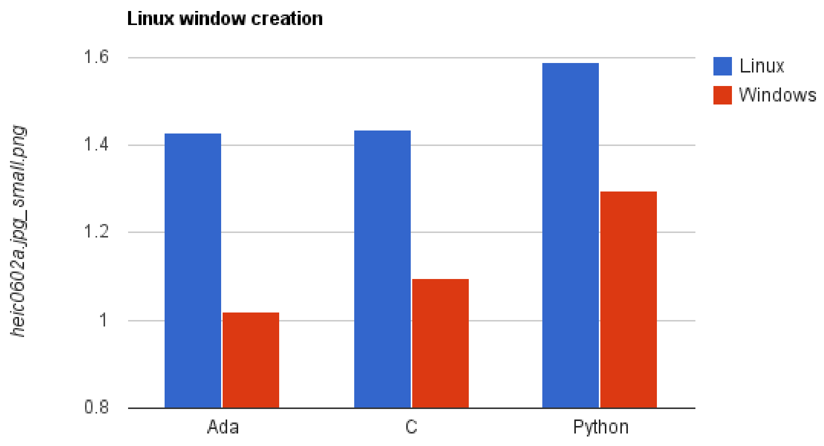


Figure 3.12: Comparison of Linux and Windows window creation

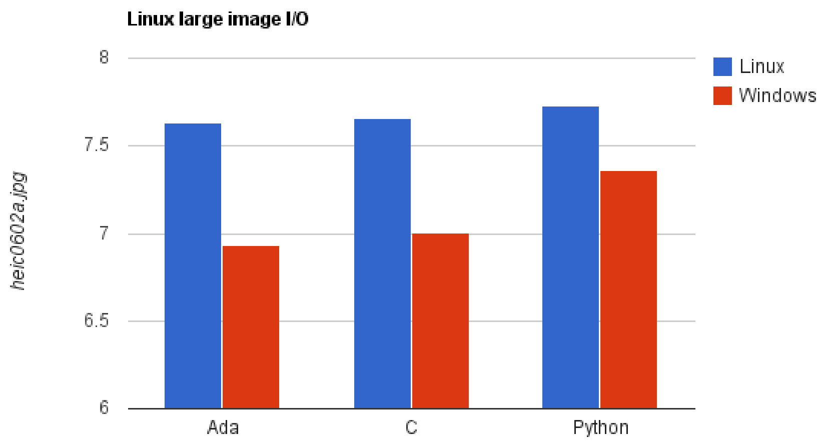


Figure 3.13: Comparison of Linux and Windows large image I/O

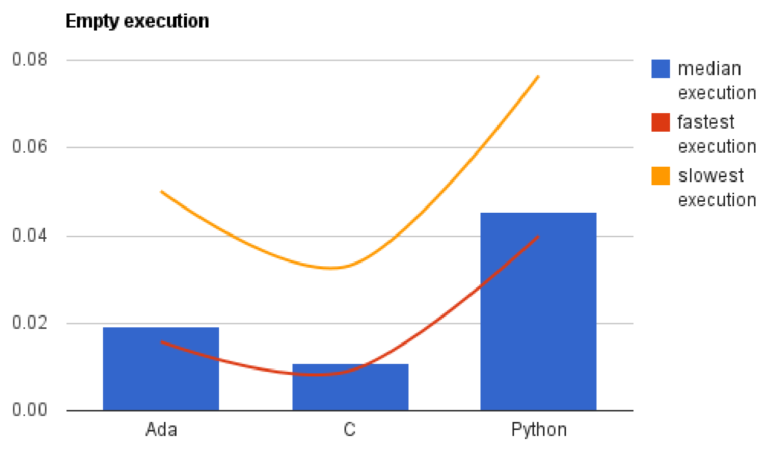


Figure 3.14: Benchmark of Windows reference.

3.3.6 Executable reference

To get reference data this test was designed to execute an empty application from Ada, C and Python in both Windows and Linux. This test is used as a baseline and also to make sure that the other tests are not influenced by the language or platform to a large degree. As we can see in the diagrams the overhead caused in choice of language and platform is negligible, this means that the platform and language can be chosen on other merits then performance unrelated to OpenCv. The tests results can be seen for Windows in Figure 3.14 and Linux in Figure 3.15.
Memory usage varies between programming languages and compilers, different languages allocate different resources to optimize execution times and other functionality. The representation of data is also different between languages which influences the memory usage.
A simple application (an infinite loop that prints the numbers 0-100) was implemented in Ada and C and the memory usage of both applications was measured to get a baseline for difference in memory usage between the languages.
The test in Figure 3.16 shows that Ada in this case utilize approximately 25 percent more memory than C. Although the test is very simplistic, the memory difference percentage will be used as a criteria for assessing the memory usage of the OpenCVAda samples compared to their OpenCV equivalent.

3.3.7 Correctness

By taking the samples supplied with OpenCV and re-implementing them in OpenCVAda it is possible to compare the results from both libraries. If the results are not identical there is some problem that needs to be isolated and repair. In order to get identical results it is not possible to use a live camera to capture frames to process, a video clip will be used instead. Some of the

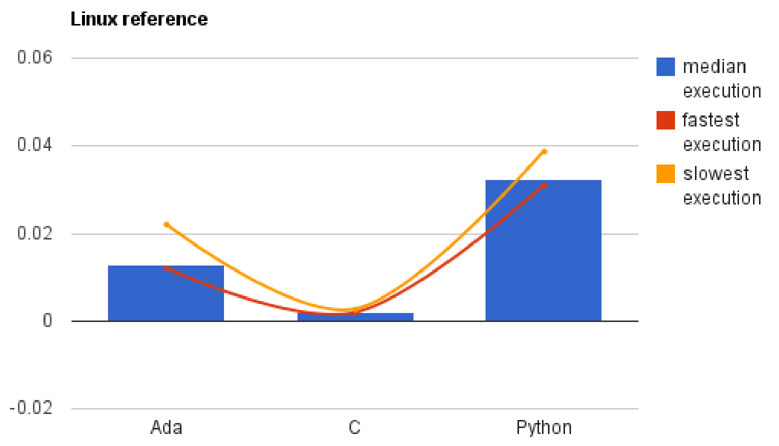


Figure 3.15: Benchmark of Linux reference.

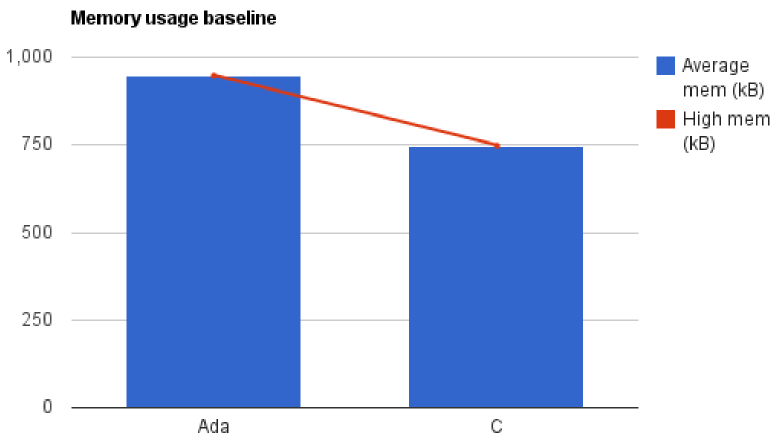


Figure 3.16: Memory usage baseline reference.

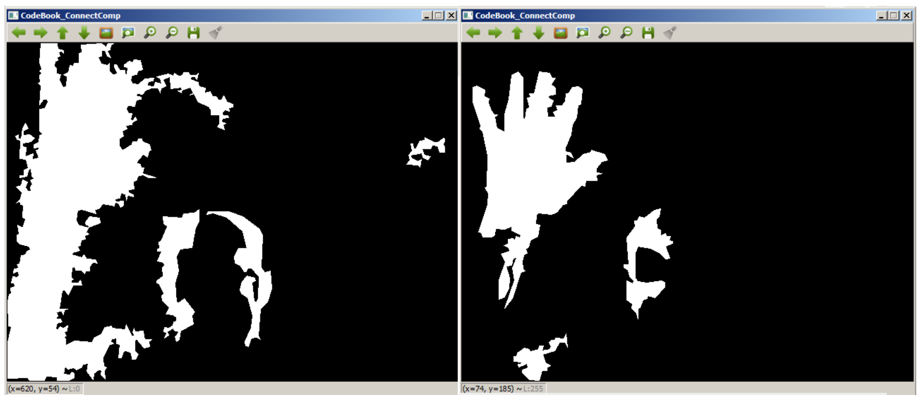


Figure 3.17: Comparison of bgfg_codebook sample.

OpenCV samples have a problem with video playback, in these cases a live camera capture was used instead and as a result it is not possible to simply compare the output. In these cases the assessment will be based on the general functionality and similarity of the output.
This test involves the samples supplied with OpenCv and the ported versions in OpenCvAda and comparing the results of the executions of the samples. This should show any major errors with the imported functions from C.
Due to issues with the video play back in OpenCv (C/C++, Python) samples, in some cases (figure Alpha and figure Delta) where a video file could be used a camera is used instead. So those samples needs to be compared for the actual functionality and not only the results.
We decided on presenting a subset of the samples and the results here for comparing the OpenCvAda samples with OpenCv. The Figures are 3.17, 3.18, 3.19,3.20 and 3.21.

3.4 Benchmark Results

Performance of OpenCvAda in Windows 7 is better then expected with OpenCvAda having slightly lower average execution time then the C code in two out of three test cases. As a reference we can look at the execution of an empty application in each language to see that Ada is not always faster then C. Even though OpenCvAda can not be considered a huge performance boost it can not be consider a performance liability either, this allows for the language with the best features for the application to be chosen instead of performance dictating which language should be used. So regarding performance we can summarise it as avoid Python if performance is a key factor but the choice of Ada or C will not hugely impact performance. Similar results can be seen in Linux but with the reservation that OpenCvAda in Windows 7 is showing better performance in the benchmarks.
The correctness test is used as more of a internal test that was used during implementation of OpenCvAda to make sure that OpenCvAda does not break any functionality from OpenCv and that a port between the two is not too complicated. If we look at both the complexity of the samples compared to the

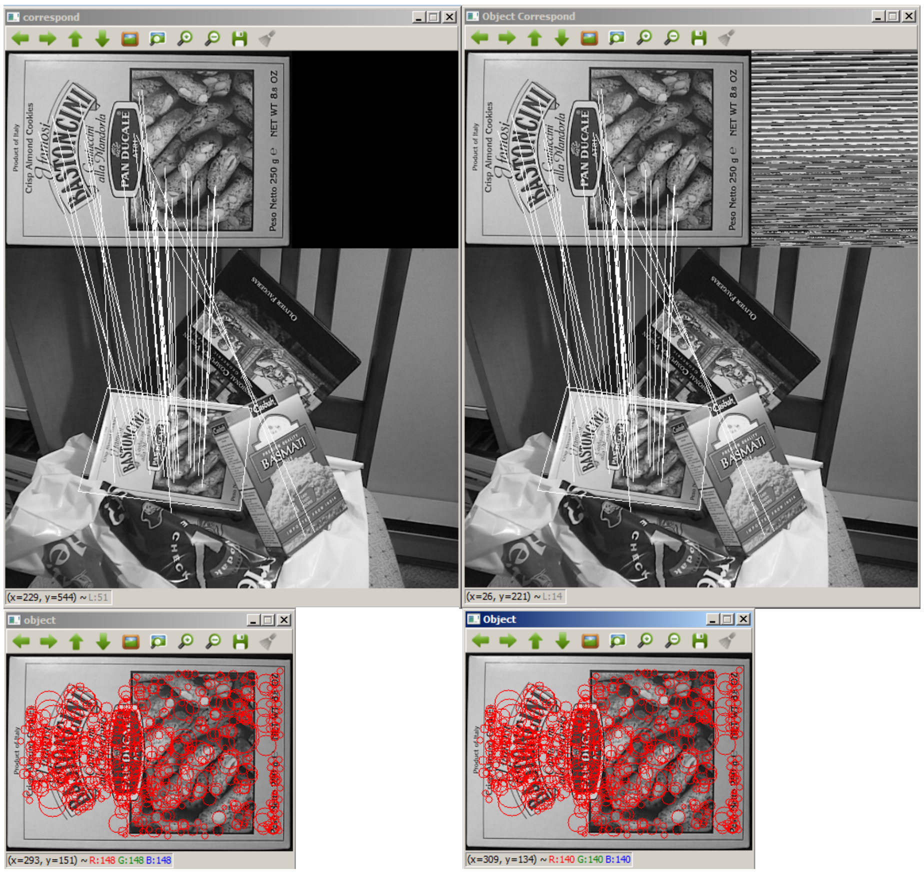


Figure 3.18: Comparison of find_object sample.

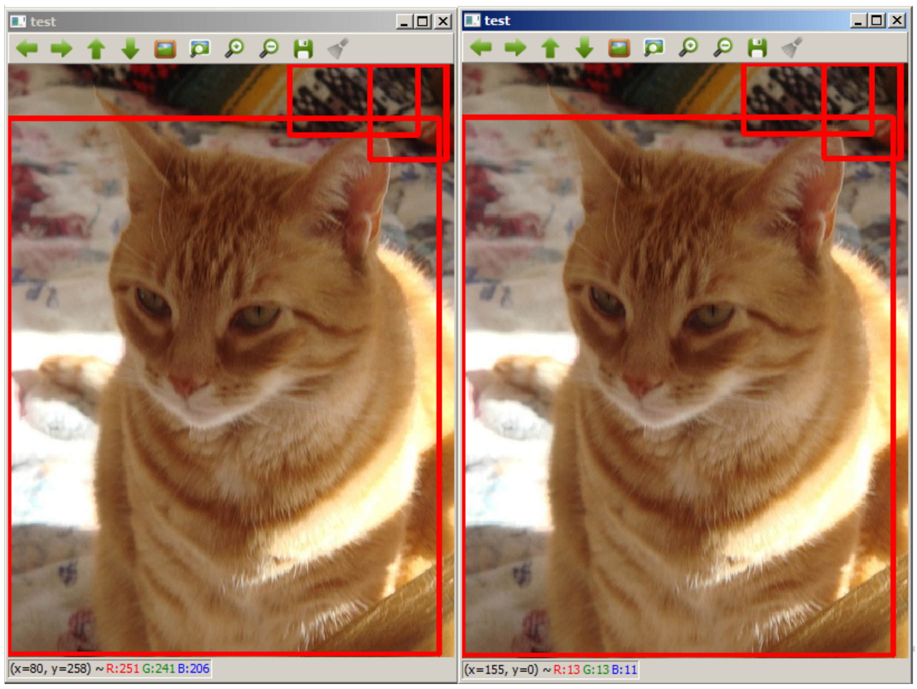

test
test
(x=80, y=258) ~ R:251 G:241 B:206
(x=155, y=0) ~ R:13 G:13 B:11

Figure 3.19: Comparison of latentsvm sample.

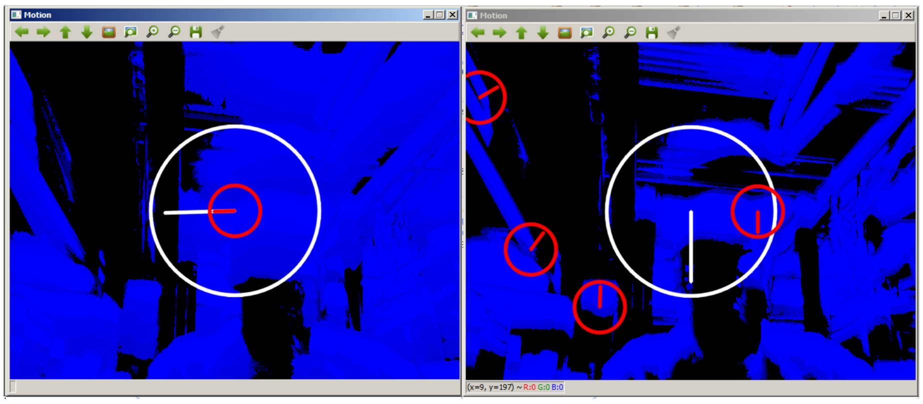


Figure 3.20: Comparison of motempl sample.

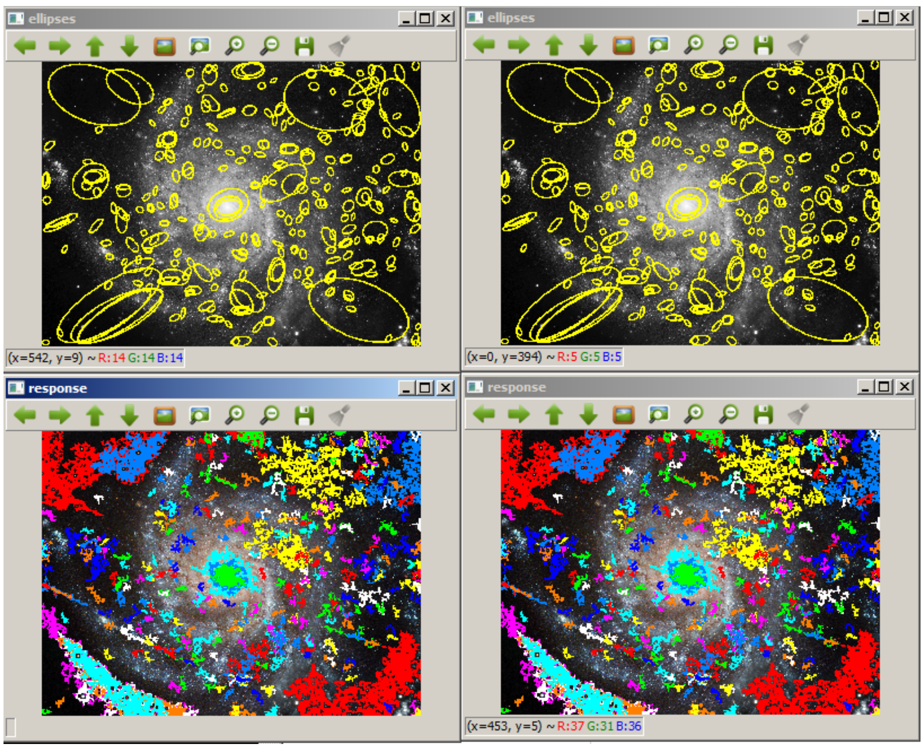

ellipses
(x=542, y=9) ~ R:14 G:14 B:14
ellipses
(x=0, y=394) ~ R:5 G:5 B:5
response
response
(x=453, y=5) ~ R:37 G:31 B:36

Figure 3.21: Comparison of mser sample.

OpenCv version and the functionality, OpenCvAda can be considered according to the results found an equal version compared to the C version.
The memory management in OpenCVAda is less efficient than OpenCV, in the best case approximately 10 percent and in the worst case 17 percent more memory was used. The average memory use over all four applications were 14.24 percent higher than OpenCV. All of the results are within the criteria specified in Executable reference (section 3.3.6).

Chapter 4

OpenCvAda User Guide

4.1 Getting Started

4.1.1 Installation

All that is required to install OpenCVAda is to extract it to a folder of choice, optionally this folder can be added to the system path in order to make it easier to include OpenCVAda in projects that will to utilize it.

4.1.2 Creating a OpenCVAda project in GNAT

OpenCVAda requires some functionality that is only available in the Ada 2005 standard. This means that all projects that utilizes this library are required to set the Ada 2005 syntax flag in the project settings window. Alternatively the flag can be set manually in the project file by adding the following code:

```
package Compiler is
        for Default_Switches ("ada")
            use ("-g", "-gnat05");
end Compiler;
```

In order to use the library its project file needs to be included, this can be achieved either by adding a dependency through the GNAT GPS IDE (right-click the project, select "Dependencies" in the "Project" sub-category) or by adding a with clause at the top of the project file:

```
with "<path_to_opencvada>/opencvada";
```

After adding the mentioned lines to a project file, OpenCVAda is linked in and will compile together with the project.

4.1.3 Hello World

This section presents a simple example of a minimalistic program that utilizes OpenCVAda to capture an image from a camera and showing it in a window to the user.

```
with Core; use Core; -- Ipl_Image
with Highgui; use Highgui; -- Cv_Capture and GUI functions

procedure Hello_World is
        Capture   : aliased Cv_Capture_Ptr :=
                          Cv_Create_Camera_Capture(0);
        Frame     : aliased Cv_Ipl_Image_Ptr := null;
begin
        Cv_Named_Window("Hello World",
                          Cv_Window_Autosize);

        Frame := Cv_Query_Frame(Capture);
        Cv_Show_Image("Hello World", Frame);

Cv_Wait_Key(0);

        Cv_Release_Capture(Capture'Access);
        Cv_Destroy_Window("Hello World");
end Hello_World;
```

When compiled and run, this program captures a single frame from an automatically selected camera. The program then waits until the user press a key, after which the program close the window and terminates. In the declaration stage of the program we have two variables. The first variable, Capture, is used to retrieve

```
        Capture   : aliased Cv_Capture_Ptr :=
                  Cv_Create_Camera_Capture(0);
```

Cv_Create_Camera_Capture returns a reference to a camera and allows the program to retrieve images from it. The parameter 0 means that OpenCV should use the first camera it is able to find. The next variable, Frame, is used to reference an image.

```
        Frame     : aliased Cv_Ipl_Image_Ptr := null;
```

The first thing the program does is to create a window where the captured image will be displayed.

```
        Cv_Named_Window("Hello World",
                          Cv_Window_Autosize);
```

In this case Cv_Named_Window creates a window with the title "Hello World", which is displayed in the window frame and is later used to reference the window in the code. By creating the window with the Cv_Window_Autosize flag OpenCV will control the window size itself and does not allow for re-sizing the window. The program then captures an image from the camera.

```
        Frame := Cv_Query_Frame(Capture);
```

Cv_Query_Frame asks for the next frame available from the buffer in Capture and returns that image to Frame. If there was a frame to retrieve there is now something that can be displayed in the window.

```
        Cv_Show_Image("Hello World", Frame);
```

With Cv_Show_Image the program can send an image to be displayed in a window. In this case the window handle is “Hello World” and the image to be displayed is referenced by Frame. In order to keep the program from terminating instantly after displaying the image the next statement can be used.

```
        Cv_Wait_Key(0);
```

This function waits until the user presses a key and returns the pressed key. The parameter specifies how many milliseconds Cv_Wait_Key should wait before returning in case no key-press event occurs. By setting the parameter to 0 Cv_-Wait_Key will wait indefinitely. All that remains is to free the resources that has been allocated and then terminate the program.

```
        Cv_Release_Capture(Capture'Access);
        Cv_Destroy_Window("Hello World");
```

Note that the Frame variable is not released in this case, the reason for that is because it references an image buffer stored inside Capture, which will be released when we release the Capture variable.

4.2 Basics

4.2.1 Naming Guide

Functions

Functions in OpenCvAda uses three rules for naming:

1. Words should be separated with underscore and have a capital first letter. (cvAvg becomes Cv_Avg)
2. "Constructors" should have Create added after Cv to distinguish themselves from the types they create. (cvPoint becomes Cv_Create_Point)
3. Except for rule 1 and 2 OpenCvAda should use the same names as OpenCv in C.

Remember that Ada unlike C is case insensitive and types can't have the same names as a function in the same package. Macros will keep their names when possible since they use a similar naming convention in C.

Types and Constants

Basic types uses two rules for naming in OpenCvAda:

1. Words should be separated with underscore and a have a capital first letter. (CvSize becomes Cv_Size)
2. Except for rule 1 OpenCvAda should use the same names as OpenCv in C.

OpenCvAda specific types uses the same naming convention.
Constants keep their names from C but will sometimes when appropriate be of a type, for example enumerations are changed to be constants of a specific type.

Pointers

There are two types of pointers in OpenCvAda first we have the Ada access types and the second is the C compatible Array/Pointer found in Interfaces.C.Pointers.
Ada access types uses the suffix _Ptr while C pointers uses the suffix _Pointer. (There are packages asociated with each C pointer that have the suffix _Pkg, these packages are needed for pointer operations or converting the C pointers to more user friendly Ada types)

Arrays

Array types names are built from the type in the array with the suffix _Array added to the type name.
For example the type Cv_Size have an array type associated with it called Cv_Size_Array. (In some cases there exists Arrays of pointers like Cv_Size_-Ptr_Array or a pointer of an array like Cv_Point_array_Ptr)

4.2.2 Standard Types

Integer and Float arrays

Arrays of integer and floating point values can be passed to functions as is. However, passing an array access is not possible because of the array bound information that is stored in the access. Therefore it is necessary to pass these types of parameters as C compatible pointers.

```
procedure Some_Procedure is
        Arr     : Cv_32f_Array_Ptr :=
                  new Cv_32f_Array(0 .. 100);
        Arr_Ptr : Cv_32f_Pointer := Arr.all(0'Access);
begin
        Cv_Procedure_With_Pointer_Param (Arr_Ptr);
end Some_Procedure;
```

Note that you need an access to the first element in the array and not the array itself.

Images and Matrices

Cv_Arr_Ptr

Many of the functions in OpenCVAda work with different types of data, such as Ipl_Image, Cv_Mat and Cv_Sparse_Mat. In most cases these functions are overloaded to take each compatible type separately, but in some cases this is not possible. When there are too many different types (or multiple parameters that do not share a common type) it is necessary to make an unchecked conversion to a Cv_Arr_Ptr which is used as a generalised type.

Ipl_Image_Ptr

Images are represented with the Ipl_Image type, however this type should not be used directly, instead the access type Ipl_Image_Ptr should be used whenever an image is required. The reason for this is that image data is allocated dynamically with the Cv_Create_Image function and should be released with the Cv_Release_Image procedure.

The Cv_Mat, Cv_Sparse_Mat and Cv_Mat_ND packages

The matrix types in OpenCVAda use generics in order to store data of an arbitrary type. In order to use these types the developer needs to instantiate the relevant package with the type of data to be stored in the matrix. For example,

```
package Mat_32f is new Core.Mat(Float);
```

will instantiate a package that can be used to declare matrices containing Float data. A matrix can now be declared in the following way,

```
Mat : Mat_32f.Cv_Mat_Ptr :=
        Mat_32f.Cv_Create_Mat(10, 10, Cv_32f, 1);
```

The above statement will declare and create an empty 10x10 matrix with 1 channel. The generic packages contains various unchecked conversions in order to make them compatible with the non-generic packages. For example,

```
Cv_Show_Image("Example", To_Arr_Ptr(Mat));
```

will convert the generic matrix to a Cv_Arr_Ptr.
The Core package also contains the three matrix types as non-generic versions, these types are not meant to be used directly but rather acts as place holders for functions that take a matrix as a parameter. Unchecked conversions are available to and from these placeholder types as well. For example the Cv_-Show_Image function can also take a Cv_Mat_Ptr as the image to show.

```
Cv_Show_Image("Example", To_Mat_Ptr(Mat));
```

Null Records (“Black Box” types)

In OpenCvAda there are a few null record types that have no representation in actual Ada code and should only be used with the provided access type. The null records are these types:

- Cv_Face_Tracker_Ptr in Legacy
- Cv_File_Storage_Ptr in Core
- Cv_Contour_Scanner_Ptr in Imgproc
- Cv_Feature_Tree_Ptr in Imgproc
- Cv_Lsh_P in Imgproc
- Cv_Lsh_Operations_Ptr in Imgproc
- Cv_Capture_Ptr in Highgui
- Cv_Video_Writer_Ptr in Highgui
- Cv_Hid_Haar_Classifier_Cascade_Ptr in Objdetect

Strings

OpenCvAda uses standard Ada strings in all functions and the user do not need to add any ASCII characters to make them C compatible.

4.2.3 Highgui

Highgui is the built in GUI for OpenCv, it is not a full GUI but only usable for displaying images and a few helpful controls like buttons and trackbars. Some of the functions in the Highgui package have two versions one that is a function that returns a status value and one that is a procedure.

Creating a Window

OpenCvAda allows you to create windows in two ways, using the Cv_Named_-Window function or by using a function that references a window on a nonexistent window.

- Cv_Named_Window takes two parameters the string is the name of the window used to reference it and the flag sets the way a window can be re-sized:
- Cv_Window_Normal allows the user to re-size the window but keeps the aspect ratio.
- Cv_Window_Freeratio allows the user to re-size the window without keeping the aspect ratio.
- Cv_Window_Autosize automatically sets the size of the window to match the image shown in the window.

Cv_Window_Autosize is the standard flag. Every window used in OpenCvAda should be destroyed before ending the program or the program will raise a segfault.

```
begin
        Cv_Named_Window("Main",Cv_Window_Normal);
        ...
        Cv_Destroy_Window("Main");
```

Displaying Images

OpenCvAda can display both Ipl_Image_Ptr and Cv_Mat with the Cv_-Show_Image function. Cv_Show_Image takes the window name and the image as parameters.

```
        Cv_Show_Image("Main", Image);
```

Cv_Show_Image is only able to display image with 8 bit depth so if other depths are used the image needs to be converted.

Input

There are several ways to interact with the user with the help of Highgui. Cv_Wait_Key reads the next key pressed on the keyboard on a delay, Cv_Create_Trackbar adds a trackbar to interact with. There are also QT specific buttons that can be created in a toolbar window. Cv_Wait_Key parameter ms_delay(the standard value 0 waits indefinitely or until a key is pressed) is the time in miliseconds the function should wait for a key press to occur before returning with a null value, the function returns the Character pressed while the procedure ignores the return value. (however it will still wait until a key is pressed if ms_delay is 0) Here are three common usages of Cv_Wait_Key:

```
        -- wait until a key is pressed
        Cv_Wait_Key(0);
```

```
        -- wait until a key is pressed
        -- or 100 ms have passed
        Char := Cv_Wait_Key(100);
```

```
loop
        -- ends a loop when ESC is pressed
        exit when Cv_Wait_Key(30) = Ascii.Esc;
end loop;
```

Cv_Create_Trackbar is used to create trackbar or slider in a window so that it is possible to change a value on the fly. The parameters are:

- Trackbar_Name, string with the name of the trackbar.
- Window_Name, string with the name of the window where the trackbar should be attached.
- Value, access to an integer that contains the current value of the trackbar.
- Count, maximum value of the trackbar.
- On_Change, access to a procedure of the type Cv_Trackbar_Callback that is called when the value of the trackbar changes.

Example with the Cv_Trackbar_Callback type: (First the code needed to change the procedure into a C compatible procedure, then the actual code)

```
procedure On_Trackbar(Position : Integer);
pragma Convention (C, On_Trackbar);

Position_Value : aliased Integer := 3;
Max_Value : constant Integer := 10;
begin
Cv_Create_Trackbar("Main␣trackbar", -- trackbar name
        "Main", -- name of the window
        Position_Value'Access,
        Max_Value,
        On_Trackbar'Access);
```

Each time the trackbar changes the code in On_Trackbar will be executed. In this example we assume a global image called Global_Image exists and each time the code is executed the Cv_Erode function will be called with iterations parameter set to the position of the trackbar. Remember to release any local variables declared in a callback function.

```
procedure On_Trackbar(Position : Integer) is
        Local_Image : aliased Ipl_Image_Ptr;
begin
        -- Create a new image to store the result
        Local_Image := Cv_Create_Image
                (Cv_Get_Size(Global_Image),
                 Global_Image.all.Depth,
                 Global_Image.all.N_Channels);

        -- Run Cv_Erode with Iterations = Position
        Cv_Erode(Global_Image,Local_Image,null,Position);

        -- Show the new "eroded" Image in window Main
        Cv_Show_Image("Main",Local_Image);

        -- Release the local image
        Cv_Release_Image(Local_Image'Access);
end On_Trackbar;
```

Output

The highgui packages contains a few ways to giving the user an output string. Cv_Add_Text adds a string to an image, Cv_Display_Overlay shows a text string at the top of the image and Cv_Display_Status_Bar changes the value of the status bar below the image. Cv_Add_Text takes four parameters, the image where the text is added, the string that contains the text to be added, a Cv_Point where the text should start in the image and the Cv_Font to use.

```
        Cv_Add_Text     (Image,
                         "Hello World!",
                          -- top left corner
                         Cv_Create_Point(0,0),
                         -- scale and thickness
                         Cv_Create_Font(10.0, 2));
```

In addition to Cv_Add_Text the Cv_Put_Text function is also able to change the color of the text.

```
        Cv_Put_Text     (Image,
                         "Hello World!",
                          -- top left corner
                         Cv_Create_Point(0,0),
                          -- scale and thickness
                         Cv_Create_Font(10.0, 2),
                         -- the color red
```

```
                    Cv_RGB(255,0,0));
```

Cv_Display_Overlay and Cv_Display_Status_Bar both takes the same parameters the only difference is where the text appears in the window.

- Name, is the name of the window where the text should be displayed.
- Text, is the string containing the text.
- Delay_Ms, is how long the text should be displayed.

Cv_Display_Overlay shows the text at the top of the image below the toolbar.

```
        Cv_Display_Overlay("Main",
                           "Hello␣World!",
                           5000);
```

Cv_Display_Status_Bar shows the text at the bottom of the window in the status bar.

```
        Cv_Display_Status_Bar("Main",
                              "Hello␣World!",
                              5000);
```

4.2.4 Camera, Video and Images

How to get an image from a camera

To get an image from a camera two variables are needed, Cv_Capture_Ptr and Ipl_Image_Ptr.

- Cv_Capture_Ptr is an internal representation used to keep track of a camera.
- Ipl_Image_Ptr is a representation of an image and is fully accessible from Ada.

Cv_Create_Camera_Capture(<Camera ID>) is used to create the Cv_Capture_Ptr variable. CV_Query_Frame(<Cv_Capture_Ptr>) returns the current frame from the camera as an Ipl_Image_Ptr.

```
        Capture : aliased Cv_Capture_Ptr;
        Image : Ipl_Image_Ptr;
begin
        -- autopick the camera.
        Capture := Cv_Create_Camera_Capture(0);
        Image := Cv_Query_Frame(Capture);
```

An image from a Cv_Capture_Ptr is a special case when using release functions to free memory and should not be freed. However the Cv_Capture_Ptr should be released when it is not used anymore.

```
        Cv_Release_Capture(Capture'Access);
```

Stereo Camera

When using the Cv_Capture_Ptr type for a stereo camera it is recommended to use the function Cv_Grab_Frame and Cv_Retrieve_Frame instead of Cv_-Query_Frame (a wrapper of the two functions) since you will want the frames as close in time as possible. Cv_Grab_Frame saves an image buffer and is considered to be “fast”, while Cv_Retrieve_Frame returns the Ipl_Image_Ptr of the buffer. An example of usage of a stereo camera with Cv_Capture_Ptr:

```
        Left_Capture  : aliased Cv_Capture_Ptr;
        Right_Capture : aliased Cv_Capture_Ptr;
        Left_Image, Right_Image : Ipl_Image_Ptr;
begin
        Left_Capture := Cv_Create_Camera_Capture(1);
        Right_Capture := Cv_Create_Camera_Capture(2);

        -- Grab the frames from the cameras
        -- (This is considered to be fast in OpenCv)
        Cv_Grab_Frame(Left_Capture);
        Cv_Grab_Frame(Right_Capture);

        -- Now the buffers should be
        -- as close as possible in time.
        -- We can now access the buffers
        -- as Ipl_Image_Ptr when ever we want.
        Left_Image := Cv_Retrieve_Frame(Left_Capture);
        Right_Image := Cv_Retrieve_Frame(Right_Capture);
```

How to get an image from a video file

Using the Cv_Create_File_Capture function instead of a Cv_Capture_Ptr tied to a camera we can have a Cv_Capture_Ptr that retrieves images from a video file. Except for the parameter being a string instead of an integer reference to the camera the Cv_Capture_Ptr created with Cv_Create_Camera_Capture is used exactly the same.

```
        Capture :=
                Cv_Create_File_Capture("hello_world.avi");
```

How to get an image from an image file

To open a separate image file the Cv_Load_Image function is used. Cv_-Load_Image takes two parameters, a string containing the file name (path can be added if needed) and an integer with a flag that decides which color type should be used(Cv_Load_Image_Color is the standard value and creates an image that contains colors).

```
        Image : aliased Ipl_Image_Ptr;
begin
        -- opens the image as grayscale.
        Image := Cv_Load_Image("Hello.jpg",
                                Cv_Load_Image_Grayscale);
```

Images created with Cv_Load_Image should be released when they are not used anymore.

```
      Cv_Release_Image(Image'Access);
```

Saving images

With Cv_Save_Image images can be saved to the hard drive to a normal image file(png,jpg, etc), Cv_Save_Image takes two parameters, the file name with a file name extension that decides the file format to save the image as and the Ipl_Image_Ptr or Cv_Mat_Ptr that will be saved.

```
      -- saves the image as a png file.
      Cv_Save_Image("Output.png", Image);
```

Saving video files

Cv_Create_Video_Writer and Cv_Write_Frame allows for the creation of video files. Cv_Create_Video_Writer creates a pointer that is passed to Cv_-Write_Frame and contains the information used to add a frame to the video file.

- Cv_Create_Video_Writer needs six parameters:
- Filename, the name of the video file and the extension.
- Fourcc, an integer created from four characters that decides the codec see Cv_Fourcc below.
- Fps, a long float with the amount of frames that should be shown per second.
- Width, integer width.
- Height, integer height.
- Is_Color, 1 for color, 0 for gray scale.

The Cv_Fourcc function creates a codec integer from four characters, or Cv_-Fourcc_Prompt can be used in windows to pick a codec from a list.

```
      Writer : Cv_Video_Writer_Ptr;
begin
      Writer := Cv_Create_Video_Writer
                  ("output.avi",
                  -- uses the ffdshow codec.
                  Cv_Fourcc('F','F','D','S'),
                  24.0, -- fps
                  640, -- width
                  480, -- height
                  1); -- color
```

Cv_Write_Frame adds a Ipl_Image_Ptr to the specified Cv_Video_Writer_-Ptr.

```
        Cv_Write_Frame(Writer, Image);
```

4.3 Advanced

4.3.1 Unchecked Conversions

In certain cases it is required to use an Ada.Unchecked_Conversion between two pointers in OpenCvAda, the reason for this is that in many cases the actual type returned by functions or contained in records are not known until they are put into a context, this is more often then not due to Cv_Seq_Ptr and related records and or functions.
Here is a larger example of how Ada.Unchecked_Conversions are used with OpenCvAda, that uses the Cv_Canny and Cv_Hough_Lines2 to find lines in an image.
Since we know that the Cv_Seq returned by Cv_Hough_Lines2 will contain four integers representing the start and end point of a line, we can see it as an array of Cv_Point with the length of two, the values will be in a Cv_Void_-Ptr that can be retrieved from the Cv_Seq. Unchecked_Conversions can only convert between actual types of the same size (e.g. access type to access type) so we need to create a type that is an access to a array of Cv_Point, that can be converted from a Cv_Void_Ptr.

```
        subtype Line_Array is Cv_Point_Array (0 .. 1);
        type Line_Array_Ptr is access Line_Array;

        function To_Line_Array_Ptr is
                 new Ada.Unchecked_Conversion
                        (Target => Line_Array_Ptr,
                         Source => Cv_Void_Ptr);

        Line     : Line_Array;
        Lines    : Cv_Seq_Ptr;
begin
```

Cv_Hough_Lines2 returns a Cv_Seq_Ptr, the data can be retrieved from the Cv_Seq_Ptr using the Cv_Get_Seq_Elem function that returns a Cv_Void_-Ptr so here is when we need to use the Unchecked_Conversion that was created above.

```
        Lines := Cv_Hough_Lines2(Dst,
                                 To_Void_Ptr (Storage),
                                 Cv_Hough_Probabilistic,
                                 1.0, Cv_Pi / 180.0,
                                 100, 5.0, 10.0);

        for I in Integer range 0 .. Lines.all.Total - 1
        loop
```

```
                  Line :=
                          To_Line_Array_Ptr
                                  (Cv_Get_Seq_Elem
                                          (Lines, I)).all;
                                  Cv_Line (Image,
                                  Line (0),
                                  Line (1),
                                  Cv_Rgb (255, 0, 0),
                                  3,
                                  8);
        end loop;
```

The full example can be found in OpenCvAda/Samples/Hough_Lines.adb or as an appendix to this document.

4.4 The Demo Application

This application allows the user to test the Cv_Corner_Harris function with different values on the fly by using trackbars.
Packages needed for this application:

```
with Highgui;                   use Highgui;
with Core;                      use Core;
with Core.Operations;           use Core.Operations;
with Imgproc;                   use Imgproc;
with Imgproc.Operations;        use Imgproc.Operations;
```

The variables used in the application, the aliased variables except for the slider values are aliased so that they can be released when not used anymore.

```
procedure Demo is
        -- Parsed variables created from the trackbars
        Aperture_Size           : Integer := 1;
        Block_Size              : Integer := 1;

        -- Unparsed variables used with the trackbars
        Slider_A_Value          : aliased Integer := 1;
        Slider_B_Value          : aliased Integer := 1;

        -- Variables used with the camera.
        Capture                 : aliased Cv_Capture_Ptr;
        Image                   : Ipl_Image_Ptr;

        -- Temporary images
        Corners, Image_Gray     : aliased Ipl_Image_Ptr;

        -- Used to convert a 32F image to a 8U image.
        Scale, Shift            : Long_Float;
        Min, Max                : aliased Long_Float;
```

Trackbar procedure that is used to change the parameter Aperture_Size in the Cv_Corner_Harris procedure. The value must be between 1 and 31, and can not be even.

```
        -- Change the convention to match a C function.
        procedure Slider_A (Position : Integer);
        pragma Convention (C, Slider_A);

        procedure Slider_A (Position : Integer) is
        begin
                -- Value is 0 or even.
                if Position = 0 or Position mod 2 = 0 then
                        Aperture_Size := Position + 1;
                else
                        Aperture_Size := Position;
                end if;
        end Slider_A;
```

This trackbar procedure is for the Block_Size parameter and the value can not be 0.

```
        -- Change the convention to match a C function.
        procedure Slider_B (Position : Integer);
        pragma Convention (C, Slider_B);

        procedure Slider_B (Position : Integer) is
        begin
                if Position > 0 then
                        Block_Size := Position;
                else
                        Block_Size := 1;
                end if;
        end Slider_B;
```

Creates two windows, one for showing the original image and the other one for showing the result of Cv_Corner_Harris. Also the trackbars are added to the Corner window. Unrestricted_Access to the trackbar callbacks is needed since they are subprograms.

```
begin
        -- Create Windows,
        -- Corners window will be resizeable.
        Cv_Named_Window ("Original",
                          Highgui.Cv_Window_Autosize);
        Cv_Named_Window ("Corners",
                          Highgui.Cv_Window_Normal);

        -- Move the windows,
        -- so they are not on top of eachother.
        Cv_Move_Window ("Corners", 700, 50);
        Cv_Move_Window ("Original", 0, 50);
        Cv_Create_Trackbar("Aperture",
                           "Corners",
                           Slider_A_Value'Access,
                           31, -- Max value.
                           Slider_A'Unrestricted_Access);
        Cv_Create_Trackbar("Block",
                           "Corners",
                           Slider_B_Value'Access,
                           11, -- Max value.
                           Slider_B'Unrestricted_Access);
```

Loop until the ESC button is clicked, at the start of each loop retrieve a new image from the capture device and show it in the Original window.

```
        Capture := Cv_Create_Camera_Capture (0);

        loop
                exit when Cv_Wait_Key (100) = Ascii.Esc;
                Image := Cv_Query_Frame (Capture);
                Cv_Show_Image ("Original", Image);
```

Create two temporary images, Image_Gray is the gray scale version of Image created with Cv_Cvt_Color while Corners contains the result of the Cv_-Corner_Harris procedure.

```
                Corners := Cv_Create_Image
                        (Cv_Get_Size(Image),
                        Ipl_Depth_32f,
                        1);
                Image_Gray := Cv_Create_Image
                        (Cv_Get_Size (Image),
                        Ipl_Depth_8u,1);
                Cv_Cvt_Color
                        (Image, Image_Gray, Cv_Rgb2gray);
                Cv_Corner_Harris(Image_Gray,
                                 Corners,
                                 Block_Size,
                                 Aperture_Size);
```

Convert the result from Cv_Corner_Harris back to a 8U image so that it can be show in a window.

```
                Cv_Min_Max_Loc (Corners,
                                Min'Access,
                                Max'Access,
                                null, null, null);

                Scale := 255.0 / (Max - Min);
                Shift := -Min * Scale;

                Cv_Convert_Scale
                        (Corners,
                          Image_Gray, Scale, Shift);
                    Cv_Show_Image ("Corners", Image_Gray);
```

Release the temporary images so that there is no memory leak and then end the loop.

```
                    Cv_Release_Image (Image_Gray'Access);
                    Cv_Release_Image (Corners'Access);
          end loop;
```

Chapter 5

Interfacing Ada with C

5.1 Basic types

Types found in the Standard, Interfaces and Interfaces.C in Ada can be used to map C types. If portability is a huge concern (between architectures while using a different compiler mostly) Interfaces.C should be more correct, in the cases where Interfaces.C and Standard and Interfaces are not the same types(which is the case in most of the types i GNAT so this is a void issue).
To find out if two types are equal in C and Ada a compression of the sizes is done, and as long as a type is accessed in only one of the languages having the same size is enough. However for the types to make sense in both languages the actual representation of the types needs to be the same. So C will not care if an Integer is mapped to a Float, it will be hard to track changes made in both languages if such a mapping is made.
Enumerations types in Ada is fitted into the smallest possible Integer type available, while in C int is used. Using pragma Convention C on the enumeration type will force the Ada type to be equal to the C type.

5.2 Composite types

C structs and unions are mapped in Ada as records, unions use variant records with the C union convention. Unlike the variant records a union ignores the option variable and because of this it can be type equal to a C union. When structs are used directly as parameters the corresponding record must have the C_Pass_By_Copy convention to mimic the behaviour of the C struct. When a struct is instead used with a pointer the C_Pass_By_Copy convention is not needed.
C struct for comparison:

```
typedef struct {
        int x;
        int y;
} foo;
```

Ada record that is possible to pass as a parameter in a C function:

```
type foo is record
        x : Integer;
        y : Integer;
end record;

pragma Convention (C, foo);
pragma Convention (C_Pass_By_Copy, foo);
```

An Ada variant record is not type equal to a C union since the “option variable” is part of the record, however if the pragma Unchecked_Union[?] is used the “option variable” is ignored.
C union for reference:

```
typedef union{
        int ivalue;
        unsigned uvalue;
        float fvalue;
} foo;
```

Ada union that is mappable to a C union:

```
type Foo_Options is (One,Two,Three);
type Foo(Option : Foo_Options := Foo_Options'First) is
record
        case Option is
                when One =>
                        Ivalue : Integer;
                when Two =>
                        Uvalue : Unsigned_32;
                when Three =>
                        Fvalue : Float;
        end case;
end record;
pragma Unchecked_Union (Foo);
pragma Convention (C,Foo);
```

5.3 Interfacing C functions

Functions from C can be imported to Ada by using the import pragma. The pragma takes three parameters, the language convention determines which language the function is written in, the entity parameter is which local function will be used for the binding and the external name is the name of the function to be imported.
If the function to be imported does not have a return value (i.e. void foo(...)) the local entity needs to be a procedure, otherwise it should be a function with a matching return value as the C function.
For example, consider two functions c_foo and c_bar like they are declared below.

```
int c_foo(int i, char c);
void c_bar(int i, char c);
```

In order to import these functions it is required to declare a function and a procedure for c_foo and c_bar respectively. After the declarations the functions can now be imported with the import pragma.

```
function Foo (I : Integer; C : Character) return Integer;
procedure Bar (I : Integer; C : Character);

pragma Import (C, Foo, "c_foo");
pragma Import (C, Bar, "c_bar");
```

Matching the parameters from C can be problematic without access to the complete source code or very well documented functions. The problem is discerning whether pointers passed as parameters point to a single variable or an array of variables. For example,

```
int func(int *param);
```

Without further information about the function it is not possible to determine if param is just the address to an integer or an array of integers. For further information see Pointers (section 5.4).

5.3.1 Macros from C

. Some C implementations make extensive use of macros for short snippets of code in order to avoid the overhead a function call results in. If a macro needs to be imported to Ada it must be wrapped inside a C function since macros are not represented as symbols in the object files. The cost of doing this is the added overhead of a function call in Ada because for all intents and purposes the macro is now a regular function. As and example, consider a macro that sets a particular bit in a variable. To import this macro to Ada we first create a wrapper function in C.

```
#define SET_BIT(value, bit) ((value) | (1 << (bit)))
int set_bit_wrap(int value, int bit)
{
        return SET_BIT(value, bit);
}
```

After the wrapper function has been created we import the function with the import pragma as described earlier in Interfacing C functions (section 5.3.

```
function Set_Bit (Value : Integer;
                  Bit : Integer) return Integer;
pragma Import (C, Set_Bit, "set_bit_wrap");
```

Depending on the nature of the macro it might be necessary to create several wrapper functions. For instance, the set_bit example macro could be used with other types than standard integers, if more types need to be used with this macro it is necessary to declare additional wrapper functions for each different type to be used.

5.4 Pointers

Ada can reference a pointer in three ways: the access of a type, an array or by using the Interfaces.C.Pointers. Also In the special case of a char * the Ada type String and the package Interfaces.C.Strings can instead be used, see more in Strings (section 5.6.3). Another special case is the record, where the array must be used with an access type.
When a C function can be mapped as a procedure it is possible in Ada to use an in out parameters to map a pointer instead, this should be considered for future use starting in Ada 2012 see In out parameter(section 5.5).
If an array is created in C it will begin on index 0 regardless of what you try to bound it to in Ada, also most C implementations will assume an array starts on index 0.

5.4.1 What could a pointer from C be?

Consider a function in C with the parameter of the type T * param, what can it actually be?

- A pointer to one element of the type T.
- An array of T.

What about the T ** type what different meanings can it have?

- A pointer to a pointer of type T.
- A pointer to an array of T.
- A two dimensional array of type T.

This of course continues for each level of the pointer with adding a N-level array and pointers to the previous level alternatives.
The single pointer is trivial to map in Ada since the Interfaces.C.Pointers can be used in both cases, if a more high level Interface is required, an analysis of the code that is imported is needed to map it to either an array or an access type. The second level pointer is trickier since to map a two dimensional array successfully, it is needed in most cases to use an array of Interfaces.C.Pointers. So that each array element is a pointer from Interfaces.C.Pointers.

5.4.2 Mapping int * to Ada

Consider the following function in C:

```
void foo(int *value);
```

Without knowing if the parameter is an array or just a pointer to a single value we present the three ways that is always usable to map the parameter in Ada.

Access to a variable

Using the access keyword a direct translation from a C pointer to an Ada type is possible, however since pointer arithmetic's in Ada do not work on access types this way will not be able to map an array without using type conversions... Using type conversions is not recommended when it is not necessary (see Void * in section 5.6), due to unsafe practices and added complexity.
Ada specification of a function comparable to the C function (in section 5.4.2)using access:

```
procedure foo(value : access Integer);
```

Arrays

It is possible to map a C pointer directly to an Ada array. An array in Ada can even be used to reference a single value pointer if an array with only one element is used.
Ada specification of a function comparable to the C function(in section 5.4.2) using array and type declaration of an array type:

```
type Integer_Array is array (Integer range <>)
        of aliased Integer;

procedure foo(values : Integer_Array);
```

Mapping a bounded array as a pointer will of course cause problems, since a C pointer or array does not not share its bounds with Ada. If an element that does not exist in C is referenced in Ada an access violation will occur. This of course works both ways so if C is expecting an array with ten elements and Ada provides one with five elements an access violation will occur in C instead.
Something worth nothing is that access to an array does not equal a pointer to an array in C and is instead equal to just a normal array in C.

Interfaces.C.Pointers

Using the generic package Interfaces.C.Pointers to map C pointers in Ada might add a bit more complexity then using the previous alternatives. Interfaces.C.Pointers does make up for the added complexity with additional functionality like pointer arithmetic's and the possibility of terminated arrays to avoid the problems found when using arrays to map C pointers.
Ada specification of the C function (in section 5.4.2) using Interfaces.C.Pointers to map a C pointer into Ada:

```
type Integer_Array is array (Integer range <>)
        of aliased Integer;
package Integer_Pointer is
        new Interfaces.C.Pointers(Integer,
                                  Integer,
                                  Integer_Array,
                                  0);

procedure foo(values : Integer_Pointer.Pointer);
```

5.4.3 Mapping int ** to Ada

Consider the following C function taking a double pointer integer as a parameter:

```
void foo (int ** value);
```

We already know what a single level pointer can be referencing, but how is this translated in Ada?

Access to access

If the value variable is a pointer to a pointer then an access type is needed since Ada does not support anonymous types (type without a name) that is access access.

```
type Integer_Ptr is access Integer;
procedure foo (value : access Integer_Ptr);
```

While an int ** might not be used a lot, if we considered value instead as a linked list then it is a lot more common to use the ** , so that it is possible to make changes on the linked list. The only change needed to use this way to reference a C Pointer with a record instead, is to change the access type to be an access to a record instead.

Access to array

Access to Ada arrays do not equal a pointer to an array in C, so to map a int ** as a pointer to an array in Ada it is required to create an access type to an array and use access to that type instead.

```
type Integer_Array is array (Integer range <>)
        of aliased Integer;
type Integer_Array_Access is access Integer_Array;
procedure foo (value : access Integer_Array_Access);
```

When this is actually used an array with bounds needs be be passed even if the array is created in C.

Access to Interfaces.C.Pointers

A pointer to an array can also be mapped with Interfaces.C.Pointers by passing it as access to the C function.

```
type Integer_Array is array (Integer range <>)
        of aliased Integer;

package Integer_Pointer is
        new Interfaces.C.Pointers(Integer,
                                  Integer,
                                  Integer_Array,
                                  0);

procedure foo(values : access Integer_Pointer.Pointer);
```

Two dimensional arrays

Mapping an unbounded two dimensional array from C in Ada is a bit trickier. Since it needs to be an array of Interfaces.C.Pointers pointers.

```
type Integer_Array is array (Integer range <>)
        of aliased Integer;
package Integer_Pointer is
        new Interfaces.C.Pointers(Integer,
                                  Integer,
                                  Integer_Array,
                                  0);
type Integer_Pointer_Array is array (Integer range <>)
        of Integer_Pointer.Pointer;

procedure foo(value : Integer_Pointer_Array);
```

Just like one dimensional arrays if the two dimensional array has bounds in C then an other method can be used see Constrained Arrays(section 5.6.2) for more information.

5.5 In out parameter

Another way to map C pointers in Ada is, when they are passed as parameters is to use the in and out keywords on the parameter instead. So a variable that is a pointer in C can be called as a in out variable in Ada instead. In Ada 2005 only procedures can use in out parameters, in the upcoming Ada 2012[?] functions will also have support for in out parameters.
An in out parameter is equivalent to a parameter passed as access.
C void function taking a pointer as parameter:

```
void foo(int * value);
```

The corresponding Ada function mapping the C pointer using an in out parameter:

```
procedure foo (value : in out Integer);
```

In Ada 2012 using in out parameters to map C pointers should be a homogeneous way of mapping different pointer types seamlessly in Ada. More testing will be required for the more advanced types of pointers to make sure they are still working correctly when passed as in out parameters.

5.6 Void *

In C the void * is often used to allow the use of arbitrary types in functions or structs, since Ada does not have a void type we need a way to work around it. The first way of mapping a void * in Ada is by using generics to allow arbitrary types, if generics is used any Ada type can be sent to a function by sending an access to the generic type. The second way is to use an access type that is converted to the desired type with Ada.Unchecked_Conversion.
C void pointer example:

```
void foo(void * param);
```

```
double value = 0.1;
foo((void *)&value);
```

Ada void pointer using generics example:

```
generic
        type Element is private;
procedure foo (Param : access Element);
```

```
procedure Foo_Lf is new foo(Long_Float);
Foo_Lf(0.1);
```

Void pointer in Ada using Unchecked_Conversion example:

```
type Lf_Ptr is access all Long_Float;
type Void_Ptr is access all Integer;

function To_Void is new Ada.Unchecked_Conversion
        (Target => Void_Ptr, Source => Lf_Ptr);
procedure foo (Param : Void_Ptr);
```

```
Value : aliased Long_Float := 0.1;
foo(To_Void(Value'access));
```

5.6.1 Pointers in records

Normal pointers work in the same way as when mapping C pointers with Ada functions, however when mapping pointers as arrays in records instead access to an array is needed. Unless the array is bounded in C then a similar bounded array in Ada will be equal.
A C struct with three different types of pointers:

```
typedef struct {
        int * value;
        int * value_array;
        int values[10];
} foo;
```

The corresponding Ada records with all three pointers mapped to Ada:

```
type Integer_Array is array (Integer range <>) of Integer;
type Integer_Array_Ptr is access Integer_Array;

type foo is record
        value : access Integer;
        value_array : Integer_Array_Ptr;
        values : Integer_Array(0 .. 9);
end record;
```

The reason for the unbounded array to be used as an access type instead of directly as an array is due to Ada having bounds information embedded in the type. In a similar way unbounded two dimensional arrays must also be used as access in records.

5.6.2 Constrained Arrays

Arrays with known constraints can be directly mapped between Ada and C. Although it is not common to use constrained arrays in parameters in C consider the following function that takes such a parameter.

```
void c_proc (int arr[5]);
```

The equivalent in Ada is to declare a constrained array type and a procedure that takes the new type as a parameter. For example,

```
type Constrained_Array is array (Integer range 1 .. 5)
        of Integer;

procedure Proc (Arr : Constrained_Array);
pragma Import (C, Proc, "c_proc");
```

It is also possible to use unconstrained array types from Ada to map to a constrained array in C.

```
type Unconstrained_Array is array (Integer range <>)
        of Integer;

procedure Proc (Arr : Unconstrained_Array);
pragma Import (C, Proc, "c_proc");
```

However, this is prone to errors as an array that is smaller than the bounds expected by the C function might cause undefined behaviour.

```
-- Expected parameter by Proc (c_proc), works
Proc((1, 2, 3, 4, 5));
-- Array to short, potential heap corruption
Proc ((1, 2, 3));
-- A longer array causes no problems
Proc ((1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
```

The three different cases above shows the different cases an unconstrained array type might be used. The array in the first case is the appropriate size and therefore causes no problems. The second array is too short, this might lead to undefined behaviour. The last case might look as though it could potentially cause undefined behaviour, however, this works fine and the function can access data outside the specified bounds (although that would constitute an improper implementation).

5.6.3 Strings

A C char * can be mapped like a regular pointer or by using the string specific types from Ada String and Interfaces.C.Strings.Chars_Ptr. What needs to be remembered is that char * needs to be nul terminated in most cases, Ada strings are not, Interfaces.C.Strings.Chars_Ptr on the other hand is nul terminated.

5.6.4 Function pointers

Ada provide accesses to subprograms that can be mapped directly to the function pointers available in C. A function pointer in C can be declared in one of two ways, either by creating a new type with the keyword typedef.

```
typedef int (*Function_Pointer) (int);

void Function_Pointer_Example (Function_Pointer func);
```

The above example creates a type Function_Pointer that is a pointer to a function which takes an integer as a parameter and returns an integer as result. If the type will only be used for one specific function it can be more convenient to declare the function pointer anonymously directly in the functions parameter list, for example

```
void Function_Pointer_Example
        (int (*Function_Pointer) (int));
```

Ada 2005 supports both methods of having accesses to subprograms as well, however when interfacing with C it is not possible to use anonymous types because the Ada access types require a convention pragma, which is not possible to specify for an anonymous type.

```
type Function_Access is
        access function (Param : Integer) return Integer;
pragma Convention (C, Function_Access);

procedure Access_Example (Func : Function_Access);
pragma Import
        (C, Access_Example, "Function_Pointer_Example");
```

The example above works because it is possible to set the convention of the access type, making it compatible with C pointers. Although anonymous subprogram access types were introduced in Ada 2005, it is not possible to use an anonymous access when interfacing with C. Consider the following example:

```
procedure Access_Example
        (Func : access function (param : Integer)
                         return Integer);
pragma Import
        (C, Access_Example, "Function_Pointer_Example");
```

Although the code is syntactically correct it will not compile because of a convention mismatch in the Func parameter. Since access types require a C convention when used in C interfaces it is impossible to use an anonymous type because no convention can be applied to these types.

5.7 Type conformance between Ada and C

5.7.1 Basic types

C	Ada Standard[1]	Ada Interface	Ada Interfaces.C
int	Integer	Integer_32	int
long	Long_Integer	Integer_32	long
long long	Long_Long_Integer	Integer_64	
short	Short_Integer	Integer_16	short
unsigned		Unsigned_32	unsigned
unsigned long		Unsigned_32	unsigned_long
unsigned long long		Unsigned_64	
unsigned short		Unsigned_16	unsigned_short
float	Float	IEEE_Float_32[2]	C_float
double	Long_Float	IEEE_Float_64[3]	double
long double	Long_Float	IEEE_Float_64	long_double
char	Character	Integer_8	char
unsigned char		Unsigned_8	unsigned_char

5.7.2 Structs , arrays and pointers

Using integer as example should be same for all types. We assume here that * does not equal array if it does it maps like []. For more information see the appropriate section.

struct	record, with pragma Convention(C_Pass_By_Copy, <type name>);
struct *	access to record
int[]	array or access to array
int[][]	array of Interfaces.C.Pointers.Pointer
int[n]	array (0 .. n - 1)
int[n][m]	array (0 .. n -1, 0 .. m - 1)
int *	access Integer
int **	access Integer_Access[4]
char[]	String, ends with Nul or Interfaces.C.Strings.Chars_Ptr
char[][]	Interfaces.C.Strings.Chars_Ptr_Array
char *	access Character
char **	access Character_Access[5]

5.8 Pragmas, Conventions and Keywords

5.8.1 Convention C_Pass_By_Copy

Used with records so that they can be passed like a C struct when not using them as a pointer.

5.8.2 Convention C

Required on certain types like enumerations and function pointers to make them behave like their C equals. In GNAT most types have this convention by default.

5.8.3 Pragma Unchecked_Union

Used on an Ada variant record to make it compatible into a C union.

5.8.4 Pragma Import

Allows Ada programs to call C functions.

5.8.5 Aliased

Suggests to the compiler that it should give an element or a type its own address. This is required on arrays to make them C compatible.

Chapter 6

Interfacing Ada with C++

When interfacing Ada with C++ the problem is not how to do something compared to interfacing with C, but why it can not be done and what tricks and how to bypass certain features in C++ that allows the code to interface with Ada easier or even at all.

6.1 Importing functions from C++

C uses a standard Application Binary Interface(ABI) for compilers, C++ does not have this and because of that for all functions that are imported the mangled name that is often unique for different compilers is needed. For simple functions this might not be a huge problem but when we have overloaded functions, namespaces, functions inside classes or templates we need to spend time going through a demangled library file to find the names that match the function that is imported.
There are several cases when importing a function from C++ can not be done without changing the C++ code or writing C++ code. During the work on this thesis the following cases have been the most common to cause problems when importing the functions.

- Function is declared inline.
- Function body is in a header file.
- Function is made inline as a compiler optimisation.
- Function is a template or part of a template class.

Each of these four cases causes the same problem, namely that they are not exported to a library file and therefor can not be imported to Ada. For the first three cases solving this can be done in two ways, first is to move code to body files and making sure they are not inline in the third case this might actually be impossible. Two other methods are to create wrapper functions in C or export the functions using the extern "C" command, the result of both of these should be the same namely C++ functions appearing as C functions. The issue is that to use extern "C" no C++ specific constructs are allowed this limits the usage dramatically since this for example includes classes, vectors and strings.

In the fourth case when importing a template function to Ada the issue is that to be able to import a function we need to have an instance created in C++ so that the object file or library file actually have any information about the function, the only working way to import a function is to make an instance and import that instance and loose the template functionality. One way to import a template function to Ada that was tried is to have a wrapper in C but the issue that makes this improbable is that even tough Ada have its own generic functions, it was not possible for C to pass enough information to the C++ to be able to use the templates. Another method that was not fully explored is to use a more advanced but stupid C wrapper that would have the possibility to create only a hard coded set of template versions, however if like in the case of OpenCv when template functions are created to be used with any type and be completely generic this is method is very unrealistic.
Without using any of the methods discussed above a function imported from C++ would look like this in Ada:

```
function WaitKey (Delay_Ms : Integer := 0) return Integer;
pragma Import (CPP, WaitKey, "_ZN2cv7waitKeyEi");
```

But this example is only working on one specific compiler on one platform so for each platform or if the library is recompiled with another compiler we need to find the mangled name again.
If we instead use the method with extern “C” described above we would need to first change the C++ code or create a wrapper like this:

```
extern "C" {
        int WaitKey_wrap(int Delay_Ms);
}
```

Then we can import the function to Ada using the same method used with C functions.

```
function WaitKey (Delay_Ms : Integer := 0) return Integer;
pragma Import (C, WaitKey, "WaitKey_wrap");
```

6.2 Importing Classes from C++

A class as seen from Ada is more or less a record that is passed as a pointer, however to have a working class we need constructors and destructors. It is possible to create wrapper functions in C for constructors and destructors, for simple classes importing them to Ada and allowing Ada to have access to internal variables is not very difficult as long as no C++ specific types or functionality is used. Even the C++ specific things that can be imported to Ada with a bit of tricks with C wrappers can not be used in an Ada record that tries to map the C++ record anyway, so while it is possible to import a function that uses the vector class or string type if they are used as a member of a class then the possible import method is not working anymore since the Ada record will not be equal to the C++ class anymore. For a possible method to map strings and vectors to Ada see Vector and String (section 6.2.1).

6.2.1 Vector and String

Even though the vector class and string type have very little in common the method to map these successfully to Ada is the same. The method involves creating a wrapper function in C that instead of a vector or a string takes a array of the type instead and in that wrapper function the array is converted to a vector and then back again if the function returns the vector as well. The problem with this method is that for every type the vector can have, either a new wrapper function must be created or the wrapper function must be able to chose between the types. If we consider these restrictions then we understand that it is nearly impossible to map a vector used with more then one type, string on the other hand is possible to map using this method.
If we consider the function used to create a window in C++ that has a string as one of the parameters.

```
void namedWindow( const string& winname, int flags ):
```

Then we create a wrapper function in C that takes a char pointer instead and uses the string constructor that creates a string from a char pointer.

```
extern "C"{
        void namedWindow( const char* winname, int flags)
        {
                cv::namedWindow(string(winname), flags);
        }
}
```

Then we can create an Ada function that uses the Chars_Ptr type or the Ada string type or any other method that is explained in Interfacing Ada with C, Strings (section 5.6.3).

```
procedure NamedWindow (Window_Name : Chars_Ptr;
                       Flags : Integer);
pragma Import (C, NamedWindow, "namedWindow");
```

Chapter 7

GIMME Communication Protocol

The GIMME Communication Protocol (GCP) is meant to be used with the GIMME system currently being developed at Mälardalens Högskola. GCP can communicate through raw Ethernet or USB and is primarily designed to transfer raw image data and Harris data but it also supports other custom data types in matrix or array form. GCP is designed as a layered header model, where a frame header is actually multiple headers layered on each other. Fault tolerance is not a priority in GCP because it is intended for use with streaming data, as a result there is no error correction if a frame is corrupt with the exception of certain control frames that require an acknowledgment to be sent as a reply.

7.1 Requirements

The GIMME system supports raw Ethernet and USB communication, GCP should be able to use both medias to transmit data.
The data that is transmitted is real-time streaming data which favors a high throughput over error correction. If some frames are lost or corrupt it is not important to re-transmit them as it will not influence the overall result to a great extent.
Although error correction is less important, the order of the data is important because it is mainly sequential image data that is transferred. If a frame is lost or corrupt it should be ignored and that data region should be skipped to prevent distorting the remainder of the image. For example, consider an image measuring 20x20 pixels and that each Ethernet frame could transfer 10 pixels. If one frame is lost (illustrated by the red rectangle in Figure 7.1) the image would be distorted like the left image in Figure 7.2 if the protocol ignored the lost frame completely. This result is unacceptable and can be solved by bypassing the missing pixels, thus continuing to reconstruct the image from the correct coordinates in the image, as can be seen in the right image in Figure 7.3.

Protocol headers should be expandable and have room for future extensions. The headers follow a layered model, where the headers are put on top of each other. This provides an easy way to extend the protocol with other types of headers in the future and allows for more flexibility when attempting to mini-

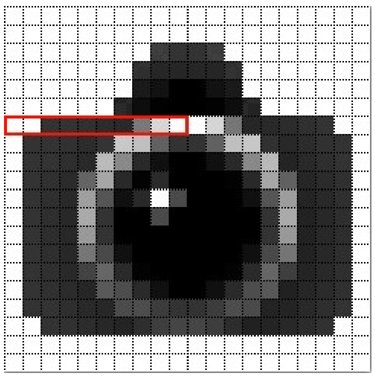


Figure 7.1: Original image to transmit

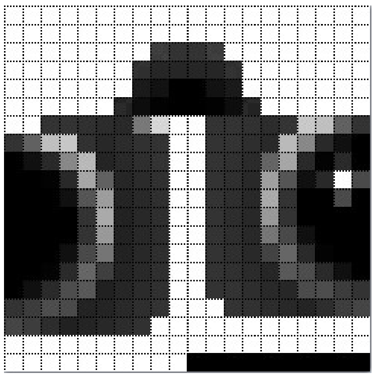


Figure 7.2: If missing data after a lost frame is not skipped the resulting image look distorted

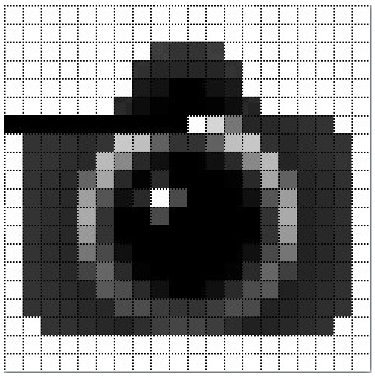


Figure 7.3: Skipping over lost data by filling it with a solid color gives a more correct result

mize the headers to provide further throughput. GCP will use one mandatory header and a series of supplementary headers that describes the actual contents of a transaction. The supplementary headers will only appear in the first frame of a transaction, the subsequent frames will omit the header to further optimize throughput. If the frame containing the supplementary header is lost, the subsequent frames will be discarded since the contents of the transaction is unknown.
Configuring compatible devices as well as being able to read and write the memory of these devices. The configuration options can vary between different devices and therefore needs to be as general as possible.
In addition to simple polling for data there is support for subscription based transactions where it is possible to subscribe to specific types of data from individual devices on the network is.
Compatible devices on the network should be automatically detected and identified through a handshake containing information that describes the features of the device.

7.1.1 GCP Functionality

GCP allows the following functionality based on the requirements,

- Transferring data through
  - Raw Ethernet
  - USB
- Transfer various types of predefined and general data
  - Image data
  - Harris algorithm data
  - General arrays
  - General matrices
- Dynamic headers
  - Layered headers
  - Minimizing use of headers to increase throughput
- Subscription based on
  - Device ID
  - Data type
- Automatic handshaking that
  - Identifies a GCP compatible device
  - Creates a handle to the device
  - Get information about supported data modes and protocol versions
- Ping devices to verify they are still on the network
  - Broadcast

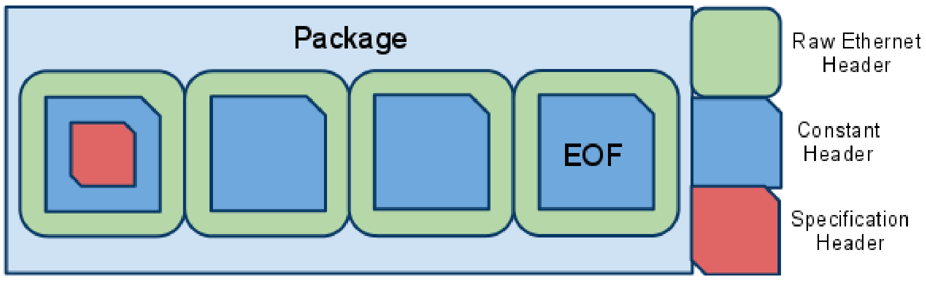


Figure 7.4: Representation of a transaction (series of raw Ethernet frames)

  - Single target
- Device configuration over the network
- Read on-board memory
- Write on-board memory

7.1.2 PC requirements

GCP needs to work on both Windows and Linux since both operating systems are used in conjunction with the GIMME device. This requires a driver that handles the communication layer and a API that enables developers to communicate through GCP. The driver will assemble and disassemble data into frames forming transactions. Transactions that are in the process of being received will be put in a temporary buffer until the transaction is completed, after completion the data will be moved to a buffer, either waiting for the user to fetch it or being dispatched to a callback function.
OpenCVAda will be used to process image and data received from the network so an interface between GCP and OpenCVAda is beneficial to make development easier. This will add support to convert transactions to appropriate OpenCVAda types such as Ipl_Image and Cv_Mat.
Since GCP supports subscription based transactions the driver should provide callback functions to automatically handle incoming subscribed transactions. When a new transaction is received it will be compared against the subscription list, if the device is subscribed and contains the desired data the transaction will be dispatched to a callback associated with the subscription.

7.2 Design

7.2.1 Header structure

Each transaction sent from a camera device is sent as a series of raw Ethernet frames (or equal ones over USB) with a constant header. The first frame in each transaction will have a specification header that details what type of transaction is sent, the specification header is different for all types of transactions. The last frame in a transaction will have an EOF flag set to signal that the transaction is terminated.

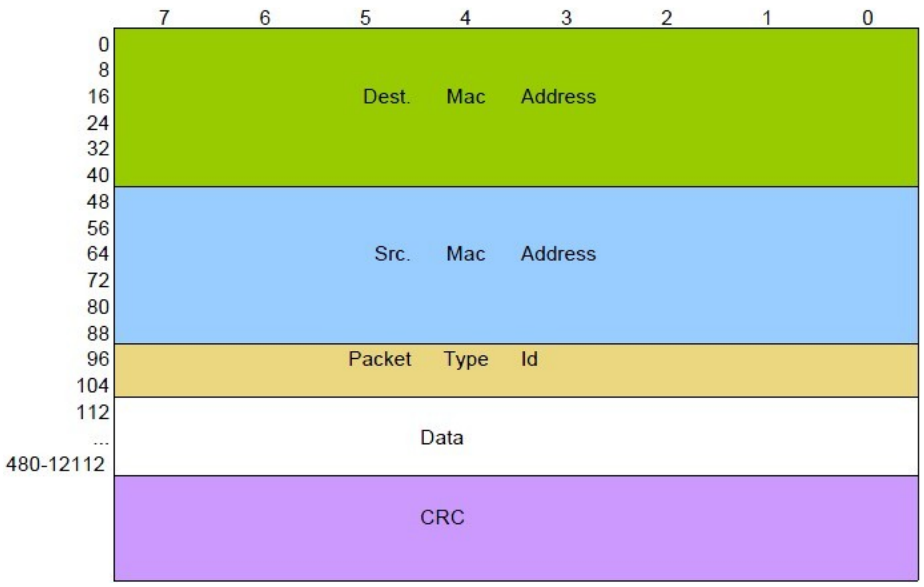


Figure 7.5: Raw Ethernet header

7.2.2 Raw Ethernet header

Bit 0:47 - Dest. Mac Address The address to the intended recipient of the Ethernet frame. The mac address FF-FF-FF-FF-FF-FF is used to broadcast messages to all devices on the network.

Bit 48:95 - Src. Mac Address The senders address.

Bit 96:111 - Packet Type Id Describes which type of Ethernet protocol is used, for this protocol only the IEEE802.3 Length Field is viable which range from 0x0000-0x05DC (0-1500).

Bit 112:479-12111 - Data This field contains the payload of the Ethernet frame, it can range from 48-1500 byte.

Bit 480:501-12112:12143 - CRC The end of the frame is a cyclic redundancy check to verify that the message has not been corrupted.

7.2.3 Constant and specification headers

All Ethernet frames sent will use the constant header in addition to the raw Ethernet header, the first Ethernet frame in each transaction will be a specification header for the different types of transactions.

Constant header

Bit 0:3 - Version Version of the protocol being used.

Bit 4:7 - Header Length If the header contains extra data (i.e. handshake) this field specifies how many extra bytes of data the header contains. See Bit 40:159 - Data for further information.

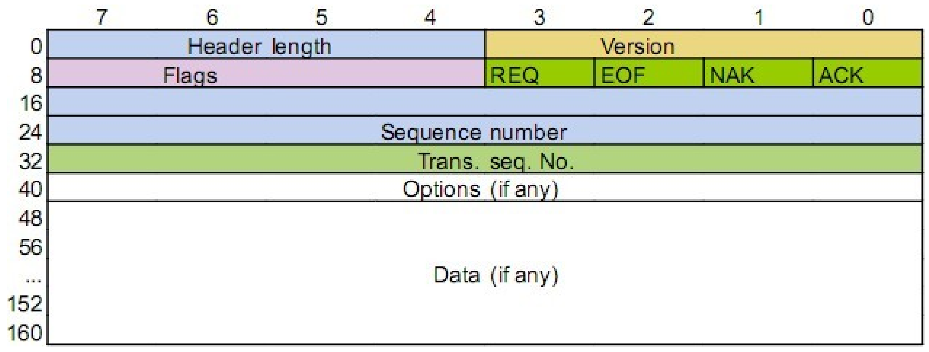


Figure 7.6: Mandatory protocol header

Bit 8 - ACK This bit signifies that the transaction is a response to a request.

Bit 9 - NAK If a corrupt or erroneous frame is received, a mirrored frame with this bit set will be transmitted back to the sender.

Bit 10 - EOF The last frame of each transaction will have this bit set to signify that the end of the transaction is reached.

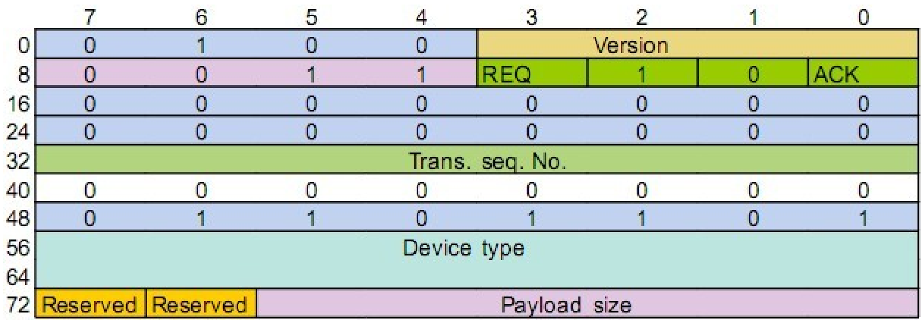
Bit 11 - REQ This bit is used to make a request of some description (i.e. requesting image data).

Bit 12:15 - Flags The flags field is used for the remaining flags that are mutually exclusive.

- 0000: No flags
- 0001: Ping
- 0010: Configure
- 0011: Handshake
- 0100: Subscribe
- 0101: Data
- 0110: Memory
- 1111: Extended flags

Bit 16:31 - Seq. No. A sequence number that describes which frame within a transaction this is. The sequence number is reset for each transaction of frames.

Bit 32:39 - Trans. seq. No. If a device is transmitting multiple transactions with separate data of the same type in parallel it is necessary to be able to differentiate between them. The transaction sequence number is used to separate transactions from each other. When a transaction is transmitted it will keep the same transaction sequence number throughout the transmission of the transaction, the sequence number will however be incremented for each frame within the transaction.



	7	6	5	4	3	2	1	0
0	0	1	0	0	Version			
8	0	0	1	1	REQ	1	0	ACK
16	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0
32	Trans. seq. No.							
40	0	0	0	0	0	0	0	0
48	0	1	1	0	1	1	0	1
56	Device type							
64								
72	Reserved	Reserved	Payload size					

Figure 7.7: Handshake header

Bit 40:47 - Options This field is not part of the standard header. If Extended flags is set this field will give access to one additional byte of customizable flags that can be application specific.

Bit 48:167 - Data This field is not part of the standard header. Some header flags require additional data (i.e. handshake), this field is used in those cases to store the extra data associated with such flags. The Header Length field specifies the size of this field in whole bytes, if a whole byte is not used the remaining bits shall be padded with zeros'. The bit offset for this can be either 32 or 40 if Extended flags is false or true respectively.

Handshake

When a device becomes available on the network it is necessary to determine if it is a compatible device and also what features are available on the device. The handshake is performed with a 1 byte mask followed by information about the device. The mask is used as an identifier which together with the rest of the header make up a unique message to identify compliant devices. The reply from any compatible device will be mirrored from the handshake request up to the 32nd bit with the exception of having the ACK flag set in the reply. After the 32nd bit follows the information about the device such as device type and available data structures that can be used when communicating with it.

Bit 0:3 - Version Version of the protocol being used.

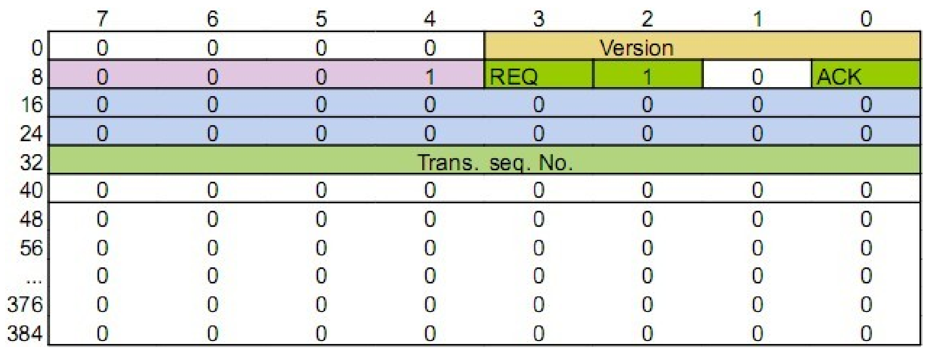
Bit 4:7 - Header Size Always 0100 since there is 4 extra bytes in the handshake header.

Bit 16:31 - Seq. No. Always 0 not used.

Bit 32:39 - Hndshake mask This field contains a mask that is used to confirm that the handshaking device is compatible. The fields value is constant and should be set to 0x6D.

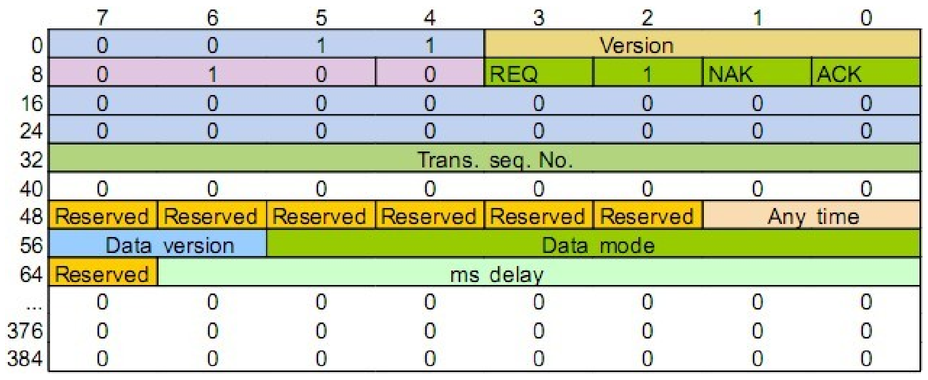
Bit 40:55 - Device type This field describes what type of device the sender is.

Bit 56:61 - Payload size Amount of bytes added as payload, used to describe the functionalities available.



	7	6	5	4	3	2	1	0
0	0	0	0	0	Version			
8	0	0	0	1	REQ	1	0	ACK
16	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0
32	Trans. seq. No.							
40	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0
56	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0
376	0	0	0	0	0	0	0	0
384	0	0	0	0	0	0	0	0

Figure 7.8: Ping header



	7	6	5	4	3	2	1	0
0	0	0	1	1	Version			
8	0	1	0	0	REQ	1	NAK	ACK
16	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0
32	Trans. seq. No.							
40	0	0	0	0	0	0	0	0
48	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Any time	
56	Data version		Data mode					
64	Reserved	ms delay						
...	0	0	0	0	0	0	0	0
376	0	0	0	0	0	0	0	0
384	0	0	0	0	0	0	0	0

Figure 7.9: Subscription header

Bit 62:63 - Reserved Reserved for future use.

Ping

Message used to check for devices currently on the network. This can be used with a broadcast address to check all devices on the network at once.

Bit 0:3 - Version Version of the protocol being used.

Bit 8 - ACK Acknowledges a request.

Bit 10 - EOF Always 1 since a ping can only be one frame long.

Bit 11 - REQ Requests a response from the destination.

Bit 12:15 - Flags Always 0001 which is the flag for ping.

Subscription

Bit 0:3 - Version Version of the protocol being used.

Bit 4:7 - Header length Set to 0011 to denote that the frame header is two bytes longer then the normal constant header.

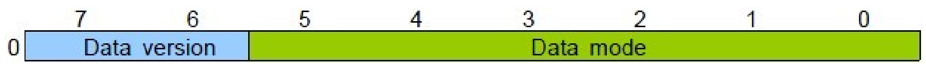


Figure 7.10: Data type header

Bit 8 - ACK Set when acknowledging a new subscription.

Bit 9 - NAK Set when not acknowledging a new subscription.

Bit 10 - EOF Always set to 1 since a subscription transaction is only one frame.

Bit 11 - REQ Set when requesting that a new subscription should be started.

Bit 12:15 - Flags Always 0100 which is the flag for subscription.

Bit 16:31 - Seq number Always 0.

Bit 32 - Any Subscribe to any data sent from the device. (Byte 6 and 7 is ignored.)

Bit 33 - Time If set to 1 requested data will be sent at the chosen interval. (Byte 7)

Bit 34:39 - Reserved Reserved bits for future use.

Bit 40:45 - Data mode See Image and data headers for the possible data modes that can be subscribed to.

Bit 46:47 - Data version See above.

Bit 48:55 - Ms Delay The delay between the transaction sent, in milliseconds. (I.E. how many “image frames” per second that is sent.)

Bit 56 - Reserved Reserved bit for future use.

Specification headers

The specification headers are used at the start of each transaction to specify the type of transaction and what it contains due to the dynamic headers requirement, this header will only be used on the necessary frames and the constant and Ethernet headers will be used on every Ethernet frame including the one with the specification header.

Image and data headers

Except for an image the data headers can reference with two types of data templates array and matrix. There is also room for around 60 specific data types like Harris data. The 2 bit version flag is for when the device supports more then one version of a single header or to indicate a rewrite between versions of the device.

Bit 0:5 - Data Mode Data modes available:

- 000000 - Image

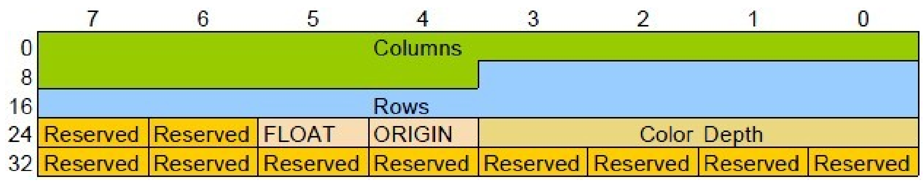


Figure 7.11: Image data header

- 000001:011111 - Matrices
- 100000 - Harris
- 100001:111111 - Arrays

Bit 6:7 - Data Version Version explanation:

- 00 - Current version
- 01 - Extra version 1
- 10 - Extra version 2
- 11 - Deprecated version

While the matrix and array headers might not be able to do a full specification that can be used to create a type without more knowledge on the PC side, the headers should be able to represent any realistic type needed. Using the array or the matrix header without creating a new definition on both sides should not be done, since it will not be possible to distinguish between different implementations of the templates.

Image header

Bit 0:11 - Columns Columns in the image, between 1 and 4096.

Bit 12:23 - Rows Rows in the image, between 1 and 4096.

Bit 24:27 - Color Depth Color depths available:

- 0000 - 1 bit
- 0001 - 8 bit gray (8)
- 0010 - 8 bit color(3,3,2)
- 0011 - 15 bit (5,5,5)
- 0100 - 24 bit (8,8,8)
- 0101 - 30 bit (10,10,10)
- 0110 - 36 bit (12,12,12) (GIMME camera standard)
- 0111 - 48 bit (16,16,16)
- 1000:1111 - Reserved for future colors

Bit 28 - ORIGIN Top left origin (1) or Bottom left origin (0).

Bit 29 - FLOAT Color is represented as 0.0 to 1.0 instead of an integer value.

Bit 30:39 - Reserved Reserved bits for future use.

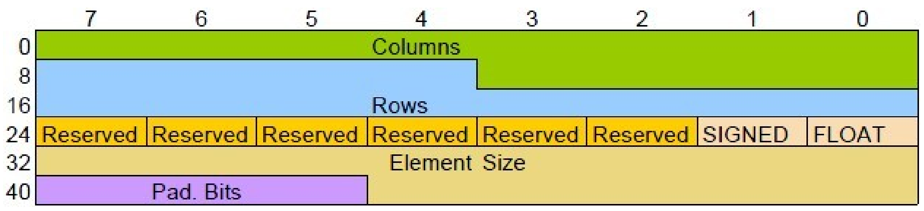


Figure 7.12: Matrix data header

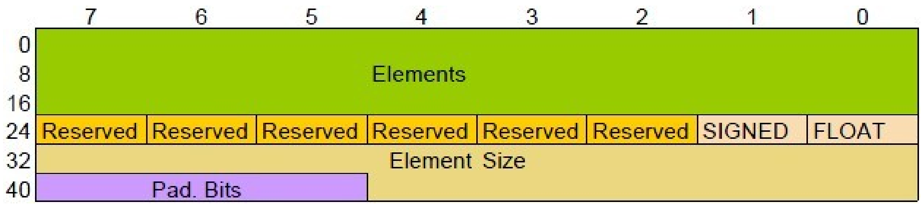


Figure 7.13: Array data header

Matrix header

A matrix behaves like a image except that it allows for anonymous types as elements instead of a color. Each element have a maximum size of 8191 bytes with a possible padding of 7 bits.

Bit 0:11 - Columns Columns in the matrix, between 1 and 4096.

Bit 12:23 - Rows Rows in the matrix, between 1 and 4096.

Bit 24 - FLOAT Elements are float values instead of integer values.

Bit 25 - SIGNED Elements are signed or unsigned.

Bit 26:31 - Reserved Reserved bits for future use.

Bit 32:44 - Element Size Total size of the element in the matrix.

Bit 45:47 - Padding Bits Used if element size is not a even byte.

Array header

The array type has a maximum length of roughly 16.8 million elements where each element has a combined maximum size of 8191 bytes with a padding of 7 bit possible.

Bit 0:23 - Elements Number of elements in the array.

Bit 24 - FLOAT Elements are float values instead of integer values.

Bit 25 - SIGNED Elements are signed or unsigned.

Bit 26:31 - Reserved Reserved bits for future use.

Bit 32:44 - Element Size Total size of the element in the array.

Bit 45:47 - Padding Bits Used if element size is not a even byte.

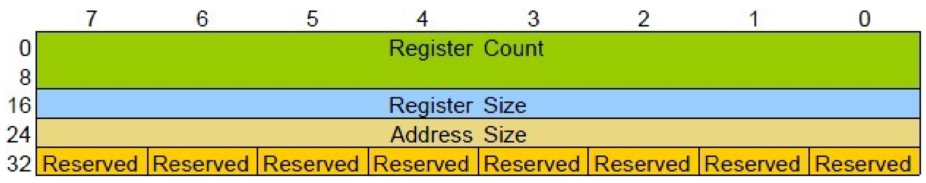


Figure 7.14: Configuration header

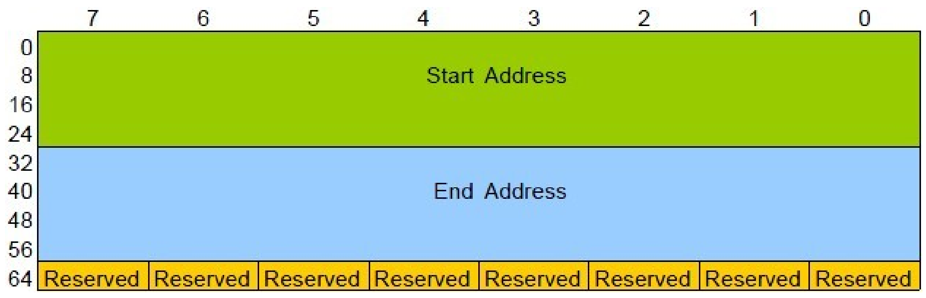


Figure 7.15: Memory header

Harris data header

Will be a array header with only elements as the parameter allowed to be changed.

Configuration header

Configuration of a device on the network can be done with the configuration header. The header contains the number of registers to configure, the size of the registers and the size of the registers addresses.
The header is followed by a payload consisting of pairs of addresses and values. The EOF flag will be set in the last frame denoting the end of the configuration transaction.
Each frame will be acknowledged by the receiver with a ACK-frame, a corrupt message will trigger a NAK-frame to be sent. If a frame is lost or corrupt the transmitter will re-transmit the frame upon receiving the NAK-frame or after failing to receive a response within a certain delay.

Bit 0:15 - Register Count Number of registers that will be configured.

Bit 16:23 - Register Size The size of the registers in bytes.

Bit 24:31 - Address Size The address size in bytes.

Bit 32:39 - Reserved Reserved bits for future use.

Memory header

Send or receive memory from start address to end address.

Bit 0:31 - Start Address The beginning of the memory space where data will be read/written.

Bit 32:63 - End Address The end of the memory space where data will be read/written.

Bit 64:71 - Reserved Reserved bits for future use.

7.2.4 PC GCP driver design

System design

The driver on the PC side will be driven by different threads that takes care of the communication between a user API and the network. Data will be received according to these steps:

1. Frames from the network are sorted into their appropriate frame buffer.
2. A thread responsible for the frame buffer will start assembling the frames into a complete transaction.
3. When the transaction is finished, the transaction is put into a transaction buffer.
4. The API fetches the transaction on a users request, or dispatches it to an appropriate callback.

When transmitting data the steps are done in reverse

1. The API creates a transaction from data provided by the user.
2. The transaction is put into an outgoing transaction buffer.
3. A thread managing the outgoing communication disassembles a transaction into frames.
4. The frames are put into a frame buffer waiting to be sent out on the network.

A simplified view of the communication interface (see figure 7.16) could be described as different layers. There is a network layer which is the physical medium actually transporting the data between endpoints. A frame buffer layer hold incoming and outgoing frames waiting to be transmitted or processed. After the frame buffer layer is a layer of threads that assemble and disassemble data from the buffer layers. The threads are followed by layer of transaction buffers that contain complete data either waiting to be fetched by the API or to be disassembled and transmitted out on the network. The last layer is the API layer which is used by developers to send and receive data from the network.

Network layer

Handles sending and receiving of raw Ethernet frames. Frames received will be sent to the appropriate frame buffer, while frames that are sent is taken from the outgoing frame buffer.

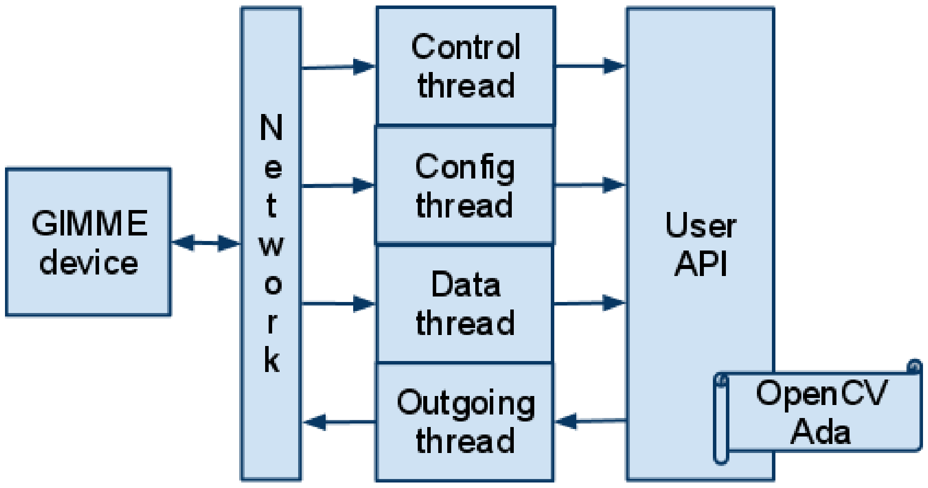


Figure 7.16: Communication protocol design

Frame buffer layer

The buffers in this layer contain frames received from the network or waiting to be transmitted out on the network. Each connected device has its own set of buffers that are associated with it. In the case of a buffer overflow the oldest frames will be dropped since new data is preferred over old data with the exception that a transactions first frame will always have a higher priority since it contains vital information about the transaction. If that frame is dropped the remaining frames of that transaction are useless.

Thread layer

This layer consists of a number of threads, each responsible for a frame buffer and a transaction buffer (see figure 7.17). The threads handling incoming frames read a frame from the frame buffer and extracts the data into the appropriate place in a transaction buffer. There is one thread that handles outgoing data, the thread works the opposite way from the incoming threads. It reads data from the transaction buffer and disassembles it into frame that are put into the frame buffer waiting to be transmitted.

Transaction buffer layer

This layer contains buffers where complete transactions are stored. Transactions that have been received from the network are assembled and put in the appropriate buffer ready to be fetched by the user. Each connected device has its own set of buffers that are associated with it. These buffers can only contain one transaction since it is only the latest data that is relevant, so if a new transaction is completed before the current one has been fetched it will be overwritten by the new transaction.

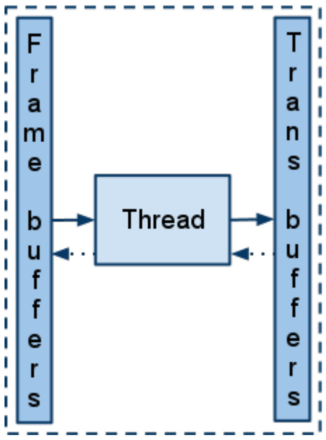


Figure 7.17: Thread layer layout

API layer

The API contains functions to enable developers to interact with the network. It stores information about available devices and maintains handles to interact with those devices. The API provides functions to create GCP headers and transactions as well as functions that transmit and receive data from the network. Some of the native OpenCVAda types are supported directly in the API to make the transition from a GIMME device to OpenCVAda easier.

Chapter 8

Summary and Conclusions

"If you wish to make an apple pie from scratch, you must first invent the universe."
Carl Sagan

Our thesis was done in three stages first was to investigate how to interface C or C++ with Ada, then to create an Ada interface for OpenCv called OpenCvAda the last stage was a communication protocol for the GIMME stereo camera device.
Types in Ada and C are in almost all of the cases equal and when mapping back or forward nothing except the standard types needs to be used, the only case that causes problems is arrays. But the solutions for C arrays in Ada are straightforward and does not move away from standard Ada coding styles, as long as unbounded arrays in Ada are not used. C pointers in Ada might seem tricky but if truth be told, the only case when Ada access and C style pointers do not reference the same things are when used with arrays. One thing that can not be used in Ada from C are macros which needs to be moved to either a C or Ada function instead. Using the C to Ada guide in this report should allow Ada to interface with any C library or object code with access to headers.
Except for C++ code that only uses C types in the parts that is visible to Ada, this would allow only basic types, pointers and references to be used, the first noticeable problem with C++ the name mangling will still not be solved and requires either the C wrapper or extern "C" solutions. The main problem with using C++ code is always in one way or another related to a template class or template function and that they are not added as part of the library or object file until an actual instance is created, this was not solvable in the scoop of this report and possible solutions to investigate is discussed in the future work section.
When designing and implementing OpenCvAda the goal was to make it behave like an Ada library while keeping it as familiar as possible to users of OpenCv. This inadvertently created some weird design choices where access types are used a bit to much and should have been hidden away from the user to make OpenCvAda more intuitive to work with. Another problem is regarding the Cv_Seq and Cv_Void types and the design that they should behave like they do in C instead of making any functions using these generic, the reason for them not be generic was the assessment that it would require the user to create several

instances of the same package and that it would break dependencies between packages, this should have been looked at more to make OpenCvAda behave more like an Ada library instead of mimicking a C library. Other problems arise due to OpenCv not working properly including broken functionality (camera and video playback to name a few) or samples not tested on current versions making it impossible to even run them in some cases, this is most likely due to OpenCv trying to keep three language versions running at the same time and release testing that seemed to be skipped or just sloppy. If we disregard the changes that should be looked at that we discussed above OpenCvAda is a working interface to OpenCv in C but lacking support for the new functionality only available in the C++ version.
Benchmarks was created and executed to test the performance of OpenCvAda compared to OpenCV in C and Python in six different categories: Highgui window creation, large image I/O, algorithms, memory usage, OS comparison and correctness. Also a test was done as a reference where an empty application was executed to make sure no unknown bottlenecks existed in the tested platforms. The results are overall positive in 4-2 advantage in observed performance in favor of OpenCvAda, and according to our data from a performance standpoint Windows 7 is the favored platform, but for a better and more detailed picture more tests needs to be run since it makes very little sense for OpenCvAda to be faster then OpenCv in C. The expected results of the OpenCvAda benchmarks is close to the results measured but more like memory and algorithm test where C edges out Ada with small margin and Python being a clear bit away from the other two in performance, without more detailed, different and more executions in the tests it is not possible to find a reason for the actual results where Ada has a lower average then C.
The GIMME communication protocol was designed to be expandable and favouring high throughput over error handling and correctness. Some error handling still exists and are mostly designed to allow for simultaneous receiving and sending of several transactions at the same time and to be able to replace missing Ethernet frames on the receiving end with dummy data.
The perceived result is that OpenCvAda is not invalidated in either performance or correctness compared to C and Python, so this should be considered a huge success, but changes could be made to make OpenCvAda more user friendly and to change the package layout to not just match the C/C++ version but to make it more clear where to find the different functions and types and easier to find functions and types.

Chapter 9

Future Work

For future work we have discussed and debated several ways of extending and improving OpenCvAda. We believe that OpenCvAda would benefit greatly from a new system to simplify the mapping of pointers to OpenCv. One possible solution becomes available when the new Ada 2012 ISO standard is released. In Ada 2012 *in out* parameters will be usable in both procedures and functions [2], it will then be possible to use these parameter types as a homogeneous pointer type instead of the access types currently being used.
Another change would be look into the increased usage of generics on the Ada side or a change in the package layout. The C interface changes are not big and maybe not very interesting compared to what can be done with an Ada interface to the C++ version of OpenCv instead.
The first steps would be to investigate, design and implement ways of mapping the two biggest culprits from C++ to Ada namely vectors and templates. For vectors the idea would be an Ada type that would make it possible to have a direct mapping to a C++ vector. While for templates what would be needed is not as clear and a lot more information is needed, for example it could involve some way of creating instance of C++ templates from Ada without being forced to hard code the allowed types.
For the tentatively named OpenCvAda++, first a handmade interface could be made to map with the C++ version of OpenCv but the long term goal would be to have an automated way of creating the interface between OpenCvAda++ and the C++ version OpenCv similar to the Python version of OpenCv.
Another interesting area would be the performance of OpenCv and OpenCvAda and creating a good way of testing releases of all versions to avoid the problems we have seen with OpenCv.

Bibliography

[1] Ada 95 quality and style guide.

[2] Ada comparison chart.

[3] Nasa featured images and galleries.

[4] Opencv wiki.

[5] Qt - cross-platform application and ui framework.

Appendix A

Hough_Lines.adb

```
with Highgui;              use Highgui;
with Core;                 use Core;
with Core.Operations;      use Core.Operations;
with Imgproc;              use Imgproc;
with Imgproc.Operations;   use Imgproc.Operations;
with Ada.Unchecked_Conversion;


procedure Hough_Lines is
        subtype Line_Array is Cv_Point_Array (0 .. 1);
        type Line_Array_Ptr is access Line_Array;

        function To_Void_Ptr is
                new Ada.Unchecked_Conversion
                        (Target => Cv_Void_Ptr,
                        Source => Cv_Mem_Storage_Ptr);
        function To_Line_Array_Ptr is
                 new Ada.Unchecked_Conversion
                        (Target => Line_Array_Ptr,
                        Source => Cv_Void_Ptr);

        Capture            : aliased Cv_Capture_Ptr;
        Image              : Ipl_Image_Ptr;
        Dst, Color_Dst, Gray : aliased Ipl_Image_Ptr;
        Storage            : aliased Cv_Mem_Storage_Ptr;
        Lines              : Cv_Seq_Ptr;
        Line               : Line_Array;

begin
        Capture := Cv_Create_Camera_Capture (0);
        Cv_Named_Window ("Hough");
        loop
                Image := Cv_Query_Frame (Capture);
                Dst := Cv_Create_Image
```

```
                    (Cv_Get_Size (Image),
                    8, 1);
          Gray := Cv_Create_Image
                    (Cv_Get_Size (Image),
                    8, 1);
          Color_Dst := Cv_Create_Image
                    (Cv_Get_Size (Image),
                    8, 3);

          Storage := Cv_Create_Mem_Storage (0);
          Cv_Cvt_Color (Image, Gray, Cv_Bgr2gray);

          Cv_Canny (Gray, Dst, 50.0, 200.0, 3);
          Cv_Cvt_Color (Dst,
                        Color_Dst, Cv_Gray2bgr);

          Lines := Cv_Hough_Lines2
                     (Dst, To_Void_Ptr (Storage),
                     Cv_Hough_Probabilistic ,
                     1.0,
                     Cv_Pi / 180.0, 100, 5.0, 10.0);

          for I in Integer
                    range 0 .. Lines.all.Total - 1
          loop
                    Line := To_Line_Array_Ptr
                              (Cv_Get_Seq_Elem
                                        (Lines, I)).all;
                    Cv_Line (Color_Dst,
                               Line (0),
                               Line (1),
                               Cv_Rgb (255, 0, 0),
                               3, 8);
          end loop;

          Cv_Show_Image ("Hough", Color_Dst);

          Cv_Release_Mem_Storage (Storage'Access);
          Cv_Release_Image (Dst'Access);
          Cv_Release_Image (Gray'Access);
          Cv_Release_Image (Color_Dst'Access);

          exit when Cv_Wait_Key (30) = Ascii.Esc;

     end loop;
     Cv_Destroy_Window ("Hough");
     Cv_Release_Capture (Capture'Access);

end Hough_Lines;
```

Appendix B

Demo Application

B.1 Demo.gpr

```
with "OpenCvAda";
        -- A demo application to show how to use OpenCvAda
project Demo is
        for Source_Dirs use ("Source\Demo");
        for Object_Dir use "Build";
        for Exec_Dir use "Exec";
        for Languages use ("Ada");

        package Builder is
                for Default_Switches
                        ("ada") use ("-s", "-m");
        end Builder;

        package Compiler is
                for Default_Switches ("ada")
                        use ("-gnatw.hkd", "-gnat05");
        end Compiler;

        for Main use ("demo.adb");
end Demo;
```

B.2 Demo.adb

```
with Highgui;                         use Highgui;
with Core;                            use Core;
with Core.Operations;                 use Core.Operations;
with Imgproc;                         use Imgproc;
with Imgproc.Operations;              use Imgproc.Operations;

procedure Demo is
```

```
        -- Parsed variables created from the trackbars
        Aperture_Size           : Integer := 1;
        Block_Size              : Integer := 1;

        -- Unparsed variables used with the trackbars
        Slider_A_Value          : aliased Integer := 1;
        Slider_B_Value          : aliased Integer := 1;

        -- Variables used with the camera.
        Capture                 : aliased Cv_Capture_Ptr;
        Image                   : Ipl_Image_Ptr;

        -- Temporary images
        Corners, Image_Gray     : aliased Ipl_Image_Ptr;

        -- Used to convert a 32F image to a 8U image.
        Scale, Shift            : Long_Float;
        Min, Max                : aliased Long_Float;

        -- Change the convention to match a C function.
        procedure Slider_A (Position : Integer);
        pragma Convention (C, Slider_A);

        procedure Slider_A (Position : Integer) is
        begin
                -- Value is 0 or even.
                if Position = 0 or Position mod 2 = 0 then
                        Aperture_Size := Position + 1;
                else
                        Aperture_Size := Position;
                end if;
        end Slider_A;

        -- Change the convention to match a C function.
        procedure Slider_B (Position : Integer);
        pragma Convention (C, Slider_B);

        procedure Slider_B (Position : Integer) is
        begin
                if Position > 0 then
                        Block_Size := Position;
                else
                        Block_Size := 1;
                end if;
        end Slider_B;

begin
        -- Create Windows,
        -- Corners window will be resizeable.
        Cv_Named_Window ("Original",
                     Highgui.Cv_Window_Autosize);
   Cv_Named_Window ("Corners",
                     Highgui.Cv_Window_Normal);

   -- Move the windows,
   -- so they are not on top of eachother.
   Cv_Move_Window ("Corners", 700, 50);
   Cv_Move_Window ("Original", 0, 50);

   Cv_Create_Trackbar("Aperture",
                      "Corners",
                      Slider_A_Value'Access,
                      31, -- Max value.
                      Slider_A'Unrestricted_Access);
   Cv_Create_Trackbar("Block",
                      "Corners",
                      Slider_B_Value'Access,
                      11, -- Max value.
                      Slider_B'Unrestricted_Access);
   Capture := Cv_Create_Camera_Capture (0);

   loop
      exit when Cv_Wait_Key (100) = Ascii.Esc;
      Image := Cv_Query_Frame (Capture);
      Cv_Show_Image ("Original", Image);
      Corners := Cv_Create_Image
         (Cv_Get_Size(Image),
         Ipl_Depth_32f,
         1);
      Image_Gray := Cv_Create_Image
         (Cv_Get_Size (Image),
         Ipl_Depth_8u,1);
      Cv_Cvt_Color
         (Image, Image_Gray, Cv_Rgb2gray);
      Cv_Corner_Harris(Image_Gray,
                       Corners,
                       Block_Size,
                       Aperture_Size);
      Cv_Min_Max_Loc (Corners,
                      Min'Access,
                      Max'Access,
                      null, null, null);

      Scale := 255.0 / (Max - Min);
      Shift := -Min * Scale;

      Cv_Convert_Scale
         (Corners,
         Image_Gray, Scale, Shift);
      Cv_Show_Image ("Corners", Image_Gray);
```

```
                Cv_Release_Image (Image_Gray'Access);
                Cv_Release_Image (Corners'Access);
        end loop;
        Cv_Destroy_Window ("Original");
        Cv_Destroy_Window ("Corners");
        Cv_Release_Capture (Capture'Access);
end Demo;
```