© 2018 IJSRST | Volume 4 | Issue6 |Print ISSN: 2395-6011 | Online ISSN: 2395-602X

National Conference on Smart Computation and Technology in Conjunction with The Smart City Convergence 2018

Machine Vision Algorithms implementation in Ruby

Bhaskar Sharma*, Ankit Sarswa, Gopesh Kumar Sharma

Computer Engineering Department, Institute Of Engineering and Technology , Jaipur, Rajasthan, India

ABSTRACT

In current scenario, machine vision systems (at least machine vision algorithms(MVA)) are preponderantly implemented using statically typed programming languages such as C, C++, or Java. However, statically typed languages are not suitable for development and maintenance of large scale systems. Dynamically typed languages are generally not considered while choosing a programming language due to their lack of support for high-performance array operations. This review paper presents efficient implementations of MVA with the dynamically typed programming language Ruby. The Ruby programming language is used in this paper review because it offers the best support to meta-programming from other conventional programming languages. A Ruby library-Hornetseye is reviewed for performing array operations as portion of this paper. It is shown that the library eases brief implementations of MVA that are commonly used in industrial automation. That is, this paper is about implementing machine vision systems in different way.The performance of general operations in ruby is compared with the performance of equivalent C/C++ programs to validate the approach.

Keywords: Ruby, Machine Vision Algorithms, Dynamic Programming Language

I. INTRODUCTION

Machine vision is a vast field and in many cases there are various independent approaches solving a specific problem. Also, it is oftentimes difficult to preconceive which particular approach will give the best results. Therefore it is significant to maintain the agility of the system to be able to implement necessary changes in the concluding stages of a project.

A traditional application of computer vision is industrial automation. That is, the cost of implementing a machine vision system eventually needs to be recovered by savings in labour cost, increased productivity, and/or better quality in manufacturing. However, statically typed programming language such as C, C++, or Java are still mostly used to implement machine vision systems. Development and maintenance of large scale systems using a statically typed language is much more expensive compared to when using a dynamically typed languages.

This paper shows how the dynamically typed programming language Ruby can be used to reduce the cost of implementing machine vision algorithms. A Ruby library is reviewed which facilitates rapid prototyping and development of machine vision systems.

The downside of using a compiled language is that a developer is required to make changes to the source code, save them in a file, compile that file to create a binary file, and then re-run that binary file. In contrast, interpreted languages o er considerable savings in development time. In an interpreted language the developer can enter code and have it run straight away.

II. RUBY PROGRAMMING LANGUAGE

The Ruby programming language is an interpreted, pure object-oriented, and dynamically typed general purpose programming language. Furthermore Ruby sup-ports closures and meta-programming. Also Ruby has a straightforward API for writing extensions. Finally Ruby currently is on place 11 of the Tiobe Programming Community

Index.. Ruby is a multi-paradigm language and it is inspired by Perl, Python, Smalltalk, Eiffel, Ada, and Lisp. Ruby supports the following language features:

- Object-oriented
- Single-dispatch dynamic typing
- Exception handling
- Garbage collection (i.e. managed environment)
- Mixins
- Closures
- Continuations introspection
- Meta programming
- Reification.

Software integration in Ruby is easy because:

- ✓ interfacing with native code for writing extensions is simple.
- ✓ classes can still be modified after declaration.
- ✓ Ruby uses duck-typing, i.e. two objects are compatible if they support the same methods and properties.

The design philosophy of the Ruby programming language follows the following principles:

- Brevity: The language is expressive so that programs written in that language are succinct.
- Conservatism: Ruby sticks to traditional control structures to reduce the cost of adoption.
- Simplicity: The Ruby programming language supports simple solutions.
- Flexibility: Ruby should adapt to the user instead of the user adapting to Ruby.
- Balance: The Ruby programming language tries to achieve a balance between all these concepts.
  - Energy/cost consumed,
  - Time/cost to network partition,
  - Variation in node power levels,
  - Cost/packet ,and
  - Maximum node cost

III. STATICALLY TYPED LIBRARIES

Most computer vision libraries are implemented in the statically typed C/C++ language. However C++ has a split type system. There are primitive types which directly correspond to registers of the hardware and there are class types which support inheritance and dynamic dispatch. In C++ not only integers and floating point numbers but also arrays are primitive types. However these are the most relevant data types for image processing. To implement a basic operation such as adding two values so that it will work on different types, one needs to make extensive use of template meta-programming. That is, all combinations of operations, element-type(s), and number of dimensions have to be instantiated separately. For example the FrameWave1 C-library has 42 explicitly instantiated different methods for multiplying arrays. For this reason most libraries do not support all possible combinations of element-types and operations.

Static typing not only leads to an explosion of methods to instantiate. A related problem caused by static typing is that when a developer wants to modify one aspect of the system, the static typing can force numerous rewrites in unrelated parts of the source code (Tratt and Wuyts, 2007). Static typing enforces unnecessary “connascence” (a tech-nical term introduced by Weirich (2009)) which interferes with the modularity of the software. In practise this causes problems when implementing operations involving scalars, complex numbers, and RGB-triplets.

OpenCV: It (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. The library consist of more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

IV. STATICALLY TYPED EXTENSIONS

Some computer vision libraries come with bindings in order to use them as an exten-sion to a dynamically typed language. For example for the OpenCV2 library there are Python bindings (PyCV3) as well as Ruby bindings (opencv.gem4).This allows one to use a statically typed extension in an interpreted language and it becomes possible to develop machine vision

software interactively without sacrificing performance.Open classes and dynamic typing make it possible to seamlessly integrate the functionality of one library into the application programming interface (API) of another. However supporting all possible combinations of types and operations with a statically typed library is hard (see Section 3). In practise most computer vision extensions only provide a subset of all combinations. n general it is not possible to instantiate e cient implementations of all the possible combinations of operations and compile them ahead-of-time. dynamic typing facilitates much more concise code than static typing.

V. DYNAMICALLY TYPED LIBRARIES

There are a number of active free and open source software projects in the area of machine vision. These contain ITK, NASA Vision Workbench, OpenCV, OpenVidia, Camellia, PyGPU, and Gamera to name only a few. Machine vision systems require software for handling video and image files, accessing cameras, and visualizing results. To keep the size of the project manageable it is mandatory to make use of existing software projects. Although open source packages and libraries are available for free, integrating it requires significant time and effort.

To port all required software to Ruby is desirable so as to take full advantage of the language properties. However for input and output (e.g. capturing camera images and displaying videos) it is necessary to interface with native code. In addition it is necessary to implement computationally expensive parts of the code in C/C++ as long as there is no sufficiently strong run-time optimizer for Ruby.
The quickest way to integrate an existing C/C++ library into Ruby is to use the bindings-generators. However simply making the static data types of a C/C++ library visible in Ruby is insufficient for fully exploiting the features of Ruby. An array data type to handle multi-dimensional arrays with elements of a single type was implemented in Ruby library HornetsEye, inspired by NArray.

NArray: NArray is an Numerical N-dimensional Array class. Element types supported are 1/2/4-byte Integer, single/double-precision Real/Complex, and Ruby Object. This extension library incorporates fast calculation and easy manipulation of large numerical arrays into the Ruby language. It has features similar to NumPy, but NArray has vector and matrix subclasses. NArray provides fast element-wise operations combined with methods to manipulate single elements or subarrays. However in contrast to NArray our data type is largely implemented in Ruby and thus allows definition of custom element-types.

HornetsEye: HornetsEye is a real-time computer vision library for the Ruby programming language. HornetsEye is maybe the first free software project providing a solid platform for implementing real-time computer vision software in a scripting language. The library could effectively be used in industrial automation, robotic applications, and human computer interfaces. It is an extension for Ruby which facilitates rapid development of machine vision software and and provide a high amount of flexibility without sacrificing real-time capabilities.

VI. PERFORMANCE

Comparison of HornetsEye with NArray and C++:
Figure 6.1 shows the time required for running the operation “m + 1” for arrays of different size. The array “m” is single-precision floating point array with 500 500 elements.

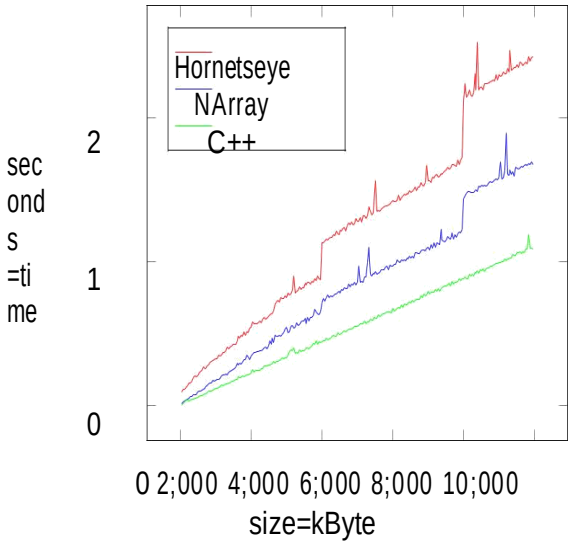

Figure 6.1. Processing time of running "m + 1" one-hundred times for different array sizes

Code Size of Programs: Ruby implementation is significantly shorter and the semantics of Ruby is simpler. Ruby+Hornetseye requires half as many lines of code as the Python+OpenCV implementation. Also the semantics of the Ruby implementation is much more concise.

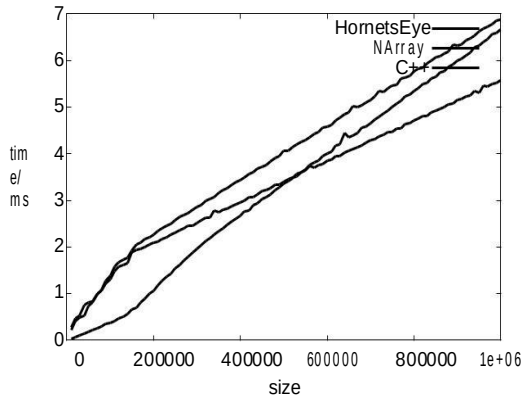

Figure 6.2. Speed comparison for array-scalar multiplication

Contrary to common belief, an interpreted language can be faster than a static implementation. Table Ishows that the garbage collector of Ruby can be faster than the static memory management of a naive C++ implementation.

Table 1

	Mimas/Boost	NArray	HornetsEye
constructor	2.7 ms	8.4 ms	7.8 ms
m.fill(1)	2.7 ms	2.7 ms	2.8 ms
m*m	6.8 ms	10.0 ms	8.1 ms
m*2	6.7 ms	8.9 ms	7.2 ms
subarray	3.0 ms	2.2 ms	3.7 ms

The C++ library seems to be much faster when copying arrays or when filling them with a value is required. This is probably due to the fact that neither NArray nor HornetsEye are currently making use of the highly optimized routines of the C++ standard template library.

The C++ implementation is much faster for small sizes than both Ruby implementations. The reason is that the array manipulations in Ruby and the garbage collector have a larger overhead. For larger arrays the benefits of the garbage collector become dominant.For bigger arrays HornetsEye is the most efficient implementation.

VII. CONCLUSION

Existing free and open source software (FOSS) for machine vision is predominantly implemented in C/C++. Albeit the performance of machine code generated by C/C++ compilers is high, the static type system of the C++ language makes it exceedingly difficult to provide a complete and coherent basis for developing machine vision software.It is hard to support all possible combinations of operations and native data types in a statically typed language. Therefore most libraries implemented in such a programming language only support some combinations (e.g. OpenCV and NArray). In contrast Ruby already comes with a set of numeric data types which can be combined seamlessly.

The contribution of this paper is a machine vision system which brings together productivity and performance by implementing machine vision algorithms in ruby. The dynamic programming languages facilitates concise and flexible implementations which means that developers can achieve high productivity. It is presented how the library reviewed in this paper can competitively perform to implement machine vision algorithms.

VIII. FUTURE WORK

Although in this review paper, the Ruby programming language was used, machine vision field could significantly benefit from any dynamic programming language which offers equal or stronger

for meta-programming support. Meta-programming is the ability of program to change itself during run-time. This facilitates implementation of optimization algorithms which would be hard to do in currently popular programming languages. Possible future work is the development of efficient libraries for dynamic languages out of conventional approaches.

IX. REFERENCES

[1] M. Boissenin, J. Wedekind, A. N. Selvan, B. P. Amavasai, F. Caparrelli, and J. R. Travis. "Computer vision methods for optical microscopes, Image and Vision Computing." http://dx.doi.org/10.1016/j.imavis.2006.03.009

[2] Hal Fulton. The Ruby Way. Addison Wesley. http:// rubyhacker.com/

[3] Baker and I. Matthew. Lucas-Kanade 20 years on: A unifying framework. International Journal of Computer Vision. http://www.ri. cmu.edu/projects/project_515.html