# PRODUCTION QUALITY DECISION SUPPORT USING REAL-TIME COMPUTER VISION FRAMEWORK

**Harijs Grinbergs**
Riga Technical University, Latvia
harijs.grinbergs@rtu.lv

**Abstract.** Development of computer vision systems is a complex and tedious task requiring the development of algorithms, prototypes and the final system – these three parts are usually made separately. This means the process is slow and the reuse of algorithms developed by others are rare. One way to speed up such system development could be a unification of these three steps as closely as possible. Therefore, a graph-based framework is proposed, which is designed to be fast in three areas – fast prototyping, fast execution and fast visualization. The framework is based on nodes because most computer vision systems can be efficiently broken down into independent algorithm steps and it allows the creation of a computer vision system without writing any code. Fast prototyping is achieved by allowing any part of the system to be added, changed or removed while the system is running without recompiling or interrupting the execution in any other way. The computer vision algorithms themselves are implemented in a scripting language enabling them to be changed and improved at run-time. Fast execution is done through optimizations provided by the graph-based representation. It is possible to disregard blocks that would not contribute to the output or to control when and how data is copied. It is also possible to run the algorithm on GPU using OpenCL thus significantly increasing the performance for computationally intensive calculations. Specific use cases and adoption of computer vision is hindered by the complexity of making a domain specific system, so a framework that allows creating such system - even without coding - is very useful in these automation cases. Current use cases include, but are not limited to, visual landmark based robot localization for an agricultural robot, tracking livestock or measuring certain parameters like shape or size of vegetables. Creation time of such systems is reduced by an order of magnitude using the proposed framework and the resulting system runs on a wide range of hardware - from desktops PC's to embedded devices.

**Keywords:** computer vision, robot vision systems, precision agriculture.

## Introduction

Like other branches of modern automation, agriculture might be very dynamic in terms of needs for production line reorganization addressing the actual needs of the production process. If at one time the processing facility packs one vegetable, it might need to adapt to a different kind at short notice. For most situations like this the mechanical automation may be easily adaptable, but there are parts of the production system, like automatic quality control, which usually isn't due to specific algorithms and calibration that is needed. Computer vision is used increasingly more in modern quality control because it allows working with variable and non-uniform objects, which is true for organic shapes. Another important advantage is its non–destructive nature allowing quality control without physical handling. Currently, computer vision is used for very diverse tasks in agriculture, from detecting back posture of dairy cows [1] to finding blight disease in tomato leafs [2]. While the whole algorithmic process for each of these cases is different, they often share some of the same image processing blocks. Reusing these blocks and doing so efficiently is key for efficient precision agriculture and rapid reorganization of the quality control systems.

Many of the vegetables and fruits are already classified using computer vision (a brief review can be found in [3]), but implementing several computer vision algorithms together in a single solution usually is nontrivial. There are already block or graph-based methods available for computer vision, like Matlab Simulink with Computer Vision System Toolbox [4], but it is often too slow for many real-time systems and the system cannot be easily transferred to stand-alone products. The same issues are appearing with the existing frameworks like Cassandra [5], which are much faster, but are platform locked on Windows and not usable in embedded systems. So a new framework is required, which could run on modern embedded systems (e.g., for flying drones or mobile robots) and be efficient enough to run in real-time.

The rest of the paper is organized as follows. The framework is described in the first section. Prototyping using the framework is described in section two. How rapid execution speed is achieved is described in section three. Experimental assessment using a precision agriculture task is in section four and conclusions are in section five.

## Framework

The proposed framework uses directed and acyclic graphs to describe and execute a computer vision system consisting of separate image processing steps. Graphs are employed to describe computer vision systems because most of them consist of several self-contained steps that usually run in a linear order. For example, a simple rectangular marker detection consists of these steps [6]: acquire image, threshold image to get a binary image, find contours, filter the contours based on how many points they have, transform the potential marker rectangles to a plane and then compare this transformed rectangle to an existing marker image. Markers are often used in industrial facilities for very precise robot localization. These steps can be seen in Fig. 1. These individual steps also mean that individual nodes in a graph can be implemented separately, and thus allow greater interoperability with other algorithms.

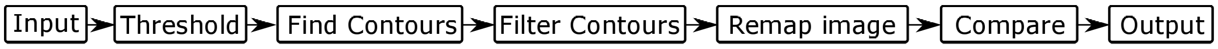

Fig. 1. **Marker detector example**

Analysis of other image processing algorithms used in practice is allowing to identify the following main requirements for the framework:

1. Computer vision systems made in the framework must work at real-time speeds and have as little overhead as possible. This requirement means that the system should run at the speed as closely as possible to code that was implemented in C++ manually.
2. It must be able to run on wide range of hardware systems. This is required for embedded system development, like a vision system for a mobile robot. This means there must be as few software dependencies as possible.

The graphs created using the framework always are acyclic and directed, meaning that there should be a clear input and clear output for the system. Outputs for the system are called sinks because data from other nodes flows to it. When the system is running, these special sink nodes are used to determine which nodes in the whole system need to be run and in what order if they are called in a serial mode. This is done by calculating the transitive closure of the graph using Warshall's Algorithm [7]. This calculates all nodes that are reachable from the sinks. The result is a list. Any node not on this list won't be executed. So it is possible to have large parts of a graph that won't impact performance, as they won't run at all. As the framework is created for computer vision the execution is split into frames. In a single frame the whole system is executed once, i.e. each of the algorithm steps is executed at least one time. When all the nodes in the execution list are executed the frame ends and a new one starts. Currently, the framework doesn't support *double buffering,* i.e., the second frame cannot start before the first one ends.

While the graph as discussed above is directed and acyclic it still allows looping using a token approach inspired by Petri Nets. This allows the framework to perform complex dynamic calculations while still having a clear execution order, e.g., it is possible for one part of the graph to execute once in one frame and several times in another frame. An example showing a loop that prints numbers from 0 to 24 is outlined in **Error! Reference source not found.**. Cycles are automatically detected and this removes the risk of creating an infinite loop. There are watchdog timers in place for both individual nodes and the graph itself, stopping or restarting the system if a calculation is not done in a preset time. This adds error tolerance important for industrial applications. Each node can be specified to run in serial mode, instead of parallel, in cases where threading can cause race conditions. This is important for robot control logic, input/output operations or drawing. In these cases, required nodes can be executed in a serial manner, the order of which can be easily changed in the graphical user interface.

There are two ways nodes can be connected with each other. One is trough data – e.g., node output with type string, can only be connected to node input with type string. Only matching types can be connected and only outputs to inputs and inputs to outputs. This is useful when the system doesn't have any control flow, e.g., *if/else* or *switch* statements. It is sometimes possible that control is necessary and in these cases another connection can be made, called execution connection or *exec* for short. In the graphical user interface they are represented as triangle inputs or outputs like depicted in **Error! Reference source not found.**. Every node has these *exec* input and output controls but are by default hidden. *Exec* connections allow unrelated nodes to execute each other while data connections only allow nodes with some data relation to do it. For example, *image input* node shouldn't have any data relation with *draw circle* node, so it is not possible for *image input* to execute *draw circle* through data, but it is possible through *exec* connection.

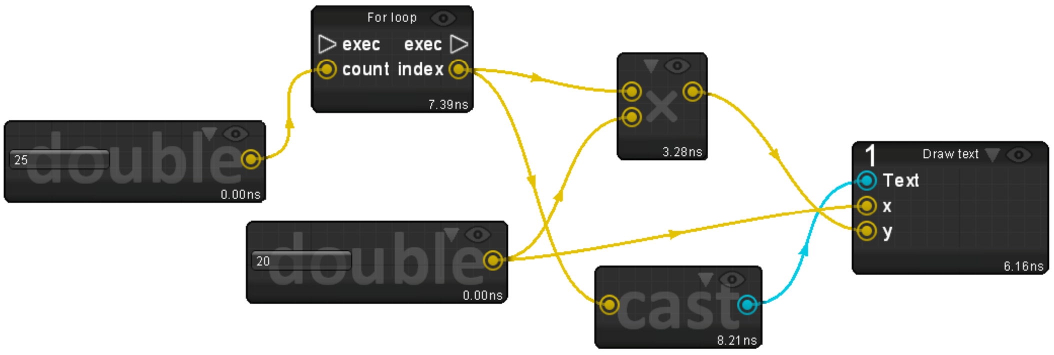

Fig. 2. **For loop example as shown in graphical user interface**

## Prototyping

Prototyping is a key part in creating a complex computer vision system because different tasks require different algorithms. The proposed framework allows implementing algorithms in nodes and then use these nodes as individual pieces to create a system. Nodes are connected via inputs and outputs and don't share any global state with any other node. This means the nodes can be disconnected and replaced at will while the system is running. Nodes in the framework are implemented in a scripting language called AngelScript [8] which shares C++ syntax. It allows easy interfacing with C++, is efficient to execute and runs on a wide range of platforms. The scripts are loaded and compiled at runtime so it is possible to implement and change a node while the system is running. This increases testing iteration speed for new algorithms.

The nodes are defined in an XML file. The definition includes all the higher level information about the node – what inputs and outputs it has, if the node is deterministic (outputs doesn't change if the inputs don't change), can it be run in parallel, is it an output node and more. An example definition for *Image Box Blur* node with two inputs and one output can be seen in Fig. 3.

```
<node name="Image box blur" script="nodeboxblur.as">
<input name="Image" type="image" />
<input name="Kernel size" type="double" />
<output name="Output" type="image" />
</node>
```

Fig. 3. **Node definition in XML**

The definition specifies a script file, which will be loaded and used for this node. They are reloaded automatically when changed. For higher performance it is possible to call compiled functions or to use existing optimized computer vision libraries, like OpenCV [9]. An example script can be seen in Fig. 4. Scripting languages are often a lot slower than compiled languages. In this case, AngelScript is up to 10 times slower than equivalent C++ code. This is especially true for iterative or repetitive calculations, like using *for loops*. So it is still beneficial to use C++ after the prototyping stage is done. But as the scripting language syntax is very similar to pure C++, the required effort is greatly reduced. AngelScript also has a Just-In-Time compiler support which can speed up execution of the script.

```
class Node
{
  bool update(NoviNode &node){
    int ks = int(node.input(2).get_double());
    if (ks<=0){
      node.set_error("Kernel size is <=0!");
      return false;
    }
    cv::blur(node.input(1).get_cvmat(), node.output(1).get_cvmat(), cv::Size(ks,ks));
    return true;
  }
}
```

Fig. 4. **Node script for Image Box Blur using OpenCV function**

The framework provides an optional graphical user interface, which shows a graphical representation of the computer vision system's graph and allows interactive changes while the system is running. The previously described node would be visualized as shown in Fig. 5. Execution speed is very important when developing a real-time based system, so by default, all node execution times are measured and displayed at the bottom of the node itself. It is possible to look at data coming in and out of the node and that is very useful for debugging. The colors of inputs and outputs show type, so it is easy to see which can be connected. The graphical user interface is separate from the framework and uses OpenGL for rendering.

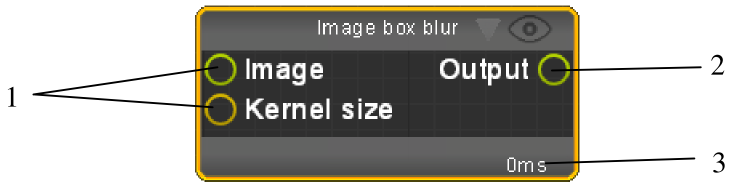

Fig. 5. **Node visualization in graphical user interface:** 1 – inputs; 2 – output; 3 – execution time

## Execution

Normally after algorithm prototyping is complete the result is rewritten to gain more performance or to run in embedded systems. This means that either the program has to be rewritten from scratch in a more efficient programming language or a lot of time needs to be spent making the existing code more efficient. The proposed framework is designed to be very fast at execution, so even complex systems can be usable as production ready implementations. The framework can also be easily included in C++ projects and can be run from command line (headless) allowing integration into existing projects both local and cloud. The framework uses graphs as a way to increase the systems performance. It is often possible to skip execution of nodes if their input doesn't change (when the node is deterministic) or the nodes can be run in parallel. Graphs allow these improvements to be done at much higher level and require little knowledge about the implemented algorithms themselves. There are some computer vision based systems which work a lot better at higher framerates [10], so it is important to have high performance even when prototyping the system. The framework itself is C++11 based. One key consideration is using GPU's which have a lot of computation cores for calculations, which are useful as most image processing algorithms are *Embarrassingly parallel*, i.e., it takes little effort to split them into parallel tasks [11]. For that OpenCL is used allowing runtime compilation of GPU-executable code [12]. This means that it is possible to implement an algorithm running on GPU while the system is running, thus greatly improving prototyping speed. Graph based system allows passing input and output data to algorithms more efficiently, as it is possible to detect whether the node needs the data copied. This feature is especially useful when running nodes on GPU's and CPU's interchangeably.

## Experimental assessment

To assess the framework a practical implementation has been made for use in precision agriculture domain. Particularly for quality control of vegetables – checking the circularity of tomatoes. The shape is one of the criteria for tomatoes classification in Europe [13] and automatic determination of defects improves classification speed and precision. To accurately determine the shape of an object it is required that the object is not occluded, so tomatoes which are wholly visible are used for inputs. In a factory processing phase when the tomatoes are on conveyor belt this is not an issue, but if the fruit is still hanging on a vine it might not always be possible. In this test tomatoes are on vines to simulate an automatic classification by a mobile robot.

The system consists of an image input, the pre-processing stage that uses filtering in HSV color space to find the tomatoes and circularity calculation as described in [14]. The output for drawing the image and circularity value is presented as well. In the graphical user interface it is possible to group these steps in larger blocks like depicted in Fig. 6.

The circularity function returns from 0.0 to 1.0, so the result can be then used for classification with simple thresholds. If there are several tomatoes in an image only the largest one is returned for comparing results in order to keep the example simple. The input images, as well as system output, can be seen in Fig. 7. The first four dataset images show round undamaged tomatoes, which as a result have high circularity. Note that in fifth dataset image the tomato has a large healed scar. The segmentation algorithm doesn't segment it and as a result, the circularity is only 0.478 instead of >0.75 for healthy tomatoes. In this way it is possible to detect scarred tomatoes which need to be put in a different class.

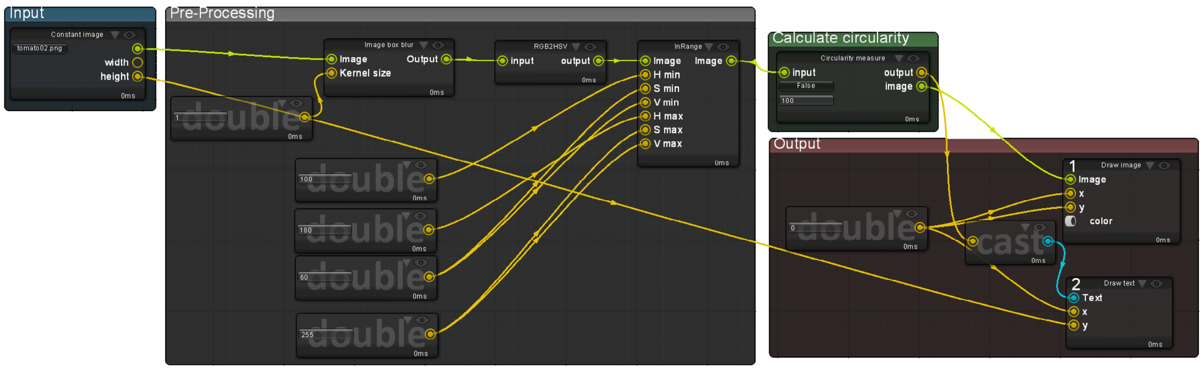

Fig. 6. **Computer vision system as seen in the framework for finding circularity of tomatoes**

As the whole computer vision system is implemented intuitively using a graphical user interface, it is easy to tune algorithm input values and observe the result instantly. And the system can also be rapidly changed without any coding for a different task. For example, if a cucumber straightness needs to be determined instead of tomato circularity, then only a single node needs to be replaced in the graph – *Calculate circularity* with *Calculate linearity*. Then parameters in pre-processing stage need to be modified to segment cucumbers instead of tomatoes. This doesn't require any code changes, takes very little time and can be done while the system is running. This allows rapid adaptation to manufacturing changes. And as all nodes are independently implemented it is possible to upgrade to better and more precise computer vision algorithms as they evolve. This means any production level computer vision system can always be state-of-the-art.

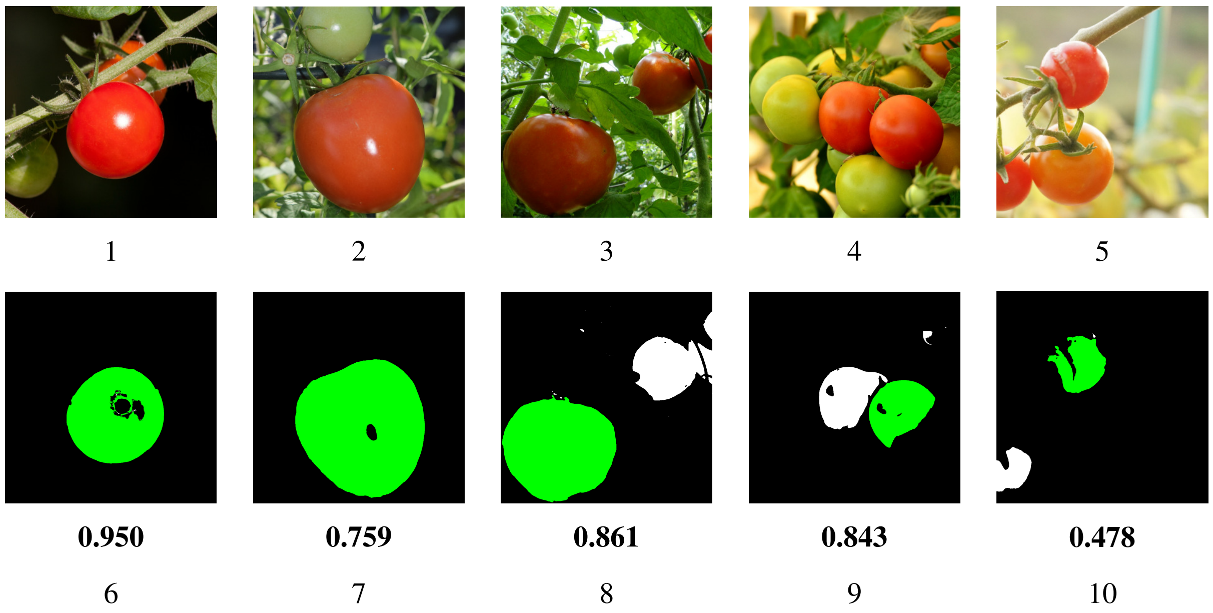

Fig. 7. **Calculating circularity for tomatoes:** 1-5 – input data;
6-10 – segmentation output with circularity values

## Conclusions

Existing computer vision systems are inflexible and require a lot of engineering effort to build and maintain. Current computer vision frameworks are more suited for static factory settings and cannot be used for more dynamic environments, like fruit inspection in a greenhouse using a mobile robot. The proposed framework speeds up prototyping, development and adaptability of computer vision systems using a very efficient graph-based method which can be used without writing and maintaining complicated code. An example was demonstrated where tomato circularity was calculated with a simple graph-based model while still having the flexibility to easily switch to a different kind of inspection.

**Acknowledgement**

This research has been supported by the Latvian state research program 2014-2017 project "The Next Generation of Information and Communication Technologies (NexIT)".

## References

1. Viazzi S., Bahr C., Van Hertem T., Schlageter-Tello A., Romanini C.E.B., Halachmi I., Lokhorst C., Berckmans D., Comparison of a three-dimensional and two-dimensional camera system for automated measurement of back posture in dairy cows, Computers and Electronics in Agriculture, Volume 100, January 2014, pp. 139-147, ISSN 0168-1699
2. Arakeri M., Arun M., Padmini R K,Analysis of Late Blight Disease in Tomato Leaf Using Image Processing Techniques, IJEM, vol.5, no.4, November 2015, pp. 12-22.
3. Kodagali J. and Balaji S. Article: Computer Vision and Image Analysis based Techniques for Automatic Characterization of Fruits - A Review. International Journal of Computer Applications 50(6), July 2012, pp. 6-12.
4. The MathWorks inc. Computer Vision System Toolbox [online] [15.03.2016]. Available at: http://se.mathworks.com/products/computer-vision/
5. Hella Aglaia Mobile Visionn. Cassandra Vision [online] [17.03.2016]. Available at: http://www.cassandra-vision.com/
6. Garrido-Jurado S., Muñoz-Salinas R., Madrid-Cuevas F. J., and Marín-Jiménez M. J. Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recogn. 47, 6, June 2014, pp. 2280-2292.
7. Warshall S. 1962. A Theorem on Boolean Matrices. *J. ACM* 9, 1 (January 1962), pp. 11-12.
8. Jönsson A. AngelCode scripting library. [online] [20.03.2016] Available at: http://angelcode.com/angelscript/
9. Bradski G. The OpenCV Library. Dr. Dobb's Journal of Software Tools. 2000
10. Grinbergs H., Mednis A., and Greitans M. Real-time object tracking in 3D space using mobile platform with passive stereo vision system. N. Tagoug (Ed.): Proceedings of World Congress on Multimedia and Computer Science (WCMCS 2013), Association of Computer Electronics and Electrical Engineers, 2013, pp. 60-68.
11. Maurice Herlihy and Nir Shavit. The Art of Multiprocessor Programming, Revised Reprint (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2012, pp. 14.
12. Stone J., Gohara D., and Shi G. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. IEEE Des. Test 12, 3 May 2010, pp. 66-73.
13. ANNEX I to Commission Implementing Regulation (EC) No 543/2011 of 7 June 2011 laying down detailed rules for the application of Council Regulation (EC) No 1234/2007 in respect of the fruit and vegetables and processed fruit and vegetables sectors [online] [20.03.2016]. Available at: http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32011R0543&qid=1458937284160&from=en
14. Stojmenovic M. and Nayak A. Shape based circularity measures of planar point sets. Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on, Dubai, 2007, pp. 1279-1282.