ImageNets – Framework for Fast Development of Robust and High Performance Image Processing Algorithms

ImageNets – Rahmenwerk zur schnellen Entwicklung von robusten Hochleistungs-Bildverarbeitungs-Algorithmen

Uwe Lange*, Henning Kampe, Axel Gräser, University of Bremen

* Correspondence author: ulange@iat.uni-bremen.de

Summary In this publication, we describe the novel ImageNets framework which enables developers with low programming knowledge to rapidly create efficient high performance image processing algorithms and which is available under an open source license[1]. The performance of this novel tool has been compared with the simple to use Image Processing Toolbox of MATLAB/Simulink. A group of 49 students, attendees of a robot vision lecture, had to solve an image processing task using both tools. Different evaluation criteria have been measured or determined by interrogation of the students. The advantage of the new tool ImageNets has become clearly visible. ►►► **Zusammenfassung** In diesem Artikel stellen wir das neue Rahmenwerk ImageNets vor, das bei geringen Programmierkenntnissen die schnelle Entwicklung leistungsfähiger, effizienter Bildverarbeitungsalgorithmen ermöglicht und unter einer Open Source Lizenz frei verfügbar ist[1]. Die Leistungsfähigkeit des neuen Tools wurde in einen Vergleichstest zwischen ImageNets und der vergleichbar leicht zu nutzenden Image Processing Toolbox von MATLAB/Simulink ermittelt. Eine Gruppe von 49 Studenten, Teilnehmer einer Vorlesung über Bildverarbeitung in der Robotik, hatten dazu eine Bildverarbeitungsaufgabe mit beiden Tools zu lösen. Die verschiedenen Beurteilungskriterien wurden gemessen oder durch Abfrage der Studenten ermittelt. Der Vorteil des neuen Tools ImageNets kann so klar nachgewiesen werden.

Keywords Image Processing, robotics, framework ►►► **Schlagwörter** Bildverarbeitung, Robotik, Rahmenwerk

1 Introduction

The development of intelligent behavior for assistive robots, which operate in partially unknown environment and especially under varying illumination conditions, is a complex and time-consuming task. Especially the robot vision system has to robustly perceive the environment for collision-free robot motion and successful object recognition and grasping.

During the development of the FRIEND [1] assistive robot, several shortcomings in the design of image processing algorithms have been experienced. The FRIEND system, which supports disabled people like

[1] http://imagenets.sourceforge.net

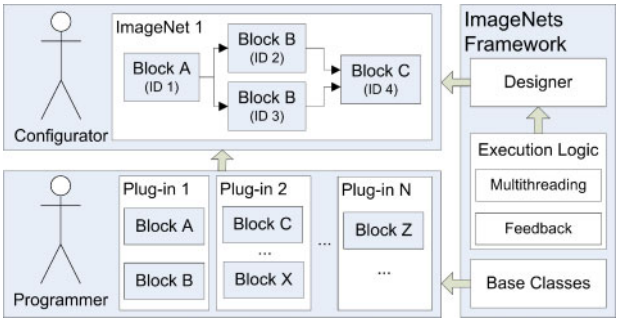

Figure 1 The user levels for the ImageNet framework.

quadriplegics[2] in professional and everyday life, consists of a wheel chair with an attached robot arm and several cameras and other sensors.

The development of the necessary image processing skills for new robot tasks proved to be one of the most challenging and time consuming tasks in service robot development. With the development of the tool ImageNets the shortcomings of available standard and advanced image processing tools are overcome. ImageNets consists of two user-levels:

- a configurator level at which new image processing algorithms are configured and parameterized with help of existing, well implemented, documented and optimized basic algorithms.
- a programmer level that allows the implementation of new algorithms that extend the number of configurable basic algorithms.

The two user-levels are depicted in Fig. 1. The main user is the *Configurator*. He only needs knowledge about the algorithm he intends to create and uses the ImageNet Designer together with the existing function blocks. This is explained in Sect. 3.1. If new functionality has to be added, a *Programmer* has to create new function blocks or change existing ones as shown in Sect. 3.7. Both, *Configurator* and *Programmer* interact with the base framework of ImageNets.

ImageNets has the following advantages:

Adv. 1 To be used with basic image processing knowledge: ImageNets provides a tool that allows people with standard image processing knowledge to configure even challenging image processing tasks. This is achieved by utilizing the concept of configuration and parameterization on image processing level.

Adv. 2 Generation of executable software: ImageNets generates executable software without the need for compilation and with a low overhead and a short execution time.

Adv. 3 Easy debugging of algorithms: ImageNets offers the possibility to view the results of algorithms and parameter changes immediately (images can be used by the developers to judge the results).

Adv. 4 3D visualization: ImageNets embeds an integrated 3D simulation environment tailored for robotics.

The outline of the paper consists of the following: Sect. 2 describes state of art and discusses to what extent comparable frameworks fulfill the above mentioned advantages. In Sect. 3 ImageNets is described in detail while the fulfillment of the advantages by ImageNets is evaluated in Sect. 4. Finally, Sect. 5 demonstrates the outcome and future work.

2 State of the Art

Since high execution performance is desired in robotics, a language with compiled code like C++ is common. The open source computer vision library OpenCV [3] provides C++ with access to much basic functionality and is therefore often used. Still it is quite challenging to program complex image processing algorithms with OpenCV and to transfer knowledge between developers. Complexity of solutions can be reduced and productivity increased by using a graphical tool [4]. One kind of tool which is commonly used is function block programming which is based on the human capability to easily comprehend graphical elements. It enables the programmer to directly use and parameterize pre-compiled functional blocks to model an algorithm. A number of graphical programming tools are available today, each tailored for a specific industry. Also industry considers graphical programming as a safe way of programming for safety related issues [5].

Another quick way to develop image processing algorithms is by using an interpreted programming language where the parameters can be modified without the need for time consuming compilation. However, interpreted languages decrease the performance of software compared to compiled programming languages like C/C++ considerably. Therefore, the combination of rapid prototyping by a graphical programmers interface and real time execution is desirable. The widely used MATLAB®&Simulink® [6] "Image Processing Blockset"

Table 1 Comparison of computer vision frameworks.

		MATLAB	LabVIEW	MeVisLab	GraphEdit	Halcon	OpenCV	ImageNets
Development	Function Block Programming	×	×	×	×			×
	Function Block Sub-Routines	×	×	×				×
	3D Data Visualisation	×	×	×		×		×
Execution	High Execution Perfomance		×	×	×	×	×	
	Feedback Structures	×	×					×
	C++ Code Connectivity	×	×	×	×	×	×	×
General	Platform Independent	×		×		×	×	×
	Open Source						×	×

[2] A quadriplegic is a human whose arms and legs are paralyzed.

and LabVIEW [7] environments provide the possibility for rapid prototyping of image processing algorithms. But a major drawback is the dependency on interpreted language. The medical image processing toolbox MeVisLab® [8] combines the advantages of graphical programming and also includes a 3D-environment for displaying reconstructed medical images. Due to its major focus on medical image processing, this toolbox is not suited well for robotics. GraphEdit® [9] and Halcon® [10] are mainly tailored for machine vision and image analysis applications but not for robot control. The various features of a representative set of existing image processing toolboxes are documented in Table 1.

None of the investigated tools met all of the targeted advantages in the important areas for robot control.

3 ImageNets

As none of the presented software tools fulfills the targeted advantages from Sect. 1, a novel open source framework for robust robot vision has been developed. The framework consists of three main components, namely:

- ImageNet Designer, a graphical user interface.
- ImageNets core library, providing the base framework including an event flow mechanism.
- A set of plug-ins[3], containing function blocks.

Each component is explained in the following subsections.

3.1 ImageNet Designer

The ImageNet Designer is a graphical user interface, enabling developers to create an ImageNet and store it in XML format. It has been implemented to support the developers of robot vision algorithms while focusing on the first advantage: "supporting fast development". The general and widely approved/accepted method [5–9] of creating function block networks builds the basis for ImageNet Designer.

As depicted in Fig. 1, the ImageNet Designer is used to model complex image processing algorithms. An algorithm consists of a set of function blocks which have input and output ports. Through these ports an event and data flow takes place by linking an output into an input. The core library of the ImageNet framework handles the execution based on the used blocks and their linkage as explained in detail in the next sections. It also loads the plug-ins with the available function blocks. In the designer multiple ways to find a desired function block exist, which consume a small amount of time. All blocks can be created from menus, sorted by name, category, input or output port data type. The transfer of knowledge of how to use a block is achieved by demos, accessible in the ImageNet Designer.

[3] A plug-in is a dynamically loadable software library to add specific abilities to a software application.

Possible configuration errors introduced by the user are prevented by limiting the allowed values for each parameter of a block. If for example a threshold is only valid in the range of [0–255], then the user cannot set other values. Furthermore, ports ensure valid linking by using defined data types. Most data types are based on OpenCV [3], which is a high performance image processing library based on C++, to support Adv. 2. These data types are visualized in the ImageNet Designer by meaningful icons.

The visualization of intermediate results is very useful for intuitive debugging. In conventional programming, output of intermediate results have to be hardcoded e. g. as console outputs or pop-up windows.

While designing an ImageNet with the Designer, intermediate results can be visualized either two dimensional e. g. with an image or as three dimensional entities such as point clouds and bodies. This direct visual feedback to the user eases algorithm development (Adv. 1). Some data types like scalar values over time can only be visualized in 2D, whereas images with camera matrices can be visualized in 2D and 3D (using OpenGL[4]). In addition, the Designer is able to visualize point clouds, calculated by stereo vision or directly captured by Time-of-Flight (ToF) cameras.

While the 2D visualization is a part of every vision library, a 3D visualization is not found in the existing image processing toolboxes. That extension of ImageNets fits to the needs of robotics. This capability requires the combination of the robot model, camera images and point clouds, which is possible with ImageNet Designer (Adv. 3, 4).

Documentation is included in every ImageNet block, and available for every of its ports and properties. Moreover every block has a link to its own page in the ImageNets Wiki where a detailed description of the algorithm and the implementation is stored. All blocks can easily be used as components-of-the-shelf (Adv. 1).

Further developing aids include warnings for missing input links on net execution and an event logger. The event logger gathers messages from the blocks during processing sorted by time and log level (e. g. error or warning). This textual data can also be visualized as a diagram. Corresponding blocks are directly accessible from the log entries, thus testing is simplified (Adv. 1).

For debugging purpose, visualization blocks (e. g. draw 2D points on an image) can be marked for "execution only in the Designer" mode. Visualization is automatically deactivated for blocks that are used inside a sub-net. This functionality supports both fast development and high execution performance (Adv. 1, 2).

ImageNet Designer provides short help documentation for explanation for ImageNet beginners and also links to a growing number of tutorials for self-learning purposes.

[4] http://www.opengl.org/

3.2 ImageNets Logic

ImageNets uses a modified version of the function block network concept from IEC 61131-3 using a combined data and event flow. The general concept of function block networks is not new but applied in several programs like PLC programming. Function blocks can be handled as reusable black boxes with images as inputs, outputs and parameters while the actual processing code is hidden. Thus, irrelevant details are hidden from the developer and the overview about the whole algorithm is easily maintained. An algorithm is built by placing a set of blocks inside a net and by linking outputs into inputs. Each input must have exactly one ingoing link. Outputs can be linked to any amount of inputs. The structure of this net also defines the execution order of the blocks. How this is handled by the framework is explained later in this section. Blocks may be configured with parameters to define execution details like a threshold.

The fundamental entity of ImageNets is referred as block *b*. It has a unique name, identification number (ID), a finite number of input/output ports and a set of parameters as described below.

Block b := (Name, ID, P_{In}, P_{Out}, P)
ID = (1…*m*)
Name = (Category + Block Name)
Input ports $\mathbf{P_{In}} := \{p_{in,i} | i = 0…q\}$
Output ports $\mathbf{P_{Out}} := \{p_{out,i} | i = 0…r\}$
Parameter P := $\{p_i | i = 0…s\}$

The set of all available blocks is denoted as B*. All blocks in B* have a unique name.

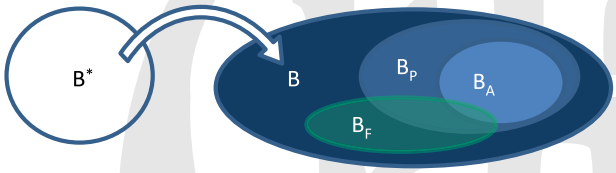

Figure 2 Relations between the ImageNets block sets.

Set of all blocks $\mathbf{B^*} := \{b_i | i = 0…k\}$

Consider an ImageNet *N* with a set of blocks B. Two identical copies of a block from B* used in the same net N are distinguished using their unique ID. Let δ represent the connection function between the blocks using the IDs. All established connections in the net are stored in the set δ^*. The set of blocks to be processed is described by B_P and B_A represents the set of currently active blocks. The framework supports feedback structures which may be applied in the designer to any existing block. B_F is the set of blocks for which special feedback rules take place during execution as explained in Sect. 3.4. The formal definition of the ImageNet and its components can be found below.

ImageNet N := $(\mathbf{B}, \boldsymbol{\delta}^*, \mathbf{B_P}, \mathbf{B_A}, \mathbf{B_F})$
Blocks B := $\{b_i | b_i \in B^*, i = 0…m\}$
Connection function $\boldsymbol{\delta}$:= $(ID_{Start}, P_{OutStart}, ID_{End}, P_{InEnd})$
$\mathbf{ID_{Start}}$ = Start block ID
$\mathbf{P_{OutStart}}$ = Start block out port
$\mathbf{ID_{End}}$ = End block ID
$\mathbf{P_{InEnd}}$ = End block in port
Set of all connections $\boldsymbol{\delta}^*$:= $\{\delta_i | i = 0…l\}$
$\mathbf{B_P} \subseteq B$:= Blocks to be processed
$\mathbf{B_A} \subseteq B_P$:= Active blocks
$\mathbf{B_F} \subseteq B$:= Feedback/optimization blocks

The relation between the block sets is visualized in Fig. 2.

Figure 3 shows an ImageNet for the calculation of a point cloud which is based on a pair of stereo images. The color images are loaded and converted to gray level images. After a disparity calculation, which is based on block matching, the point cloud is calculated and additionally the color information of the original left image is added to build a 6-dimensional data vector for each 3D point.

The execution order of an ImageNet is determined automatically by Algorithm 1. For the ImageNet, shown in Fig. 3 the execution order is displayed in Table 2.

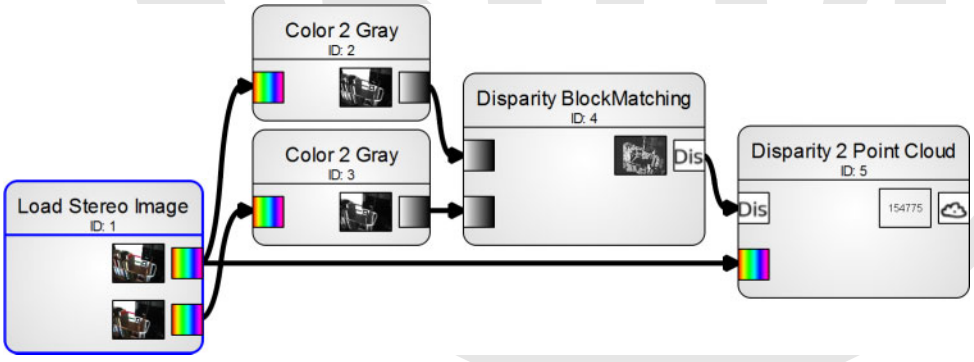

Figure 3 ImageNet which calculates a point cloud from stereo images.

Table 2 Active and to be processed block sets before and while execution of the ImageNet of Fig. 3.

Step	B_P	B_A	B'_A
0	{}	{}	{}
1	{1, 2, 3, 4, 5}	{1}	{}
2	{2, 3, 4, 5}	{2, 3}	{1}
3	{4, 5}	{4}	{2, 3}
4	{5}	{5}	{4}
5	{}	{}	{5}

Algorithm 1 (Process entire ImageNet).
$B_P := B$
while $(B_P \neq \{\})$
  $B_A := \text{PreconditionsFullfilled}(B_P)$,
  $B'_A := \text{DoneProcessing}(B_A)$
  $B_A := B_A \backslash B'_A$
  $B_P := B_P \backslash B'_A$
end while

In this example, the blocks are executed from left to right. At start, only the leftmost block, which has no pre requisite, can be executed. In the next step, the blocks 2 and 3 can be processed in parallel. When parallel execution is enabled, the framework executes all parallel blocks in multiple threads parallel. The remaining blocks are executed serially from left to right. During the execution, all blocks to be processed are in the set B_P and all active blocks in the set B_A. Processed blocks are removed from B_P. During execution, data flows between the ports of the blocks, depending on the connection functions in δ^*.

One or more blocks are grouped in a plug-in using the Qt library[5] and an arbitrary number of plug-ins can be loaded at run-time by ImageNets. In this way, ImageNets can be easily extended by completely independent units of code. Thus, the compilation errors in one plug-in do not affect other plug-ins. Every plug-in is completely free in its scope of application so that dependencies can be kept to a minimum. More details to plug-ins follow in Sect. 3.7.

Figure 4 shows the class structure of the ImageNet framework. All blocks inherit from the class CBlock, which implements core functionalities and defines the interface functions, like the *process* function. The required lines of code for most derived blocks is very limited, provides short compile times and therefore is easily manageable in code inspection. All these features help to achieve the first advantage "To be used with basic Image processing knowledge" of fast algorithm development.

Basis of the ImageNet framework is the core library of ImageNets, which is called LogicNet (see Fig. 4, center). It holds the main functionality to load the plug-ins with the blocks, which result in B*. An ImageNet is represented by the class CImageNet which owns its blocks B and can process the net as described.

The special CImageNetExecutor block in LogicNet runs a specified ImageNet, which enables building of hierarchies (see Sect. 3.3). Furthermore, this block can be used to load and execute an ImageNet with its blocks from either the ImageNet Designer (see Sect. 3.1) or using any other external application. This potential allows the nets and blocks to be used as black boxes and improves the reusability. Since the graphical part and the logical part of ImageNets are completely decoupled, the ImageNet Designer does not have to be loaded for execution of an ImageNet, which saves a significant amount of memory and supports Adv. 2 "Easy debugging of algorithms".

Additionally, this approach combines the design of the algorithm and its actual implementation. The algorithm parameters in the executed ImageNet can be configured in real-time using the Designer.

3.3 Hierarchical Modeling

Hierarchical modeling is a common method to subdivide algorithms into separate parts. This breaks down the complexity and facilitates reusability (Adv. 1).

A small example of hierarchical modeling is depicted in Fig. 5. *Color2Color3D* is an executor block which performs the sub-net shown beneath.

Every ImageNet can easily be adapted to be used as a sub-net by adding executor input and output blocks to the net in the Designer. The executor input and output *blocks* in the sub-net are then mapped to *ports* of the

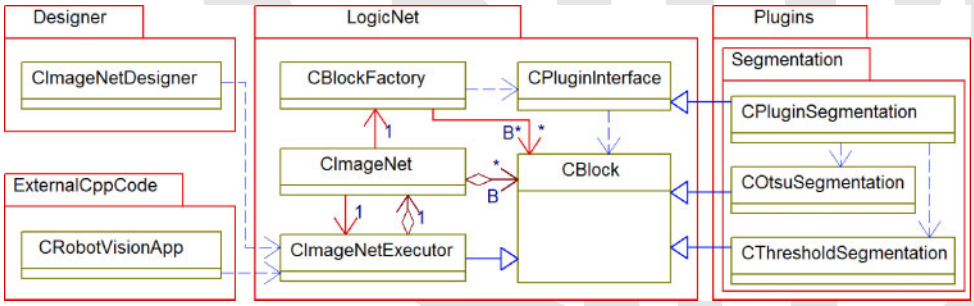

Figure 4 Class diagram of the ImageNets structure.

[5] http://qt.digia.com/

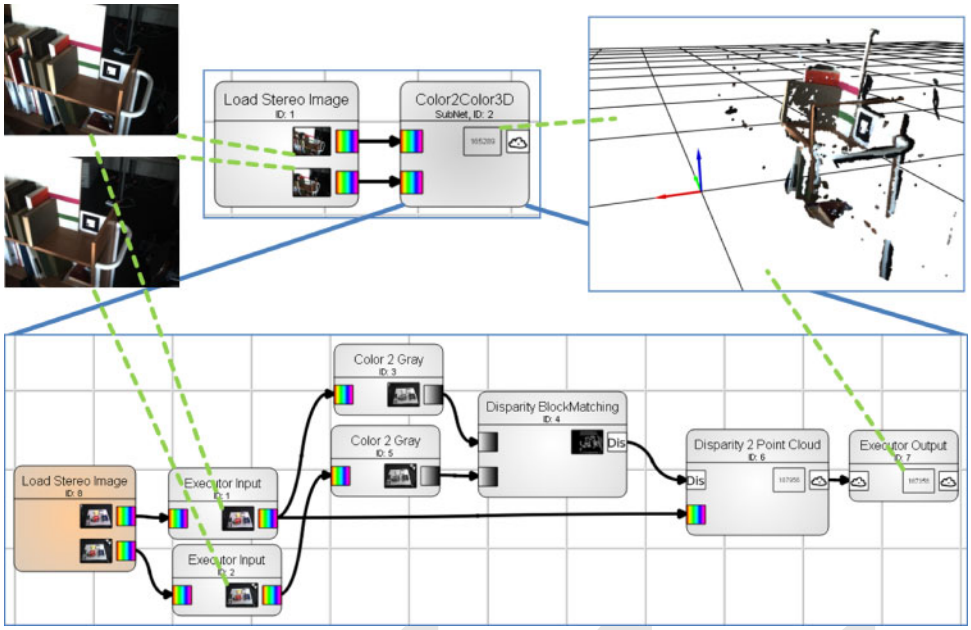

Figure 5 Hierarchy example.

executor block in the outer net. To support advantage 2, any block before an executor input block and after an executor output block is ignored. The orange block in Fig. 5 on the left is ignored when an image net is used as a sub-net.

The sub-nets can either be used as a black box or their parameters can be overwritten in an executor block. Since executor blocks can be integrated in sub-nets, hierarchical modeling is possible with any depth of layers.

3.4 Feedback Structures

In [11] the machine vision framework ROVIS representing Robust Vision Framework for Service Robotics is presented. The robustness of ROVIS against external influences is achieved through integrated feedback structures at different levels of the vision system. Feedback structures are useful to achieve robust object detection under various lighting conditions and are superior to conventional open loop vision algorithms [14].

For this reason, feedback is one of the core aspects of ImageNets to have automatic parameter adaptation. The general feedback mechanism can be applied to any block from the Designer and is therefore very flexible to use.

Table 3 shows the analogies which hold when comparing control theory with feedback structures in ImageNets. To implement feedback, an evaluator has to be introduced, which measures the quality of the outcome of the algorithm. Based on this, parameters of the algorithm can be adjusted until an optimal result is achieved.

Figure 6 illustrates an algorithm which uses feedback to optimize a threshold operation based on entropy. In the first part of execution all blocks from left to right are executed. Normally, only the first two blocks could be executed and afterwards the net would stop since the next two blocks would wait for each other's result. Only

Table 3 Analogies of control theory terms and their ImageNets equivalents.

Control Theory	ImageNets
System	Algorithm
Sensor	Evaluator, quality criterion
Controller	Rules to change parameters
Actuator	Parameter of a Block

because feedback ports are ignored in the beginning of net execution, all blocks will be executed once. During this first execution, the input of the feedback port is empty since the linked block was not processed yet. As already mentioned above, this feedback was applied to the threshold segmentation block in the designer and no programming is necessary. It varies the threshold of the segmentation. A defined start value is used for the first execution which leads to a bump free initialization of the control loop. After the first complete execution, the block with the feedback port and its following blocks are executed until the desired output is obtained or another abort criterion holds. An abort criterion can be whether a certain feedback port value leaves its range limits. In this example the condition is to minimize the entropy but ignore a value of zero in the complete threshold range (0 to 255 with step size 1). To visualize the entropy values, they are collected after every completion of the loop and for visual inspection the entropy graph is plotted by the last block in the net.

Since all blocks after a block with feedback are executed again when the feedback block was processed, it is necessary to place feedback loops that have to finish before the following part of the algorithm is executed inside a sub-net. The loop finishes then during the execution of

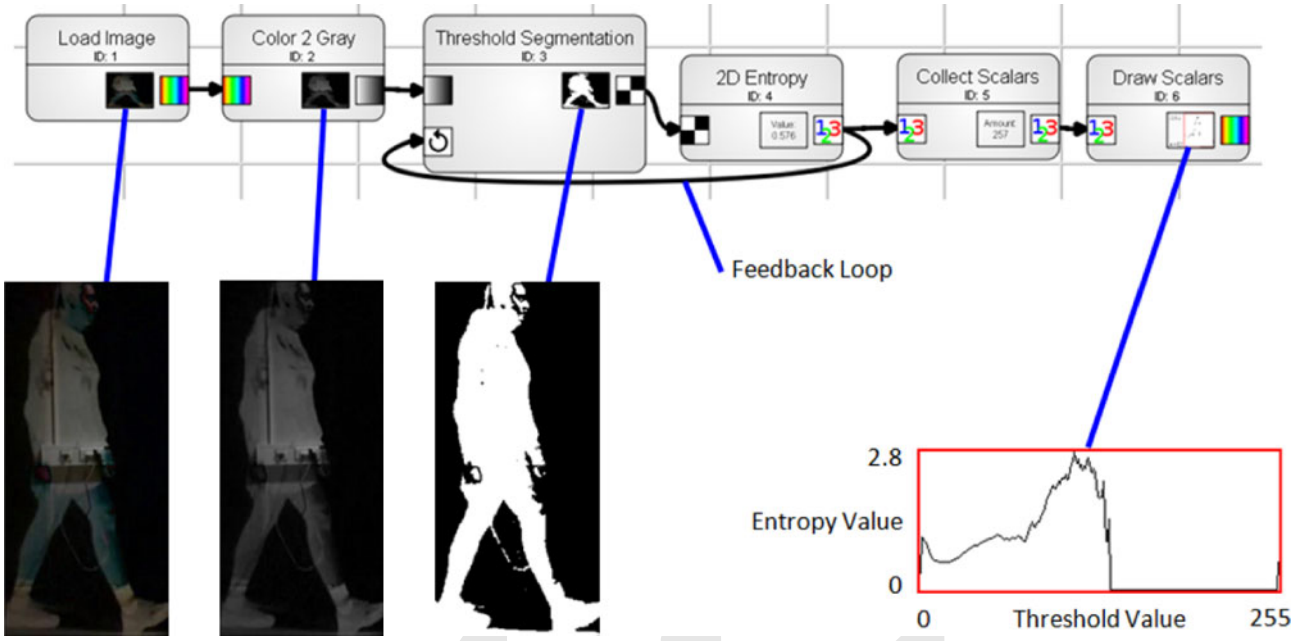

Figure 6 Feedback optimization example.

the sub-net and only the final results are passed on to the outer net. This enables cascaded feedback structures.

The extended algorithm which executes a net with feedback is:

Algorithm 2 (Process entire ImageNet with feedback).

$B_P := B$
while$(B_P \neq \{\})$
  $B_F :=$ FeedbackBlocks(B_P)
  while$(B_P \neq \{\})$
  $B_A :=$ PreconditionsFullfilled(B_P)
  $B'_A :=$ DoneProcessing(B_A)
  $B_A := B_A \backslash B'_A$
  $B_P := B_P \backslash B'_A$
  end while
  $B'_F := \{\forall b_j \in B_F | \text{FeedbackConditionTrue}(b_j)\}$
  $B_P := B_P \cup B'_F$
  $B_P := B_P \cup \{\forall b_j \in B'_F | \text{ExtendedPostBlocks}(b_j)\}$
end while

Algorithm 2 extends Algorithm 1 by feedback execution rules. The algorithm processes the whole net normally in the first run. Afterwards, feedback blocks are checked for reprocessing depending on their feedback rules. The part of the net starting from the feedback block will then be reprocessed in a loop, until an abort criterion like "maximum number of executions" or "value is in goal range" is met. Feedback rules to set the parameters of a block can be defined to find an optimal value. This can be achieved either by changing the parameter based on the evaluator or, if that is not possible, by a numerical optimization procedure.

The user has to configure the linkage of the blocks and the condition for the loop correctly to prevent unrealistic, false starting conditions or infinite loops. Simple range based tests can help to easily visualize the effect of a parameter to the result of an algorithm. Each iteration loop's result can be displayed at the end of the execution, depending on the user requirement. Based on this overview, optimization rules can be generated.

Feedback structures can also be used to simplify the processing of complete videos. A video is loaded frame by frame. Single frames are processed with the same method that may contain additional feedback structures and afterwards fused to a result video. The outer feedback ensures processing of all frames in subsequent execution loops. With the internal feedback of the processing method the optimal result for each frame is ensured.

3.5 Integrated Multithreading

The high performance of ImageNets is improved by the integrated multithreading capability (Adv. 2). Each block in ImageNets is executed by the framework in a separate thread. On the one hand, this keeps the ImageNet De-

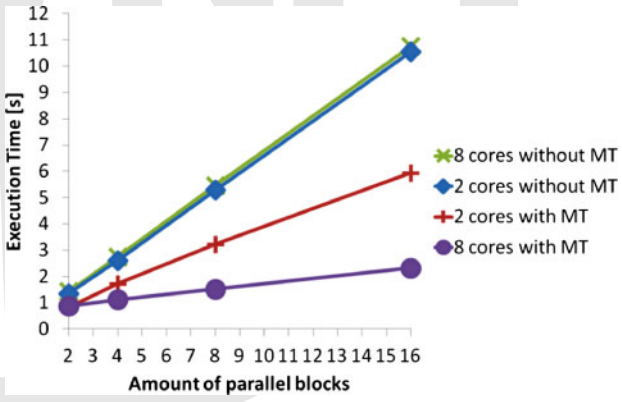

Figure 7 Multithreading performance of ImageNets.

signer always responding to user inputs and on the other hand this also offers the possibility of speeded up execution by the configuration of a net. Thus, the developer does not have to deal with the details of multithreading but can make use of its improved performance.

Figure 7 displays time measurements of the integrated multithreading capability of ImageNets. Two computer systems have been compared: one with two and one with eight cores. With deactivated multithreading, both computers performed almost equally. After activating multithreading, the 8-core system has speeded up the calculations by more than 4 times and the 2-core system has been speeded up by almost 2 times. The values have been calculated for an intrinsic camera calibration process where 10 images have been loaded and processed by a set of same blocks.

3.6 Hardware Access

Hardware access to the system with its cameras, pan-tilt-head and robot is very important for robot development environments. For other applications different hardware components may be used. In ImageNets, hardware access to a component is encapsulated in its own block (e. g. acquire an image from a specific camera or move the robot to a defined joint configuration). In the FRIEND reference system, a method based on CORBA communication is used. This enables the execution of an ImageNet at a separate computer that is not directly attached to the specific hardware component. ImageNets can also be easily integrated into other robot software like ROS [12]. Existing systems can easily be made available to ImageNets by placing the corresponding hardware access into a function block. Afterwards, every ImageNets-user can use this hardware.

3.7 Development of Function Blocks and Plug-ins

All algorithms are built with function blocks that are connected to each other to form nets. These blocks must be provided to the modeler of an algorithm by programmers. They and only they must know the required details to implement new blocks. All other users of ImageNets may use the function of available blocks and don't need to know the internal structures (see also Fig. 1).

Figure 4 shows the involved classes to implement a new block and a new plug-in. A new block is derived from CBlock. It holds the core functionality and the interfaces that have to be used by the block programmer. The process method of the interface must be implemented by the block programmer. All functionality has to be placed there. If the block has parameters, also the edit method has to be implemented to enable editing of the parameters.

A set of function blocks is made available to the framework by placing them into a plug-in. The new plug-in is derived from CPluginInterface. It knows all new blocks and can produce a copy of them.

4 Performance Evaluation

In this chapter, the ImageNets framework is compared with other development tools to evaluate whether the advantages of chapter 1 are achieved. On the one hand ImageNets has to support fast development while on the other hand the resulting algorithms, developed with ImageNets, must have high performance.

OpenCV [3] and MATLAB/Simulink [6] have been chosen as comparisons to ImageNets, as they are both very well known to the image processing community and each of them fulfills at least one of the four advantages. Both development tools stand for a group of similar tools. MATLAB/Simulink supports fast development but lacks high performance and OpenCV, on the other side, has high performance but is relatively complex to program. The combination of MATLAB and OpenCV also has to be considered, because it fulfills the first two advantages while adding the disadvantage of having to convert MATLAB code to C++ which also requires some programming knowledge.

The results in Table 4 were achieved using Intel Core 2 Duo computer system with 1.86 GHz and 2 GB RAM. It shows that the high performance of OpenCV and the short development time of MATLAB are both achieved by ImageNets – at least for the very simple image processing example. By using MATLAB with Simulink, the lines of code can also be brought to zero so that a non-programmer can achieve the goal but still the disadvantageous low performance remains. If the combination of MATLAB and OpenCV is used, the efforts sum up. The relatively high development time of OpenCV can be traced back to many compile and test cycles and the syntax and the arguments of used functions had to be referred from the documentation.

Due to the fact that ImageNets uses OpenCV at its core, the comparable execution performance was anticipated. With an increasing number of blocks in an ImageNet, it is expected that the overhead in ImageNets slows down the execution with respect to pure OpenCV. The overhead consists of e. g. checking preconditions of input images and calculating the next executable blocks. In the Sect. 3.5 about multithreading it is shown, that ImageNets can even improve the performance with re-

Table 4 Performance comparison of image processing tools. The task is to load a color image, convert it to gray scale and perform a thresholding operation on it.

	execution time [s]	development time [min]	lines of code
MATLAB	0.10	1	3
MATLAB / SIMULINK	0.10	1	0
OpenCV	0.06	20	30
MATLAB + OpenCV	0.06	21	33
ImageNets	0.06	1	0

spect to OpenCV by making use of widespread multi-core processors without further work for the developer.

4.1 Comparison of ImageNets and Simulink under Realistic Conditions

The lecture Real Time Software Design 2 at University of Bremen focuses in the summer term 2012 on the implementation of real time image processing algorithms.[6] In the attached exercise the lecture's theory is intensified with practical tasks to be solved by the participants. The tools used in the exercises are MATLAB/Simulink and ImageNets. OpenCV was also considered as a third tool to be used but as Table 4 shows OpenCV would not be faster in development of image processing algorithms. Also it is not function block based and is therefore hard to compare directly.

After the exercises the students were asked to compare the two tools and give their preferences regarding *finding* of the blocks for the algorithm, *building* a net with them to model the algorithm, *running* the net and *debugging* the net. To avoid a biased statistic the students have been split into groups of two (one group with three students) with comparable pre-knowledge about the used tools and assigned half of the groups to start with Simulink and the other half with ImageNets. In the second exercise the tasks were the same but the tool was switched. This way the differences coming from pre-knowledge about the tasks and regarding comparing different tasks for the tools were evened out.

Altogether 49 students participated in the exercises. Their preferences are displayed in Fig. 8. The x-axis shows the preference where a value of 0 means that ImageNets is completely preferred and 6 represents a strong preference on Simulink. On the y-axis the amount of students who voted for a specific answer can be seen. A clear preference for ImageNets is visible. Especially finding the required blocks is much easier but also for the other subtasks a preference to Image Nets can be seen. For these preferences it made no difference with which tool the students had started.

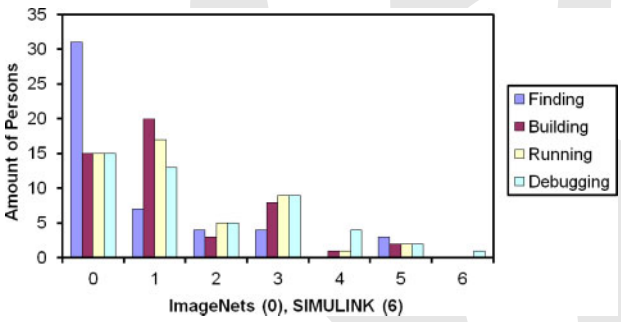

Figure 8 Preference of tool of the students for image processing. Values close to 0 represent a preference for ImageNets and close to 6 for Simulink.

[6] The theory part was read by Prof. Vasile Gui, Temeswar, who was visiting professor in the scheme "Internationalization at home" funded by University of Bremen.

Table 5 Success rate of the exercises.

Tool	Ex. 1	Ex. 2	Mean
ImageNets	70.83%	92.00%	81.42%
Simulink	0.00%	54.17%	27.08%

While the starting tool made no difference in the final preferences, it influenced the difficulty of finishing the tasks. Table 5 shows the success rate of the students to finish the exercises with the different tools. Generally, the second exercise had a higher success rate for each tool since the students could use their pre-knowledge for the tasks from the first exercise. Looking at the change from one tool to the other it can be seen that a change from Simulink to ImageNets results in a big increase of the success rate. The students that switched from ImageNets to Simulink had a significant drop in their success rate. This and the mean over both exercises show clearly the ability of ImageNets to enable an easy access to image processing functionality for inexperienced users to model their own algorithms.

4.2 Development Example

Figure 9 shows a photograph and its mapped virtual reality of a grasping scene where a robot must grasp a book from a shelf. As the color mono camera, which is attached to the gripper, can only perceive 2D images it is difficult to estimate the 3D position of the book, which is essential for the robot to perform grasping.

The idea is to simulate a stereo camera by acquiring images at two viewpoints separated by their adjustable baseline. Subsequently, depth can be calculated based on disparity calculation [13]. With ImageNets, the time from idea to first working version took only 15 minutes, by using a combination of existing blocks and sub-nets. Not a single line of code had to be written.

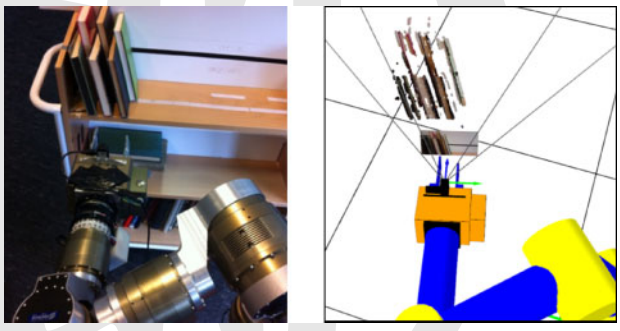

Figure 9 Depth perception of a mono camera by robot movement.

5 Conclusion

With the presented open source framework called ImageNets, the design process of robust robot vision algorithms is speeded up significantly by reusable function blocks and easy to use hardware integration. The

developed 3D environment combines robot vision results with a model of the robot. High run-time performance is achieved by using C++, OpenCV and integrated multi-threading. So far about 20 developers have implemented over 300 blocks for ImageNets, which are directly accessible for any user of the framework. These blocks could be used easily for robust image processing applications in other projects of the robotics community. With the help of the ImageNet Designer, all functions in the plug-ins are directly accessible and grouped by functionality, name and input/output ports.

References

[1] Gräser, A., Kuzmicheva, O., Ristic-Durrant, D., Natarajan, S. K., and Fragkopoulos, C.: Vision-based Control of Assistive Robot FRIEND: Practical Experiences and Design Conclusions, Automatisierungstechnik, 60(5):297–308, 2012.

[2] Prenzel, O., Lange, U., Kampe, H., Martens, C., and Gräser, A.: Programming of Intelligent Service Robots with the Process Model "FRIEND::Process" and Configurable Task-Knowledge; *Robotic Systems – Applications*, 2012, AshishDutta (Ed.) ISBN 978–953-307-941-7, InTech, 2012.

[3] Bradsky, G. and Kaehler, A.: Learning OpenCV Computer Vision with the OpenCV Library, O'Reilly, ISBN 978-0-596-51613-0, 2008.

[4] Telelogic: Rhapsody Open Workshop – Hand-out Documentation, http://www.telelogic.com, 2007.

[5] Pilz: PNOZ multi Configurator, http://www.pilz.de/products/control_communication/safety_relay/f/pnozmulti/s/00258.

[6] The Mathworks: MATLAB & Simulink Video and Image Processing Blockset, http://www.mathworks.de/products/viprocessing.

[7] National Instruments: LabView with image processing plug-in, http://www.ni.com/analysis/lvaddon_vision.htm.

[8] Heckel, F., Schwier, M., and Peitgen, H.-O.: Object Oriented Application Development with MeVisLab and Python, in *Lecture Notes in Informatics (Informatik 2009: Im Focus das Leben)*, 154:1338–1351, 2009.

[9] Microsoft: GraphEdit, http://msdn.microsoft.com/de-de/library/ms787460.aspx.

[10] MVTec: Halcon, http://www.mvtec.com/halcon/.

[11] Grigorescu, S. M.: Robust Machine Vision for Service Robotics, PhD Thesis, Institute of Automation, University of Bremen, Shaker-Verlag, Aachen, 2010.

[12] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y.: ROS: an open-source Robot Operating System, in: Proc. Of ICRA Workshop on Open Source Software, 2009.

[13] Scharstein, D. and Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, International Journal of Computer Vision, 47(1/2/3):7–42, April–June 2002, 2002.

[14] Ristic-Durrant, D. and Gräser, A.: Performance Measure as Feedback Variable in Image Processing, EURASIP J. Adv. Sig. Proc. (EJASP) 2006.

Received: October 5, 2012

Uwe Lange received the B.Sc. and M.Sc. degrees in Systems Engineering from the University of Bremen, Germany, in 2007 and 2008 respectively. He is currently a Ph.D. student at the Institute of Automation, University of Bremen. His research interest is digital image processing for service robotic applications.

Address: Institute of Automation, University of Bremen, Otto-Hahn-Allee, NW1, 28359 Bremen, e-mail: ulange@iat.uni-bremen.de

Henning Kampe received the diploma degree in Electrical Engineering and Information Technology from University of Bremen, Germany, in 2009. Since 2009 he is working as a Ph.D. student at the IAT, University of Bremen, in the field of software-architecture design. His main areas of interest are software development and system design.

Address: Institute of Automation, University of Bremen, Otto-Hahn-Allee, NW1, 28359 Bremen, e-mail: kampe@iat.uni-bremen.de

Prof. Dr.-Ing. Axel Gräser received the diploma in electrical engineering from the University of Karlsruhe, Germany, in 1976 and the Ph.D. degree in control theory from the TH Darmstadt, Germany, in 1982. Since 1994, he has been the Director of the Institute of Automation, University of Bremen, and the Head of the Department of Robotics and Process Automation. His research interests include service robotics, brain robot interface, digital image processing and augmented reality.

Address: Institute of Automation, University of Bremen, Otto-Hahn-Allee, NW1, 28359 Bremen, e-mail: ag@iat.uni-bremen.de